CICS Transaction Server for z/OS
5.4

Performance Guide

IBM

Note

Before using this information and the product it supports, read the information in "Notices" on page 259.

This edition applies to the IBM CICS® Transaction Server for z/OS® Version 5 Release 4 (product number 5655-Y04) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this PDF

This PDF describes how to identify performance constraints, and make adjustments to the operational CICS system and its application programs. Other PDFs, listed below, describe how to address problems with certain areas of CICS and you might need to refer to those as well as this PDF. (In IBM Knowledge Center, all this information is under one section called "Improving performance".) You might also need two companion reference PDFs: *Monitoring Data Reference* and *Statistics Reference*. Before CICS TS V5.4, these two reference PDFs were included in the *Performance Guide* .

Performance information for areas of CICS is in the following PDFs:

- ONC/RPC interface is in the *External Interfaces Guide* .
- Java and Liberty are in *Java Applications in CICS*.
- Front End Programming Interface is in the *Front End Programming Interface User's Guide*.
- DBCTL is in the *IMS DB Control Guide*.
- Shared data tables are in the *Shared Data Tables Guide*.
- Intersystem performance is in *Intercommunication Guide*.

For details of the terms and notation used in this book, see Conventions and terminology used in the CICS documentation in IBM Knowledge Center.

Date of this PDF

This PDF was created on 2024-01-04 (Year-Month-Date).

Chapter 1. Improving performance

To achieve and maintain the best possible performance for your CICS systems, you can tune the environment and CICS settings to best match the requirements of your application workloads.

This information includes general recommendations that are based on current knowledge of CICS. These recommendations cannot be guaranteed to improve the performance of a specific system.

To maximize performance and throughput, as well as tuning your CICS systems, you must ensure that your CICS applications are written or updated to take advantage of performance features within CICS.

Measuring, tuning, and monitoring: the basics

Good performance is the achievement of maximizing the use of your system resources, which helps towards reaching service level agreements efficiently.

About this task

You must consider the performance of a CICS system at the following times:

- When you plan to install a new system
- When you review an existing system
- When you plan major changes to a system

The following procedure shows the principal steps to tune a system.

Procedure

1. Agree what good performance is.
2. Set up performance objectives and decide how you measure them.
3. Measure the performance of the production system.
4. Adjust the system as necessary.
5. Continue to monitor the performance of the system and anticipate future constraints.

24-bit, 31-bit, and 64-bit addressing

How does bittage affect performance?

The zOS architecture uses virtual storage and address spaces. A program running in an address space can reference all of the storage associated with that address space. The original MVS™ (pre zOS) architecture defined storage addresses as 24 bits in length, which allowed an allocation to each user an address space of 16 MB. Later, the addressability of the architecture was extended to 31 bits, which increased the addressability of virtual storage, and the size of the address space, from 16 MB to 2 GB. The new architecture did not require customers to change existing application programs, but provided compatibility for existing programs originally designed to run with 24-bit addressing. Subsequently, with the release of IBM® eServer™ zSeries mainframes in 2000, IBM further extended the addressability of the architecture to 64 bits. With 64-bit addressing, each address space, called a 64-bit address space, is potentially 16 EB in size (an exabyte is slightly more than one billion gigabytes).

Note: Although the size of an address space is potentially 16 EB, z/OS, by default, continues to create address spaces with a size of 2 GB. The address space exceeds this limit only if a program running in it allocates virtual storage above the 2 GB address. If so, z/OS increases the storage available to the user from 2 GB to 16 EB. Programs cannot execute in 64-bit (above-the-bar) storage; it is limited to storing user data for programs that are loaded into the address space below 2 GB.

A program running on z/OS and the zSeries mainframe can run with 24-, 31-, or 64-bit addressing (and can switch among these if needed). To address the high virtual storage available with the 64-bit architecture, the program uses 64-bit-specific instructions, and must be running in 64-bit addressing mode (AMODE(64)). Although the architecture introduces the unique 64-bit instructions, the program can use both 31-bit and 64-bit instructions as needed. In zOS terminology:

- 24-bit storage (up to 16M) is known as below-the-line storage.
- 31-bit storage (16M to 2 GB) is know as above-the-line storage.
- 64-bit storage (2 GB to 16 EB) is know as above-the-bar storage.

CICS transaction flow

This section outlines how CICS processes transactions.

To begin an online session with CICS, you usually begin by "signing on," which is the process that identifies you to CICS. Signing on to CICS gives you the authority to invoke certain transactions. When signed on, you invoke the particular transaction that you intend to use. A CICS transaction is usually identified by a one to four-character transaction identifier or TRANSID, which is defined in a table that names the initial program to be used for processing the transaction.

Application programs are stored in a library on a direct access storage device (DASD) that is attached to the processor. They can be loaded when the system is started or simply loaded as required. If a program is in storage and is not being used, CICS can release the space for other purposes. When the program is next needed, CICS loads a fresh copy of it from the library.

In the time it takes to process one transaction, the system might receive messages from several terminals. For each message, CICS loads the application program (if it is not already loaded) and starts a task to execute it. Thus, multiple CICS tasks can run concurrently.

CICS maintains a separate thread of control for each task. When, for example, one task is waiting to read a disk file, or to get a response from a terminal, CICS can give control to another task. Tasks are managed by the CICS task control program.

CICS manages both multitasking and requests from the tasks themselves for services (of the operating system or of CICS itself). This process allows CICS processing to continue while a task is waiting for the operating system to complete a request on its behalf. Each transaction that is being managed by CICS is given control of the processor when that transaction has the highest priority of those that are ready to run.

While it runs, your application program requests various CICS facilities to handle message transmissions between it and the terminal and to handle any necessary file or database accesses. When the application is complete, CICS returns the terminal to a standby state.

Program Control

A transaction (task) can execute several programs in the course of completing its work.

The program definition contains one entry for every program used by any application in the CICS system. Each entry holds, among other things, the language in which the program is written. The transaction definition has an entry for every transaction identifier in the system, and the important information kept about each transaction is the identifier and the name of the first program to be executed on behalf of the transaction.

These two sets of definitions, transaction, and program work in concert:

1. The user types in a transaction identifier at the terminal (or the previous transaction determined it).
2. CICS looks up this identifier in the list of installed transaction definitions, which tells CICS which program to invoke first.
3. CICS looks up this program in the list of installed transaction definitions, finds out where it is, and loads it (if it is not already in the main storage).
4. CICS builds the control blocks necessary for this particular combination of transaction and terminal using information from both sets of definitions. For programs in command-level COBOL, this includes making a private copy of working storage for this particular execution of the program.

5. CICS passes control to the program, which begins running using the control blocks for this terminal. This program can pass control to any other program in the list of installed program definitions, if necessary, in the course of completing the transaction.

There are two CICS commands for passing control from one program to another. One is the LINK command, which is similar to a CALL statement in COBOL. The other is the XCTL (transfer control) command, which has no COBOL counterpart. When one program links another, the first program stays in main storage. When the second (linked-to) program finishes and gives up control, the first program resumes at the point after the LINK. The linked-to program is considered to be operating at one logical level lower than the program that does the linking.

In contrast, when one program transfers control to another, the first program is considered terminated and the second operates at the same level as the first. When the second program finishes, control is returned not to the first program, but to whatever program last issued a LINK command.

Some people like to think of CICS itself as the highest program level in this process, with the first program in the transaction as the next level down, and so on.

A sound principle of CICS application design is to separate the presentation logic from the business logic. Communication between the programs is achieved by using the LINK command, and data is passed between such programs in the COMMAREA. Such a modular design provides not only a separation of functions, but also much greater flexibility for the web enablement of existing applications using new presentation methods.

CICS functions for monitoring and gathering performance data

You can use CICS statistics, monitoring, and trace facilities to gather and monitor performance data to help you tune your CICS system optimally.

CICS statistics

CICS statistics are the simplest and the most important tool to monitor a CICS system permanently. They collect information about the CICS system as a whole, without regard to tasks.

For more information, see Introduction to CICS statistics.

CICS monitoring

CICS monitoring collects data about the performance of all user and CICS transactions during online processing for later offline analysis.

For more information, see Introduction to CICS monitoring.

CICS trace

For the more complex problems that involve system interactions, you can use the CICS trace to record the progress of CICS transactions through the CICS management modules.

CICS trace provides a history of events leading up to a specific situation.

The CICS trace facilities can also be useful for analyzing performance problems such as excessive waiting on events in the system, or constraints resulting from inefficient system setup or application program design.

For more information, see CICS trace.

CICS tools for monitoring and gathering performance data

You can use CICS tools to gather and monitor performance data to help you tune your CICS systems.

CICS Performance Analyzer for z/OS (CICS PA)

CICS Performance Analyzer is a performance reporting and analysis tool that provides information on the performance of your CICS systems and applications, and helps you tune, manage, and plan your CICS systems effectively. CICS PA provides historical performance and statistical information about CICS systems and applications using data collected from CICS and any connected sub-systems including DB2, IMS, and MQ.

CICS PA can be used to produce comprehensive performance reporting and analysis help to evaluate CICS system efficiency, eliminate system bottlenecks and proactively tune system performance including threadsafe analysis. It provides over 250 pre-defined, customizable reports, along with the facilities to create your own reports and extracts.

The CICS PA plug-in for CICS Explorer integrates performance data into CICS Explorer and allows you to, for example, right-click CICS resource in the CICS Explorer and display historical performance information about it.

For more information about CICS PA, see CICS Performance Analyzer for z/OS.

CICS Interdependency Analyzer for z/OS (CICS IA)

CICS Interdependency Analyzer is a productivity tool used to discover and analyze CICS resources and identify relationships between them.

CICS IA dynamically discovers runtime relationships among key resources within your CICS system. It does this by monitoring applications for API and SPI commands along with optional DB2, IMS, MQ and COBOL calls to give you a complete picture of your application and the interactions and resources that are referenced along with their inter-relationship. CICS IA can help you analyze applications for Threadsafe considerations to improve the overall execution efficiency.

CICS IA stores its data in a DB2 database and this can be accessed offline for querying and reporting by the CICS IA reporter component. CICS IA also provides a CICS Explorer plug-in, which allows you to, for example, right-click a CICS resource in the CICS Explorer and display all the resources related to it.

The CICS IA Command Flow feature gives you the ability to capture and view all EXEC CICS, SQL, MQ, and IMS calls in chronological order. This feature can be helpful in diagnosing the flow of your application as it executes along with the ability to identify TCB mode switches to help with tuning and Threadsafe analysis.

For more information about CICS IA, see CICS Interdependency Analyzer for z/OS

Other tools for obtaining performance data

You can use a number of tools that are not provided by CICS to provide performance-related information to help you optimally tune your CICS system.

The IBM Redbooks® publication ABCs of z/OS System Programming contains information about capacity planning, performance management, RMF, and SMF.

Resource measurement facility (RMF)

The resource measurement facility (RMF) collects system-wide data that describes the processor activity (WAIT time), I/O activity (channel and device usage), main storage activity (demand and swap paging statistics), and system resources manager (SRM) activity (workload).

RMF is a centralized measurement tool that monitors system activity to collect performance and capacity planning data. The analysis of RMF reports provides the basis for tuning the system to user requirements. They can also be used to track resource usage.

RMF measures the following activities:

- Processor usage
- Address space usage
- Channel activity:
  - Request rate and service time per physical channel
  - Logical-to-physical channel relationships
  - Logical channel queue depths and reasons for queuing.
- Device activity and contention for the following devices:
  - Unit record
  - Graphics
  - Direct-access storage
  - Communication equipment
  - Magnetic tapes
  - Character readers.
- Detailed system paging
- Detailed system workload
- Page and swap data set
- Enqueue
- CF activity
- XCF activity.

RMF allows the z/OS user to:

- Evaluate system responsiveness:
  - Identify bottlenecks. The detailed paging report associated with the page and swap data set activity can give a good picture of the behavior of a virtual storage environment.
- Check the effects of tuning:
  - Results can be observed dynamically on a screen or by postprocessing facilities.
- Perform capacity planning evaluation:
  - The workload activity reports include the interval service broken down by key elements such as processor, input/output, and main storage service.
  - Analysis of the resource monitor output (for example, system contention indicators, swap-out broken down by category, average ready users per domain) helps in understanding user environments and forecasting trends.
  - The post-processing capabilities make the analysis of peak load periods and trend analysis easier.
- Manage the larger workloads and increased resources that MVS can support
- Identify and measure the usage of online channel paths

For more information about RMF, see the IBM Redbooks publication ABCs of z/OS System Programming and z/OS Resource Measurement Facility (RMF) User's Guide.

Tivoli OMEGAMON XE for CICS on z/OS

Tivoli® OMEGAMON® XE for CICS on z/OS helps you to proactively manage performance and availability of complex CICS systems.

Tivoli OMEGAMON XE for CICS on z/OS (OMEGAMON XE for CICS on z/OS) is a remote monitoring agent that runs on z/OS managed systems. It assists you in anticipating performance problems and warns you when critical events take place in your CICS environments. You can set threshold levels and flags to alert you when events within your CICS regions reach critical points.

When running under the Tivoli Enterprise Portal, IBM Tivoli OMEGAMON XE for CICS on z/OS offers a central point of management for CICS Transaction Server and provides a comprehensive means for gathering the information you need to detect and prevent problems within your CICS regions. You view data that Tivoli Enterprise Portal gathers in tables and charts that show you the status of your managed CICS regions.

With this data you can perform a number of tasks:

- Collect and analyze reliable, up-to-the-second data that allows you to make faster, better informed, operating decisions
- Manage all CICS regions from a single point to identify problems at any time
- Balance workloads across various regions
- Track performance against goals

With OMEGAMON XE for CICS on z/OS, systems administrators can set threshold levels and flags to alert them when system conditions reach these thresholds. These are the advanced monitoring facilities:

- User-defined and predefined situations based on thresholds to raise different types of alerts
- At-a-glance status of all CICS regions
- The capability to monitor multiple CICS regions simultaneously from one or more centralized workstations

Used in conjunction with other OMEGAMON XE monitoring products, the data, analyses, and alerts presented by OMEGAMON XE for CICS on z/OS can help you develop an overall view of the health of your entire computing enterprise from a single console.

For more information about OMEGAMON XE for CICS on z/OS, see IBM Tivoli OMEGAMON XE for CICS on z/OS

Tivoli Decision Support for z/OS

Tivoli Decision Support for z/OS is an IBM product that collects and analyzes data from CICS and other IBM systems and products.

The reports generated by Tivoli Decision Support are useful when:

- Getting an overview of the system
- Ensuring service levels are maintained
- Ensuring availability
- Performance tuning
- Capacity planning
- Managing change and problems
- Accounting

A large number of ready-made reports are available, and in addition you can generate your own reports to meet specific needs.

In the reports Tivoli Decision Support uses data from CICS monitoring and statistics. Tivoli Decision Support also collects data from the MVS system and from products such as RMF, TSO, IMS and NetView®. This means that data from CICS and other systems can be shown together, or can be presented in separate reports.

Reports can be presented as plots, bar charts, pie charts, tower charts, histograms, surface charts, and other graphic formats. Tivoli Decision Support for z/OS passes the data and formatting details to Graphic Data Display Manager (GDDM) which does the rest. Tivoli Decision Support can also produce line graphs and histograms using character graphics where GDDM is not available, or the output device does not support graphics. For some reports, where you need the exact figures, numeric reports such as tables and matrices are more suitable.

To use Tivoli Decision Support for z/OS to report on CICS performance, you will need to use the Tivoli Decision Support for z/OS CICS performance feature. For more information about Tivoli Decision Support for z/OS and the Tivoli Decision Support for z/OS CICS performance feature, see IBM Tivoli Decision Support for z/OS.

Performance monitoring and review

CICS performance can be monitored, measured, and analyzed by implementing a strategy that best suits your needs.

You can use a number of monitoring techniques to set your performance objectives, and analyze CICS performance.

Establishing monitoring activities and techniques

Establishing an ongoing strategy involving monitoring activities and monitoring techniques provides an understanding of your CICS production system that helps to ensure optimum performance and avoid unexpected problems.

Monitoring is used to describe regular checking of the performance of a CICS production system, against objectives, by the collection and interpretation of data. *Analysis* describes the techniques used to investigate the reasons for performance deterioration. *Tuning* can be used for any actions that result from this analysis.

Monitoring is an ongoing activity for a number of reasons:

- It can establish transaction profiles (that is, workload and volumes) and statistical data for predicting system capacities
- It can give early warning through comparative data to avoid performance problems
- It can measure and validate any tuning you might have done in response to an earlier performance problem.

A performance history database (see "Tivoli Decision Support for z/OS" on page 34 for an example) is a valuable source from which to answer questions on system performance, and to plan further tuning.

Monitoring can be described in terms of strategies, procedures, and tasks.

Strategies include these elements:

- Continuous or periodic summaries of the workload. You can track all transactions or selected representatives.
- Snapshots at normal or peak loads. Monitor peak loads for these reasons:
  - Constraints and slow responses are more pronounced at peak volumes.
  - The current peak load is a good indicator of the future average load.

Procedures, such as good documentation practices, provide a management link between monitoring strategies and tasks.

Tasks (not to be confused with the task component of a CICS transaction) include:

- Running one or more of the tools; see "Performance measurement tools" on page 21
- Collating the output
- Examining it for trends

Allocate responsibility for these tasks between operations personnel, programming personnel, and analysts. Identify the resources that are to be regarded as critical, and set up a procedure to highlight any trends in the use of these resources.

Because the tools require resources, they can disturb the performance of a production system.

Give emphasis to peak periods of activity, for both the new application and the system as a whole. Run the tools more frequently at first if required to confirm that the expected peaks correspond with the actual ones.

It is often not practical to keep all the detailed output. File summarized reports with the corresponding CICS statistics, and hold output from the tools for an agreed period, with customary safeguards for its protection.

Do not base conclusions on one or two snapshots of system performance, but rather on data collected at different times over a prolonged period. Emphasise peak loading. Because different tools use different measurement criteria, early measurements might give apparently discrepant results.

Plan your monitoring procedures ahead of time. In your procedures, explain the tools to be used, the analysis techniques to be used, the operational extent of those activities, and how often they are to be performed.

Developing monitoring activities and techniques

To collect and analyze data that is consistent with your strategy, you must have the correct tools and processes in place. When you are developing a main plan for monitoring and performance analysis, consider these points:

- Establish a main schedule of monitoring activity. Coordinate monitoring with operations procedures to allow for feedback of online events and instructions for daily or periodic data gathering.
- Consider your business in relation to system performance, for example, what will be the growth of transaction rates and changes in the use of applications and future trends. Consider the effects of nonperformance system problems such as application abends, frequent problems, and excessive attempts.
- Decide on the tools to be used for monitoring. The tools used for data gathering must provide for dynamic monitoring, daily collection of statistics, and more detailed monitoring. See “Planning your monitoring schedule” on page 9 for more information.
- Consider the kinds of analysis to be performed. Take into account any controls you have already established for managing the installation. Document what data is to be extracted from the monitoring output, identifying the source and usage of the data. Although the formatted reports provided by the monitoring tools help to organize the volume of data, design worksheets to assist in data extraction and reduction.
- Compose a list of the personnel who are to be included in any review of the findings. The results and conclusions from analyzing monitor data should be shared with the user liaison group and system performance specialists.
- Create a strategy for implementing changes to the CICS system design resulting from tuning recommendations. Incorporate the recommendations into installation management procedures, and include items such as standards for testing and the permitted frequency of changes to the production environment.

Planning the performance review process

A plan of the performance review process includes a checklist of the tools and analysis that are required to implement monitoring procedures. Establish a simple schedule for monitoring procedures. To create a performance review process, perform the following tasks:

- List the CICS requests made by each type of task. This helps you decide which requests or which resources (the high-frequency or high-cost ones) need to be looked at in statistics and CICS monitoring facility reports.
- Create a checklist of review questions.
- Estimate resource usage and system loading for new applications. This is to enable you to set an initial basis from which to start comparisons.

Planning your monitoring schedule

A comprehensive monitoring plan includes the scheduling of various system activities at different time intervals. This approach provides a broad collection of data to measure and analyze the performance your CICS system. Plan for both dynamic monitoring and scheduled monitoring.

Dynamic monitoring

Dynamic monitoring is â€œon-the-spotâ€ monitoring that you can carry out at all times. This type of monitoring includes the following activities:

- Observing the operation of the system continuously to discover any serious short-term deviation from performance objectives. End-user feedback is essential for this activity. You can also use the Resource Measurement Facility (RMF) to collect information about processor, channel, coupling facility, and I/O device usage.
- Obtaining status information. You can get status information about system processing during online execution. This information might include the queue levels, active regions, active terminals, and the number and type of conversational transactions. You can get this information with the aid of an automated program started by the main terminal operator. At prearranged times in the production cycle (such as before scheduling a message, at shutdown of part of the network, or at peak loading), the program can capture the transaction processing status and measurements of system resource levels.
- Using CICSPlex® SM monitoring data. CICSPlex SM can accumulate information produced by the CICS monitoring facility to assist in dynamic monitoring activities. The data can then be immediately viewed online, giving instant feedback on the performance of the transactions. CICS monitoring must be active for CICSPlex SM to collect CICS monitoring information.

Daily monitoring

Measure and record key system parameters by monitoring data daily. The daily monitoring of data usually consists of counts of events and gross level timings. In some cases, the timings are averaged for the entire CICS system. To monitor data daily, perform a series of tasks. For example:

- Record both the daily average and the peak period (usually one hour) average of items such as messages, tasks, processor usage, I/O events, and storage used. Compare these events against your major performance objectives and look for adverse trends.
- List the CICS-provided statistics at the end of every CICS run. Date-stamp and time-stamp the data that is provided, and file it for later review. For example, in an installation that has settled down, you might review daily data at the end of the week; generally, you can carry out reviews less frequently than collection, for any one type of monitoring data. If you know there is a problem, you might increase the frequency; for example, reviewing daily data as soon as it becomes available.
- Be familiar with all the facilities in CICS for providing statistics at times other than at shutdown. The main facilities are invocation from a terminal (with or without reset of the counters) and automatic time-initiated requests.
- File an informal note of any incidents reported during the run, including, for example, a shutdown of CICS that causes a gap in the statistics, a complaint from your users of poor response times, a terminal going out of service, or any other significant item. These notes are useful when reconciling disparities in detailed performance figures that might be discovered later.
- Print the system console log for the period when CICS was active, and file a copy of the console log in case it becomes necessary to review the CICS system performance in the light of the concurrent batch activity.
- Run one of the performance analysis tools described in "Performance measurement tools" on page 21 for at least part of the day if there is any variation in load. File the summaries of the reports produced by the tools you use.
- Transcribe onto a graph any items identified as being consistently heavily used in the post-development review phase.
- Collect CICS statistics, monitoring data, and RMF data into the Tivoli Decision Support database.

Weekly monitoring

Periodically collect detailed statistics on the operation of your system for comparison with your system-oriented objectives and workload profiles. To monitor data weekly, perform these steps:

- Run the CICS monitoring facility with performance class active, and process it. You might not need to run the facility every day, but it is important to do it regularly and to keep the sorted summary output and the detailed reports. Whether you run the facility on the same day of the week depends on the nature of the system load. For example, if one day of the week has a heavier system load than others, monitor on this day. Bear in mind, however, that the use of the monitoring facility causes additional load, particularly with performance class active.
- If the load is apparently the same each day, run the CICS monitoring facility daily for a period sufficient to confirm the load. If there really is little difference from day to day in the CICS load, check the concurrent batch loads in the same way from the logs. Checking the batch loads helps you identify any obscure problems because of peak volumes or unusual transaction mixes on specific days of the week. The first few weeks of output from the CICS statistics also provide useful information.You might not need to review the detailed monitor report output every time, but always keep this output in case the summary data is insufficient to answer questions raised by the statistics or by user comments. Label the CICS monitoring facility output and keep it for an agreed period in case further investigations are required.
- Run RMF, because this shows I/O use, channel use, and other uses. File the summary reports and archive the output information for some agreed period.
- Review the CICS statistics, and any incident reports.
- Review the graph of critical parameters. If any of the items is approaching a critical level, check the performance analysis and RMF output for more detail.
- Tabulate or produce a graph of values as a summary for future reference.
- Produce weekly Tivoli Decision Support or CICS Performance Analyzer reports.

Monthly monitoring

Monitor and assess trends that are better reflected when tracked regularly over a longer period of time. The following list includes some tasks for monitoring data on a monthly basis:

- Run RMF.
- Review the RMF and performance analysis listings. If there is any indication of excessive resource usage, follow any previously agreed procedures (for example, notify your management), and do further monitoring.
- Date-stamp and time-stamp the RMF output and keep it for use in case performance problems start to arise. You can also use the output in making estimates, when detailed knowledge of component usage might be important. The RMF output provides detailed data on the usage of resources within the system, including processor usage, use of DASD, and paging rates.
- Produce monthly Tivoli Decision Support reports showing long-term trends.

Monitoring for the future

When performance is acceptable, establish procedures to monitor system performance measurements and anticipate performance constraints before they become response-time problems. Exception-reporting procedures are a key to an effective monitoring approach. In a complex production system there is often too much performance data for it to be comprehensively reviewed every day. Key components of performance degradation can be identified with experience, and those components are the ones to monitor most closely. Identify trends of usage and other factors (such as batch schedules) to aid in this process.

Typical performance review questions

Use the following questions as a basis for your own checklist when carrying out a review of performance data. Many of these questions can be answered by performance reporting packages such as CICS Performance Analyzer or Tivoli Decision Support for z/OS.

Some of the questions are not strictly to do with performance. For instance, if the transaction statistics show a high frequency of transaction abends with usage of the abnormal condition program, there might be sign-on errors and, therefore, a lack of terminal operator training. This situation is not a performance problem, but is an example of the additional information that can be provided by monitoring.

1. What are the characteristics of your transaction workload?

   a. Has the frequency of use of each transaction identifier altered?

   b. Does the mix vary from one time of the day to another?

   c. Should statistics be requested more frequently during the day to verify this?

   A different approach must be taken:

   - In systems where all messages are channeled through the same initial task and program (for user security routines, initial editing or formatting, statistical analysis, and so on)
   - For conversational transactions, where a long series of message pairs is reflected by a single transaction
   - In transactions where the amount of work done relies heavily on the input data.

   In these cases, you must identify the function by program or data set usage, with appropriate reference to the CICS program statistics, file statistics, or other statistics. In addition, you might be able to put user tags into the monitoring data (for example, a user character field in the case of the CICS monitoring facility), which can be used as a basis for analysis by products such as CICS Performance Analyzer for z/OS, or Tivoli Decision Support for z/OS.
2. What is the usage of the telecommunication lines?

   a. Do the CICS terminal statistics indicate any increase in the number of messages on the terminals on each of the lines?

   b. Does the average message length on the CICS performance class monitor reports vary for any transaction type? This can easily happen with an application where the number of lines or fields output depends on the input data.

   c. Is the number of terminal errors acceptable? If you are using a terminal error program or node error program, are there any line problems?
3. What is the DASD usage?

   a. Is the number of requests to file control increasing? Remember that CICS records the number of logical requests made. The number of physical I/O operations depends on the configuration of indexes, and on the data records per control interval and the buffer allocations.

   b. Is intrapartition transient data usage increasing? Transient data involves a number of I/O operations depending on the queue mix. Review the number of requests made to see how it compares with previous runs.

   c. Is auxiliary temporary storage usage increasing? Temporary storage uses control interval access, but writes the control interval out only at sync point or when the buffer is full.
4. What is the virtual storage usage?

   a. How large are the dynamic storage areas?

   b. Is the number of GETMAIN requests consistent with the number and types of tasks?

   c. Is the short-on-storage (SOS) condition being reached often?

   d. Have any incidents been reported of tasks being purged after deadlock timeout interval (DTIMOUT) expiry?

   e. How much program loading activity is there?

   f. From the monitor report data, is the use of dynamic storage by task type as expected?
   g. Is storage usage similar at each execution of CICS?
   h. Are there any incident reports showing that the first invocation of a function takes a lot longer than subsequent ones? This situation can occur if programs are loaded that then need to open data sets, particularly in IMS, for example. Can a change in application design rectify the problem?
5. What is the processor usage?
   a. Is the processor usage as measured by the monitor report consistent with previous observations?
   b. Are batch jobs that are planned to run, able to run successfully?
   c. Is there any increase in usage of functions running at a higher priority than CICS? Include MVS readers and writers, MVS JES, and z/OS Communications Server if running above CICS, and overall I/O, because of the lower-priority regions.
6. What is the coupling facility usage?
   a. What is the average storage usage?
   b. What is the link utilization?
7. Do any figures indicate design, coding, or operational errors?
   a. Are any of the resources heavily used? If so, was this situation expected at design time? If not, can the heavy usage be explained in terms of heavier usage of transactions?
   b. Is the heavy usage associated with a particular application? If so, is there evidence of planned growth or peak periods?
   c. Are browse transactions issuing more than the expected number of requests? In other words, is the count of browse requests issued by a transaction greater than what you expected users to cause?
   d. Is the CICS CSAC transaction (provided by the DFHACP abnormal condition program) being used frequently? If so, is this occurring because invalid transaction identifiers are being entered? For example, errors are signaled if transaction identifiers are entered in lowercase on IBM 3270 terminals but automatic translation of input to uppercase has not been specified.

      A high use of the DFHACP program without a corresponding count of CSAC could indicate that transactions are being entered without correct operator signon. This situation might indicate that some terminal operators need more training in using the system.

In addition, review regularly certain items in the CICS statistics, such as:

- Times the MAXTASK limit is reached (transaction manager statistics)
- Peak tasks (transaction class statistics)
- Times cushion is released (storage manager statistics)
- Storage violations (storage manager statistics)
- Maximum number of RPLs posted (z/OS Communications Server statistics)
- Short-on-storage count (storage manager statistics)
- Wait on string total (file control statistics)
- Use of DFHSHUNT log streams
- Times auxiliary storage is exhausted (temporary storage statistics)
- Buffer waits (temporary storage statistics)
- Times string wait occurred (temporary storage statistics)
- Times NOSPACE occurred (transient data global statistics)
- Intrapartition buffer waits (transient data global statistics)
- Intrapartition string waits (transient data global statistics)
- Times the MAXSOCKETS limit is reached (TCP/IP statistics)
- Pool thread waits (DB2® connection statistics)

Review the effects of and reasons for system outages and their duration. If there is a series of outages, there might be a common cause.

CICS performance analysis techniques

A number of techniques are available for analyzing CICS performance.

There are four main uses for performance analysis:

- You currently have no performance problems, but you want to adjust the system to give better performance.
- You want to characterize and calibrate individual stand-alone transactions as part of the documentation of those transactions, and for comparison with some future time when, perhaps, they start behaving differently.
- A system is departing from previously identified objectives, and you want to find out precisely where and why. Although an online system might operate efficiently when it is installed, the characteristics of the system usage can change and the system might not run so efficiently. This inefficiency can usually be corrected by adjusting various controls. Some adjustments usually need to be made to any new system when it goes live.
- A system might or might not have performance objectives, but it appears to be suffering severe performance problems.

If the current performance does *not* meet your needs, consider tuning the system. To tune your system, you must perform the following tasks:

1. Identify the major constraints in the system.
2. Understand what changes could reduce the constraints, possibly at the expense of other resources. Tuning is usually a trade-off of one resource for another.
3. Decide which resources could be used more heavily.
4. Adjust the parameters to relieve the constrained resources.
5. Review the performance of the resulting system in the light of these criteria:
   - Your existing performance objectives
   - Progress so far
   - Tuning effort so far
6. Stop at this point if performance is acceptable; otherwise do one of the following actions:
   - Continue tuning
   - Add suitable hardware capacity
   - Reduce your system performance objectives.

The tuning tasks can be expressed in flowchart form as follows:

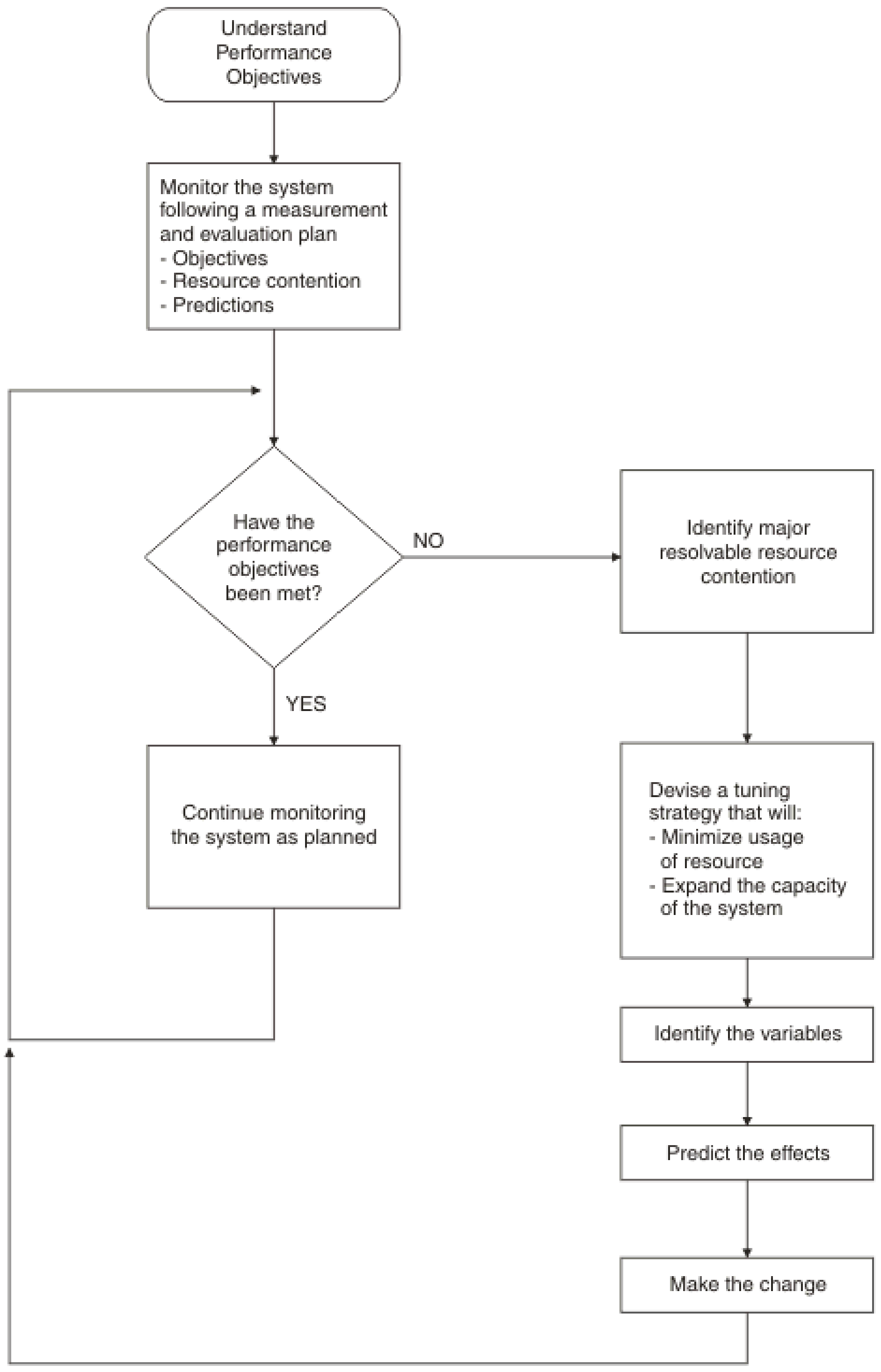

Understand Performance Objectives
Monitor the system following a measurement and evaluation plan
- Objectives
- Resource contention
- Predictions
Have the performance objectives been met?
NO
YES
Identify major resolvable resource contention
Continue monitoring the system as planned
Devise a tuning strategy that will:
- Minimize usage of resource
- Expand the capacity of the system
Identify the variables
Predict the effects
Make the change

Figure 1. Flowchart to show rules for tuning performance

What to investigate when analyzing performance

Always start by looking at the overall system before you decide that you have a specific CICS problem. Check total processor usage, DASD activity, and paging.

Performance degradation is often due to application growth that has not been matched by corresponding increases in hardware resources. If so, solve the hardware resource problem first. You might still need to follow on with a plan for multiple regions.

Information from at least three levels is required:

1. *CICS*: Examine the CICS interval or end-of-day statistics for exceptions, queues, and other symptoms that suggest overloads on specific resources. A shorter reporting period can isolate a problem. Consider software and hardware resources; for example, utilization of VSAM strings or database threads, files, and TP lines. Check runtime messages that are sent to the console and to transient data destinations, such as CSMT and CSTL, for persistent application problems and network errors.

   Use tools such as the CICS Explorer® and RMF, to monitor the online system and identify activity that correlates to periods of bad performance. Collect CICS monitoring facility history and analyze it, using tools such as CICS Performance Analyzer or Tivoli Decision Support to identify performance and resource usage exceptions and trends. For example, note processor-intensive transactions that perform little or no I/O. These transactions can monopolize the processor, causing erratic response in other transactions with more normally balanced activity profiles. These transactions might be candidates for isolation in another CICS region.
2. *MVS*: Use SMF data to discover any relationships between periods of bad CICS performance and other concurrent activity in the MVS system. Use RMF data to identify overloaded devices and paths. Monitor CICS region paging rates to make sure that there is sufficient real storage to support the configuration.
3. *Network*: The proportion of response time spent in the system is small compared with transmission delays and queuing in the network. Use tools such as Tivoli NetView for z/OS to identify problems and overloads in the network. Without automatic tools, you are dependent on the subjective opinions of a user that performance has deteriorated.

In CICS, the performance problem is either a poor response time or an unexpected and unexplained high use of resources. In general, you must look at the system in some detail to see why tasks are progressing slowly through the system, or why a given resource is being used heavily. The best way of looking at detailed CICS behavior is by using CICS auxiliary trace. But note that switching on auxiliary trace, though the best approach, can worsen existing poor performance while it is in use.

The approach is to get a picture of task activity first, listing only the task traces, and then to focus on particular activities: specific tasks, or a specific time interval. For example, for a response time problem, you might want to look at the detailed traces of one task that is observed to be slow. There might be a number of possible reasons; for example, the tasks might be trying to do too much work for the system, or the system is real-storage constrained, or many of the CICS tasks are waiting because there is contention for a particular function.

Information sources to help analyze performance

Potentially, any performance measurement tool, including statistics and the CICS monitoring facility, can help in diagnosing problems. Consider each performance tool as usable in some degree for each purpose: monitoring, single-transaction measurement, and problem determination. CICS statistics can reveal heavy use of a particular resource. For example, you might find a large allocation of temporary storage in main storage, a high number of storage control requests per task (perhaps 50 or 100), or high program use counts that imply heavy use of program control LINK.

Both statistics and CICS monitoring might show exceptional conditions arising in the CICS run. Statistics can show waits on strings, waits for VSAM shared resources, waits for storage in GETMAIN requests, and other waits. These waits also generate CICS monitoring facility exception class records.

While these conditions are also evident in CICS auxiliary trace, they might not be obvious, and the other information sources are useful in directing the investigation of the trace data.

In addition, you can gain useful data from the investigation of CICS outages. If there is a series of outages, investigate common links between the outages.

Establishing a measurement and evaluation plan

For some installations, a measurement and evaluation plan might be suitable. A measurement and evaluation plan is a structured way to measure, evaluate, and monitor the performance of the system.

To set up a measurement and evaluation plan, perform the following steps:

1. Devise the plan.
2. Review the plan.

3. Implement the plan.
4. Revise and upgrade the plan as necessary.

To use the plan, perform the following major activities:

- Collect information periodically to determine:
  - Whether objectives have been met
  - Transaction activity
  - Resource utilization
- Summarize and analyze the information. For this activity:
  - Plot volumes and averages on a chart at a specified frequency
  - Plot resource utilization on a chart at a specified frequency
  - Log unusual conditions on a daily log
  - Review the logs and charts weekly
- Make or recommend changes if objectives have not been met.
- Relate past, current, and projected transaction activity and resource utilization to determine if objectives continue to be met, and whether resources are being used beyond an efficient capacity.
- Keep interested parties informed with informal reports, written reports, and monthly meetings.

A typical measurement and evaluation plan might include the following items as objectives, with statements of recording frequency and the measurement tool to be used:

- Volume and response time for each department
- Network activity:
  - Total transactions
  - Tasks per second
  - Total by transaction type
  - Hourly transaction volume (total, and by transaction)
- Resource utilization examples:
  - DSA utilization
  - Processor utilization with CICS
  - Paging rate for CICS and for the system
  - Channel utilization
  - Device utilization
  - Data set utilization
  - Line utilization
- Unusual conditions:
  - Network problems
  - Application problems
  - Operator problems
  - Transaction count for entry to transaction classes
  - SOS occurrences
  - Storage violations
  - Device problems (not associated with the communications network)
  - System outage
  - CICS outage time

Assessing the performance of your system

The following performance measurements can be helpful in determining the performance of a system: processor usage, I/O rates, terminal message or data set record block sizes, paging rates, and error rates.

Processor usage
: This item reflects how active the processor is. Although the central processor is of primary concern, 37X5 communications controllers and terminal control units can also increase response time if they are heavily used.

I/O rates
: These rates measure the amount of access to a disk device or data set over a given period. Again, acceptable rates vary depending on the speed of the hardware and response time requirements.

Terminal message or data set record block sizes
: These factors, when combined with I/O rates, provide information about the current load on the network or DASD subsystem.

Indications of internal virtual storage limits
: These indications vary by software component, including storage or buffer expansion counts, system messages, and program abends because of system stalls. In CICS, program fetches on nonresident programs and system short-on-storage or stress messages reflect this condition.

Paging rates
: CICS can be sensitive to a real storage shortage, and paging rates reflect this shortage. Acceptable paging to DASD rates vary with the speed of the DASD and response time criteria.

Error rates
: Errors can occur at any point in an online system. If the errors are recoverable, they can go unnoticed, but they put an additional load on the resource on which they are occurring.

Investigate both system conditions and application conditions.

System conditions

A knowledge of the following conditions can help you evaluate the performance of the system as a whole:

- System transaction rate (average and peak)
- Internal response time and terminal response time, preferably compared with transaction rate
- Working set, at average and peak transaction rates
- Average number of disk accesses per unit time (total, per channel, and per device)
- Processor usage, compared with transaction rate
- Number of page faults per second, compared with transaction rate and real storage
- Communication line usage (net and actual)
- Average number of active CICS tasks
- Number and duration of outages

Application conditions

Application conditions, measured both for individual transaction types and for the total system, give you an estimate of the behavior of individual application programs. Gather data for each main transaction, and average values for the total system. This includes the following data:

- Program calls per transaction
- CICS storage GETMAIN and FREEMAIN requests (number and amount)
- Application program and transaction usage
- File control (data set, type of request)
- Terminal control (terminal, number of inputs and outputs)
- Transaction routing (source, target)

- Function shipping (source, target)
- Other CICS requests

Methods of performance analysis

You can use two methods for performance analysis: measuring a system under full production load (*full-load* measurement), to get all information that is measurable only under high system-loading, and measuring single-application transactions (*single-transaction* measurement), during which the system must not carry out any other activities.

Because a system can have various problems, it is not possible to recommend which option to use to investigate the behavior of a system. When in doubt about the extent of a problem, always use both methods.

Rapid performance degradation often occurs after a threshold is exceeded and the system approaches its ultimate load. You can see various indications only when the system is fully loaded (for example, paging, short-on-storage condition in CICS, and so on), and you should usually plan for a full-load measurement.

The IBM Redbooks publication ABC's of z/OS System Programming, Volume 11 contains further information about performance analysis methods.

Performance analysis: Full-load measurement

A full-load measurement highlights latent problems in the system. It is important that you take the measurement when, from production experience, the peak load is reached.

Many installations have a peak load for about one hour in the morning and again in the afternoon. CICS statistics and various performance tools can provide valuable information for full-load measurement. In addition to the overall results of these tools, it might be useful to have the CICS auxiliary trace or RMF active for about 1 minute.

CICS auxiliary trace

CICS auxiliary trace can be used to find situations that occur under full load. For example, all ENQUEUE operations that cannot immediately be honored in application programs result in a suspension of the issuing task. If this situation happens frequently, attempts to control the system by using the main transaction are not effective.

Trace is a heavy overhead. Use trace selectivity options to minimize this overhead.

RMF

It is advisable to do the RMF measurement without any batch activity.

For full-load measurement, the system activity report and the DASD activity report are important.

The most important values for full-load measurement are as follows:

- Processor usage
- Channel and disk usage
- Disk unit usage
- Overlapping of processor with channel and disk activity
- Paging
- Count of start I/O operations and average start I/O time
- Response times
- Transaction rates.

Expect stagnant throughput and sharply climbing response times as the processor load approaches 100%.

It is difficult to forecast the system paging rate that can be achieved without serious detriment to performance, because too many factors interact. Observe the reported paging rates; note that short-duration severe paging leads to a rapid increase in response times.

In addition to taking note of the count of start I/O operations and their average length, find out whether the system is waiting on one device only. With disks, for example, it can happen that several frequently accessed data sets are on one disk and the accesses interfere with each other. In each case, investigate whether a system wait on a particular unit could not be minimized by reorganizing the data sets.

The RMF DASD activity report includes the following information:

- A summary of all disk information
- Per disk, a breakdown by system number and region
- Per disk, the distribution of the seek arm movements
- Per disk, the distribution of accesses with and without arm movement.

Use the IOQ(DASD) option in RMF monitor 1 to show DASD control unit contention.

After checking the relationship of accesses with and without arm movement, for example, you might want to move to separate disks those data sets that are periodically frequently accessed.

Comparison charts
Consider using a comparison chart to measure key aspects of your system performance before and after tuning changes have been made. A suggested chart is as follows:

Table 1. Comparison chart

Observations to make		**Run A**	**Run B**	**Run C**	**Run D**
DL/I transactions	Number				
DL/I transactions	Response				
VSAM transactions	Number				
VSAM transactions	Response				
Response times	DL/I				
Response times	VSAM				
Most heavily used transaction	Number				
Most heavily used transaction	Response				
Average-use transaction	Number				
Average-use transaction	Response				
Paging rate	System				
Paging rate	CICS				
DSA virtual storage	Maximum				
DSA virtual storage	Average				
Tasks	Peak				
Tasks	At MXT				
Most heavily used DASD	Response				
Most heavily used DASD	Utilization				
Average-use DASD	Response				

Table 1. Comparison chart (continued)

Observations to make		Run A	Run B	Run C	Run D
Average-use DASD	Utilization				
CPU utilization					

This type of comparison chart requires the use of TPNS, RMF, and CICS interval statistics running together for about 20 minutes, at a peak time for your system. It also requires you to identify the following items:

- A representative selection of terminal-oriented DL/I transactions accessing DL/I databases
- A representative selection of terminal-oriented transactions processing VSAM files
- The most heavily used transaction
- Two average-use nonterminal-oriented transactions writing data to intrapartition transient data destinations
- The most heavily used volume in your system
- A representative average-use volume in your system

To complete the comparison chart for each CICS run before and after a tuning change, you can obtain the figures from the following sources:

- *DL/I transactions*: Identify a selection of terminal-oriented DL/I transactions accessing DL/I databases.
- *VSAM transactions*: Identify a selection of terminal-oriented transactions processing VSAM files.
- *Response times*: External response times are available from the TPNS terminal response time analysis report; internal response times are available from RMF. The “DL/I” subheading is the average response time calculated at the 99th percentile for the terminal-oriented DL/I transactions you have previously selected. The “VSAM” subheading is the average response time calculated at the 99th percentile for the terminal-oriented VSAM transactions you have previously selected.
- *Paging rate (system)*: The RMF paging activity report shows a figure for total system non-VIO non-swap page-ins added to the figure shown for the total system non-VIO non-swap page-outs. This figure is the total paging rate per second for the entire system.
- *Tasks*: Transaction manager statistics (part of the CICS interval, end-of-day, and requested statistics). The “Peak” subheading is the figure shown for “Peak Number of Tasks” in the statistics. The “At MXT” subheading is the figure shown for “Number of Times at Max. Task” in the statistics.
- *Most heavily used DASD*: The RMF direct access device activity report, which relates to the most heavily used volume in your system. The “Response” subheading is the figure shown in the “Avg. Resp. Time” column for the volume you have selected. The “Utilization” subheading is the figure shown in the “% Dev. Util.” column for that volume.
- *Average-use DASD*: The RMF direct access device activity report, which relates to a representative average-use volume in your system. The “Response” subheading is the figure shown in the “Avg. Resp. Time” column for the volume you have selected. The “Utilization” subheading is the figure shown in the “% Dev. Util.” column for that volume.
- *Processor utilization*: The RMF processor activity report.

This chart is most useful when comparing before-and-after changes in performance while you are tuning your CICS system.

Performance analysis: Single-transaction measurement

You can use full-load measurement to evaluate the average loading of the system per transaction. However, this type of measurement cannot provide you with information about the behavior of a single transaction and its possible excessive loading of the system. If, for example, nine different transaction types issue five start I/Os (SIOs) each, but the 10th issues 55 SIOs, this results in an average of 10 SIOs per transaction type. This situation should not cause concern if the transactions start at the same time; however, an increase of the transaction rate of the 10th transaction type might lead to poor performance overall. To investigate this type of problem, you can perform a single-transaction measurement.

Sometimes, response times are good with existing terminals, but adding a few more terminals leads to unacceptable degradation of performance. In this case, the performance problem might be present with the existing terminals, and has been highlighted by the additional load.

To investigate this type of problem, do a full-load measurement and a single-transaction measurement. The single-transaction measurement must be done when no batch region is running, and there must be no activity in CICS apart from the test screen. Halt the polling of remote terminals.

Measure each existing transaction that is used in a production system or in a final test system. Test each transaction two or three times with different data values, to exclude an especially unfavorable combination of data. Document the sequence of transactions and the values entered for each test as a prerequisite for subsequent analysis or interpretation.

Between the tests of each single transaction, insert a pause of several seconds, to make the trace easier to read. Use a copy of the production database or data set for the test, because a test data set containing 100 records can often result in different behavior when compared with a production data set containing 100,000 records.

The condition of data sets can cause performance degradation, especially when many segments or records have been added to a database or data set. Do not measure directly after a reorganization, because the database or data set is only in this condition for a short time. If the measurement reveals an unusually large number of disk accesses, reorganize the data and perform a further measurement to evaluate the effect of the data reorganization.

Single-transaction measurement with only one terminal might not be an efficient tool for revealing a performance degradation that might occur when, perhaps, 40 or 50 terminals are in use. Practical experience has shown, however, that single-transaction measurement is usually the only means for revealing and rectifying, with justifiable expense, performance degradation under full load.

Ideally, carry out single-transaction measurement during the final test phase of the transactions, for these reasons:

- Any errors in the behavior of transactions can be revealed and rectified before production starts, without loading the production system.
- The application is documented during the measurement phase, helping to identify the effects of later changes.

CICS auxiliary trace

Auxiliary trace is a standard feature of CICS, and gives an overview of transaction flows so that you can quickly and effectively analyze them. From this trace, you can find out whether a specified application is running as expected.

If you have many transactions to analyze, you can select, in a first pass, the transactions whose behavior does not comply with what is expected.

If all transactions last much longer than expected, there might be a system-wide error in application programming or in system implementation. The analysis of a few transactions is then sufficient to determine the error.

If, only a few transactions remain, analyze these transactions next, because it is highly probable that these transactions are creating most of the performance problems.

Performance measurement tools

There are a number of tools that you can use to measure performance and to understand where constraints in the system might develop.

Performance of a production system depends on the utilization of resources such as CPU, real storage, ISC links, coupling facility, and the network. A variety of programs could be written to monitor all these resources. Many of these programs are currently supplied as part of IBM products such as CICS or IMS, or are supplied as separate products. These topics describe some of the products that can give performance information on different components of a production system.

The list of products in these topics is far from being an exhaustive summary of performance monitoring tools, although the data provided from these sources comprises a large amount of information. To monitor all this data is an extensive task. Furthermore, only a small subset of the information provided is important for identifying constraints and determining necessary tuning actions, and you have to identify this specific subset for your particular CICS system.

Consider that there are two different types of tools:

1. Tools that directly measure whether you are meeting your objectives
2. Additional tools to look into internal reasons why you might not be meeting objectives.

None of the tools can directly measure whether you are meeting end-user response time objectives. The lifetime of a task within CICS is comparable, that is, usually related to, response time, and bad response time is usually correlated with long lifetime within CICS, but this correlation is not exact because of other contributors to response time.

Obviously, you want tools that help you to measure your objectives. In some cases, you might choose a tool that looks at some internal function that contributes towards your performance objectives, such as task lifetime, rather than directly measuring the actual objective, because of the difficulty of measuring it.

When you have gained experience of the system, you should have a good idea of the particular things that are most significant in that particular system and, therefore, what things might be used as the basis for exception reporting. Then, one way of monitoring the important data might be to set up exception-reporting procedures that filter out the data that is not essential to the tuning process. This involves setting standards for performance criteria that identify constraints, so that the exceptions can be distinguished and reported while normal performance data is filtered out. These standards vary according to individual system requirements and service level agreements.

Often, you need to gather a considerable amount of data before you can fully understand the behavior of your own system and determine where a tuning effort can provide the best overall performance improvement. Familiarity with the analysis tools and the data they provide is basic to any successful tuning effort.

Remember, however, that all monitoring tools cost processing effort to use. Typical costs are 5% additional processor cycles for the CICS monitoring facility (performance class), and up to 1% for the exception class. The CICS trace facility overhead is highly dependent on the workload used. The overhead can be in excess of 25%.

In general, then, we recommend that you use the following tools in the sequence of priorities shown:

1. CICS statistics
2. CICS monitoring data
3. CICS internal and auxiliary trace.

Tuning your system

When you have identified specific constraints, you will have identified the system resources that need to be tuned. The three major steps in tuning a system are determining acceptable tuning trade-offs, making tuning changes to your system and reviewing the results of tuning.

Determining acceptable tuning trade-offs

The art of tuning can be summarized as finding and removing constraints. In most systems, the performance is limited by a single constraint. However, removing that constraint, while improving performance, inevitably reveals a different constraint, and you might often have to remove a series of constraints. Because tuning generally involves decreasing the load on one resource at the expense of increasing the load on a different resource, relieving one constraint always creates another.

A system is always constrained. You do not remove a constraint; you can only choose the most satisfactory constraint. Consider which resources can accept an additional load in the system without themselves becoming worse constraints and causing a performance degradation.

Making tuning changes to your system

The next step in the tuning process is to make the actual system modifications that are intended to improve performance. You should consider several points when adjusting the system:

- Tuning is the technique of making small changes to the system's resource allocation and availability to achieve relatively large improvements in response time.
- Tuning is not always effective. If the system response is too long and all the system resources are lightly used, you see very little change in the CICS response times. (This is also true if the wrong resources are tuned.) In addition, if the constraint resource, for example, line capacity, is being fully used, the only solution is to provide more capacity or redesign the application (to transmit less data, in the case of line capacity).
- Do not tune just for the sake of tuning. Tune to relieve identified constraints. If you tune resources that are not the primary cause of performance problems, this has little or no effect on response time until you have relieved the major constraints, and it may make subsequent tuning work more difficult. If there is any significant improvement potential, it lies in improving the performance of the resources that ***are*** major factors in the response time.
- In general, tune major constraints first, particularly those that have a significant effect on response time. Arrange the tuning actions so that items having the greatest effect are done first. In many cases, one tuning change can solve the performance problem if it addresses the cause of the degradation. Other actions may then be unnecessary. Further, improving performance in a major way can alleviate many user complaints and allow you to work in a more thorough way. The 80/20 rule applies here; a small number of system changes normally improves response time by most of the amount by which it can be improved, assuming that those changes address the main causes of performance problems.
- Make one tuning change at a time. If two changes are made at the same time, their effects may work in opposite directions and it may be difficult to tell which of them had a significant effect.
- Change allocations or definitions gradually. For example, when reducing the number of resident programs in a system, do not change all programs in a system from RES=YES to RES=NO at once. This could cause an unexpected lengthening of response times by increasing storage usage because of fragmentation, and increasing processor usage because of higher program loading activity. If you change a few programs at a time, starting with the lesser-used programs, this can give you a better idea of the overall results.

  The same rule holds true for buffer and string settings and other data set operands, transaction and program operands, and all resources where the operand can be specified individually for each resource. For the same reason, do not make large increases or decreases in the values assigned to task limits such as MXT.
- Continue to monitor constraints during the tuning process. Because each adjustment changes the constraints in a system, these constraints vary over time. If the constraint changes, tuning must be done on the new constraint because the old one is no longer the restricting influence on performance. In addition, constraints may vary at different times during the day.
- Put fallback procedures in place before starting the tuning process. As noted earlier, some tuning can cause unexpected performance results. If this leads to poorer performance, it should be reversed and something else tried. If previous definitions or path designs were not saved, they have to be redefined to put the system back the way it was, and the system continues to perform at a poorer level until these restorations are made. If the former setup is saved in such a way that it can be recalled, back out of the incorrect change becomes much simpler.

Reviewing the results of tuning

After each adjustment has been done, review the performance measurements that have been identified as the performance problem to verify that the intended performance changes have occurred and to quantify that change. If performance has improved to the point that service level agreements are being met, no more tuning is required. If performance is better, but not yet acceptable, investigation is required to determine the next action to be taken, and to verify that the resource that was tuned is still a constraint. If it is not still a constraint, new constraints need to be identified and tuned. This is a return to the first step of the tuning process, and you should repeat the next steps in that process until an acceptable performance level is reached.

CICS tools to obtain performance data

You can use CICS statistics, monitoring, and trace facilities to gather and monitor performance data to help you tune your CICS system optimally. Additional sources of information are also listed.

CICS statistics

CICS statistics are the simplest and the most important tool to monitor a CICS system permanently. They collect information about the CICS system as a whole, without regard to tasks.

For more information, see Introduction to CICS statistics.

CICS monitoring

CICS monitoring collects data about the performance of all user and CICS transactions during online processing for later offline analysis.

For more information, see Collecting and processing data for CICS monitoring.

CICS trace

For the more complex problems that involve system interactions, you can use the CICS trace to record the progress of CICS transactions through the CICS management modules.

CICS trace provides a history of events leading up to a specific situation.

The CICS trace facilities can also be useful for analyzing performance problems such as excessive waiting on events in the system, or constraints resulting from inefficient system setup or application program design.

For more information, see CICS trace.

Other sources of information

The measurement tools just described do not provide all the data necessary for a complete evaluation of current system performance. They do not provide information about how, and under what conditions, each resource is being used, or the system configuration that exists when the data is collected. Therefore, it is important to use as many techniques as possible to get information about the system. Additional sources of information include the following:

- Hardware configuration
- VTOC listings
- LISTCAT (VSAM)
- Installed resource definitions
- Link pack area (LPA) map
- Load module cross-reference of the CICS nucleus
- SYS1.PARMLIB listing
- z/OS Workload Manager (WLM) service definition
- MVS System Logger configuration - LOGR couple data set listing
- Dump of the CICS address space
- TCP/IP Profile data set.

System management facility (SMF)

The z/OS system collects statistical data for each task when certain events occur in the life of the task. The System Management Facility (SMF) formats the information that it gathers into system-related (or job-related) records.

System-related SMF records include information about the configuration, paging activity, and workload. Job-related records include information about the processor time, SYSOUT activity, and data set activity of each job step, job, APPC/MVS transaction program, and TSO/E session.

The information gathered by SMF is useful when completing the following tasks:

- Billing users
- Reporting reliability
- Analyzing the configuration
- Scheduling jobs
- Summarizing direct-access volume activity
- Evaluating data set activity
- Profiling system resource use
- Maintaining system security.

For more information, see z/OS MVS System Management Facilities (SMF).

Generalized trace facility (GTF)

GTF is part of the MVS system that you can use to record CICS trace entries.

You can use GTF to record CICS trace entries and use the interactive problem control system (IPCS) to produce reports. More generally, GTF is an integral part of the MVS system, and traces the following system events: DASD seek addresses on start I/O instructions, system resources manager (SRM) activity, page faults, I/O activity, and supervisor services. Execution options specify the system events to be traced.

GTF is generally used to monitor short periods of system activity and you should run it accordingly.

The amount of processing time that GTF uses can vary considerably, depending on the number of events to be traced. You should request the time-stamping of GTF records with the TIME=YES operand on the EXEC statement for all GTF tracing.

Run GTF at a dispatching priority (DPRTY) of 255 so that records are not lost. If the DPRTY is specified at 255 and GTF records are lost, specify the BUF operand on the execute statement as greater than 10 buffers.

You can use the following options to get the data that is generally needed for CICS performance studies:

```
TRACE=SYS,RNIO,USR     (VTAM)
TRACE=SYS              (Non-VTAM)
```

Note: VTAM® is now known as z/OS Communications Server.

If you need data on the units of work dispatched by the system and on the length of time it takes to execute events such as SVCs and LOADs, the options are as follows:

```
TRACE=SYS,SRM,DSP,TRC,PCI,USR,RNIO
```

The TRC option produces the GTF trace records that indicate GTF interrupts of other tasks that it is tracing. This set of options uses a higher percentage of processor resources, so use it only when you need a detailed analysis or timing of events.

No data-reduction programs are provided with GTF. To extract and summarize the data into a meaningful and manageable form, you can either write a data-reduction program or use one of the program offerings that are available.

For further details, see z/OS MVS Diagnosis Tools and Service Aids.

Generalized trace facility (GTF) reports
You can produce reports from GTF data with the interactive problem control system (IPCS). The reports generated by IPCS are useful in evaluating both system and individual job performance.

IPCS produces job and system summary reports, and also an abbreviated detail trace report. The summary reports include information about MVS dispatches, SVC usage, contents supervision, I/O counts and timing, seek analysis, page faults, and other events traced by GTF. The detail trace reports can be used to follow a transaction chronologically through the system.

Other reports are available that map other data:

- seek addresses for a specific volume
- arm movement for a specific volume
- references to data sets and members within partitioned data sets
- page faults and module reference in the link pack area (LPA).

Before GTF is run, you should plan the events to be traced. If specific events such as start I/Os (SIOs) are not traced, and the SIO-I/O timings are required, the trace must be re-created to get the data needed for the reports.

If there are any alternative paths to a control unit in the system being monitored, you should include the PATHIO input statement in the report execution statement. Without the PATHIO operand, there are multiple I/O lines on the report for the device with an alternative path: one line for the primary device address and one for the secondary device address. If this operand is not included, the I/Os for the primary and alternate device addresses must be combined manually to get the totals for that device.

Seek histogram report

The seek histogram report (SKHST) can help you find out if there is any arm contention on that volume, that is, if there are any long seeks on the volume being mapped. It produces two reports: the first shows the number of seeks to a particular address, and the second shows the distance the arm moves between seeks. These reports can be used to determine if you should request a volume map report to investigate further the need to reorganize a specific volume.

Volume map report

The volume map report (VOLMAP) shows information about data sets on the volume being mapped and about seek activity to each data set on that volume. It also maps the members of a partitioned data set and the count of seeks issued to each member. This report can be useful in reorganizing the data sets on a volume and in reorganizing the members within a partitioned data set to reduce the arm movement on that specific volume.

Reference map report

The reference map report (REFMAP) shows the page fault activity in the link pack area (LPA) of MVS™. This reference is by module name and separates the data faults from the instruction faults. The report also shows the count of references to the specific module. This reference is selected from the address in the stored PSW of the I/O and EXT interrupt trace events from GTF. This report can be useful if you want to change the current MVS pack list in order to reduce real storage or to reduce the number of page faults that are being encountered in the pageable link pack area of MVS.

CICS Performance Analyzer for z/OS (CICS PA)
CICS Performance Analyzer (CICS PA) is a reporting tool that provides information on the performance of your CICS systems and applications, and helps you tune, manage, and plan your CICS systems effectively.

CICS PA can help:

- System Programmers to track overall CICS system performance and evaluate the results of their system tuning efforts
- Application Programmers to analyze the performance of their applications and the resources they use

- Database Administrators to analyze the usage and performance of database systems such as IMS and DB2
- WebSphere® MQ Administrators to analyze the usage and performance of their WebSphere MQ messaging systems
- Managers to ensure transactions are meeting their required Service Levels and measure trends to help plan future requirements and strategies

CICS PA provides an ISPF menu-driven dialog to help you request and submit your reports and extracts. The available reports and extracts are grouped by category:

- Performance reports
  - List
  - List extended
  - Summary
  - Totals
  - Wait analysis
  - Transaction profiling
  - Cross-system work
  - Transaction group
  - BTS
  - Workload activity
  - Transaction tracking list
  - Transaction tracking summary
- Exception reports
  - List
  - Summary
- Transaction resource usage reports
  - File usage summary
  - Temporary storage usage summary
  - DPL usage summary
  - Transaction resource usage list
- Statistics reports
  - List
  - Alert
  - CICS Transaction Gateway
- Subsystem reports
  - DB2
  - WebSphere MQ
  - OMEGAMON
- System reports
  - System logger
- Extracts
  - Cross-system work
  - Performance
  - Record selection

  - HDB load
  - System logger
  - Statistics

CICS PA also provides a Historical Database (HDB) facility to help you manage the performance and statistics data for your CICS transactions. SMF data is saved in HDB container data sets that are managed from the CICS PA dialog. The following types of HDB are available:

- Performance List HDB

  A List HDB is built from CMF performance class data. In a List HDB data set, one record represents one transaction. Typically, List HDBs are used to analyze recent transaction events.
- Performance Summary HDB

  A Summary HDB is built from CMF performance class data. In a Summary HDB data set, one record represents a summary of transaction activity over a user-specified time interval. Typically, Summary HDBs are used for long-term trend analysis and capacity planning.
- Statistics HDB

  A Statistics HDB contains collections of CICS statistics and server statistics and CICS Transaction Gateway statistics over a specified time interval.

For more information about CICS Performance Analyzer for z/OS, see the CICS Performance Analyzer documentation.

The CICS PA dialog
You use the CICS PA dialog to create, maintain, and submit your report requests. You can also use it to specify your input data and tailor requests specific to your requirements without needing to understand the CMF data.

The dialog requires no special customization or setup. Reporting can commence immediately.

The following steps explain how to use the dialog for reporting.

1. Define your CICS (and other related) systems and their SMF files and log streams. After your systems are defined, you can start reporting against them. You can fast-track this process by using the take-up facility. CICS PA extracts information about your CICS systems from your SMF files and makes it available in the dialog. If you define your own CMF user fields, specify your MCT definition. The user fields can then be incorporated into your CICS PA reports. The following example panel shows some CICS systems, a DB2 subsystem, a WebSphere MQ subsystem, and an MVS System Logger defined to CICS PA.

   ```
                                 System Definitions                 Row 1 from 8

   Command ===> ________________________________________________ Scroll ===> CSR

   Select a System to edit its definition, SMF Files and Groups.
                                                                          SMF Files
   /   System   Type     Image           Description                      System
   _   MVS1     Image             Production MVS system                   MVS1
   _   CICSP1   CICS     MVS1     CICS Production System 1                MVS1
   _   CICSPTOR CICS     MVS1     CICS Production TOR                     MVS1
   _   CICSPAOR CICS     MVS2     CICS Production AOR                     CICSPAOR
   _   CICSPFOR CICS     MVS2     CICS Production FOR                     CICSPFOR
   _   DB2P     DB2      MVS3     DB2 Production Subsystem                DB2P
   _   MQSP     MQ       MVS4     MQ Production Subsystem                 MQSP
   _   MVS1LOGR Logger   MVS1     System Logger for MVS1                  MVS1
   ```

   Figure 2. CICS PA: System Definitions

   Related CICS systems, such as those systems that connect through IRC/MRO or ISC/APPC, can be grouped together for reporting purposes. For example, if you assign the CICS MRO systems (CICSPTOR, CICSPAOR, CICSPFOR, CICSPDOR) to a group, you can report on these systems as a single entity. CICS PA reports can then show a complete end-to-end picture of your MRO transaction activity, incorporating detailed DB2 statistics derived from the DB2 accounting data of subsystem DB2P.

2. To build, submit and save your report requests, you can define Report Sets. A Report Set contains the set of reports that you want to run in a single job. Simply select the required reports and submit a report request.

   Figure 3 on page 29 shows a Report Set. The available reports are displayed in a tree structure (folder style) and grouped by category. Report categories can be expanded or collapsed as required. The Active status controls which reports in the Report Set are run when you submit a report request.

```
EDIT                      Report Set - DAILY                      Row 1 of 45
Command ===> ________________________________________________Scroll ===> CSR

Description  . . . . Daily Reports for our production MRO system

Enter "/" to select action.

___          ** Reports **                           Active
     -  ___  Options                                   Yes
             ___  Global                               Yes
     -  ___  Selection Criteria                        Yes
             ___  Performance                          Yes
             ___  Exception                            No
     -  ___  Performance Reports                       Yes
             ___  List                                 Yes
             ___  List Extended                        Yes
             ___  Summary                              Yes
             ___  Totals                               Yes
             ___  Wait Analysis                        No
             ___  Transaction Profiling                No
             ___  Cross-System Work                    No
             ___  Transaction Group                    Yes
             ___  BTS                                  No
             ___  Workload Activity                    No
             ___  Transaction Tracking List            No
             ___  Transaction Tracking Summary         No
     -  ___  Exception Reports                         No
             ___  List                                 No
             ___  Summary                              No
     -  ___  Transaction Resource Usage Reports        No
             ___  File Usage Summary                   No
             ___  Temporary Storage Usage Summary      No
             ___  DPL Usage Summary                    No
             ___  Transaction Resource Usage List      No
     -  ___  Statistics Reports                        No
             ___  Alert                                No
             ___  CICS Transaction Gateway             No
     -  ___  Subsystem Reports                         No
             ___  DB2                                  No
             ___  WebSphere MQ                         No
             ___  OMEGAMON                             No
     -  ___  Performance Graphs                        No
             ___  Transaction Rate                     No
             ___  Transaction Response Time            No
     -  ___  Extracts                                  Yes
             ___  Cross-System Work                    Yes
             ___  Performance                          No
             ___  Record Selection                     No
             ___  HDB Load                             No
             ___  System Logger                        No
             ___  Statistics                           No
             ** End of Reports **
```

   Figure 3. CICS PA: Report Set

   Report Sets can contain selection criteria, which are used to filter CMF records. This enables you to tailor your reporting to include only the information that you are interested in. For example, you can specify selection criteria to restrict reporting to:

   - A particular date/time range
   - A group of related transaction IDs
   - Transaction response times that exceed your thresholds

3. To tailor the format and content of your reports, you can define Report Forms. You can use an editor to design your own report by selecting the required CMF fields. You can select most CMF fields for reporting, and detailed explanations of each CMF field is available from the dialog. Report Forms can

contain selection criteria. When a report specifies a Report Form and both have selection criteria specified, records must match both sets of selection criteria to be included in the report.

Figure 4 on page 30 shows a Report Form tailored to show File Control statistics.

```
                     EDIT LIST Report Form - FCLIST        Row 1 of 16 More: >
Command ===> ________________________________________________ Scroll ===> CSR

Description . . . .  File Control List Form           Version (VRM): 710

Selection Criteria:
 _  Performance  *                                    Page width . . 132

   Field
/  Name +       Type     Description
__ TRAN                  Transaction identifier
__ USERID                User ID
__ STOP         TIMET    Task stop time
__ RESPONSE              Transaction response time
__ DISPATCH     TIME     Dispatch time
__ CPU          TIME     CPU time
__ FCWAIT       TIME     File I/O wait time
__ FCAMCT                File access-method requests
__ FCADD                 File ADD requests
__ FCBROWSE              File Browse requests
__ FCDELETE              File DELETE requests
__ FCGET                 File GET requests
__ FCPUT                 File PUT requests
__ FCTOTAL               File Control requests
__ EOR                   ---------------- End of Report ----------------
__ EOX                   ---------------- End of Extract ---------------
```

Figure 4. CICS PA: Report Form

4. Define and maintain Historical Databases (HDBs) as repositories of performance data. Generate reports against your HDBs or export HDB data to DB2 tables for further analysis.

Using CICS PA to analyze CICS performance
CICS PA provides reports and extracts to help you analyze and tune the performance of your CICS systems and applications.

- The Performance List, List Extended, and Summary reports provide a detailed analysis of transaction activity.
- The Performance Totals report provides comprehensive resource usage analysis of your entire CICS system, or individual transactions.
- The Wait Analysis report summarizes transaction activity by wait time. For each transaction ID, the resources that cause this transaction to be suspended are shown in the order of most to least expensive. This report highlights the system resource bottlenecks that might be causing bad response time. More detailed analysis can then be performed, focusing on the problem resources identified.
- The Transaction Profiling report compares two sets of CMF performance class data. For example, you can compare the performance data for a specific CICS application in two different time periods, or the performance data for all applications on two systems.
- The Cross-System Work report combines CMF records from your connected systems (such as MRO and APPC) to produce a consolidated unit-of-work report.
- The Cross-System Work extract consolidates CMF records for the same unit-of-work into a single record in CMF format. CICS PA can then process the extracted data set to produce any of the reports. For example, "Summarize all multi-system UOWs whose originating transaction ID is TR01".
- The Transaction Group report provides a detailed list of incoming work requests. Transactions that CICS executes under the same incoming work request (for example, the CWXN and CWBA transactions for CICS web support requests) are grouped together in the report.
- The BTS report provides a detailed list of CICS Business Transaction Services activity. Transactions with the same CICS BTS process identifier (root activity identifier) are grouped together in the report.
- The Workload Activity report provides a transaction response time analysis by z/OS Workload Manager (WLM) service and report class. You can use this information to understand, from a CICS perspective,

how well your CICS transactions are meeting their response time goals. The Workload Activity List report is a cross-system report that correlates CMF performance class data from single or multiple CICS systems for each network unit-of-work. Importantly, this report ties MRO and function shipping tasks to their originating task so that their impact on response time can be assessed.

- The Transaction Tracking List report provides performance data for groups of related transactions. This allows monitoring and measurement of transaction performance from the perspective of transaction flow. The report shows how a process flowed from one transaction or system to the next and back again. The report combines CMF records for each originating transaction and its subordinate (group) transactions.
- The Transaction Tracking Summary report combines CMF records for each originating transaction and its subordinate (group) transactions. The summarized data is presented on a single line for each grouped originating transaction.
- The Exception List and Summary reports provide a detailed analysis of the exception events recorded by CMF.
- The Transaction Resource Usage reports process CMF performance data and CMF resource class data to provide a detailed analysis of file, temporary storage, and distributed program link (DPL) usage.
- Statistics Alerts enable you to define conditions, in terms of CICS Transaction Server or CICS Transaction Gateway statistics field values, that interest you. You can then use those conditions to report on CICS statistics stored in SMF files or historical databases.
- The CICS Transaction Gateway reports provide reporting of CICS Transaction Gateway Statistics SMF 111 records. The following reports are available:
  - Activity
  - Usage and Capacity
  - Configuration
  - Client Workload
  - CICS Workload
- The DB2 report processes CICS CMF records and DB2 accounting records to produce a consolidated and detailed view of DB2 usage by your CICS systems. With this report you can view CICS and DB2 resource usage statistics together in a single report. The DB2 List report shows detailed information of DB2 activity for each transaction. The DB2 Summary reports summarize DB2 activity by transaction and program within APPLID.
- The WebSphere MQ report processes WebSphere MQ accounting (SMF 116) records to produce a detailed view of WebSphere MQ usage by your CICS systems. The WebSphere MQ List report provides a trace of WebSphere MQ accounting records. The WebSphere MQ Summary report provides two summarized views of your WebSphere MQ transactions: by CICS transaction ID showing the WebSphere MQ system and queue resources used, and by WebSphere MQ queue name showing the transactions they service and resources used.
- The OMEGAMON report processes OMEGAMON XE for CICS (SMF 112) records to produce a detailed view of how CICS transactions use Adabas, CA-Datacom, CA-IDMS, and Supra. For each type of DBMS, you can request the following reports:
  - A List report for database usage for each transaction
  - A Transaction Summary report for database usage summarized by transaction ID
  - A Database Summary report for database usage summarized by database
- The System Logger report processes System Logger records to provide information on the System Logger log streams and coupling facility structures that are used by CICS Transaction Server for logging, recovery and backout operations. The report can assist with measuring the effects of tuning changes and identifying log stream or structure performance problems.
- The Performance Graph reports provide a graphical representation of transaction rates and response times.

- The Extract data sets are produced from SMF data and are suitable for further manipulation and analysis. You can use the following extracts to import data into external programs such as DB2, or PC tools such as Lotus® 1-2-3:
  - A Performance Data Extract for CMF performance class data
  - A System Logger Extract for System Logger data
  - A Statistics Extract for CICS statistics

  The Cross-System Work extract is described earlier in this topic. You can use the Record Selection Extract to reduce the volume of data processed by CICS PA, for more efficient reporting. You can use HDB Load to load SMF data into a Historical Database (HDB).

You can use Report Forms to tailor the format of reports and extracts, for example, to specify which fields, the order of columns, and the sort sequence.

You can use Selection Criteria to filter your reporting, for example to include data for only a specific transaction ID, and only for a specific period of time.

For more information about CICS Performance Analyzer for z/OS, see the CICS Performance Analyzer documentation.

Other tools for obtaining performance data

You can use a number of tools that are not provided by CICS to provide performance-related information to help you optimally tune your CICS system.

The z/OS Resource Measurement Facility collects data and produces reports for activity in a sysplex. For more information, see z/OS Resource Measurement Facility (RMF) User's Guide.

The IBM Redbooks publication ABCs of z/OS System Programming contains information about capacity planning, performance management, RMF, and SMF.

Resource measurement facility (RMF)

The resource measurement facility (RMF) collects system-wide data that describes the processor activity (WAIT time), I/O activity (channel and device usage), main storage activity (demand and swap paging statistics), and system resources manager (SRM) activity (workload).

RMF is a centralized measurement tool that monitors system activity to collect performance and capacity planning data. The analysis of RMF reports provides the basis for tuning the system to user requirements. They can also be used to track resource usage.

RMF measures the following activities:

- Processor usage
- Address space usage
- Channel activity:
  - Request rate and service time per physical channel
  - Logical-to-physical channel relationships
  - Logical channel queue depths and reasons for queuing.
- Device activity and contention for the following devices:
  - Unit record
  - Graphics
  - Direct-access storage
  - Communication equipment
  - Magnetic tapes
  - Character readers.
- Detailed system paging

- Detailed system workload
- Page and swap data set
- Enqueue
- CF activity
- XCF activity.

RMF allows the z/OS user to:

- Evaluate system responsiveness:
  - Identify bottlenecks. The detailed paging report associated with the page and swap data set activity can give a good picture of the behavior of a virtual storage environment.
- Check the effects of tuning:
  - Results can be observed dynamically on a screen or by postprocessing facilities.
- Perform capacity planning evaluation:
  - The workload activity reports include the interval service broken down by key elements such as processor, input/output, and main storage service.
  - Analysis of the resource monitor output (for example, system contention indicators, swap-out broken down by category, average ready users per domain) helps in understanding user environments and forecasting trends.
  - The post-processing capabilities make the analysis of peak load periods and trend analysis easier.
- Manage the larger workloads and increased resources that MVS can support
- Identify and measure the usage of online channel paths

For more information about RMF, see the IBM Redbooks publication ABCs of z/OS System Programming and z/OS Resource Measurement Facility (RMF) User's Guide.

Tools provided by IMS to obtain performance data

You can use IMS Performance Analyzer (IMS PA) and the IMS program isolation (PI) trace to monitor information on various access methods and other programs used with CICS and the operating system.

IMS Performance Analyzer (IMS PA)

IMS Performance Analyzer is a performance analysis and tuning aid for database and transaction manager systems for IMS. It processes IMS log and monitor data, including fast path data, to provide comprehensive performance, usage, and availability reports that help you to analyze and tune your IMS systems.

IMS PA:

- Uses log and monitor data to produce comprehensive DBCTL reports showing application and internal resource utilization, processor usage, and full function and fast path database activity
- Uses IMS log data to produce comprehensive information about transit times (actual system performance time), and IMS resource usage and availability
- Creates extracts of transit time by time interval data, which can be graphed, exported for processing by external programs, or downloaded to a PC
- Creates extracts of total transaction traffic and exception transactions (MSGQ or fast path), for direct import by external programs
- Processes logs from a single IMS system, or from multiple IMS subsystems running in a sysplex and using shared queues
- Uses monitor data to produce summary and analysis reports for regions, resources, programs, transactions, databases, and the total system, organized by level of detail and area of analysis

For further information, see IMS Performance Analyzer for z/OS.

IMS program isolation (PI) trace

The program isolation (PI) trace can point out database contention problems arising from the nature of task's access to a particular database.

Because only one task can have access to a record at one time, and any other task waits till the record is freed, high contention can mean high response time. This trace is part of IMS. For information about the format of the PI trace report, see System administration in IMS product documentation.

TCP/IP monitoring

TCP/IP is a communication protocol used between physically separated computer systems. TCP/IP can be implemented on a wide variety of physical networks. TCP/IP is a large family of protocols that is named after its two most important members, Transmission Control Protocol and Internet Protocol.

Internet Protocol (IP) is a network-layer protocol. It provides a connectionless data transmission service, and supports both TCP and User Datagram Protocol (UDP). Data is transmitted link by link; an end-to-end connection is never set up during the call. The unit of data transmission is the datagram.

Transmission Control Protocol (TCP) is a transport-layer protocol. It provides a connection-oriented data transmission service between applications, that is, a connection is established before data transmission begins. TCP has more error checking that UDP.

UDP is a transport-layer protocol and is an alternative to TCP. It provides a connectionless data transmission service between applications. UDP has less error checking than TCP. If UDP users want to be able to respond to errors, the communicating programs must establish their own protocol for error handling. With high-quality transmission networks, UDP errors are of little concern.

For more information about TCP/IP, see Internet, TCP/IP, and HTTP concepts.

You can use TCP/IP management and control to save the data collected by CICS so that it can be examined offline, at some point after the tasks and resources to which it relates are no longer available. You can also use TCP/IP management and control to obtain a CICSplex-wide view of the TCP/IP network and examine items in real time:

- The TCP/IP network resources that a particular CICS region is using.
- The work passing in and out of a particular CICS region over the TCP/IP network.
- The CICS resources and tasks associated with a distributed transaction that flows across the CICSplex over the TCP/IP network.
- The CICS region in which a distributed transaction originated.

You can use TCP/IP management and control to diagnose problems such as connectivity problems and transaction delays, to track work across the CICSplex, to monitor the CICSplex, and to capture system data over time for use in capacity planning.

Tivoli Decision Support for z/OS

Tivoli Decision Support for z/OS is an IBM product that collects and analyzes data from CICS and other IBM systems and products.

The reports generated by Tivoli Decision Support are useful when:

- Getting an overview of the system
- Ensuring service levels are maintained
- Ensuring availability
- Performance tuning
- Capacity planning
- Managing change and problems
- Accounting

A large number of ready-made reports are available, and in addition you can generate your own reports to meet specific needs.

In the reports Tivoli Decision Support uses data from CICS monitoring and statistics. Tivoli Decision Support also collects data from the MVS system and from products such as RMF, TSO, IMS and NetView. This means that data from CICS and other systems can be shown together, or can be presented in separate reports.

Reports can be presented as plots, bar charts, pie charts, tower charts, histograms, surface charts, and other graphic formats. Tivoli Decision Support for z/OS passes the data and formatting details to Graphic Data Display Manager (GDDM) which does the rest. Tivoli Decision Support can also produce line graphs and histograms using character graphics where GDDM is not available, or the output device does not support graphics. For some reports, where you need the exact figures, numeric reports such as tables and matrices are more suitable.

Using Tivoli Decision Support for z/OS to report on CICS performance

To understand performance data, you must first understand the work CICS performs at your installation. Analyze the work by its basic building blocks: transactions. Group the transactions into categories of similar resource or user requirements and describe each category's characteristics. Understand the work that CICS performs for each transaction and the volume of transactions expected during any given period. Tivoli Decision Support for z/OS can show you various types of data for the transactions processed by CICS.

A service-level agreement for a CICS user group defines commitments in several areas of quantifiable CICS-related resources and services. CICS service commitments can belong to one of these areas:

- Response times
- Transaction counts
- Exceptions and incidents
- Availability.

The following topics describe certain issues and concerns associated with systems management and how you can use the Tivoli Decision Support for z/OS CICS performance feature.

Performance measuring with Tivoli Decision Support for z/OS
Tivoli Decision Support for z/OS is a reporting system which uses DB2. You can use it to process utilization and throughput statistics written to log data sets by computer systems. You can use it to analyze and store the data into DB2, and present it in a variety of forms.

Tivoli Decision Support consists of a base product with several optional features that are used in systems management:

- **Tivoli Decision Support for z/OS and optional features:**
- CICS Performance Feature
- IMS Performance Feature
- Network Performance Feature
- System Performance Feature
- Workstation Performance Feature
- iSeries Performance Feature
- Accounting Feature

The Tivoli Decision Support for z/OS base includes:

- Reporting and administration dialogs that use the Interactive System Productivity Facility (ISPF)
- A collector function to read log data, with its own language
- Record mapping (definitions) for all data records used by the features

Each feature provides:

- Instructions (in the collector language) to transfer log data to DB2 tables

- DB2 table definitions
- Reports.

The Tivoli Decision Support for z/OS database can contain data from many sources. For example, data from System Management Facilities (SMF), Resource Measurement Facility (RMF), CICS, and Information Management System (IMS) can be consolidated into a single report. In fact, you can define any nonstandard log data to Tivoli Decision Support for z/OS and report on that data together with data coming from the standard sources.

The Tivoli Decision Support for z/OS CICS performance feature provides reports for your use when analyzing the performance of CICS Transaction Server, based on data from the CICS monitoring facility (CMF) and CICS statistics. These are some of the areas that Tivoli Decision Support can report on:

- Response times
- Resource usage
- Processor usage
- Storage usage
- Volumes and throughput
- CICS and DB2 activity
- Exceptions and incidents
- Data from connected regions, using the unit of work as key
- CICS availability
- CICS resource availability

The Tivoli Decision Support for z/OS CICS performance feature collects only the data required to meet CICS users' requirements. You can combine that data with more data (called *environment data*), and present it in a variety of reports. Tivoli Decision Support for z/OS provides an administration dialog for maintaining environment data. Figure 5 on page 36 illustrates how data is organized for presentation in Tivoli Decision Support z/OS reports.

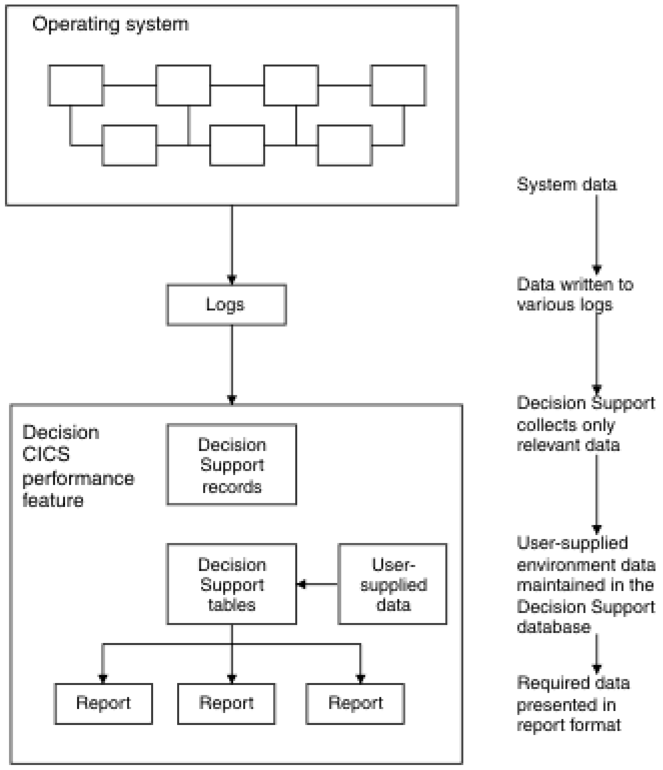

Operating system
Logs
Decision CICS performance feature
Decision Support records
Decision Support tables
User-supplied data
Report
Report
Report
System data
Data written to various logs
Decision Support collects only relevant data
User-supplied environment data maintained in the Decision Support database
Required data presented in report format

Figure 5. Organizing and presenting system performance data

The Tivoli Decision Support for z/OS CICS performance feature processes these records:

CMF

- CICS Transaction Server performance
- CICS Transaction Server exceptions
- CICS Transaction Server accounting, performance, and exceptions

Statistics

- CICS Transaction Server statistics

Monitoring response time
The response time is the total time from the start to the finish of the transaction's activity, subdivided into suspend time and dispatch time. The dispatch time includes service time. You can use the Tivoli Decision Support for z/OS CICS response-time reports to see the CICS application internal response times.

The elements of the response time report are shown in Figure 6 on page 37.

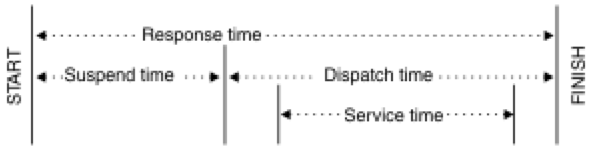


Figure 6. CICS internal response-time elements

As described in IBM Z Decision Support product documentation, the Network Performance feature generates reports that show the total, end-to-end average response time (operator transit time) for SNA applications (for example, a CICS region) by logical unit. The operator transit time consists of the host transit time and the network transit time, which are also shown in the Network Performance feature reports. Using these reports, you can isolate a response-time problem either to the network or to CICS and act on it accordingly. Should the problem be in CICS, you can use the Tivoli Decision Support for z/OS CICS performance feature reports to identify the application causing the response-time degradation.

Monitoring processor and storage use
Poor response time usually indicates inefficient use of either the processor or storage (or both). Tivoli Decision Support-supplied reports can help you isolate a resource as the cause of a CICS performance problem.

If both the Tivoli Decision Support for z/OS CICS performance feature's statistics component and the Decision Support System Performance feature's MVS component are installed and active, these reports are available for analyzing transaction rates and processor use by CICS region:

- The CICS Transaction Processor Utilization, Monthly report shows monthly averages for the dates you specify.
- The CICS Transaction Processor Utilization, Daily report shows daily averages for the dates you specify.

Tivoli Decision Support for z/OS produces several reports that can help analyze storage usage. For example, the CICS Dynamic Storage (DSA) Usage report, shows pagepool usage, under the headings **Pagepool name**, **DSA (bytes)**, **Cushion (bytes)**, **Free storage (bytes)**, **Free storage (pct)**, **Largest free area**, **Getmains**, and **Freemains**.

```
                          CICS Dynamic Storage (DSA) Usage
                        MVS ID ='MV28'      CICS ID ='IYCSCTSK'
                         Date:  '2001-01-17' to '2001-01-18'


                                         Free     Free      Largest
Pagepool       DSA     Cushion   storage  storage     free
  name      (bytes)    (bytes)    (bytes)   (pct)      area    Getmains  Freemains
--------   ---------  --------  ---------  -------  ---------  --------  ---------
CDSA         524288     65536     299008       57     245760      2668       2470
ECDSA       5242880    131072    1122304       21     868352   1084154    1067000
ERDSA      11534336    262144    1130496        9     966656       710         16
ESDSA             0         0          0        0          0         0          0
EUDSA       2097152         0    2097152      100    1048576     73620      73620
RDSA         524288     65536     204800       39     122880        40          0
SDSA         262114     65536     249856       95     249856        12          6
UDSA         524288     65536     524288      100     262114    301922     301922

                  Tivoli Decision Support Report: CICS809
```

Figure 7. CICS Dynamic storage (DSA) usage report

Monitoring volumes and throughput

If you suspect that a performance problem is related to excessive paging, you can use Tivoli Decision Support for z/OS to report on page-ins, using RMF data.

Because CICS Transaction Server for z/OS, Version 5 Release 4 uses an MVS subtask to page and because an MVS page-in causes an MVS task to halt execution, the number of page-ins is a performance concern. Page-outs are not a concern because page-outs are scheduled to occur during lulls in CICS processing.

The best indicator of a transaction's performance is its response. For each transaction ID, the CICS transaction performance detail report (in Figure 8 on page 38) shows the total transaction count and the average response time. The headings are Tran ID, Tran count, Average resp time (sec), Average CPU time (sec), Prog load reqs (avg), FC calls (avg), Exceptions, Program storage bytes (max), Getmains < 16 MB (avg), and Getmains > 16 MB (avg). Use this report to start verifying that you are meeting service-level objectives. First, verify that the values for average response time are acceptable. Then check that the transaction counts do not exceed agreed-to limits. If a transaction is not receiving the appropriate level of service, you must determine the cause of the delay.

```
                      CICS Transaction Performance, Detail
                       MVS ID ='MV28'  CICS ID ='IYCSCTSK'
                      Date: '2001-01-17'  to  '2001-01-18'


                Avg     Avg   Prog                        Program
                resp    CPU   load  Prog   FC             storage    Getmains  Getmains
Tran   Tran     time    time  reqs  loads  calls  Excep-  bytes       < 16 MB > 16 MB
 ID    count    (sec)   (sec) (avg) (avg) (avg)  tions   (max)        (avg)     (avg)
------------ -------- ------- ----  ----- -----  ------ ---------     --------  --------
QUIT    7916    0.085   0.017    0     0    18      0    74344          22         0
CRTE    1760    4.847   0.004    0     0     0      0   210176           1         0
AP00    1750    0.184   0.036    0     0     8      0   309800          66         0
PM94    1369    0.086   0.012    0     0     6      0   130096          24         0
VCS1     737    0.073   0.008    2     0     7      0    81200          14         0
PM80     666    1.053   0.155    1     0    62      0   104568         583         0
CESN     618    8.800   0.001    0     0     0      0    41608           0         0
SU01     487    0.441   0.062    4     0   126      0   177536          38         0
...
GC11       1    0.341   0.014    1     0     2      0    37048          10         0
DM08       1    0.028   0.002    0     0     0      0     5040           3         0
    ========                                            =========
       20359                                            309800

                       Tivoli Decision Support Report: CICS101
```

Figure 8. CICS transaction performance, detail report

Combining CICS and DB2 performance data
You can create reports that show the DB2 activity caused by a CICS transaction by combining CICS and DB2 performance data.

For each CICS task, CICS generates an LU6.2 unit-of-work ID. DB2 also creates an LU6.2 unit-of-work ID. Figure 9 on page 39 shows how DB2 data can be correlated with CICS performance data using the DB2 token (QWHCTOKN) to identify the task.

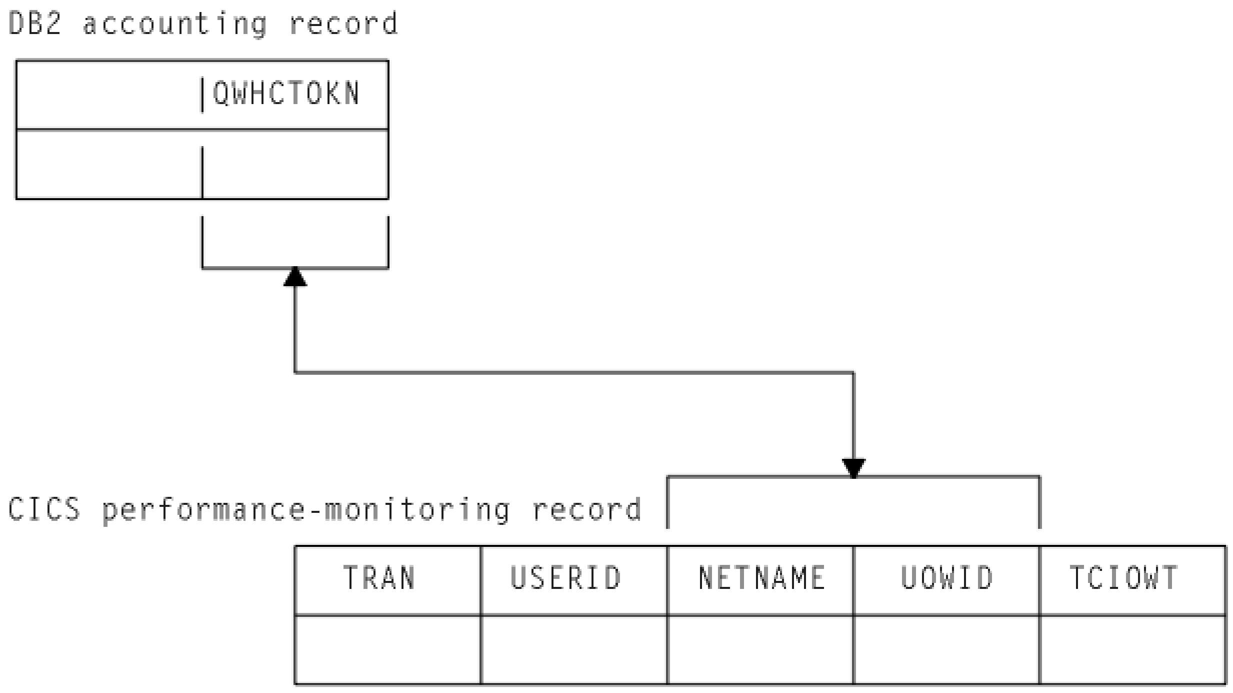


Figure 9. Correlating a CICS performance-monitoring record with a DB2 accounting record

Matching the NETUOWPX and NETUOWSX fields in a CICS record to the DB2 token, you can create reports that show the DB2 activity caused by a CICS transaction.

Monitoring exception and incident data
An *exception* is an event that you should monitor. An exception appears in a report only if it has occurred; reports do not show null counts. A single exception need not be a cause for alarm. An incident is defined as an exception with severity 1, 2, or 3.

The Tivoli Decision Support for z/OS CICS performance feature creates exception records for these incidents and exceptions:

- Wait for storage
- Wait for main temporary storage
- Wait for a file string
- Wait for a file buffer
- Wait for an auxiliary temporary storage string
- Wait for an auxiliary temporary storage buffer
- Transaction ABEND
- System ABEND
- Storage violations
- Short-of-storage conditions
- z/OS Communications Server request rejections
- I/O errors on auxiliary temporary storage
- I/O errors on the intrapartition transient data set

- Autoinstall errors
- MXT reached
- Link errors for IRC and ISC
- Log stream buffer-full conditions
- CREAD and CWRITE fails (data space problems)
- Local shared resource (LSR) pool (string waits)
- Waits for a buffer in the LSR pool
- Errors writing to SMF
- No space on transient-data data set
- Waits for a transient-data string
- Waits for a transient-data buffer
- Transaction restarts
- Maximum number of tasks in a transaction class reached (CMXT)
- Transmission errors

Figure 10 on page 40 shows an example of an incidents report, giving information on Severity, Date, Time, Terminal operator ID, User ID, Exception ID, and Exception description.

```
                                         CICS Incidents
                                DATE: '2001-01-17' to '2001-01-18'

                             Terminal
                             operator  User    Exception           Exception
Sev  Date        Time           ID      ID        ID              description
---  ----------  --------  -------- -------- ------------------ ---------------------------
03   2001-01-17  15.42.03  SYSTEM            TRANSACTION_ABEND  CICS TRANSACTION ABEND AZTS
03   2001-01-18  00.00.00  SYSTEM            TRANSACTION_ABEND  CICS TRANSACTION ABEND APCT
03   2001-01-18  17.37.28  SYSTEM            SHORT_OF_STORAGE   CICS SOS IN PAGEPOOL
03   2001-01-18  17.45.03  SYSTEM            SHORT_OF_STORAGE   CICS SOS IN PAGEPOOL

                 Tivoli Decision Support report: CICS002
```

Figure 10. Example of a Tivoli Decision Support CICS incidents report

Tivoli Decision Support for z/OS can pass the exceptions to an Information/Management system.

Unit-of-work reporting
In a CICS multiple region operation (MRO) or intersystem communication (ISC) environment, you can trace a transaction from one region (or processor complex) to another and back. Using the data from the trace, you can determine the total resource requirements of the combined transaction as a unit of work, without separately analyzing the component transactions in each region.

The ability to combine the component transactions of an MRO or ISC series makes possible precise resource accounting and chargeback, and capacity and performance analysis.

The CICS UOW Response Times report in Figure 11 on page 41 shows an example of how Tivoli Decision Support for z/OS presents CICS unit- of-work response times. The headings are Adjusted UOW start time, Tran ID, CICS ID, Program name, UOW tran count, and Response time (sec).

```
                       CICS UOW Response Times
                   Time: '09.59.00' to '10.00.00'
                          Date: 2001-01-18


 Adjusted
   UOW                                     UOW   Response
  start    Tran  CICS      Program   tran     time
   time     ID    ID        name     count   (sec)
 --------  ----  --------  --------  -----  --------
 09.59.25  OP22  CICSPROD  DFHAPRT       2     0.436
           OP22  CICSPRDC  OEPCPI22

 09.59.26  AP63  CICSPRDE  APPM00        2     0.045
           AP63  CICSPROD  DFHAPRT

 09.59.26  ARUS  CICSPROD  DFHAPRT       3     0.158
           CSM5  CICSPRDB  DFHMIRS
           ARUS  CICSPRDC  AR49000

 09.59.27  CSM5  CICSPRDB  DFHMIRS       4     0.639
           CSM5  CICSPRDB  DFHMIRS
           MQ01  CICSPROD  DFHAPRT
           MQ01  CICSPRDD  CMQ001

  ...

         Tivoli Decision Support report: CICS902
```

Figure 11. Tivoli Decision Support for z/OS CICS UOW response times report

Monitoring availability

In some cases, an application depends on the availability of many resources of the same and of different types, so reporting on availability requires a complex analysis of data from different sources.

Users of CICS applications depend on the availability of several types of resources:

- Central site hardware and the operating system environment in which the CICS region runs
- Network hardware, such as communication controllers, telecommunication lines, and terminals through which users access the CICS region
- CICS region
- Application programs and data. Application programs can be distributed among several CICS regions.

Tivoli Decision Support for z/OS can help you, because all the data is in one database.

CICS workload activity reporting

CICS records the transaction ID, the associated terminal ID, and the elapsed time at the end of each transaction. When more detailed reports are needed, Use the MVS Performance Management (MVSPM) component of System Performance feature of Tivoli Decision Support.

Transaction data is useful when you require only transaction statistics, rather than the detailed information that CMF produces. In many cases, it is sufficient to process only this data, since RMF records it as part of its SMF type-72 record. Analysis (and even recording) of SMF records from CMF can then be reserved for those circumstances when the detailed data is needed. Use the MVSPM component of the System Performance feature of Tivoli Decision Support to report on this data.

When running under goal mode in MVS 5.1.0 and later, CICS performance can be reported in workload groups, service classes, and periods. These are a few examples of Tivoli Decision Support reports for CICS in this environment. Figure 12 on page 42 shows how service classes were served by other service classes. This report is available only when the MVS system is running in goal mode. The headings are Workload group, Service class, Served class, No of times served, No of transactions, and No of times served per transaction.

```
                        MVSPM Served Service Classes, Overview
                        Sysplex: 'SYSPLEX1' System: IP02
                        Date: '2001-01-18' Period: 'PRIME'


     Workload Service  Served    No of times         No of    No of times
      group    class   class       served            tx's    served per tx
     -------- -------- -------- ------------  ------------  --------------
     CICS     CICSREGS CICS-1          15227           664            22.9
                       CICS-2           6405           215            29.8
                       CICS-3          24992          1251            20.0
                       CICS-4          87155          1501            58.1
                       CICSTRX         67769          9314             7.3

                       Tivoli Decision Support report: MVSPM79
```

Figure 12. Example of an MVS Performance Management served service classes overview report

Figure 13 on page 42 shows the average transaction response time trend and how the various transaction states contribute to it. (The times shown for the various transaction states are calculated based on transaction state samples, and so are not necessarily a precise record of the time spent in each state.) Adding the time spent in each of the transaction states (the shaded areas on the graph) gives the average execution time, which is shorter than the average response time (the line on the graph). The difference between the response time and the execution time is mainly made up of switch time — for example, the time the transactions spend being routed to another region for processing.

This report is available when the MVS system is running in goal mode and when the subsystem is CICS or IMS.

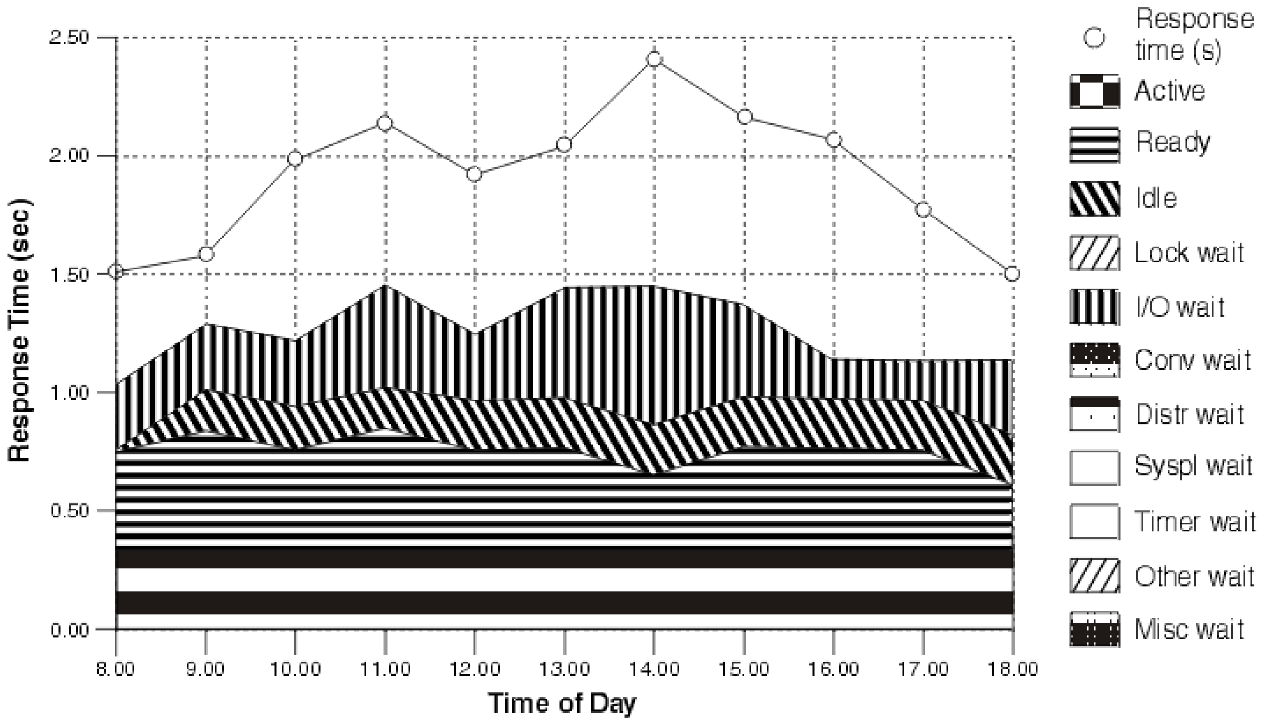


Figure 13. Example of an MVS Performance Management response time breakdown, hourly trend report

Figure 14 on page 43 shows how much the various transaction states contribute to the average response time. This report is available when the MVS system is running in goal mode and when the subsystem is CICS or IMS. The report gives information on Workload group, Service class/Period, Ph, MVS sys ID, and Total state, followed by the percentage of response time spent in each of the states listed in Figure 13 on page 42.

```
                                      MVSPM Response Time Breakdown, Overview
                                      Sysplex: 'SYSPLEX1' Subsystem: IP02
                                      Date: '2001-01-18' Period: 'PRIME'


          Service        MVS  Total Activ Ready Idle   Lock  I/O   Conv Distr Local  Netw Syspl Timer
Other  Misc
 Workload  class         sys  state state state state  wait  wait  wait wait  wait   wait wait  wait
wait   wait
   group   /Period    Ph  ID    (%)   (%)   (%)   (%)   (%)   (%)   (%)   (%)   (%)   (%)   (%)   (%)
(%)   (%)
 --------- ---------- --- ---- ----- ----- ----- ----- ----- ----- --     --- ----- ----- ----- -----
----- ----- -----
 CICS     CICS-1  /1 BTE CA0    6.6   0.0   0.0   0.0   0.0   0.0   6.5   0.0   0.0   0.0   0.0   0.0
0.0   0.0
                         C80   29.4   0.0   0.0   0.0   0.0   0.0  14.7   0.0   0.0   0.0   0.0   0.0
14.6   0.0
                         C90    3.8   0.4   1.3   1.5   0.0   0.2   0.5   0.0   0.0   0.0   0.0   0.0
0.0   0.0
                              ----- ----- ----- ----- ----- ----- -----  --- ----- ----- ----- -----
----- ----- -----
                            *  13.3   0.1   0.5   0.5   0.0   0.1   7.2   0.0   0.0   0.0   0.0   0.0
4.9   0.0


                  /1 EXE CA0   16.0   0.1   0.2   0.1   0.0  15.5   0.0   0.0   0.0   0.0   0.0   0.0
0.1   0.0
                         C80   14.9   0.1   0.1   0.1   0.0   3.7   0.0   0.0   0.0   0.0   0.0   0.0
11.0   0.0
                         C90   14.0   1.6   4.5   4.8   0.0   3.2   0.0   0.0   0.0   0.0   0.0   0.0
0.0   0.0
                              ----- ----- ----- ----- ----- -----   --- ----- ----- ----- ----- -----
----- -----
                            *  14.9   0.6   1.6   1.7   0.0   7.4   0.0   0.0   0.0   0.0   0.0   0.0
3.7   0.0


 IMS      IMS-1   /1 EXE CA0   20.7   0.4   0.7   0.0   0.0   0.0  19.6   0.0   0.0   0.0   0.0   0.0
0.0   0.0
                         C80    1.1   0.2   0.1   0.7   0.0   0.1   0.0   0.0   0.0   0.0   0.0   0.0
0.0   0.0
                         C90   22.2   5.3  11.9   1.2   0.0   0.2   3.6   0.0   0.0   0.0   0.0   0.0
0.0   0.0
                              ----- ----- ----- ----- ----- -----  ---- ----- ----- ----- ----- -----
----- -----
                            *  14.7   2.0   4.2   0.6   0.0   0.1   7.8   0.0   0.0   0.0   0.0   0.0
0.0   0.0

                           Tivoli Decision Support report: MVSPM73
```

Figure 14. Example of an MVS Performance Management response time breakdown overview report

Tivoli OMEGAMON XE for CICS on z/OS

Tivoli OMEGAMON XE for CICS on z/OS helps you to proactively manage performance and availability of complex CICS systems.

Tivoli OMEGAMON XE for CICS on z/OS (OMEGAMON XE for CICS on z/OS) is a remote monitoring agent that runs on z/OS managed systems. It assists you in anticipating performance problems and warns you when critical events take place in your CICS environments. You can set threshold levels and flags to alert you when events within your CICS regions reach critical points.

When running under the Tivoli Enterprise Portal, IBM Tivoli OMEGAMON XE for CICS on z/OS offers a central point of management for CICS Transaction Server and provides a comprehensive means for gathering the information you need to detect and prevent problems within your CICS regions. You view data that Tivoli Enterprise Portal gathers in tables and charts that show you the status of your managed CICS regions.

With this data you can perform a number of tasks:

- Collect and analyze reliable, up-to-the-second data that allows you to make faster, better informed, operating decisions
- Manage all CICS regions from a single point to identify problems at any time
- Balance workloads across various regions

- Track performance against goals

With OMEGAMON XE for CICS on z/OS, systems administrators can set threshold levels and flags to alert them when system conditions reach these thresholds. These are the advanced monitoring facilities:

- User-defined and predefined situations based on thresholds to raise different types of alerts
- At-a-glance status of all CICS regions
- The capability to monitor multiple CICS regions simultaneously from one or more centralized workstations

Used in conjunction with other OMEGAMON XE monitoring products, the data, analyses, and alerts presented by OMEGAMON XE for CICS on z/OS help you develop a holistic view of your entire computing enterprise from a single console.

OMEGAMON XE for DB2

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS is a single, comprehensive assessment tool, Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS helps you resolve critical performance issues.

Performance Expert

OMEGAMON XE for DB2 Performance Expert (PE) is a performance analysis, monitoring, and tuning tool for DB2 on z/OS environments. This product is part of the integrated and complete cross z Systems® monitoring solution of the IBM Tivoli OMEGAMON XE family that monitors all DB2 subsystems on z/OS and other resources, such as IMS, MVS, or CICS. OMEGAMON XE for DB2 PE simplifies and supports system and application performance monitoring, reporting, trend analysis, charge back usage, and buffer pool analysis. If problems are encountered you are notified and advised how to continue.

Performance Monitor

Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS permits you to monitor, analyze and optimize the performance of DB2 on z/OS applications in two key modes: online, in real time with immediate alerts when problems occur, and batch, in reports.

Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS helps you resolve critical performance issues. Use it to monitor:

- Individual data-sharing members or entire data-sharing groups.
- Applications running in a parallel query environment, even in the parallel tasks are executed on different processors.
- Near-term performance history to see problems that otherwise go unnoticed and prevent them in the future.
- Object analysis of database disks, tables, table spaces, and other elements to tune performance.

Identifying CICS performance constraints

Major constraints on a CICS system are often identified by external symptoms such as stress conditions and longer response times. CICS can resolve some constraint problems; others must be resolved manually.

Many indications of poor performance can occur in a system that is congested. For example, if there is a slowdown in direct access storage device (DASD) activity, the following symptoms might occur:

- Transactions that perform data set activity accumulate
- Waits on strings occur
- More transactions are waiting in the system
- Demands on virtual storage increase
- Demands on real storage increase

- Increased paging occurs
- The task dispatcher uses more processor power scanning task chains
- Task constraints occur
- The MXT or transaction class limit is exceeded; the processor is required to do additional work because more retries are required

As a result the system shows heavy use of all resources, resulting in typical system stress. This situation does not indicate problems with all resources; it shows that a constraint has yet to be found. To identify the constraint, you must find out what is affecting task life.

If performance is unacceptable, the performance constraints (the causes of the symptoms) must be identified so that they can be tuned.

When dealing with limit conditions, you might find it helpful to check the various hardware and software locations in the system where performance constraints are occurring.

Hardware contentions

Contentions can occur on processor cycles, real storage, database associated hardware I/O operations, and network-associated hardware operations.

- *Processor cycles*. It is not uncommon for transactions to execute more than one million instructions. To execute these instructions, transactions must contend with other tasks and jobs in the system. Sometimes these tasks and jobs must wait for activities such as file I/O. Transactions give up their use of the processor at these points and must contend for use of the processor again when the activity has completed. Dispatching priorities determine which transactions or jobs get use of the processor, and batch or other online systems affect response time by receiving preferential access to the processor. Batch programs that access online databases also tie up those databases for longer periods of time if their dispatching priority is low. At higher usages, the wait time for access to the processor can be significant.
- *Real storage (working set)*. Just as transactions must contend for the processor, they also must be given a certain amount of real storage. A real storage shortage can be particularly significant in CICS performance because a normal page fault that occurs when acquiring real storage results in synchronous I/O. The basic design of CICS is asynchronous, which means that CICS processes requests from multiple tasks concurrently to make maximum use of the processor. Most paging I/O is synchronous and causes the MVS task that CICS is using to wait, and that part of CICS cannot do any further processing until the page operation completes. Most, but not all, of CICS processing uses a single MVS task (called "QUASI" in the dispatcher statistics).
- *Database-associated hardware (I/O) operations*. When data is being accessed to provide information that is required in a transaction, an I/O operation passes through the processor, the processor channel, a disk control unit, the head of string on a string of disks, and the actual disk device where the data resides. If any of these devices are overused, the time taken to access the data can increase significantly. This overuse can be the result of activity on one data set, or on a combination of active data sets. Error rates also affect the usage and performance of the device. In shared DASD environments, contention between processors also affects performance. This, in turn, increases the time that the transaction ties up real and virtual storage and other resources.

  Large amounts of central and expanded storage, very large data buffers, and keeping programs in storage, can significantly reduce DB I/O contention and somewhat reduce processor utilization while delivering significant internal response time benefits.
- *Network-associated hardware operations*. The input and output messages of a transaction must pass from the terminal to a control unit, a communications link, a network controller, a processor channel, and finally the processor. Just as overuse of devices to access data can affect response time, so excessive use of network resources can cause performance degradation. Error rates also affect performance. In some cases, the delivery of the output message is a prerequisite to freeing the processor resources that are accessed, and contention can cause these resources to be tied up for longer periods.

Design considerations

The length of time between data set reorganizations can affect performance. The efficiency of access decreases as the data set becomes increasingly fragmented. Fragmentation can be kept to the minimum by reducing the length of time between data set reorganizations.

The following factors can limit performance:

- *Database design*. A data set or database needs to be designed to meet the needs of the application it is supporting. Such factors as the pattern of access to the data set (especially whether it is random or sequential), access methods chosen, and the frequency of access determine the best database design. Such data set characteristics as physical record size, blocking factors, the use of alternate or secondary indexes, the hierarchical or relational structure of database segments, database organization (HDAM, HIDAM, and so on), and pointer arrangements are all factors in database performance.
- *Network design*. This item can often be a major factor in response time because the network links are much slower than most components of an online system. Processor operations are measured in nanoseconds, line speeds in seconds. Screen design can also have a significant effect on overall response time. A 1200-byte message takes one second to be transmitted on a relatively high-speed 9600 bits-per-second link. If 600 bytes of the message are not needed, half a second of response time is wasted. Besides screen design and size, such factors as how many terminals are on a line, the protocols used (SNA, bisynchronous), and full-duplex or half-duplex capabilities can affect performance.
- *Use of specific software interfaces or serial functions*. The operating system, terminal access method, database manager, data set access method, and CICS must all communicate in the processing of a transaction. Only a given level of concurrent processing can occur at any one time, and this can also cause a performance constraint. Examples of concurrent processes include the SNA receive any pool (RAPOOL), VSAM data set access (strings), CICS temporary storage, CICS transient data, and CICS intercommunication sessions. Each of these can have a single or multiserver queueing effect on a transaction's response time, and can tie up other resources by slowing task throughput.

One useful technique for isolating a performance constraint in a CICS system with SNA is to use the IBMTEST command issued from a user's terminal. This terminal must not be in session with CICS, but must be connected to the z/OS Communications Server for SNA.

At an SNA LU enter the following:

```
    IBMTEST (n)(,data)
```

where *n* is the number of times you want the data echoed, and *data* consists of any character string. If you enter no data, the alphabet and the numbers zero through nine are returned to the terminal. This command is responded to by SNA LU.

IBMTEST is an echo test designed to give the user a rough idea of the z/OS Communications Server component of terminal response time. If the response time is fast in a slow-response system, the constraint is not likely to be any component from the z/OS Communications Server onward. If the response time is slow, the z/OS Communications Server or the SNA network may be the reason. This sort of deductive process in general can be useful in isolating constraints.

To avoid going into session with CICS, you may have to remove APPLID= from the LU statement or CONNECT=AUTO from the TERMINAL definition.

Observing response time

The basic criterion of performance in a production system is response time. Good performance depends on a variety of factors including user requirements, available capacity, system reliability, and application design. Good performance for one system can be poor performance for another.

In straightforward data-entry systems, good response time implies sub-millisecond response time. In normal production systems, good response time is measured in the five to ten millisecond range. In scientific, compute-bound systems or in print systems, good response time can be one or two minutes.

When checking whether the performance of a CICS system is in line with the system's expected or required capability, you should base this investigation on the hardware, software, and applications that are present in the installation.

If, for example, an application requires 100 accesses to a database, a response time of three to six milliseconds may be considered to be quite good. If an application requires only one access, however, a response time of three to six milliseconds for disk accesses would need to be investigated. Response times, however, depend on the speed of the processor, and on the nature of the application being run on the production system.

You should also observe how consistent the response times are. Sharp variations indicate erratic system behavior.

Typically, the response time in the system varies with an increasing transaction rate, is gradual at first, then quickly deteriorates. The typical curve shows a sharp change when, suddenly, the response time increases dramatically for a relatively small increase in the transaction rate.

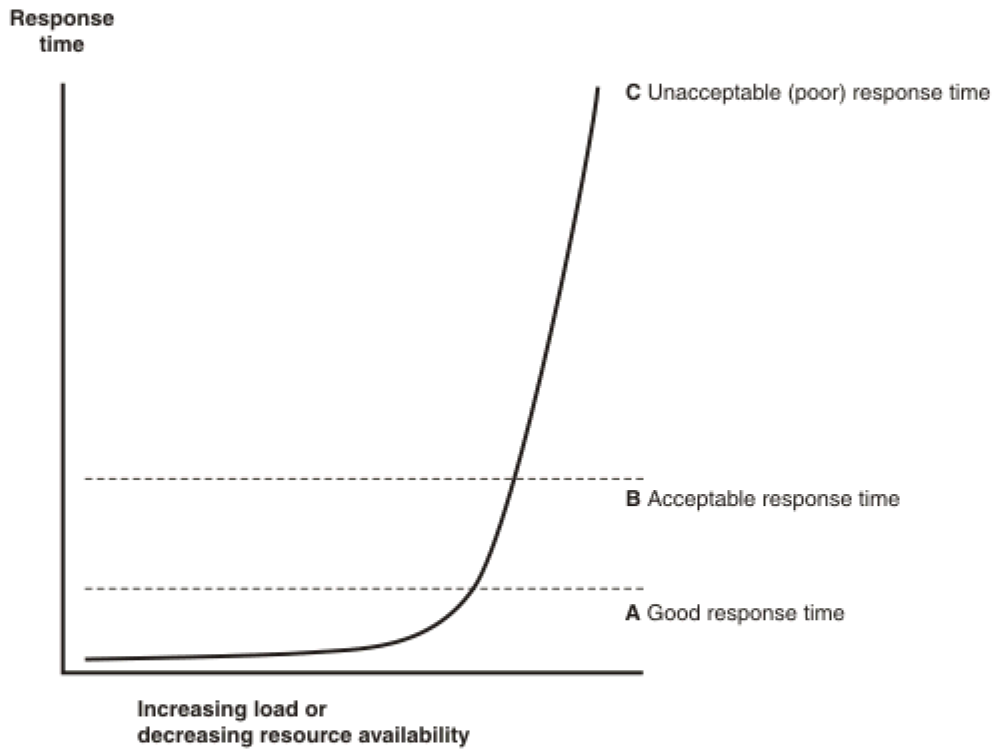


Figure 15. Graph to show the effect of response time against increasing load

For stable performance, it is necessary to keep the system operating below this point where the response time dramatically increases. In these circumstances, the user community is less likely to be seriously affected by the tuning activities being undertaken by the DP department, and these changes can be done in an unhurried and controlled manner.

Response time can be considered as being made up of queue time and service time. Service time is generally independent of usage, but queue time is not. For example, 50% usage implies a queue time approximately equal to service time, and 80% usage implies a queue time approximately four times the service time. If service time for a particular system is only a small component of the system response, for example if it is part of the processor, 80% usage might be acceptable. If it is a greater portion of the system response time, for example, in a communication line, 50% usage may be considered high.

If you are trying to find the response time from a terminal to a terminal, you should be aware that the most common “response time” obtainable from any aid or tool that runs in the host is the “internal response time.” Trace can identify only when the software in the host, that is, CICS and its attendant software, first “sees” the message on the inbound side, and when it last “sees” the message on the outbound side.

Internal response time gives no indication of how long a message took to get from the terminal, through its control unit, across a line of whatever speed, through the communication controller (whatever it is), through the communication access method (whatever it is), and any delays before the channel program that initiated the read is finally posted to CICS. Nor does it account for the time it might take for CICS to start processing this input message. There may have been lots of work for CICS to do before terminal control regained control and before terminal control even found this posted event.

The same is true on the outbound side. CICS auxiliary trace knows when the application issued its request, but that has little to do with when terminal control found the request, when the access method ships it out, when the controllers can get to the device, and so on.

While the outward symptom of poor performance is overall bad response, there are progressive sets of early warning conditions which, if correctly interpreted, can ease the problem of locating the constraint and removing it.

The information in this topic has been based on the assumption that CICS is the only major program running in the system. If batch programs or other online programs are running simultaneously with CICS, you must ensure that CICS receives its fair share of the system resources and that interference from other regions does not seriously degrade CICS performance.

Poor response time: Causes and solutions

This table shows four levels of response time, in decreasing order of severity. The major causes are shown for each level, together with a range of suggested solutions.

The first step is to check the causes by following the advice given in "Assessing the performance of your system" on page 17. When you have identified the precise causes, you can find information in "Improving the performance of a CICS system" on page 56 on how to implement an appropriate solution.

Table 2. CICS response time checklist

Major cause	**Solution**
Level 1: Poor response at all loads for all transactions	
High level of paging	Reduce working set, or allocate more real storage
Very high usage of major resources	Reconsider system resource requirements and redesign system, and check for application errors and resource contention
Level 2: Poor response at medium and high loads	
High level of paging	Reduce working set, or allocate more real storage
High processor usage	Reduce pathlength, or increase processor power
High DB or data set usage	Reorganize data sets, or reduce data transfer, or increase capacity
High communication network usage	Reduce data transfer, or increase capacity
TP or I/O access-method constraint	Increase buffer availability
CICS limit values exceeded	Change operands, or provide more resources, or check if errors in application
Level 3: Poor response for certain transactions only	
Identify common characteristics listed under Level 2	The solutions are as for Level 2
Lines or terminal usage	Increase capacity, or reduce data transfer, or change transaction logic
Data set usage	Change data set placement buffer allocations or change enqueue logic or data set design

Table 2. CICS response time checklist (continued)

Major cause	**Solution**
High storage usage	Redesign or tune applications
Same subprograms used by transactions	Redesign or tune application subprograms
Same access method or CICS features used by transactions	Reallocate resource or change application, and reevaluate use of feature in question
Limit conditions	Reallocate resource or change application
Level 4: Poor response for certain terminals	
Check network loading as appropriate	Increase capacity of that part of network
Check operator techniques	Revise terminal procedures
Check terminal definitions	Redefine terminal definitions

Reducing storage stress

Storage stress occurs when there is a shortage of free space in one of the dynamic storage areas.

Storage stress can be a symptom of the following situations:

- Other resource constraints that cause CICS tasks to occupy storage for longer than usual
- A sudden large number of tasks that overwhelm available free storage
- Badly designed applications that require unreasonably large amounts of storage

CICS handles storage stress as follows:

- With decreasing free storage availability, nonresident, not-in-use programs might be deleted progressively, as CICS determines appropriate, on a least-recently-used basis. Dispatch of new tasks is also progressively slowed as free storage approaches a critically small amount. This self-tuned activity tends to spread the cost of managing storage. There might be more program loading overall, but the heavy overhead of a full program compression is not incurred at the critical time.
- The loading or reloading of programs is handled by CICS with an MVS subtask. In this way, other user tasks can proceed if a processor of the MVS image is available and even if a page-in is required as part of the program load.
- User runtime control of storage usage is achieved through appropriate use of maximum task specification (MXT) and transaction class limits. This is necessary to avoid the short-on-storage condition that can result from unconstrained demand for storage.

Short-on-storage condition

CICS reserves a minimum number of free storage pages for use only when there is not enough free storage to satisfy an unconditional GETMAIN request even after all not-in-use nonresident programs have been deleted.

Whenever a request for storage results in the number of contiguous free pages in one of the dynamic storage areas falling below its respective cushion size, or failing to be satisfied even with the storage cushion, a cushion stress condition exists. Details are given in the storage manager statistics ("Times request suspended", "Times cushion released"). CICS attempts to alleviate the storage stress situation by taking a number of actions. If these actions fail to alleviate the situation, or if the stress condition is caused by a task that is suspended for SOS, a short-on-storage condition is signaled. This is accompanied by message DFHSM0131, DFHSM0133 or DFHSMSM0606.

Removing unwanted data set name blocks

The extended CICS dynamic storage area (ECDSA) is also used for data set name (DSN) blocks. One DSN block is created for every data set that CICS file control opens, and they are recovered at a warm or emergency restart. If an application creates a large number of temporary data sets, all with a unique name, the number of DSN blocks can increase to such an extent that they can cause a short-on-storage condition.

If application programs use temporary data sets, with a different name for every data set created, it is important that these programs remove the temporary data sets after use. See SET DSNAME for information about how you can use this command to remove unwanted temporary data sets from your CICS regions.

Language Environment® runtime options for AMODE(24) programs

Two of the default Language Environment runtime options for CICS are ALL31(ON) and STACK(ANY). This means all programs that require Language Environment must be capable of addressing 31-bit storage, that is, must be AMODE(31) when Language Environment is enabled. For AMODE(24) programs to run in a Language Environment environment, you can specify ALL31(OFF) and STACK(BELOW). However, if you change these options globally so that all programs can use them, a lot of storage will be put below the 16 MB line, which might cause a short-on-storage condition.

For more information, see "Short-on-storage conditions in dynamic storage areas" on page 85.

Purging tasks

If a CICS task is suspended for longer than its DTIMOUT value, it might be purged if SPURGE=YES is specified on the RDO transaction definition. That is, the task is abended and its resources freed, thus allowing other tasks to use those resources. In this way, CICS attempts to resolve what is effectively a deadlock on storage.

If purging tasks is not possible or does not solve the problem, CICS stops processing. You must then cancel and restart the CICS region.

Reducing DASD paging activity

A large amount of DASD paging activity can slow down the rate at which transactions pass through the system.

About paging

The virtual storage of a processor might far exceed the size of the central storage available in the configuration. Any excess must be maintained in auxiliary storage (DASD). This virtual storage occurs in blocks of addresses called pages. Only the most recently referenced pages of virtual storage are assigned to occupy blocks of physical central storage. When reference is made to a page of virtual storage that does not appear in central storage, the page is brought in from DASD to replace a page in central storage that is not in use and least recently used.

The newly referenced page is said to have been paged in. The displaced page may need to be paged out if it has been changed.

It is the page-in rate that is of primary concern, because page-in activity occurs synchronously (that is, an MVS task stops until the page fault is resolved). Page-out activity is overlapped with CICS processing, so it does not appreciably affect CICS throughput.

A page-in from DASD incurs a time cost for the physical I/O and a more significant increase in processor usage.

Thus, extra DASD page-in activity slows down the rate at which transactions flow through the CICS system; that is, transactions take longer to get through CICS, you get more overlap of transactions in CICS, and so you need more virtual and real storage.

If you suspect that a performance problem is related to excessive paging, you can use RMF to obtain the paging rates.

Consider controlling CICS throughput by using MXT and transaction class limits in CICS on the basis that a smaller number of concurrent transactions requires less real storage, causes less paging, and may be processed faster than a larger number of transactions.

When a CICS system is running with transaction isolation active, storage is allocated to user transactions in multiples of 1MB. This means that the virtual storage requirement for a CICS system with transaction isolation enabled is very large. This does not directly affect paging that only affects those 4K byte pages that have been touched. More real storage is required in ELSQA, however. For more information about transaction isolation and real storage, see "Allocation of real storage when using transaction isolation" on page 120.

What is an ideal CICS paging rate from DASD? Less than one page-in per second is best to maximize the throughput capacity of the CICS region. Anything less than five page-ins per second is probably acceptable; up to ten may be tolerable. Ten per second is marginal, more is probably a major problem. Because CICS performance can be affected by the waits associated with paging, you should not allow paging to exceed more than five to ten pages per second.

Note: The degree of sensitivity of CICS systems to paging from DASD depends on the transaction rate, the processor loading, and the average internal lifetime of the CICS tasks. An ongoing, hour-on-hour rate of even five page faults per second may be excessive for some systems, particularly when you realize that peak paging rates over periods of ten seconds or so could easily be four times that figure.

What paging rates are excessive on various processors and are these rates operating-system dependent? Excessive paging rates should be defined as those that cause excessive delays to applications. The contribution caused by the high-priority paging supervisor executing instructions and causing applications to wait for the processor is probably a minor consideration as far as overall delays to applications are concerned. Waiting on a DASD device is the dominant part of the overall delays. This means that the penalty of high paging rates has almost nothing to do with the processor type.

CICS systems are usually able to deliver much better response times with somewhat better processor utilization when the potential of large amounts of central storage is exploited by keeping more data and programs in memory.

Program loading and paging

CICS employs MVS load under an MVS subtask to load programs. This allows the use of the library lookaside function of MVS to eliminate most DASD I/Os by keeping copies of programs in an MVS controlled dataspace.

A page-in operation causes the MVS task that requires it, to stop until the page has been retrieved. If the page is to be retrieved from DASD, this has a significant effect. When the page can be retrieved, the impact is only a relatively small increase in processor usage.

The loading of a program into CICS storage can be a major cause of page-ins. Because this is carried out under a subtask separate from CICS main activity, such page-ins do not halt most other CICS activities.

Reducing resource contention

Stress conditions are an indication that certain limit conditions have been reached and additional processing is required. The transactions involved must wait until resources are released.

The main limit conditions or constraints that can occur in a CICS system include those listed in "Identifying CICS performance constraints" on page 44.

To summarize, limit conditions can be indicated by the following:

- Virtual storage conditions (short-on-storage or SOS). This item in the CICS storage manager statistics shows a deficiency in the allocation of virtual storage space to the CICS region.

  In most circumstances, allocation of more virtual storage does not in itself cause a degradation of performance. You should determine the reason for the condition in case it is caused by some form of error. This could include failure of applications to free storage (including temporary storage), unwanted multiple copies of programs or maps, storage violations, and high activity of nonresident exception routines caused by program or hardware errors.

All new applications should be written to run above the 16MB line. The dynamic storage areas above the 16MB line can be expanded up to the 2GB limit of 31-bit addressing. The dynamic storage areas below the 16MB line are limited to less than the region size, which is less than 16MB.

- Number of simultaneous tasks (MXT and transaction class limit) reached (shown in the transaction manager statistics).
- Maximum number of z/OS Communications Server receive-any RPLs in use (shown in the z/OS Communications Server statistics).
- Wait-on-string and associated conditions for VSAM data sets (shown in the file control statistics).

Check how frequently the limit conditions occur. In general:

- If *no* limit conditions occur, this implies that too many resources have been allocated. This is quite acceptable if the resource is inexpensive, but not if the resource is both overallocated and of more use elsewhere.
- *Infrequent* occurrence of a limit condition is an indication of good usage of the particular resource. This usually implies a healthy system.
- *Frequent* occurrence (greater than 5% of transactions) usually reveals a problem, either directly or indirectly, that needs action to prevent more obvious signs of poor performance. If the frequency is greater than about 10%, you may have to take some action quickly because the actions taken by CICS itself (dynamic program storage compression, release of storage cushion, and so on) can have a perceptible effect on performance.

  Your own actions should include:

  – Checking for errors
  – Raising the limit, provided that it does not have a degrading effect on other areas
  – Allocating more resources to remove contention
  – Checking recovery usage for contention.

Resolving resource problems

This table provides information about the symptoms that indicate resource problems, their causes, and their solutions.

Follow this general procedure for resolving resource problems:

1. Confirm that your diagnosis of the type of constraint is correct, by means of detailed performance analysis. “Methods of performance analysis” on page 18 describes various techniques.
2. Read “Tuning your system” on page 22 for general advice on performance tuning.
3. See the relevant sections in “Improving the performance of a CICS system” on page 56 for detailed information on applying the various solutions.
4. Improve virtual storage exploitation by ensuring the following:
   - Large data buffers above the 16 MB line or in Hiperspace
   - Programs that run above the 16 MB line
   - Large amounts of real storage to support the virtual storage exploitation

   Such a system can deliver better internal response times, while minimizing DASD I/O constraint and reducing processor utilization.

Typical resource problems, their symptoms, and their solutions:

Problem	Symptom	Solution
Excessive DASD I/O operations: the amount of I/O operations needed to locate and fetch modules from DASD storage is excessive.	• Slow response times (the length of the response time depends on the number of I/O operations, with a longer response time when batch mode is active) • High DSA utilization • High paging rates • MXT limit frequently reached • SOS condition often occurs	• Reduce the number of I/O operations. • Tune the remaining I/O operations. • Balance the I/O operations load.
Slow transaction response on network: the average transaction response time for the network is unacceptably slow.	• Slow response times • Good response when few terminals are active on a line, but poor response when many terminals are active on that line • Big difference between internal response time and terminal response time	• Reduce the line utilization. • Reduce delays in data transmission. • Alter the network.
Slow response from remote system: the response time from a connected remote system is excessive.	• SOS condition or MXT occurs when there is a problem with a connected region • CICS takes time to recover when the problem is fixed	• Control the amount of queuing which takes place for the use of the connections to the remote systems. • Improve the response time of the remote system.
Excessive use of virtual storage: excessive use of common storage is occurring, or storage is not being freed at the end of a job or address space.	• Slow response times • Multiple loads of the same program • Increased I/O operations against program libraries • High paging rates • Frequent SOS condition	• Tune the MVS system to obtain more virtual storage for CICS (increase the region size). • Make more efficient use of the dynamic storage area.
Insufficient real storage: a program issued a request for real (processor) storage that specified a variable length with a maximum value that was too high.	• High paging rates • Slow response times • MXT limit frequently reached • SOS condition often occurs	• Reduce the demands on real storage • Tune the MVS system to obtain more real storage for CICS

Problem	Symptom	Solution
Excessive processor cycling: storage buffering or cycle stealing with integrated channels is occurring, or the amount of the queue searching is excessive.	• Slow response times • Low priority transactions respond very slowly • Low priority work very slow to complete	• Increase the dispatching priority of CICS. • Reevaluate the relative priorities of operating system jobs. • Reduce the number of MVS regions (batch). • Reduce the processor utilization for productive work. • Use only the CICS facilities that you really require. • Turn off any trace that is not being used. • Minimize the data being traced by reducing the scope of the trace, or by tracing less frequently. • Use a faster processor.

For more information about resolving performance problems see the z/OS Resource Measurement Facility (RMF) User's Guide.

Reducing storage violations

Storage violations can be reduced if CICS has storage protection and transaction isolation enabled.

CICS can detect storage violations when the duplicate storage accounting area (SAA) or the initial SAA of a TIOA storage element has become corrupted, or when the leading storage check zone or the trailing storage check zone of a user task storage has become corrupted.

A storage violation can occur in the following situations:

- When CICS detects an error during its normal processing of a FREEMAIN request for a TIOA storage element, and finds that the two storage check zones of the duplicate SAA and the initial SAA are not identical.
- CICS also detects user violations involving user task storage by checking the storage check zones of an element of user task storage following a FREEMAIN command.

When a storage violation is detected, an exception trace entry is made in the internal trace table. A message (DFHSM0102) is issued and a CICS system dump follows if the dump option is switched on.

For more information about storage violations, see Dealing with storage violations .

Performance management and capacity planning

Performance management means monitoring and allocating existing data processing resources to applications according to a Service Level Agreement (SLA) or informal service objectives. Capacity planning is the process of planning for sufficient computer capacity in a cost-effective manner to meet the future service needs for all users.

Performance management

The goal of performance management is to make the best use of your current resources to meet your current objectives, without excessive tuning effort. To formalize your objectives, you can set up a Service

Level Agreement (SLA). An SLA is a contract that objectively describes measurable performance factors, for example:

- Average transaction response time for network, I/O, processor, or total
- Transaction volumes
- System availability

A fundamental part of performance management is to measure transaction response time and break it down into components. This process shows you where tuning can be carried out for individual transactions. For effective performance management, you need to go on to measure resource requirements at the workload level. Analyzing your workload helps you to understand the behavior of your system and how workloads interact with each other.

The following IBM product documentation about performance tuning and management is available:

- z/OS Resource Measurement Facility (RMF) User's Guide
- z/OS MVS Initialization and Tuning Guide
- z/OS MVS Planning Workload Management

Capacity planning

Capacity planning involves asking the following questions:

- How much of your computer resources (processor, storage, I/O, network) are being used?
- Which workloads are consuming the resources (workload distribution)?
- What are the expected growth rates?
- When will the demands on current resources affect service levels?

The data that you gather, and the predictions that you make, help you to plan a schedule for upgrading your z Systems hardware, or for making additional enhancements such as adding zIIP and zAAP specialty processors to your system.

For more information about capacity planning, see the IBM Redbooks publication ABCs of z/OS System Programming, SG24-6327-01.

Relating CICS transactions to hardware resources

Use information provided by CICS monitoring and statistics to see what hardware resources in your system are being used by CICS transactions. You can use this data for capacity planning, and also for accounting and billing purposes.

About this task

The SMF monitoring records for each CICS transaction identify the CEC machine type and CEC model number for the physical hardware environment where the CICS region is running. CEC (central electronics complex) is a commonly used synonym for CPC (central processing complex), which refers to a collection of physical hardware including main storage, one or more central processors, timers, and channels. You can use further monitoring fields to calculate the processor time that the transaction spends on a zIIP or zAAP specialty processor, and to see the processor time that the transaction could have spent on a specialty processor.

Procedure

- To link your CICS workload to a specific CPC in your system, use the information in the CECMCHTP and CECMDLID fields in the DFHTASK performance class group.

  CECMCHTP shows the CEC machine type for the physical hardware environment, and CECMDLID shows the CEC model number.

  This information is also in the monitoring domain global statistics for the CICS region, reported by the DFHSTUP statistics reporting utility and the DFH0STAT sample statistics program.

Tip: You can use this information with the IBM Large Systems Performance Reference (LSPR) ratios to make an accurate assessment of CICS performance and relative processor capacity, particularly when considering upgrades to your z/OS hardware. For more information about the LSPR ratios, see Large Systems Performance Reference for IBM z Systems®.

- To calculate the actual and potential use of zIIP or zAAP specialty processors by CICS transactions, use the information in the CPUTONCP and OFFLCPUT fields in the DFHTASK performance class group:
  - Field 436, CPUTONCP, shows the total task processor time spent on a standard processor. To calculate the task processor time spent on a specialty processor, subtract the time recorded in this field from the time recorded in field 008, USRCPUT.
  - Field 437, OFFLCPUT, shows the total task processor time that was eligible for offload to a specialty processor, but actually ran on a standard processor. To calculate the total task processor time that was not eligible for offload, subtract the time recorded in this field from the time recorded in field 436, CPUTONCP.
  - To calculate the total task processor time that was either actually spent on a specialty processor, or eligible to be spent on a specialty processor, use the following equation:

    ```
    (OFFLCPUT + (USRCPUT - CPUTONCP))
    ```
- On sub-capacity hardware the general CPs (GCPs) run at a reduced speed for the processor model, but zIIP or zAAP specialty processors run at full speed. When running on sub-capacity hardware, the CPU time consumed on specialty engines must therefore be normalized to represent the equivalent time that the same transaction running on a GCP would have consumed.

  The CPU time returned by CICS in the USRCPUT field includes the total CPU time consumed on general CPs, plus the normalized CPU time spent on specialty engines. The CPUTONCP and OFFLCPUT fields both represent time spent executing exclusively on a general CP and therefore are not subject to normalization.

  The normalization factor for an LPAR is fixed at IPL time. Use the following fields in SMF records to determine the normalization factor for a given processor type.
  - Resource Measurement Facility (RMF) - use R723NFFI for zAAP and R791NFFS for zIIP
  - Common Address Space Work - use SMF30ZNF for zAAP and SMF30SNF for zIIP

  Note: The normalization process can produce data where the task response time is less than the time reported in the USRCPUT field.

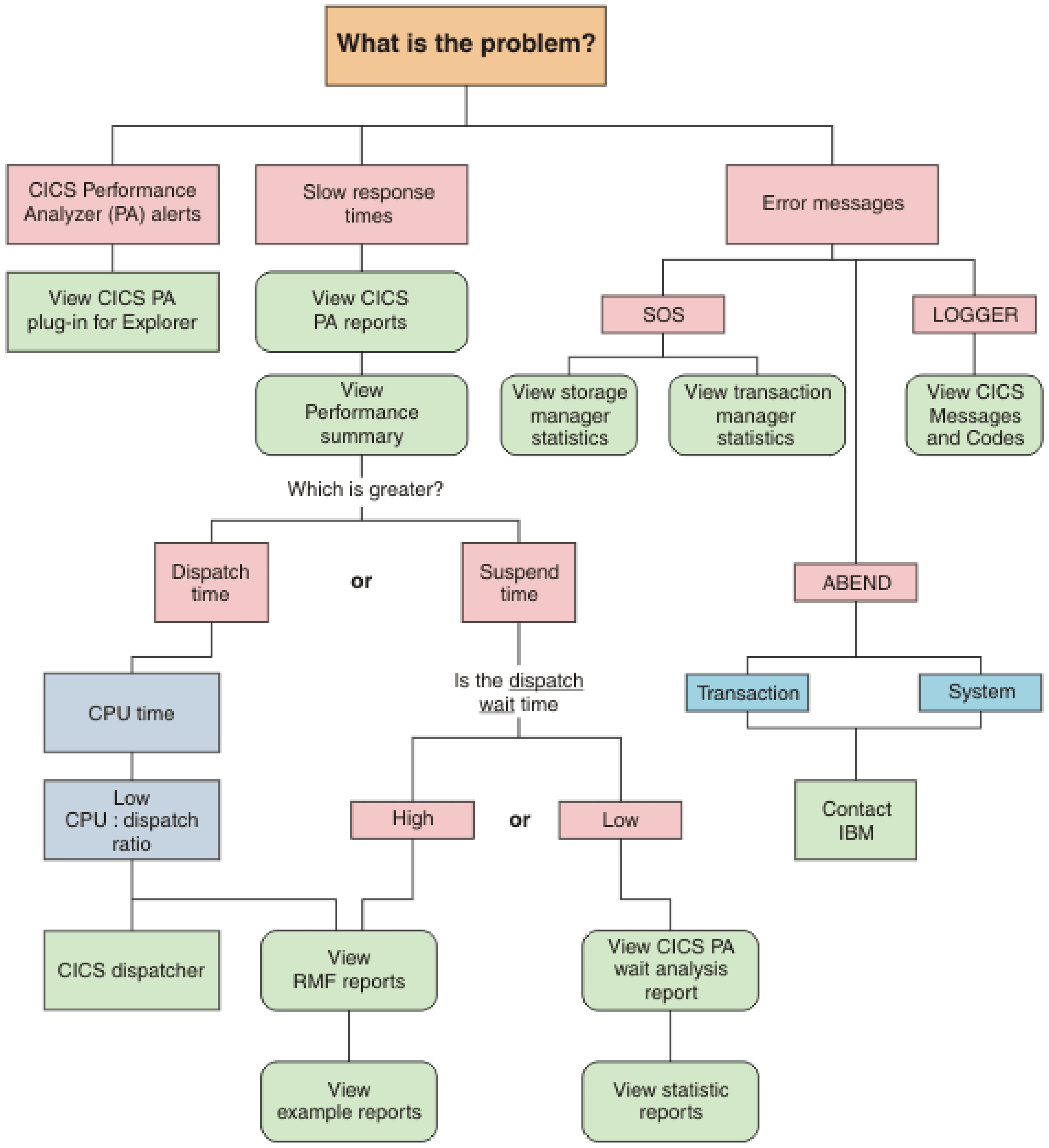
Improving the performance of a CICS system

Tuning is a key factor in improving the performance of a CICS system. You must always tune DASD, the network, and the overall MVS system before tuning any individual CICS subsystem through CICS parameters. Before you tune a system, you must understand why your CICS system is not performing as expected.

If you have concerns about the performance of your CICS system, use the performance flow diagram as a guide to understand why your CICS system is not performing as expected, and as a guide to solve the system problems.

The red boxes are selection boxes; depending on the result of your selection you are directed to different reports. The green boxes are links which, when clicked, open topics in the information center that describe how to use the performance analysis tools to improve the performance of your CICS system.

For example, if you determine that response times in your system are too slow, view your CICS Performance Analyzer (CICS PA) reports and the performance summary. Use the performance summary and the performance analyzer report to determine whether it is the dispatch time or the suspend time of tasks that is greater. If the suspend time is greater, determine whether the dispatch wait time is low or high. If the dispatch wait time is low, click the green box on the diagram that directs you to information about reports that provide the relevant information to help you improve the performance.



To help you improve performance, you can view tuning guidelines for different aspects of CICS:

- "Using data tables" on page 159
- "CICS dispatcher: performance and tuning" on page 62
- "Virtual and real storage: performance and tuning" on page 69
- "CICS storage protection facilities: Performance and tuning" on page 121
- "Tuning with Language Environment" on page 122
- "Java applications: performance and tuning" on page 124
- "MVS and DASD: performance and tuning" on page 124
- "Networking and the z/OS Communications Server: performance and tuning" on page 126
- "CICS MRO, ISC, and IPIC: performance and tuning" on page 138
- "CICS VSAM and file control: Performance and tuning" on page 147
- " Database management for performance" on page 175
- "CICS logging and journaling: Performance and tuning" on page 178

- “CICS temporary storage: Performance and tuning” on page 190
- “CICS transient data (TD) facility: Performance and tuning” on page 196
- “Global CICS enqueue and dequeue: Performance and tuning” on page 201
- “CICS monitoring facility: Performance and tuning” on page 202
- “CICS trace: performance and tuning” on page 203
- “CICS security: Performance and tuning” on page 204
- “CICS startup and shutdown time: Performance and tuning” on page 205
- “CICS business transaction services: Performance and tuning” on page 208
- “Managing workloads” on page 209
- Using RMF to monitor CICS

CICS Transaction Manager: performance and tuning

The CICS Transaction Manager domain provides transaction-related services.

The services provided by the domain are used to:

- Create tasks
- Terminate tasks
- Purge tasks
- Inquire on tasks
- Manage transaction definitions
- Manage tranclass definitions

The transaction manager domain also provides a transaction environment to enable other CICS components to implement transaction-related services.

For more information about transactions, see Transaction statistics.

Setting the maximum task specification (MXT)

The **MXT** system initialization parameter limits the total number of concurrent user tasks in the CICS system.

The **MXT** parameter controls unconstrained resource demand, particularly virtual storage usage in order to avoid short-on-storage (SOS) conditions. This parameter also affects the amount of storage allocated to the kernel stack segment and controls contention for resources, the length of queues (which can avoid excessive processor usage), and real storage usage.

The value of **MXT** affects the storage use in the CICS address space. You must ensure that enough storage is available for other users in the dynamic storage areas (DSAs).

The **MXT** parameter controls the number of user tasks that are eligible for dispatch. **MXT** can be set at startup, or by using the **SET SYSTEM** command. When **MXT** is set, the kernel and dispatcher attempt to preallocate sufficient control blocks to guarantee that the number of user tasks specified by the **MXT** value can be created concurrently. Most of the storage for this preallocation is obtained from the CDSA or ECDSA, although a small amount of MVS storage is required for each task (approximately 256 bytes in 31-bit storage, above the 16 MB line, and 32 bytes in 24-bit storage, below the 16 MB line, for each user task). The **MXT** value is interrelated with the z/OS REGION size, and the DSA size limits that you set (the **DSALIM** and **EDSALIM** parameters). See “Setting the limits for CICS storage” on page 76.

The **MXT** system initialization parameter has a default value of 250, a minimum setting of 10, and a maximum setting of 2000. Initially, set **MXT** to the total number of concurrent user tasks that you require in your system.

If you set the **MXT** value too high, throughput and response time can suffer when system resources (processor, real storage, and virtual storage) are constrained or resource contention occurs, for example file strings or buffers, DB2 threads, ENQs and so on. Also, if you set the **MXT** value too high at startup, CICS forces a smaller maximum number of tasks consistent with the available virtual storage.

Conversely, if you set the **MXT** value too low, throughput and response time can also suffer due to excessive queuing delays even though system resources (processor, real storage, and virtual storage) are not constrained.

Monitor the performance of the CICS region to ensure that the response time and other time components (such as dispatch time and suspend time) for your transactions are acceptable. In some systems, setting **MXT** too high might increase resource contention to a level that causes additional delays for transactions. You can use the transaction manager global statistics from the DFH0STAT and DFHSTUP utility programs to monitor the **MXT** value.

If performance tuning for HTTP connections is enabled, when the region is at capacity, instead of queuing HTTP requests on MXT in CICS (which involves internal processing and storage requirements), requests are queued outside of CICS in the TCP/IP backlog. MXT is no longer exceeded by a surge of HTTP requests (externalized as the **XMGPQT** field in the transaction manager global statistics), but the number of times MXT is reached (externalized as the **XMGTAMXT** field in the transaction manager global statistics) might increase. It can occur if the region remains under stress when CICS processes each request from the backlog queue; as the task to process the request is attached CICS goes back to MXT. Once the levels on MXT are decreased CICS will accept the next request, and its transaction might cause CICS to go back to MXT. It does not indicate that MXT needs to be increased, it shows that its current value is correctly protecting CICS from unconstrained resource demand.

In addition, the following performance data fields in the DFHTASK group are useful to assess the relationship between the task load during the life of a transaction, and the performance of the transaction:

- CURTASKS records the current number of active user transactions in the system at the time the user task was attached.
- MAXTASKS records the current setting for the maximum number of tasks for the CICS region at the time the user task was attached.
- MXTDELAY records the elapsed time waiting for the first dispatch when the delay is because the **MXT** value is reached.

To alter the **MXT** value while CICS is running, you can use the **SET SYSTEM MAXTASKS** command. If you set the **MXT** value too high while CICS is running, the error message: "CEILING REACHED" is displayed. The CICS transaction manager statistics show the number of times the **MXT** ceiling has been reached.

Note: If the MAXOPENTCBS or MAXXPTCBS system initialization parameters have not been specified, then the **MXT** parameter also sets the MAXOPENTCBS and MAXXPTCBS parameters.

Important: Before you change the **MXT** value, review the information in Open TCB pools.

Using transaction classes (MAXACTIVE) to control transactions

Transaction classes give you a mechanism to limit the number of CICS tasks in your system. By spreading your tasks across a number of transaction classes and controlling the maximum number of tasks that can be dispatched within each transaction class, you can control resource contention between tasks and limit the number of tasks that CICS considers eligible for dispatching at task attach.

Use the *MAXACTIVE* attribute of the transaction class definition (TRANCLASS) to control a specific set of tasks that are heavy resource users, tasks of lesser importance (for example, “Good morning” broadcast messages), and so on, allowing processor time or storage for other tasks. Together with the **MXT** system initialization parameter, transaction classes control the transaction mix, that is, ensuring that one type of transaction does not monopolize CICS. In particular, you can restrict the number of heavyweight tasks, the load on particular data sets or disk volumes, and the printer load on lines. For example, you can use transaction classes to isolate tasks, or put all user tasks into separate classes. Suggested classes are simple inquiries, complex inquiries or short browses, long browses, short updates, long updates. Separate nonconversational tasks from conversational tasks. If you need to single-thread non-reentrant code, use ENQ for preference.

Using transaction classes can be useful for tasks that consume particularly large amounts of resource, but that do not exceed the MAXACTIVE ceiling frequently. Do not use transaction classes for normal tasks or for design reasons such as serializing a function within a particular task. Application design should be reviewed as an alternative in these cases.

CICS transaction class statistics show the number of times that the number of active transactions in the transaction class reached the MAXACTIVE value (Times MaxAct). CICS defines two transaction classes for its own use, DFHTCLSX and DFHTCLQ2. For information about the effects these have, see "Using transaction classes DFHTCLSX and DFHTCLQ2 to control storage use" on page 143.

Specifying a transaction class purge threshold (PURGETHRESH)

The PURGETHRESH attribute of the transaction class definition limits the number of tasks that are newly created, but cannot be started because the MAXACTIVE limit has been reached for the associated transaction class. These tasks are queued by the transaction manager domain in priority order until they obtain class membership.

The tasks occupy small amounts of storage, but if the queue becomes very long, CICS can become short-on-storage and take a considerable time to recover. Systems where a heavy transaction load is controlled by the TRANCLASS mechanism are most prone to being overwhelmed by the queue. The tasks on the queue are not counted by the MXT mechanism. The **MXT** system initialization parameter limits the total number of tasks that have already been admitted to the system within TRANCLASS constraints.

The length of the queue of tasks waiting to be started in a transaction class is limited by the PURGETHRESH attribute of that class. Any new transaction which would cause the limit to be reached is ended with the abend code AKCC. Tasks that were queued before the limit was reached are allowed to continue waiting until they can be executed.

The PURGETHRESH attribute should be specified only where the transaction load in a transaction class is heavy. This is the case in a system which uses a terminal-owning region (TOR) and multiple application-owning regions (AORs) and where the transaction classes are associated with the AORs and are used to control the numbers of transactions attempting to use the respective AORs. In this configuration, an AOR can slow down or stop and the associated transaction class fills (up to the value defined by MAXACTIVE) with tasks that are unable to complete their work in the AOR. New transactions are then queued and the queue can grow to occupy all the available storage in the CICS DSA within a few minutes, depending on the transaction volume.

The size of each entry in the queue is the size of a transaction (256 bytes), plus the size of an interval control element (ICE) secure storage extension (2108 bytes), plus the size of the TIOA holding any terminal input to the transaction. There can be any number of queues, one for each TRANCLASS that is installed in the TOR. You can estimate a reasonable size purge threshold for the queue by multiplying the maximum length of time you are prepared for users to wait before a transaction is started by the maximum arrival rate of transactions in the TRANCLASS. Make sure that the queues cannot occupy excessive amounts of storage at their maximum lengths.

The PURGETHRESH queuing limit should not be set so low that CICS abends transactions unnecessarily, for example when an AOR slows down due to a variation in the load on the CPU. The PURGETHRESH attribute of a TRANCLASS is used to set the limit of the queue for that transaction class. The default action is not to limit the length of the queue.

To monitor the lengths of the queues for each transaction class you should use CICS transaction class statistics. Many statistics are kept for each transaction class. These are the most useful statistics for monitoring queue lengths:

XMCPI
Number of transactions abended AKCC because the size of the queue reached the PURGETHRESH limit.

XMCPQT
The peak number of transactions in the queue.

XMCTAPT
The number of times the size of the queue reached the PURGETHRESH limit.

You can monitor the number of AKCC abends in the CSMT log. The AKCC abends indicate the periods when the queue limit was reached. You must correlate the transaction codes in the abend messages with the transaction classes to determine which limit was being reached.

Prioritizing tasks

Prioritization is a method of giving specific tasks preference in being dispatched. Priority is specified in the TERMINAL definition (TERMPRIORITY), a transaction in a TRANSACTION definition (PRIORITY), and in the priority field of the user segment of the external security manager (ESM), (OPPRTY).

Overall priority is determined by summing the priorities in all three definitions for any given task, with the maximum priority being 255:

```
TERMPRIORITY+PRIORITY+OPPRTY <= 255
```

The value of the PRTYAGE system initialization parameter also influences the dispatching order; for example, the default value **PRTYAGE**=1000 causes the task's priority to increase by 1 every 1000ms it spends on the ready queue. The dispatching priority of a task is reassessed each time it becomes ready for dispatch, based on clock time as well as defined priority. A task of priority n+1 that has just become ready for dispatch is usually dispatched ahead of a task of priority n, but only if PRTYAGE milliseconds have not elapsed since the latter last became ready for dispatch. Therefore, a low priority task might be overtaken by many higher priority tasks in a busy system, but eventually arrives at the top of the ready queue for a single dispatch. The lower the value of PRTYAGE, the sooner the task is dispatched. PRTYAGE should usually remain at its default value, unless certain transactions get stuck behind higher priority transactions during very busy periods.

Note: Non-terminal transactions are attached with a priority value based on the transaction priority from the TXD, and the operator priority, while terminal control based tasks are attached with only the transaction priority. When a terminal control based task first gets dispatched, the operator priority and terminal priority are added in. For this reason, terminal and non-terminal based tasks must not be managed through the same transaction class, because a steady stream of non-terminal based transactions could take precedence over other terminal control based transactions on a sufficiently busy system.

Prioritization is useful for browsing tasks, and tasks that use a lot of processor time. Input/Output bound tasks can take the required amount of CPU, and move on to the next read/write wait. CPU-intensive tasks take higher priority over the less intensive tasks. Prioritization can be implemented in all CICS systems. It is more important in a high-activity system than in a low-activity system. With careful priority selection, you can improve overall throughput and response time. Prioritization can minimize resource usage of certain resource-bound transactions. Prioritization increases the response time for lower-priority tasks, and can distort the regulating effects of MXT and the MAXACTIVE attribute of the transaction class definition.

Priorities do not affect the order of servicing terminal input messages and, therefore, the time they wait to be attached to the transaction manager. Because prioritization is determined in three sets of definitions (terminal, transaction, and operator), it can be a time-consuming process for you to track many transactions in a system. CICS prioritization is not interrupt-driven as is the case with operating system prioritization, but determines the position on a ready queue. This means that, after a task is given control of the processor, the task does not relinquish that control until it issues a CICS command that calls the CICS dispatcher. After the dispatch of a processor-bound task, CICS can be tied up for long periods if CICS requests are infrequent. For that reason, prioritization should be implemented only if MXT and the MAXACTIVE attribute of the transaction class definition adjustments have proved to be insufficient.

You should use prioritization sparingly, if at all, and only after you have already adjusted task levels using MXT and the MAXACTIVE attribute of the transaction class definition. It is probably best to set all tasks to the same priority, and then prioritize some transactions either higher or lower on an exception basis, and according to the specific constraints in a system. Do not prioritize against slow tasks unless you can accept the longer task life and greater dispatch overhead; these tasks are slow, in any case, and give up control each time they have to wait for I/O. Use small priority values and differences and concentrate on transaction priority. Give priority to control operator tasks rather than the person, or at least to the control operator's signon ID rather than to a specific physical terminal (the control operator may move around).

Consider for high priority a task that uses large resources. However, the effects of this on the overall system need careful monitoring to ensure that loading a large transaction of this type does not lock out other transactions. Also consider for high priority those transactions that cause enqueues to system resources, thus locking out other transactions. As a result, these can process quickly and then release resources. Here are some examples:

- Using intrapartition transient data with logical recovery
- Updating frequently used records
- Automatic logging
- Tasks needing fast application response time, for example, data entry.

Lower priority should be considered for tasks that:

- Have long browsing activity
- Are process-intensive with minimal I/O activity
- Do not require terminal interaction, for example:
  - Auto-initiate tasks (except when you use transient data intrapartition queues that have a destination of terminal defined and a trigger level that is greater than zero).
  - Batch update controlling tasks.

There is no direct measurement of transaction priority. Indirect measurement can be made from:

- Task priorities
- Observed transaction responses
- Overall processor, storage, and data set I/O usage.

CICS dispatcher: performance and tuning

You can tune the performance of the CICS dispatcher by specifying dispatch intervals. You specify dispatch intervals by setting system initialization parameters for interval control values and other parameters such as FORCEQR, MROBTCH, PRTYAGE, and SUBTSKS.

For more information about dispatcher statistics, see Dispatcher TCB Modes report.

Open TCB management

The open transaction environment (OTE) is an environment where CICS application code can use non-CICS services (facilities outside the scope of the CICS API) inside the CICS address space, without causing wait issues on the quasi-reentrant task control block (QR TCB).

Applications that exploit the open transaction environment run on their own open TCB, rather than on the QR TCB. CICS does not perform subdispatching on an open TCB, whereas it does on the QR TCB. If the application that is running on an open TCB calls a non-CICS service that blocks the TCB, the TCB blocking does not affect other CICS tasks. For more information about writing applications to exploit the open transaction environment, see Multithreading: Reentrant, quasi-reentrant, and threadsafe programs .

TCB modes

Each open TCB mode has a 2-character identifier to indicate its specific purpose, and is handled by CICS in a different way.

L8 mode TCBs and L9 mode TCBs

These TCBs are used as follows:

- L8 TCBs are used for CICS-key application programs that are defined as API(OPENAPI) by their PROGRAM resource definition.
- L8 TCBs are used for CICS-key application programs that are defined as CONCURRENCY(REQUIRED), API(CICSAPI) by their PROGRAM resource definition.

- L8 TCBs are used when programs need access to a resource manager through a task-related user exit (TRUE) that was enabled using the OPENAPI option on the ENABLE PROGRAM command. Task-related user exits always run in CICS key.
- L8 TCBs are used by CICS when accessing document templates and HTTP static responses that are stored in z/OS UNIX System Services files.
- L8 TCBs are used for web service requests and parsing XML CICS programs that run in CICS key and are defined as OPENAPI.
- L9 TCBs are used for application programs that run in user key and are defined as OPENAPI.

CICS operates with an OPENAPI task-related user exit, and therefore uses L8 TCBs, when it is connected to the following products:

- IBM MQ, using the CICS-MQ adapter
- DB2, using the CICS DB2 Attachment Facility. For more information on the CICS DB2 attachment facility thread TCBs in the open transaction environment, see Overview: How threads work.
- IMS Version 12 or later, using the CICS DBCTL Database Adapter Transformer (DFHDBAT). See Enabling CICS IMS applications to use the open transaction environment (OTE) through threadsafe programming.

Other IBM products, for example, IP CICS Sockets and the z/OS Integrated Cryptographic Service Facility (ICSF), can also use an OPENAPI enabled task-related user exit. For more information about managing IP CICS sockets, see z/OS Communications Server: IP CICS Sockets Guide. For more information about the CICS-ICSF Attachment Facility, see z/OS Cryptographic Services ICSF System Programmer's Guide.

SP mode TCB and S8 mode TCBs

These TCBs are used by CICS to manage SSL connections and requests to LDAP using the DFHDDAPX XPI interface. The S8 TCBs run in a single enclave, which is owned by the SP TCB, and also contains the SSL cache.

The S8 TCBs are contained in an SSL pool, which is managed by the CICS dispatcher. Each SSL connection uses an S8 TCB from the SSL pool. All CICS processing for an SSL connection occurs on the S8 TCB (there is no switching between the S8 and SO TCBs). The web server HTTP attach task (CWXN by default) will stay on the S8 TCB until all the data have been sent or received.

An S8 TCB is allocated to a task from the SSL pool, but is locked only for the period of time that it takes to perform functions such as an SSL handshake or an LDAP request. After this function is complete, the TCB is released back into the SSL pool to be reused.

Note: If the messages being sent or received are very large, the task could hit the runaway limit and be terminated. To avoid the task being abended, you might need to increase the transaction RUNAWAY value for the web server HTTP attach transaction (CWXN by default), the web server alias transactions, and any transactions that issue the web client API commands SEND, RECEIVE, and CONVERSE.

In UNIX System Services (USS), the **MAXTHREADS** and **MAXTHREADTASKS** parameters can be used to restrict the number of pthreads that a USS process can own. Each SSL TCB requires a pthread and an MVS task. You must therefore ensure that the values of these USS parameters exceed the value of the **MAXSSLTCBS** system initialization parameter. If you do not set a large enough value for **MAXTHREADS** or **MAXTHREADTASKS** and CICS reaches one of these limits while attempting to attach an SSL TCB, CICS issues error message DFHDS0002 severe error code X'0137' from DFHDSIT.

TP mode TCB and T8 mode TCBs

These TCBs are used by a JVM server to process requests for Java programs. The JVM server is a runtime environment that can handle multiple concurrent requests for Java applications in a single JVM. The TP mode TCB owns the Language Environment enclave and the pool of T8 TCBs. Each JVM server that is running in the CICS region has one TP TCB and at least one, but not more than 256, T8 TCBs. A T8 TCB is allocated to a task from the THRD pool of the appropriate JVM server, but is locked only for the period of time that it takes to perform the system processing. T8 TCBs are not shared between JVM servers.

Each T8 TCB requires a pthread and an MVS task. The maximum number of T8 TCBs that is allowed for the CICS region is 2000. In z/OS UNIX, you can use the **MAXTHREADS** and **MAXTHREADTASKS** parameters to restrict the number of pthreads that a z/OS UNIX process can own. You must therefore ensure that the values of these parameters exceed the maximum number of T8 TCBs. If you do not set a large enough value for **MAXTHREADS** or **MAXTHREADTASKS** and CICS reaches one of these limits while attempting to attach a T8 TCB, CICS issues error message DFHDS0002 severe error code X'0137' from DFHDSIT. For more information about the thread limits of JVM servers, see Managing the thread limit of JVM servers.

X8 mode TCBs and X9 mode TCBs

Both these TCBs are used to run C and C++ programs compiled with the XPLINK option. X8 TCBs are used for programs in CICS key, and X9 TCBs are used for programs in user key. Each instance of an XPLink program uses one X8 or X9 TCB. For more information about using XPLink, see XPLink and C and C++ programming .

Open TCB pools

CICS manages open TCBs in *pools*. A pool contains open TCBs that are used for the same purposes. A pool can consist of some TCBs that are allocated to tasks, and others that have been freed by applications and are available for reuse.

CICS can create or attach open TCBs in each pool up to the limit set for the pool. The maximum number of TCBs allowed in each pool is set as follows

- The **MAXOPENTCBS** system initialization parameter, if specified, sets the value for the open TCB pool. If the MAXOPENTCBS system initialization is not specified, CICS sets the limit for the L8 and L9 mode open TCB pool automatically based on the maximum number of tasks specified for the CICS region (the MXT value), using the following formula: `(2 * MXT Value) + 32`. For information about explicitly setting the MAXOPENTCBS parameter yourself, see MAXOPENTCBS.
- The **MAXSSLTCBS** system initialization parameter specifies the value for the SSL TCB pool.
- The **MAXTHRDTCBS** system initialization parameter specifies the value for the JVM server THRD TCB pool. The number of threads reserved for each JVM server is the THREADLIMIT value on the JVMSERVER resource, plus 1, up to a limit of 2000.
- The **MAXXPTCBS** system initialization parameter, if specified, sets the value for the XP TCB pool. If the **MAXXPTCBS** system initialization is not specified, CICS sets the limit for the X8 and X9 mode XP TCB pool automatically to a value equal to the maximum number of tasks specified for the CICS region (the MXT value). For information about explicitly setting the MAXXPTCBS parameter yourself, see MAXXPTCBS .

When an application makes a request that requires an open TCB, CICS first tries to find a suitable TCB that is available for reuse in the appropriate pool. CICS can match a request with an available TCB of the correct mode only if the TCB is for the correct subspace. CICS attaches a new TCB when it cannot find a suitable match with a free TCB for the correct subspace, provided that the limit for the pool has not been reached.

A CICS task is allowed as many X8 and X9 TCBs as it requires, and these TCBs are kept only until the program finishes. However, each CICS task is allowed at most one L8 and one L9 TCB, and it keeps an L8 and an L9 TCB from the time it is allocated to the end of the task, reusing it for further requests as needed. The TCBs then become free, and CICS can allocate them to another task or destroy them.

Sometimes CICS cannot find a suitable match for an application’s request, and the limit for the pool has been reached. In this situation, CICS might fulfill the request by destroying a free TCB in the pool that has the wrong subspace or mode, and replacing it with a TCB with the right mode and subspace. This technique is called *stealing*. Stealing can be costly on performance, depending on the type of open TCB, so CICS avoids it where it makes sense to do so. CICS maintains statistics of excess TCB management and TCB stealing activities in the CICS dispatcher TCB mode and TCB pool statistics.

If the number of TCBs is at the limit for the pool and there is no free TCB to steal, the task is suspended with an OPENPOOL wait until a TCB becomes free, or the limit for the pool is increased.

To minimize the impact on storage, CICS attempts to balance the number of open TCBs in each pool against current needs. If CICS finds free TCBs in a pool, it gradually reduces the excess number by detaching them, freeing the resources used by the excess TCBs.

MAXSSLTCBS

You can use the dispatcher TCB statistics from the DFH0STAT and DFHSTUP utility programs to monitor the S8 TCBs in the SSL pool. The maximum number of TCBs is set by the **MAXSSLTCBS** system initialization parameter.

If you want to improve the performance of SSL, you can use the dispatcher reports to find out if there are many tasks waiting for an S8 TCB. Also look at the number of tasks that have queued. If both fields report a large number, increase the maximum number of S8 TCBs. If you have few tasks queued, but many waits, you can decide whether you want to increase the number of S8 TCBs. Increasing the number by one or two could make a difference to the number of waits and reduce the tasks queued, without causing significant overheads in storage.

The maximum number of S8 TCBs that you can set is 1024. However, setting many S8 TCBs can also affect performance because of the amount of storage used. If CICS runs out of storage, you get a TCB attach failure. This failure is reported in the dispatcher reports for the S8 TCB mode statistics.

Interval control value parameters: ICV, ICVR, and ICVTSD

The interval control values (ICVs) are specified in the system initialization table (SIT) to set a new value or by overrides. Setting the ICV parameters correctly can help to improve performance by reducing processor usage for low utilization CICS regions.

CICS has three types of interval control value parameters:

Interval control value (ICV)
The **ICV** system initialization parameter specifies the maximum time in milliseconds that CICS releases control to the operating system when there are no transactions ready to resume processing. This time interval can be any integer in the range 100 through 3600000 milliseconds (specifying an interval up to 60 minutes). A typical range of operation might be 100 through 2000 milliseconds.

Interval control value for runway tasks (ICVR)
The **ICVR** system initialization parameter specifies the default runaway task time interval in milliseconds as a decimal number. You can specify zero, or a number in the range 250 through 2700000, in multiples of 250. CICS rounds down values that are not multiples of 250. This is the RUNAWAY interval that is used by transactions defined with RUNAWAY=SYSTEM.

Interval control value for terminal scan delay (ICVTSD)
The **ICVTSD** system initialization parameter was used in earlier releases to limit how quickly CICS dealt with some types of terminal output requests made by applications, in order to spread the overhead for dealing with the requests. The range is 0 through 5000 milliseconds. Specifying a nonzero value was sometimes appropriate where the CICS system used non-SNA networks. However, with SNA and IPIC networks, setting ICVTSD to 0, which is the default, is appropriate to provide a better response time and best virtual storage usage.

MROBTCH

The **MROBTCH** system initialization parameter specifies how many events in a region can be accumulated in a batch before posting.

The region is started so that it can process the requests. The batching of multiregion operation (MRO) requests includes some non-MRO events:

- VSAM physical I/O completion events
- Subtasked request completion (mostly VSAM)
- DL/I request completion implemented through DBCTL

The value of the **MROBTCH** parameter can be in the range of 1 through 255, and the default is 1. Using this batching mechanism, you can spread the dispatch resource usage in CICS over several tasks. If the value is greater than 1 and CICS is in a system wait, CICS is not posted for dispatch until the specified number of events has occurred. Events include MRO requests from connected systems or DASD I/O and CHANGE_MODE processing. For these events, CICS is dispatched after:

- The current batch fills up (the number of events equals **MROBTCH**)
- An ICV interval expires

The time interval that you specify in the ICV parameter should be low enough to prevent undue delay to the system.

During periods of low utilization, a value of **MROBTCH** greater than 1 can result in increased transaction response times. Transactions that issue I/O requests might be delayed because of an increased FCIOWAIT value. For more information about the effect of MROBTCH on performance, see "Batching requests (MROBTCH)" on page 144.

FORCEQR

The **FORCEQR** system initialization parameter specifies whether you want CICS to force all CICS API user application programs that are specified as threadsafe to run under the CICS quasi-reentrant (QR) task control block (TCB), as if they were specified as quasi-reentrant programs.

If your programs are defined as quasi-reentrant, CICS always calls them under the CICS QR TCB. The requirements for a quasi-reentrant program in a multithreading context are less stringent than if the program were to execute concurrently on multiple TCBs. CICS requires that an application program is reentrant so that it guarantees consistent conditions. In practice, an application program may not be truly reentrant; CICS expects "quasi-reentrancy". This means that the application program should be in a consistent state when control is passed to it, both on entry, and before and after each **EXEC** CICS command. Such quasi-reentrancy guarantees that each invocation of an application program is unaffected by previous runs, or by concurrent multi-threading through the program by multiple CICS tasks.

CICS quasi-reentrant user programs (application programs, user-replaceable modules, global user exits, and task-related user exits) are given control by the CICS dispatcher under the QR TCB. When running under this TCB, a program can be sure that no other quasi-reentrant program can run until it relinquishes control during a CICS request, at which point the user task is suspended, leaving the program still "in use". The same program can then be reinvoked for another task, which means the application program can be in use concurrently by more than one task, although only one task at a time can actually be executing.

Running application with programs defined as threadsafe to use OTE, such as CICS DB2 applications, could cause problems if one or more programs is not threadsafe. Using the **FORCEQR** system initialization parameter, you can force all your applications onto the QR TCB.

Forcing applications on the QR TCB is useful in production regions where you cannot afford to have applications out of service while you investigate the problem.

The default for this parameter is **FORCEQR=NO**, which means that CICS honors the **CONCURRENCY** attribute on your program resource definitions. As a temporary measure, while you investigate and resolve problems connected with threadsafe-defined programs, you can set **FORCEQR=YES**. When all problems have been resolved, resetting **FORCEQR=NO** makes all programs resume use of open TCBs under the OTE.

The **FORCEQR** parameter applies to all application programs that are restricted to the current CICS programming interfaces (programs that specify API(CICSAPI)). The parameter does not apply to any of the following programs:

- Java programs that are run in JVM
- C or C++ programs using XPLINK
- OPENAPI programs
- Programs defined with CONCURRENCY(REQUIRED)

The **FORCEQR** parameter applies to all programs defined as threadsafe that are not used as task-related user exits, global user exits, or user-replaceable modules.

PRTYAGE

The system initialization parameter, **PRTYAGE**, can be set to a value that determines the rate at which tasks in the system have their priorities aged.

The priority of a task within CICS determines the order it is dispatched. Tasks can have priority values of 1 through 255. If a task's first dispatch is too slow, changing the priority to a higher value shortens the dispatch time. You have no control over the priorities of CICS system tasks. Adjusting the value of **PRTYAGE** does not control the priorities of tasks, only how CICS sets the priorities of tasks. Altering the value of **PRTYAGE** affects the rate at which tasks are dispatched.

SUBTSKS

The **SUBTSKS** system initialization parameter specifies the number of task control blocks (TCBs) that CICS uses to run tasks in concurrent mode.

The value of the **SUBTSKS** parameter is either 1 or 0. The value of 1 turns subtasking on and the value of 0 turns subtasking off.

Using the value of 0, which is the default value for **SUBTSKS**, CICS runs under the quasi-reentrant (QR) TCB and runs all applications under the QR TCB. At this value, CICS also runs tasks that open and close files under the resource-owning mode TCB.

If the parameter value is set to 1, CICS runs under the resource-owning TCB and the QR TCB, and uses an additional TCB, a concurrent mode TCB, to perform system subtasking.

TCB statistics

The task control block (TCB) dispatcher statistics report the amount of CPU time consumed by each CICS TCB since the last time statistics were reset.

To calculate the approximate time since CICS statistics were last reset, add the values of "Accum time in MVS wait" and "Accum time dispatched". To calculate the percentage usage of each CICS TCB, divide the value of "Accum CPU time/TCB" by the time since CICS statistics were last reset (as calculated earlier).

The "Accum CPU time/TCB" value does not include uncaptured time. Therefore, when you use this calculation, even a very busy CICS TCB is noticeably less than 100% busy. If the calculation indicates that a CICS region is more than 70% busy, you are approaching the capacity of the region. However, the 70% calculation can be only approximate. The capacity of the region depends on such factors as the workload in operation, the mix of activity in the workload, and which release of CICS you are currently using. You can use Resource Measurement Facility (RMF) to obtain a definitive measurement to use in your calculation, or you can use RMF with your monitoring system. For more information, see the z/OS Resource Measurement Facility (RMF) Report Analysis.

Note: "Accum time dispatched" is *not* a measurement of CPU time. MVS can run higher priority work, for example all I/O activity and higher priority regions, without CICS being aware.

TCB modes are as follows:

QR
: There is always one quasi-reentrant mode TCB. It is used to run quasi-reentrant CICS code and non-threadsafe application code.

FO
: There is always one file-owning TCB. It is used for opening and closing user data sets.

RO
: There is always one resource-owning TCB. The RO TCB is used for loading programs, unless the command to load the program (EXEC CICS LOAD, XCTL, or LINK) is issued by an application that is currently running on an open TCB. In that situation, the open TCB is used to load the program instead of the RO TCB. The RO TCB is also used for opening and closing CICS data sets, issuing RACF calls, and similar tasks.

: The CICS loader domain global statistics record the number of program load operations that took place on the RO TCB, and the time taken for them. You can compare these values to the overall statistics for the number and time of program load operations, to see the proportion of program load operations that took place on open TCBs instead of the RO TCB.

CO
The optional concurrent mode TCB is used for processes that can safely run in parallel with other CICS activity such as VSAM requests. Define the system initialization parameter **SUBTSKS** using the value 0 or 1 to specify whether there is a CO TCB.

D2
The D2 mode TCB is used to stop DB2 protected threads. Protected threads are stopped in the normal purge cycle, or when a user issues the **DSNC DISCONNECT** *plan-name* command, which stops the protected threads for a plan immediately.

SZ
The single optional SZ mode TCB is used by the FEPI interface.

RP
The single optional RP mode TCB is used to make ONC/RPC calls.

EP
The EP mode TCBs are used to run event processing in a CICS region. The TCBs either dispatch events to an appropriate EP adapter or defer filtering of system events.

L8
A task has an L8 mode TCB for its sole use when it calls a program that is enabled with the OPENAPI option and is defined with EXECKEY=CICS, or when it calls a task-related user exit program that is enabled with the OPENAPI option. An L8 TCB is used when CICS uses the CICS-MQ adapter to connect to WebSphere MQ Version 6 or later and when CICS connects to DB2 Version 8 or earlier.

L9
A task has an L9 mode TCB for its sole use when it calls a program that is enabled with the OPENAPI option and is defined with **EXECKEY=USER**.

SO
The SO mode TCB is used to make calls to the socket interface of TCP/IP.

SL
The SL mode TCB is used to wait for activity on a set of listening sockets.

S8
A task uses an S8 TCB if it needs to use the system Secure Sockets Layer (SSL). A task also uses an S8 TCB if it needs to use LDAP over the DFHDDAPX XPI interface. The TCB is used only for the duration of the SSL negotiation or the LDAP request. On completion, the TCB is released back into the SSL pool to be reused.

SP
The SP mode TCB is used for socket pthread owning tasks. It manages the SSL pool of S8 TCBs and owns the Language Environment enclave that contains the SSL cache.

T8
A Java application running in a JVMSERVER uses a T8 TCB. The T8 will also be used for DB2 requests from Java applications when CICS is connected to DB2 Version 9 or later.

TP
The TP mode TCB owns and manages the Language Environment enclave, the JVM, the THRD TCB pool, and T8 TCBs of a JVM server.

X8
A task has an X8 mode TCB for its sole use when it calls a C or C++ program that is compiled with the XPLINK compiler option and defined with **EXECKEY=CICS**. The CICS-DB2 task-related user exit may use a X8 TCB if it is run with CONCURRENCY(REQUIRED) and API(CICSAPI).

X9
A task has an X9 mode TCB for its sole use when it calls a C or C++ program that is compiled with the XPLINK compiler option and defined with **EXECKEY=USER**.

Virtual and real storage: performance and tuning

Learn about virtual and real storage use in a z/OS system and in CICS regions, and start to monitor performance and tune your use of storage.

Procedure

1. Understand how virtual and real storage is arranged and managed in a z/OS address space.

   For information about address spaces in z/OS, read the following z/OS documentation:

   - For information about 24-bit and 31-bit storage (below the bar) in an address space, see Virtual Storage Overview in the z/OS MVS Initialization and Tuning Guide.
   - For information about 64-bit (above-the-bar) storage in an address space, see Using the 64-bit Address Space in the z/OS MVS Programming: Extended Addressability Guide.
2. Understand how virtual storage is arranged and managed when a z/OS address space is used by a CICS region.

   For information about how CICS uses a z/OS address space, see “CICS virtual storage” on page 69.
3. Monitor and measure the use of virtual storage across your z/OS system using z/OS performance and monitoring tools, such as the z/OS Resource Measurement Facility (RMF).

   For an overview of RMF, see “Resource measurement facility (RMF)” on page 32.

   For further information, see the z/OS Resource Measurement Facility (RMF) User's Guide.
4. Monitor and measure the use of virtual storage and the size of the dynamic storage areas (DSAs) in each of your CICS regions, using the following facilities:

   - The CICS storage manager statistics
   - The reports produced by the sample statistics program DFH0STAT
   - CICS formatted dumps for the loader domain and storage domain
5. Tune the use of storage across your z/OS system.

   If you are using RMF, for explanations of the RMF reports relating to CICS and to storage, and strategies for tuning, see z/OS Resource Measurement Facility (RMF) Report Analysis.
6. Tune the use of storage in each of your CICS regions, using the methods and suggestions in this section of the information.

   Concentrate on the areas of storage that seem to be the most different from your expectations.

CICS virtual storage

Each CICS region operates in its own z/OS address space. The storage available in a z/OS address space is divided into several different areas.

Figure 16 on page 70 shows an outline of the storage available in a z/OS address space. Although the theoretical upper limit for this virtual storage is extremely high, there are practical limits to real storage. For this reason, in z/OS, each address space is subject to **`REGION`** and **`MEMLIMIT`** parameters that limit the amount of storage the address space can use.

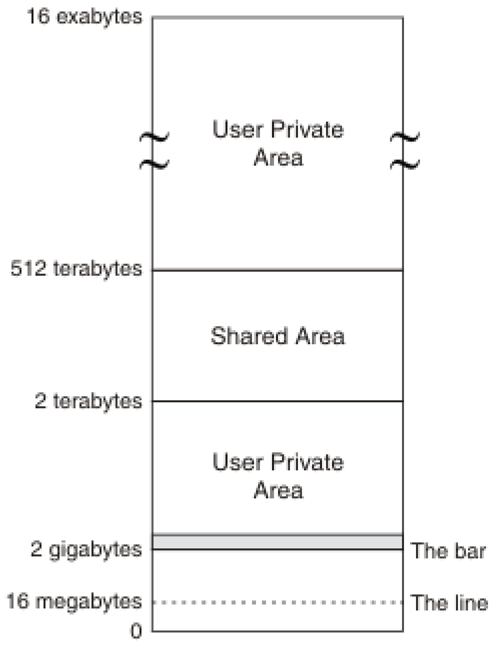


Figure 16. z/OS Address Space

CICS uses and manages virtual storage in three areas of its z/OS address space:

Storage below the line (0 MB to 16 MB)
: The storage in this area is 24-bit storage.

  Addresses below the 16 MB address are accessed by 24-bit addressing, and programs can use this storage when they run in AMODE 24 or higher. The 16 MB address is known as the line, so 24-bit storage is also called storage *below the line*.

Storage above the line (16 MB to 2 GB)
: The storage in this area is 31-bit storage.

  Addresses above the 16 MB address but below the 2 GB address are accessed by 31-bit addressing, and programs can use this storage when they run in AMODE 31 or higher. The 16 MB address is known as the line, so 31-bit storage is also called storage *above the line*.

  The area that separates the virtual storage area below the 2 GB address from the user private area is known as *the bar*. 24-bit and 31-bit storage are in storage below 2 GB and can together be referred to as storage *below the bar*.

Storage above the bar (4 GB to a theoretical 16 exabytes)
: The storage in this area is 64-bit storage.

  The area that separates the virtual storage area below the 2 GB address from the user private area is known as the bar, and 64-bit storage is also known as storage *above the bar*.

  The storage above the bar comprises a user private area between 4 GB and 2 terabytes, a shared area between 2 terabytes and 512 terabytes, and a user private area between the end of the shared area and 16 exabytes.

  Addresses above the bar are accessed by 64-bit addressing, and programs can use this storage when they run in AMODE 64.

In each private area of storage, virtual storage is used for the following purposes:

CICS dynamic storage areas
: The dynamic storage areas are used to supply the storage requirements of CICS, access methods, and applications running in CICS. See "CICS dynamic storage areas" on page 72.

MVS storage

MVS storage is available to the operating system to perform region-related services. See "64-bit MVS storage" on page 111 and "MVS storage below 2 GB" on page 111.

Note: This information and the following topics refer to other products installed with CICS, and is valid at the time of writing. For other products installed with CICS, always check the information for the versions of those products that you are using.

CICS region size

The amount of virtual storage for the address space in which CICS runs is specified by the z/OS **REGION** and **MEMLIMIT** parameters.

- The z/OS **REGION** parameter specifies your request for an amount of 24-bit and 31-bit storage, that is, storage below the bar. Up to 2047 MB of storage can be requested, but you must leave enough storage for the region-related services that require MVS storage below 2 GB.
- The z/OS **MEMLIMIT** parameter specifies the limit of 64-bit (above-the-bar) storage for the CICS region. A CICS region needs a **MEMLIMIT** value of at least 10 GB. The default value in z/OS for **MEMLIMIT** is 2 GB.

Reassess your settings for the **REGION** and **MEMLIMIT** parameters when you upgrade to a new release of CICS. Also reassess your settings when you install a new release of z/OS or a non-CICS subsystem. Changes in CICS can alter the requirements for 24-bit, 31-bit, and 64-bit storage for the CICS DSAs. Changes to other products can alter the requirements for MVS storage outside the CICS DSAs; for example, the requirements for 24-bit storage might reduce.

You cannot alter the **REGION** or **MEMLIMIT** values for the CICS region while CICS is running. You can specify new values on the next start of the CICS region. For instructions, see Setting address space storage limits for a CICS region.

You can specify the **REGION** parameter in different ways to request a specific amount of storage, or to request all the available 24-bit or 31-bit private storage. The resulting region size below the bar can be unpredictable. The z/OS message IEF374I reports the total amount of storage below the bar that z/OS assigns to a CICS region. The VIRT=nnnK portion of the message shows the 24-bit storage, and the EXT=nnnK portion shows the 31-bit storage. The CICS sample statistics program, DFH0STAT, produces a report that contains this information. You can also use RMF to monitor your use of storage in more detail.

If you plan to increase in the **REGION** value, remember the following points:

- Be aware of storage requirements in the high private area in 24-bit and 31-bit storage. Some of this storage is used by the z/OS Communications Server and other programs. An increase in the 24-bit or 31-bit storage allocated to CICS decreases the storage available for the items in the high private area. These items are the local system queue area (LSQA), scheduler work area (SWA), and subpools 229 and 230. A shortage in these subpools can cause S80A, S40D, and S822 abends. For more information about the high private area and LSQAs, see "High private area" on page 115.
- If you increase the **REGION** value, remember to increase your values for the CICS system initialization parameters **DSALIM** and **EDSALIM** as appropriate, otherwise CICS cannot use the additional storage.

For more information about the **REGION** and **MEMLIMIT** parameters and how they apply to z/OS address spaces, see the following z/OS information:

- For information about 24-bit and 31-bit storage (storage below the bar) in an address space, see Virtual Storage Overview in the z/OS MVS Initialization and Tuning Guide.
- For information about 64-bit (above-the-bar) storage in an address space, see Using the 64-bit Address Space in the z/OS MVS Programming: Extended Addressability Guide.
- REGION Parameter in the z/OS MVS JCL Reference.
- MEMLIMIT Parameter in z/OS MVS JCL Reference.

If the total amount of virtual storage required for your CICS regions increases, you might need to review the amount of space allocated for supervisor call (SVC) dumps that are requested by CICS, and the amount of auxiliary storage available. For information about SVC dump data set management, see z/OS

MVS Diagnosis Tools and Service Aids. For information about auxiliary storage management, see the z/OS MVS Initialization and Tuning Guide.

CICS dynamic storage areas

The dynamic storage areas (DSAs) supply CICS tasks with storage to run transactions and are essential for CICS operation. The DSAs in 24-bit storage are the CDSA, UDSA, SDSA, and RDSA. The DSAs in 31-bit storage are the ECDSA, EUDSA, ESDSA, ERDSA, and ETDSA. The DSAs in 64-bit storage are the GCDSA, GUDSA, and GSDSA.

The dynamic storage areas are made from virtual storage pages taken from MVS storage subpools. In the dynamic storage areas, CICS arranges the storage in CICS subpools. The subpools are dynamically acquired as needed, a page at a time, from within the dynamic storage area. The storage that individual subpools use is shown in the domain subpool statistics in the CICS storage manager statistics.

CICS manages DSA storage in extents. An individual DSA consists of one or more extents.

- 24-bit storage extents are usually allocated in multiples of 256 KB. However, when transaction isolation is in operation, the UDSA is allocated in 1 MB extents.
- 31-bit storage extents are allocated in multiples of 1 MB.
- 64-bit storage extents are allocated in multiples of 1 GB.

Only the owning DSA can use an allocated extent, and a given extent cannot be shared between more than one DSA simultaneously.

The storage for the DSAs can be allocated from CICS-key storage, user-key storage, or read-only key-0 protected storage. The type of storage that is allocated for each DSA can depend on the settings for the **STGPROT** and **RENTPGM** system initialization parameters for the CICS region.

- The storage for the CDSA, ECDSA, ETDSA and GCDSA is always allocated from CICS-key storage.
- The **STGPROT** system initialization parameter specifies whether you want storage protection to operate in the CICS region.

  When you specify **STGPROT=YES**, or allow the system initialization parameter to default, the storage for the CICS dynamic storage areas for user applications is allocated from user-key storage. These DSAs are the UDSA, SDSA, EUDSA, ESDSA, GUDSA, and GSDSA. If you specify **STGPROT=NO**, the storage for these DSAs is allocated from CICS-key storage.
- The **RENTPGM** parameter specifies whether CICS allocates the read-only DSAs from read-only key-0 protected storage.

  When you specify **RENTPGM=PROTECT**, the read-only DSAs are allocated from read-only key-0 protected storage. These DSAs are the RDSA and the ERDSA. If you specify **RENTPGM=NOPROTECT**, the storage for these DSAs is allocated from CICS-key storage.

A dynamic storage area that is too small results in increased program compression or, more seriously, short-on-storage (SOS) conditions. You can examine the pressure on virtual storage by using the CICS storage manager statistics, which report the number of times that CICS went short on storage.

DSAs in 24-bit storage: CDSA, UDSA, SDSA, and RDSA

The CICS dynamic storage areas (DSAs) below the line (below 16 MB) are in 24-bit storage. These storage areas do not have a collective name. The CICS system initialization parameter **DSALIM** specifies the limit on the total size of these dynamic storage areas.

The amount of storage specified by the **DSALIM** value is allocated as guaranteed storage at system initialization. Within this storage, CICS manages the following dynamic storage areas automatically, and you do not need to specify their individual sizes:

CDSA (CICS DSA)

The storage area for all non-reentrant CICS-key RMODE(24) programs, all CICS-key task-lifetime storage in 24-bit storage, and for CICS control blocks that reside in 24-bit storage. The CDSA is always allocated from CICS-key storage.

UDSA (User DSA)
: The storage area for all user-key task-lifetime storage in 24-bit storage. If you specify the system initialization parameter **STGPROT=YES** for the CICS region, which is the default, the UDSA is allocated from user-key storage. If you specify **STGPROT=NO**, the UDSA is allocated from CICS-key storage.

SDSA (Shared DSA)
: The storage area for any non-reentrant user-key RMODE(24) programs, and also for any storage obtained by programs issuing CICS GETMAIN commands for 24-bit storage with the SHARED option. If you specify the system initialization parameter **STGPROT=YES** for the CICS region, which is the default, the SDSA is allocated from user-key storage. If you specify **STGPROT=NO**, the SDSA is allocated from CICS-key storage.

RDSA (Read-only DSA)
: The storage area for all reentrant programs and tables in 24-bit storage. If you specify the system initialization parameter **RENTPGM=PROTECT** for the CICS region, which is the default, the RDSA is allocated from read-only key-0 protected storage. If you specify **RENTPGM=NOPROTECT**, the RDSA is allocated from CICS-key storage.

DSAs in 31-bit storage: ECDSA, EUDSA, ESDSA, ERDSA, and ETDSA
The group of CICS dynamic storage areas above the line (above 16 MB but below 2 GB) are collectively called the extended dynamic storage area (EDSA). This storage is 31-bit storage. The CICS system initialization parameter **EDSALIM** specifies the limit on the total size of these dynamic storage areas.

The amount of storage specified by the **EDSALIM** value is allocated as guaranteed storage at system initialization. Within this storage, CICS manages the following dynamic storage areas automatically, and you do not need to specify their individual sizes:

ECDSA (Extended CICS DSA)
: The storage area for all non-reentrant CICS-key RMODE(ANY) programs, all CICS-key task-lifetime storage in 31-bit storage, and CICS control blocks that reside in 31-bit storage. The ECDSA is always allocated from CICS-key storage.

EUDSA (Extended user DSA)
: The storage area for all user-key task-lifetime storage in 31-bit storage (above the line). If you specify the system initialization parameter **STGPROT=YES** for the CICS region, which is the default, the EUDSA is allocated from user-key storage. If you specify **STGPROT=NO**, the EUDSA is allocated from CICS-key storage.

ESDSA (Extended shared DSA)
: The storage area for any non-reentrant user-key RMODE(ANY) programs, and also for any storage obtained by programs issuing CICS GETMAIN commands for 31-bit storage with the SHARED option. If you specify the system initialization parameter **STGPROT=YES** for the CICS region, which is the default, the ESDSA is allocated from user-key storage. If you specify **STGPROT=NO**, the ESDSA is allocated from CICS-key storage.

ERDSA (Extended read-only DSA)
: The storage area for all reentrant programs and tables in 31-bit storage. If you specify the system initialization parameter **RENTPGM=PROTECT** for the CICS region, which is the default, the ERDSA is allocated from read-only key-0 protected storage. If you specify **RENTPGM=NOPROTECT**, the ERDSA is allocated from CICS-key storage.

ETDSA (Extended trusted DSA)
: The storage area for any security-related CICS control blocks that reside in 31-bit storage. The ETDSA is always allocated from CICS-key storage.

DSAs in 64-bit storage: GCDSA, GUDSA, and GSDSA
CICS dynamic storage areas above the bar are collectively called the above-the-bar dynamic storage area (GDSA). This storage is 64-bit storage. The z/OS **MEMLIMIT** parameter limits the 64-bit storage in the CICS region, including the GDSA.

The **MEMLIMIT** value that the z/OS operating system assigns to the CICS address space controls the upper limit for 64-bit storage in the CICS region. This 64-bit storage includes both the GDSA, and MVS storage in the CICS region outside the GDSA.

In contrast, the 24-bit storage specified by the **DSALIM** value and the 31-bit storage specified by the **EDSALIM** value relate only to the CICS DSAs.

The GDSA does not preallocate an amount of guaranteed storage. The GDSA contains the following dynamic storage areas:

GCDSA (above-the-bar CICS DSA)
The storage area for all CICS-key task-lifetime storage in 64-bit (above-the-bar) storage, and for CICS facilities that use 64-bit storage. See "CICS facilities that use 64-bit storage" on page 82. The GCDSA is always allocated from CICS-key storage.

GUDSA (above-the-bar user DSA)
The storage area for all user-key task-lifetime storage in 64-bit (above-the-bar) storage. If you specify the system initialization parameter STGPROT=YES for the CICS region, which is the default, the GUDSA is allocated from user-key storage. If you specify STGPROT=NO, the GUDSA is allocated from CICS-key storage.

GSDSA (above-the-bar shared DSA)
The storage area for any storage that programs obtain by issuing a **EXEC CICS GETMAIN64** command to obtain 64-bit storage with the SHARED option. If you specify the system initialization parameter STGPROT=YES for the CICS region, which is the default, the GSDSA is allocated from user-key storage. If you specify STGPROT=NO, the GSDSA is allocated from CICS-key storage.

Storage protection
CICS uses the storage protection facilities that are available in the operating system to prevent CICS code and control blocks from being overwritten accidentally by your user application programs. To do this, separate dynamic storage areas (DSAs), with separate storage keys, are allocated for your user application programs, and for CICS code and control blocks. Access to a storage area is not permitted unless the access key matches the key for that storage area.

The storage allocated for most CICS code and control blocks is known as CICS-key storage, and the storage allocated for your user application programs is known as user-key storage.

In addition to CICS-key and user-key storage, CICS also uses key-0 storage for separate dynamic storage areas called the read-only DSAs (RDSA and ERDSA). The ERDSA is used for eligible re-entrant CICS and user application programs that are link-edited with the RENT and RMODE(ANY) attributes. The RDSA is used for eligible reentrant CICS and user application programs that are link-edited with the RENT and RMODE(24) attributes. The allocation of key-0 storage for the read-only DSAs is from the same storage limit as the other DSAs, as specified by the **DSALIM** and **EDSALIM** system initialization parameters.

Use of the storage protection facilities is optional. You can enable the facilities by using options on the system initialization parameters that are related to storage protection. Between them, you can use these parameters to define or control the following items:

- The storage key for the common work area (**CWAKEY**)
- The storage key for the terminal control table user areas (**TCTUAKEY**)
- A storage protection global option (**STGPROT**)
- A read-only program storage key option (**RENTPGM**)
- A transaction isolation option (**TRANISO**)

Storage protection, transaction isolation, and command protection protect storage from user application code. They add no benefit to a region where no user code is executed; that is, a pure terminal-owning region (TOR) or a pure file-owning region (FOR) (where no distributed program link (DPL) requests are function-shipped).

The common work area (CWA)
The common work area (CWA) is an area of storage in your CICS region that any user application can access. You determine the size of this work area by using the **WRKAREA** system initialization parameter; you can specify sizes up to 3584 bytes.

If you omit the **WRKAREA** parameter, CICS allocates a 512-byte CWA by default. You specify the storage key for the CWA on the **CWAKEY** parameter.

This work area is available to all transactions in a CICS region, so you must ensure that the storage key is appropriate to the use of the CWA by all transactions. If any of the transactions run in user key and require write access, you must specify user-key storage for the CWA, otherwise such transactions fail with a storage protection exception (an ASRA abend). CICS obtains user-key storage for the CWA by default, and you must review the use of this storage by all programs before you decide to change it to CICS-key storage.

It is possible that you might want to protect the CWA from being overwritten by applications that should not have write access. In this case, provided all the applications that legitimately require write access to the CWA run in CICS key, you can specify CICS-key storage for the CWA.

The terminal control table user areas
A terminal control table user area (TCTUA) is an optional storage area associated with a terminal control table terminal entry (TCTTE). TCTUAs are available for application program use. You specify the storage key for TCTUAs globally for a CICS region by using the **TCTUAKEY** system initialization parameter.

By default, CICS obtains user-key storage for all TCTUAs. Review the use of TCTUAs in your CICS regions, and only specify CICS key for TCTUAs when you are sure that this is justified. If you specify CICS-key storage for TCTUAs, no user-key applications can write to any TCT user areas.

For SNA LUs, you specify that you want a TCTUA by means of the USERAREALEN attribute on the TYPETERM resource definition. The USERAREALEN attribute determines the TCTUA sizes for all terminals that reference the TYPETERM resource definition.

For sequential terminals, definitions are added to the terminal control table (TCT), and sizes are defined by means of the **TCTUAL** parameter on the DFHTCT TYPE=TERMINAL and TYPE=LINE entries. For information about the TCTUAL parameter, see Terminal control table (TCT).

The storage protection global option
You can control whether your CICS region uses storage protection by specifying the **STGPROT** system initialization parameter. By default, CICS uses storage protection.

When you specify **STGPROT=YES**, or allow the system initialization parameter to default, the storage for the CICS dynamic storage areas for user applications is allocated from user-key storage. These DSAs are the UDSA, SDSA, EUDSA, ESDSA, GUDSA, and GSDSA. If you specify **STGPROT=NO**, the storage for these DSAs is allocated from CICS-key storage.

Running a CICS region without storage protection (**STGPROT=NO**) is suitable for pure terminal-owning regions (TORs) that do not execute user transactions.

The transaction isolation global option
CICS transaction isolation builds on CICS storage protection, enabling user transactions to be protected from one another. You can specify transaction isolation globally for a CICS region using the **TRANISO** system initialization parameter.

In addition to being able to specify the storage and execution key individually for each user transaction, you can specify that CICS is to isolate the user-key task-lifetime storage of a transaction to provide transaction-to-transaction protection. You do this by using the `ISOLATE` option of the TRANSACTION resource definition.

Transaction isolation does not apply to 64-bit storage.

The read-only storage override option
CICS obtains storage for the read-only DSAs (RDSA and ERDSA) from MVS read-only storage. You can override the selection of read-only storage for the RDSA and ERDSA by specifying NOPROTECT on the **RENTPGM** system initialization parameter.

The CICS loader automatically loads eligible modules into the RDSA and ERDSA; that is, if they are link-edited with the RENT attribute, and for the ERDSA with RMODE(ANY). You can specify **RENTPGM=NOPROTECT** if you do not want such modules to be loaded into read-only storage, perhaps because you are using a development aid package that sets break points in your application programs. When you specify **RENTPGM=NOPROTECT**, CICS still allocates separate read-only DSAs, but obtains CICS-key storage for the RDSA and ERDSA instead of read-only storage.

The **RENTPGM=NOPROTECT** override is only appropriate for development regions. In production CICS regions, **RENTPGM=PROTECT** provides the right level of protection for modules in the RDSA and ERDSA.

Setting the limits for CICS storage

Use the z/OS **REGION** and **MEMLIMIT** parameters to set limits for the storage for the CICS region. Use the CICS **DSALIM** and **EDSALIM** system initialization parameters to limit the total storage for the CICS dynamic storage areas (DSAs) in 24-bit and 31-bit storage.

About this task

The z/OS **REGION** and **MEMLIMIT** parameters specify the amount of virtual storage for the address space in which the CICS region runs.

- Use the z/OS **REGION** parameter to specify the amounts of 24-bit (below-the-line) and 31-bit (above-the-line) storage that are requested for the CICS region.
- Use the z/OS **MEMLIMIT** parameter to set the limit for the amount of 64-bit (above-the-bar) storage for the CICS region.

  The 64-bit storage specified by the **MEMLIMIT** value includes the CICS dynamic storage areas above the bar (the GDSA) and MVS storage in the CICS region outside the GDSA.

For more information about these parameters, see "CICS region size" on page 71.

The storage specified by the **REGION** value includes the storage specified by the CICS system initialization parameters **DSALIM** and **EDSALIM**. These parameters determine the overall limits within which CICS can allocate storage for the CICS DSAs.

- Use the **DSALIM** parameter to set overall limits for the CICS DSAs in 24-bit (below-the-line) storage.
- Use the **EDSALIM** parameter to set overall limits for the extended dynamic storage area (EDSA), that is, the CICS DSAs in 31-bit (above-the-line) storage.

CICS allocates individual dynamic storage areas automatically, and you do not need to specify their sizes. CICS varies the size of the individual dynamic storage areas as the need arises.

- CICS allocates DSAs in 24-bit storage within the limits set by **DSALIM**.
- CICS allocates DSAs in 31-bit storage within the limits set by **EDSALIM**.
- CICS allocates DSAs in 64-bit storage within the limits set by **MEMLIMIT**.

Procedure

- To estimate and change the setting of the z/OS parameter **REGION**, see "Estimating and setting REGION" on page 76.
- To estimate and change the setting of the z/OS parameter **MEMLIMIT**, and to check its value in a running CICS system, see "Estimating, checking, and setting MEMLIMIT" on page 81.
- To estimate and change the setting of the CICS system initialization parameter **DSALIM**, see "Estimating, checking, and setting DSALIM" on page 78.
- To estimate and change the setting of the CICS system initialization parameter **EDSALIM**, see "Estimating, checking, and setting EDSALIM" on page 79.

Estimating and setting REGION

The z/OS **REGION** parameter limits the amount of 24-bit and 31-bit storage (storage below the bar) that the CICS address space can use. This value includes all the storage below the bar in the private area, except for a 16 KB system region in 24-bit storage, and the items in the high private area such as the LSQA.

About this task

For an explanation of the storage areas in the z/OS address space below 2 GB, see The Virtual Storage Address Space in the z/OS MVS Initialization and Tuning Guide.

You can request up to 2047 MB of storage below the bar for the CICS region, but you must ensure that you leave enough storage below the bar for MVS to use in the high private area. The items in the high private area are the local shared queue area (LSQA), scheduler work area (SWA), and subpools 229 and 230. These items exist in both 24-bit (below-the-line) storage and 31-bit (above-the-line) storage. The LSQA in 31-bit storage is called the extended LSQA. Some of this storage is used for control blocks, and some is used by z/OS Communications Server and other programs.

Within the value that you specify for **REGION**, the following types of storage are included:

- The CICS DSAs in 24-bit storage. The storage for these DSAs is limited by the **DSALIM** system initialization parameter.
- The CICS DSAs in 31-bit storage. The storage for these DSAs is limited by the **EDSALIM** system initialization parameter.
- Storage used by the CICS kernel.
- MVS storage obtained by MVS GETMAIN requests.
- CICS dispatcher
- CICS storage manager
- CICS lock manager

You cannot alter the **REGION** value for the CICS region while CICS is running. You can specify a new value on the next start of the CICS region.

Procedure

1. To determine the maximum value for the **REGION** parameter:
   a) Use RMF or another storage monitor to determine the size of your private area.
   b) Apply the following formula:

   ```
   Maximum possible REGION =
               Size of private area
               - Size of system region (16K)
               - (LSQA + SWA + subpools 229 and 230)
   ```

   For more information about the high private area and LSQAs, and estimates for the sizes of the items in the high private area, see "High private area" on page 115.

   c) For safety, do not use more than 80% or 90% of this maximum value for the **REGION** parameter.

   It is useful to maintain some free storage between the top of the user region and the bottom of the high private area.

   If the system is static or does not change much, use up to 90% of this number. If the system is dynamic, or changes frequently, 80% would be better.
2. To estimate the value that you require for the **REGION** parameter to meet your storage needs, add up your estimates for the following areas of storage:
   - The area of 24-bit storage for CICS DSAs below the line, specified by the **DSALIM** system initialization parameter. See "Estimating, checking, and setting DSALIM" on page 78.
   - The area of 31-bit storage for CICS DSAs above the line (the EDSA), specified by the **EDSALIM** system initialization parameter. See "Estimating, checking, and setting EDSALIM" on page 79.
   - The small amount of storage used by the CICS kernel outside the CICS DSAs. See "CICS kernel storage" on page 110.
   - MVS storage obtained by MVS GETMAIN requests outside the CICS DSAs. See "MVS storage below 2 GB" on page 111.
3. For instructions to specify the **REGION** parameter, and information about the amount of storage that z/OS allocates in response to your request, see REGION Parameter in the z/OS MVS JCL Reference.

   You cannot alter the **REGION** value for a running CICS region.

   You can set **REGION** in the following ways:

- You can specify the **REGION** parameter in the JOB statement in the CICS JCL. In this situation, each step of the job runs in the requested amount of space.
- You can specify the **REGION** parameter in the EXEC statement (program execution line) for CICS. In this situation, each step runs in its own amount of space. Use the EXEC statement instead of the JOB statement if different steps need greatly different amounts of space. For example, you could use the EXEC statement if you are using extra job steps to print auxiliary trace data sets after CICS has shut down (as in the DFHIVPOL installation verification procedure).
- The z/OS installation exit IEFUSI can limit the **REGION** value that you specify. For information about IEFUSI, see IEFUSI - Step Initiation Exit in z/OS MVS Installation Exits.

Estimating, checking, and setting DSALIM
The **DSALIM** system initialization parameter specifies the upper limit of the total amount of storage within which CICS can allocate the individual dynamic storage areas (DSAs) that reside in 24-bit storage (below 16 MB, also known as below the line). If your installation is constrained for 24-bit storage, set a value for the **DSALIM** parameter equivalent to the sum of the CDSA and UDSA. If you have sufficient virtual storage to allow a greater value for **DSALIM**, you can use the formulas given here.

About this task

Accurate sizing of the **DSALIM** value is not critical. It is better to specify a **DSALIM** value that is slightly greater than your expected requirements rather than slightly smaller. You can tune the **DSALIM** parameter to a smaller value after you obtain data from your running system.

Make sure that you understand the requirements for MVS storage in 24-bit storage outside the CICS DSAs, to avoid other subsystem problems. For more information about the operating system components that use MVS storage, see "MVS storage below 2 GB" on page 111.

The minimum **DSALIM** value is 2 MB and the default value is 5 MB. The maximum **DSALIM** value is 16 MB. The extent size for the CDSA, RDSA, and SDSA is in 256 KB increments. If transaction isolation is active, the extent size for the UDSA is 1 MB and each UDSA extent must be aligned on a megabyte boundary. If transaction isolation is not active, the allocation is in 256 KB extents.

CICS allocates 24-bit kernel stack storage when it is required. Tasks obtain 4 KB extension stack segments whenever they require 24-bit stack storage. CICS preallocates a reserve pool of 24-bit extension stack segments that tasks can use if no other 24-bit stack storage is available. For more information about kernel storage, see "CICS kernel storage" on page 110.

Procedure

- To check the **DSALIM** value that currently applies to a running CICS region, use one of the following methods:
  - CICS Explorer: **Global Dynamic Storage Areas** view
  - CICSPlex SM: **Dynamic storage areas - CICSDSA** view
  - CEMT: **CEMT INQUIRE DSAS** or **CEMT INQUIRE SYSTEM**
  - CICS SPI: **INQUIRE SYSTEM**
- To estimate a suitable **DSALIM** value, use the following steps:
  a) If you have sufficient virtual storage to specify a generous **DSALIM** value, use the following formula to estimate a value:

     CDSA + UDSA + SDSA + RDSA

     Round up the value of each component in your calculation to a 256 KB boundary.
  b) If your current installation **DSALIM** value is larger than necessary, use the following formula to estimate a **DSALIM** value:

     Peak CDSA Used + Peak UDSA Used + Peak SDSA Used + Peak RDSA Used

     Round up the value of each component in your calculation to a 256 KB boundary.

- To change the **DSALIM** value for the CICS region, use one of the following methods while CICS is running:
  - CICS Explorer: **Global Dynamic Storage Areas** view
  - CICSPlex SM: **Dynamic storage areas - CICSDSA** view
  - CEMT: **CEMT SET DSAS** or **CEMT SET SYSTEM**
  - CICS SPI: **SET SYSTEM**

Results

If there are no extents free in the CICS DSAs in 24-bit storage, CICS cannot implement a reduction of **DSALIM**. The storage manager applies MVS FREEMAIN requests to extents as they become available until the new **DSALIM** value is reached.

A short-on-storage condition can occur when you reduce **DSALIM**.

Estimating, checking, and setting EDSALIM
The **EDSALIM** system initialization parameter specifies the upper limit of the total amount of storage within which CICS can allocate the individual extended dynamic storage areas (EDSAs) that reside in 31-bit (above-the-line) storage. Set the value for the **EDSALIM** parameter as large as you can after consideration of other areas, especially MVS storage.

About this task

The maximum value that you can specify for **EDSALIM** is limited by the following factors:

- The size that you specified for the CICS region on the z/OS **REGION** parameter in the CICS job or procedure. The value of **EDSALIM** must be less that the value of **REGION**.
- The amount of MVS storage, outside the CICS DSAs, that you require to satisfy MVS GETMAIN requests for 31-bit storage. For more information about the operating system components that use MVS storage, see "MVS storage below 2 GB" on page 111.

Accurate sizing of the **EDSALIM** value is not critical. A good approach is as follows:

- Initially specify an **EDSALIM** value that is slightly greater than your expected requirements. The default setting, 800 MB, enables the CICS region to run a reasonable workload.
- Monitor the use of each CICS DSA in the EDSA while your system is running near peak loads.
- Tune your **EDSALIM** value in the running CICS system.

Try not to specify the largest possible **EDSALIM** value (for example, the maximum allowable region size). If you use the maximum possible limit, you might not receive any warnings about a shortage of virtual storage until the problem becomes difficult to resolve.

You can obtain information about your current EDSA use by looking at the CICS storage manager statistics. See the information about DSA sizes in the storage manager statistics, dynamic storage areas, and task subpools. Automatic DSA sizing removes the need for accurate storage estimates for individual DSAs, with CICS dynamically changing the size of DSAs as demand requires.

The minimum **EDSALIM** value is 48 MB, which is the minimum required to start a CICS region. The default **EDSALIM** value is 800 MB. The maximum **EDSALIM** size is 2047 MB, which is 2 GB minus 1 MB. The extent size for the ECDSA, EUDSA, ESDSA, ERDSA, and ETDSA is 1 MB.

Kernel stack storage is also allocated from the EDSA. For more information about kernel storage see "CICS kernel storage" on page 110.

Remember that for CICS regions that run with transaction isolation active (set by using the **TRANISO** system initialization parameter), the transaction isolation facility increases the allocation of some virtual storage above 16 MB.

If transaction isolation is active, CICS allocates user-key task-lifetime storage above 16 MB in multiples of 1 MB (the minimum unit of storage allocation for the EUDSA when transaction isolation is active is 1 MB).

However, MVS paging activity affects only the storage that is used (referenced), and unused parts of the 1 MB allocation are not paged.

For a CICS region that runs without transaction isolation, CICS allocates user-key task-lifetime storage above 16 MB in multiples of 64 KB.

The subspace group facility uses more real storage, because MVS creates a page and segment table from real storage for each subspace. The CICS requirement for real storage varies according to the transaction load at any one time. As a guideline, each task in the system requires 9 KB of real storage, and this should be multiplied by the number of concurrent tasks that can be in the system at any one time (governed by the **MXT** system initialization parameter).

Procedure

- To check the **EDSALIM** value that currently applies to a running CICS region, use one of the following methods:
  - CICS Explorer: **Global Dynamic Storage Areas** view
  - CICSPlex SM: **Dynamic storage areas - CICSDSA** view
  - CEMT: **CEMT INQUIRE DSAS** or **CEMT INQUIRE SYSTEM**
  - CICS SPI: **INQUIRE SYSTEM**
- To estimate a suitable **EDSALIM** value, use the following steps:

  a) Check the **MXT** value for your CICS region.

     The **MXT** system initialization parameter does not include CICS system tasks, and it might also be set to a value larger than necessary. The safest estimate for calculating an **EDSALIM** value is to assume **MXT** as the number of concurrent active tasks.

  b) Check the setting of the **TRANISO** system initialization parameter for your CICS region.

     For the EUDSA, if the **TRANISO** parameter is set to NO for the CICS region, allow 64 KB per concurrent active task. If the **TRANISO** parameter is set to YES, allow 1 MB per concurrent active task.

     If your applications use more than 64 KB per task with the **TRANISO** parameter set to NO, or more than 1 MB per task with the **TRANISO** parameter set to YES, adjust the formulas accordingly. If you adjust the formulas, use multiples of 64 KB or 1 MB.

  c) If you have sufficient virtual storage to specify a generous **EDSALIM** value, use one of the following formulas to estimate a value.

     Round up the value of each component in your calculation to a 1 MB boundary, the size of an extent, to allow for fragmentation and partially used extents.

     For TRANISO=NO:
     ECDSA + EUDSA + ESDSA + ERDSA + ETDSA + (64 K x MXT)

     For TRANISO=YES:
     ECDSA + EUDSA + ESDSA + ERDSA + ETDSA + (1 MB x MXT)

  d) If your current installation **EDSALIM** and **MXT** values are larger than necessary, use one of the following formulas to estimate an **EDSALIM** value.

     Round up the value of each component in your calculation to a 1 MB boundary, the size of an extent, to allow for fragmentation and partially used extents.

     For TRANISO=NO:
     Peak ECDSA Used + Peak EUDSA Used + Peak ESDSA Used + Peak ERDSA Used + Peak ETDSA Used - EUDSA Peak Page Storage in Task Subpools + (64 K x Peak number of tasks)

     For TRANISO=YES:
     Peak ECDSA Used + Peak EUDSA Used + Peak ESDSA Used + Peak ERDSA Used + Peak ETDSA Used - EUDSA Peak Page Storage in Task Subpools + (1 MB x Peak number of tasks)
- To change the **EDSALIM** value for the CICS region, use one of the following methods while CICS is running:

- CICS Explorer: **Global Dynamic Storage Areas** view
- CICSPlex SM: **Dynamic storage areas - CICSDSA** view
- CEMT: **`CEMT SET DSAS`** or **`CEMT SET SYSTEM`**
- CICS SPI: **`SET SYSTEM`**

Results

If you under-specify **`EDSALIM`**, your system can go short on storage and you might not be able to issue CICS commands to increase the limit. In this situation, use the CICS Explorer or CICSPlex SM to increase the **`EDSALIM`** value.

If there are no extents free in the CICS DSAs, CICS cannot implement a reduction of **`EDSALIM`**. The storage manager applies MVS FREEMAIN requests to extents as they become available until the new **`EDSALIM`** value is reached.

Estimating, checking, and setting MEMLIMIT
The z/OS **`MEMLIMIT`** parameter limits the amount of 64-bit (above-the-bar) storage that the CICS address space can use. This storage includes the CICS dynamic storage areas above the bar (collectively called the GDSA) and MVS storage in the CICS region outside the GDSA.

About this task

CICS requires a **`MEMLIMIT`** value of 10 GB; any additional use by applications or JVMs should be allowed for with a larger value of **`MEMLIMIT`**. If you attempt to start a CICS region with a **`MEMLIMIT`** value that is less than 10 GB, message `DFHSM0602` is issued, a system dump with the dump code KERNDUMP is produced, and CICS terminates.

You cannot alter the **`MEMLIMIT`** value for the CICS region while CICS is running. You can specify a new **`MEMLIMIT`** value on the next start of the CICS region.

CICS uses a different calculation to z/OS when checking **`MEMLIMIT`** to see whether a new GDSA extent can be allocated. CICS calculates the current usage as the amount that is allocated to the 64-bit Memory Objects, which includes both Usable and Hidden storage.

Procedure

- When you are setting **`MEMLIMIT`**, you should aim to avoid CICS SOS Above the Bar. **`MEMLIMIT`** can be reduced after the actual peak usage is clear.
- To check the **`MEMLIMIT`** value that currently applies to a running CICS region, use one of the following methods:
  - CICS Explorer: **Regions** view or **Global Dynamic Storage Areas** view
  - CICSPlex SM: **Dynamic storage areas - CICSDSA** view or **CICS regions - CICSRGN** view
  - CEMT: **`CEMT INQUIRE DSAS`** or **`CEMT INQUIRE SYSTEM`**
  - CICS SPI: **`INQUIRE SYSTEM`**
  - The `Storage Above 2 GB` report produced by DFH0STAT provides information about 64-bit usage from various CICS Statistics fields. `SMSMEMLIMIT - (SMSLVABYTES/1048576)` rounded down to units of GB shows by how much the GDSA can be expanded. This is reported as `"MEMLIMIT minus allocated to Private Memory Objects"`.
- To estimate a suitable **`MEMLIMIT`** value for a CICS region, add up the storage requirements for those facilities that use 64-bit storage that you use in your CICS region.

  For a list of CICS facilities that use 64-bit storage in the CICS region and the relevant storage subpools, see CICS facilities that can use 64-bit storage in Improving performance.

  CICS must allocate a GDSA extent that is a multiple of 1 GB before it can suballocate storage for CICS 64-bit subpools within it. The CICS Internal Trace Table is always allocated with a size of 1 GB.

The CICS storage manager statistics show the storage used in each CICS subpool in the GDSA in a running CICS region. For information about the subpools in the GDSA, see CICS subpools in the GCDSA and CICS subpools in the GSDSA.

- To change a **MEMLIMIT** value in z/OS, or determine the value that applies to a CICS region that you are setting up, see Limiting the use of memory objects in the z/OS MVS Programming: Extended Addressability Guide.

  MEMLIMIT can be set in one of the following ways:

  – A **MEMLIMIT** value can be specified in the JOB statement in the CICS JCL, or in the EXEC statement (program execution line) for CICS.

  – If there is no **MEMLIMIT** value specific to the CICS region, the **MEMLIMIT** value that is set in the z/OS SMFPRM*xx* PARMLIB member, or the system default, applies.

  – The z/OS installation exit IEFUSI can override any other **MEMLIMIT** values.

  The following example shows a **MEMLIMIT** value set in the program execution line:

```
//CICS EXEC PGM=DFHSIP,PARM='SI',REGION=0M,MEMLIMIT=10G
```

CICS facilities that use 64-bit storage
The CICS facilities that use 64-bit storage and the amount of storage that each facility requires are listed. You can use this information when you estimate a suitable **MEMLIMIT** value for your CICS region.

The following table lists CICS facilities that use 64-bit storage and the relevant CICS storage subpools, and indicates the amount of storage that each facility requires. You can identify the facilities that you use in your CICS region that require 64-bit storage. You can then examine the storage requirement for those facilities and use this information to estimate a suitable value for the z/OS **MEMLIMIT** parameter.

Table 3. CICS facilities that use 64-bit storage

CICS facility	**Storage use**	**CICS subpool for storage**	**Notes®**
Application context data control blocks	220 bytes for each program that is defined as an application entry point.	PGPPTE64	
Association data control blocks	Approximately 1 KB for each active task.	MN_ADCS	
CICS management client interface (CMCI)	For the details to estimate storage requirements, see Estimating storage requirements for CMCI.	WU_64	The storage is used for retained results and metadata.
CICSPlex SM API result sets	The size of the result set depends on the application.	CPSM_64	For information about result sets, see Working with result sets.
Console queue processing	1 MB per subpool	CQCQ_TR CQCQ_XT	
Containers and channels	Limited to 5% of the **MEMLIMIT** value per transaction.	PGCSDB	The storage remains in use until the channel goes out of scope, or an application deletes the container or the channel.
Event processing		EP_64	The storage is used for control blocks for items in event capture queues.

Table 3. CICS facilities that use 64-bit storage (continued)

CICS facility	Storage use	CICS subpool for storage	Notes®
Internal trace table	Minimum 16 KB Maximum 1 GB Default 12288 KB (12 MB) Controlled by the **TRTABSZ** system initialization parameter.	MVS storage outside the CICS DSAs	CICS allocates a Memory Object of the maximum size but defines the unused area as Hidden storage.
JVM servers	Apart from the values in -Xmx and -Xcmsx, the amount of storage is variable.	MVS storage outside the CICS DSAs	
Loader control blocks	Variable	LD_APES LD_CPES LD_CSECT	The storage that is used depends on the number of program load operations in the CICS region.
Loader control blocks for installed application elements (IAE)	368 bytes for each IAE	LD_IAES	Each installed application element represents a particular version of an application that is deployed on a platform.
Main temporary storage	Minimum 1 MB Maximum 32 GB Default 64 MB Controlled by the **TSMAINLIMIT** system initialization parameter.	TSDTN TSMAIN TSMN0064 TSMN0128 TSMN0192 TSMN0256 TSMN0320 TSMN0384 TSMN0448 TSMN0512 TSMN0576 TSMN0640 TSMN0704 TSMN0768 TSMN0832 TSMN0896 TSMN0960 TSMN1024 TSQUEUE TSTSI	**TSMAINLIMIT** specifies the maximum storage that can be used, and is limited to 25% of the **MEMLIMIT** value. The CICS statistics show actual use.

Table 3. CICS facilities that use 64-bit storage (continued)

CICS facility	Storage use	CICS subpool for storage	Notes®
Managed Platform	1072 bytes for each user task	DFHMPMDR DFHMPPMB DFHMPTAS	Storage in the DFHMPMDR and DFHMPPMB subpools is used for control blocks for installed policies. It remains in use until the bundle that defines the policy is disabled. Storage in the DFHMPTAS subpool contains the policy counters for a task. It is allocated for every user task if **any** polices are installed in the CICS region. The storage is freed at task end.
Message tables	Minimum 3 MB for the message modules in English. Add 1 MB if the user message table is loaded. Add 3 MB for each additional language that is loaded.	MVS storage outside the CICS DSAs	Message modules in English are always loaded.
Sockets domain	Stores the copy of the HTTPS data being sent. The size of the storage depends on the size of the HTTPS message being sent.	SOGNRL64	This storage subpool is used if the sockets domain needs to copy the HTTPS message before it is sent.
Static data item control blocks	Variable	MPSTAT	Used if a policy rule with an event action defines static data capture items.
Storage allocation control blocks	Variable	MVS storage outside the CICS DSAs	Storage used depends on the amount of storage allocation activity in your system; for example, more storage is required for subpools that keep an element chain and that have many small records.
Transaction dump trace table	Minimum 16 KB Maximum 1 GB Default 1024 KB Controlled by the **TRTRANSZ** system initialization parameter.	MVS storage outside the CICS DSAs	CICS obtains this storage only when a transaction dump is produced.

Table 3. CICS facilities that use 64-bit storage (continued)

CICS facility	Storage use	CICS subpool for storage	Notes®
Web domain	Stores the HTTP body. The size of the storage used depends on the size of the HTTP body being processed.	WB64GNRL	Where possible, the web domain holds the HTTP body in above-the-bar storage by using this subpool.
z/OS XML System Services (XMLSS) parser I/O buffers		ML64GNRL	A number of facilities, including CICS web services, use the parser for XML parsing.

DSA size limits
It is not advisable to set the size of individual dynamic storage areas (DSAs), and usually it is not necessary. However, it is possible to set the size of some DSAs by using the **CDSASZE**, **UDSASZE**, **RDSASZE**, **ECDSASZE**, **EUDSASZE**, **ESDSASZE**, and **ERDSASZE** system initialization parameters.

For example, **CDSASZE** sets the size of the CICS dynamic storage area (CDSA), and **ECDSASZE** specifies the size of the extended CICS dynamic storage area (ECDSA). The default value for these parameters is 0, indicating that the size of the DSA can change dynamically. If you specify a nonzero value, the DSA size is fixed.

If you specify DSA size values that in combination do not allow sufficient space for the remaining DSAs, CICS fails to initialize.

- The limit on the storage available for the DSAs in 24-bit storage (below 16 MB) is specified by the **DSALIM** system initialization parameter. You must allow at least 256K for each DSA in 24-bit storage for which you have not set a size.
- The limit on the storage available for the DSAs in 31-bit storage (above 16 MB but below 2 GB) is specified by the **EDSALIM** system initialization parameter. You must allow at least 1 MB for each DSA in 31-bit storage for which you have not set a size.

You cannot set the size of individual DSAs in 64-bit storage; that is, in the above-the-bar DSA (GDSA).

Coding conventions for DSA limits
You can specify the size of the DSA limits as a number of bytes, a number of kilobytes, or a number of megabytes.

Use the letter K as a suffix to indicate that the value represents a whole number of kilobytes. Use the letter M as a suffix to indicate that the value represents a whole number of megabytes. For example, 2 MB can be coded as either 2048K or 2M. (1 KB = 1024 bytes; 1 MB = 1024 KB = 1048576 bytes.)

If the value you specify is not a multiple of 256 KB for **DSALIM**, or 1 MB for **EDSALIM**, CICS rounds up the value to the next multiple.

You cannot specify fractions of megabytes; you must code sizes in bytes or kilobytes. Some examples are shown in Table 4 on page 85.

Table 4. Examples of DSA limit values in bytes, kilobytes, and megabytes

Coded as:					
bytes	2097152	3145788	3670016	4194304	4718592
kilobytes	2048K	3072K	3584K	4096K	4608K
megabytes	2M	3M	-	4M	-

Short-on-storage conditions in dynamic storage areas

If the limit for a dynamic storage area (DSA) is too small, the CICS region periodically enters a short-on-storage condition. Where possible, CICS curtails system activity until it can recover enough storage to resume normal operations. Use CICS messages and statistics to monitor when a short-on-storage (SOS) condition is entered, and when it is relieved.

CICS attempts to resolve pressures on storage before entering a short-on-storage condition. When CICS starts to become short on space in a DSA, the situation is known as a storage stress condition. Where possible, CICS takes actions such as deleting programs that are not being used (program compression), deleting cached copies of any document templates, and searching for free extents in other DSAs. If these actions fail to resolve the storage stress condition, CICS declares an SOS condition for the DSA.

During an SOS condition, CICS takes steps to limit work, so that there is enough storage to process work that is already in progress. CICS prevents acquisition of new input message areas, and defers all ATTACH requests from CICS system modules. Limiting work degrades the performance of the CICS region. In extreme circumstances, an SOS condition might also lead to storage deadlock abends.

When an SOS condition is entered, one of the following messages is issued:

- DFHSM0131 for 24-bit storage
- DFHSM0133 for 31-bit storage
- DFHSM0606 for 64-bit storage

SOS conditions are also recorded in the CICS statistics for the dynamic storage area ("Times went short on storage"). You can use the CICS commands **CEMT INQUIRE SYSTEM**, **EXEC CICS INQUIRE SYSTEM**, and **CEMT INQUIRE DSAS** to inquire about SOS conditions.

When you observe an SOS condition, first determine whether the affected storage is 24-bit, 31-bit, or 64-bit.

- For 24-bit storage, check whether the limit for the DSAs in 24-bit storage is as high as possible. If required, you can change the **DSALIM** parameter while CICS is running.
- For 31-bit storage, check whether the limit for the extended dynamic storage area (EDSA) is as high as possible. If required, you can change the **EDSALIM** parameter while CICS is running.
- For 64-bit storage, check whether there is sufficient 64-bit storage for the CICS region. If required, you can change the z/OS **MEMLIMIT** value, but only on the next start of the CICS region.

For instructions to change these limits, see "Setting the limits for CICS storage" on page 76.

CICS reserves areas of contiguous virtual storage, called storage cushions, in each DSA. A storage cushion is used only when there is not enough free storage in the DSA to satisfy an unconditional GETMAIN request. In a storage stress condition, the storage cushion might avert a storage deadlock. The CICS storage manager statistics for the dynamic storage areas show the number of times that CICS needed to use storage from the cushion. A request might be larger than all the remaining storage in the DSA, so that even the storage in the cushion is insufficient. When a request is suspended for this reason, the suspension is also shown in the CICS storage manager statistics for the dynamic storage areas.

Short-on-storage conditions for 24-bit and 31-bit storage

When an individual DSA in 24-bit or 31-bit storage, for example the CDSA, requires additional storage, the CICS storage manager allocates another extent to that DSA. Additional extents can be acquired as necessary until the **DSALIM** or **EDSALIM** limit is reached, as appropriate. When all the possible extents are allocated, CICS searches for a free extent in another DSA, to relocate it to the DSA in need. For CICS to remove an extent from one DSA so that it can be allocated to another, all pages in the extent must be free. That is, no pages must be allocated to any subpool.

Program compression might be triggered when the **DSALIM** or **EDSALIM** limit is approached and there are few free or empty extents available. The DSAs that contain programs are evaluated individually to determine whether program compression is required. In systems with a moderate proportion of loadable programs, program compression is an indicator of pressure on virtual storage.

CICS considers a short-on-storage condition for a DSA in 24-bit or 31-bit storage if all the following circumstances apply:

- No further extents can be allocated or relocated from other DSAs.

- Program compression has been attempted.
- All nonresident programs that are suitable for deletion, and that are not in use, have been deleted.
- All cached copies of document templates that are suitable for deletion. See Caching and refreshing of document templates.
- Storage from the storage cushion is in use (that is, the number of contiguous free pages is less than the number of pages in the cushion), or at least one request is suspended because there is no contiguous area of storage large enough for it, or both of these conditions apply.

Short-on-storage conditions for 64-bit storage

For 64-bit storage, CICS tracks the total amount of 64-bit storage in use for the CICS address space. This storage includes both the above-the-bar dynamic storage area (GDSA), and MVS storage in the CICS region outside the GDSA.

CICS considers an SOS condition when storage from the storage cushion is in use, or at least one request is suspended because there is no contiguous area of storage large enough for it, or both of these conditions apply. No further extents can be allocated for a DSA in 64-bit storage if the sum of all allocated above-the-bar storage and the size of a new extent would exceed the **`MEMLIMIT`** value.

The CICS storage manager statistics show 64-bit storage usage. The CICS storage manager dynamic storage areas statistics show storage usage for the DSAs in the GDSA. Statistics of interest include the following:

- Current GDSA active
- Peak GDSA active
- Number of IARV64 CONVERT(FROMGUARD) failures
- Current GDSA allocated
- Peak GDSA allocated
- Times cushion released
- Times went short on storage

An IARV64 CONVERT(FROMGUARD) failure indicates that a request for 64-bit storage has failed. A request might fail because there is not enough auxiliary storage in the system to back the request. Also, a request might fail because a component that the CICS storage manager does not control, for example, a JVM server, has allocated so much storage that the storage manager is affected. CICS cannot resolve pressures on storage caused by components outside the GDSA allocating storage, so you must use the CICS statistics to identify such problems.

Avoiding short-on-storage conditions
To optimize your use of the CICS dynamic storage areas and their storage cushions, and help to avoid short-on-storage conditions, follow these principles.

Procedure

- The lower the number of concurrent transactions in the system, the lower the usage of virtual storage. If you can improve the internal response time for transactions, for example by minimizing physical I/O, you can decrease the usage of virtual storage.
- Avoid making large GETMAIN requests in your application programs.

  The storage cushion might not be large enough to satisfy a request for a large contiguous block of storage.
- Define programs as resident only where necessary.

  CICS cannot delete resident programs to reclaim space in a DSA, even if the programs are not in use.
- Use the CICS storage manager statistics to monitor storage cushion releases and storage request suspensions. If these incidents occur frequently, investigate the cause.

If necessary, reduce the maximum number of user tasks (using the **MXT** system initialization parameter) to reduce the number of tasks that use main storage.

- Try to define a reasonable number of transactions as SPURGE(YES) and with a DTIMOUT value.

  Only transactions defined in this way can be purged during an SOS condition, if they have been waiting for storage for longer than the DTIMOUT value. If there are too few purgeable transactions, storage might become deadlocked in the CICS system.

Related tasks

Analyzing short-on-storage conditions
Analysis of short-on-storage (SOS) problems begins by obtaining a dump when the system is in an SOS condition.

Fixing short-on-storage conditions caused by subpool storage fragmentation
You might experience short-on-storage conditions in 24-bit storage or 31-bit storage despite increasing the **DSALIM** or **EDSALIM** limits, respectively. In this situation, you might need to enable the CICS self-tuning mechanism. It is also possible to fix the size of each individual DSA by using the corresponding SIT override.

Analyzing short-on-storage conditions
Analysis of short-on-storage (SOS) problems begins by obtaining a dump when the system is in an SOS condition.

About this task

Short-on-storage conditions might occur for the following reasons. Perform this task to analyze short-on-storage conditions.

1. Other resource constraints that cause CICS tasks to occupy storage for longer than is normally necessary
2. A flood of tasks that overwhelms available free storage
3. Badly designed applications that require unreasonably large amounts of storage

Procedure

1. Set an entry in the dump table to produce a dump when message DFHSM0131, DFHSM0133, or DFHSM0606 is issued.
   For example, to produce a dump the first time message DFHSM0131 is issued, use the following command:

   ```
   CEMT SET SYDUMPCODE(SM0131) SYSDUMP MAXIMUM(1) ADD
   ```

2. When you obtain the dump, enter the following IPCS commands:
   a) Use the IPCS command `VERBX CICS710 'SM=3'` to format the SM control blocks.
   b) Use the IPCS command `VERBX CICS710 'LD=3'` to format the LD control blocks.
3. Run DFH0STAT just before the statistics interval completes.

   For example, if the statistics interval is 1 hour, run DFH0STAT at 59 minutes. DFH0STAT provides useful information in the storage summary without a breakdown by subpool. See The sample statistics program, DFH0STAT for more information.
4. In the information that you have collected, examine the DSA summaries, noting which DSAs are short on storage and the amount of free space in the other DSAs.

   The amount of free space is given for each extent for each DSA. Frequently, either the UDSA or the CDSA is short on storage but there is a large amount of free storage in the SDSA.

   Also, look for evidence of large amounts of redundant program storage (RPS), which can cause a short-on-storage condition. Redundant program storage can be identified in the domain subpool summary and the loader domain summary.

Example

The dump extracts in this example are from a situation where the UDSA is short on storage.

Storage extents in 24-bit storage (below the line) are always allocated in multiples of 256 KB, except for the UDSA. If transaction isolation is active, the extent size for the UDSA is 1 MB, and each UDSA extent must be aligned on a megabyte boundary. If translation isolation is not active, the allocation is in 256 KB extents. You must allow for some fragmentation between the 256 KB extents of the CDSA, RDSA, and SDSA, compared with the 1 MB extents of the UDSA.

Storage extents in 31-bit storage (above the line) are allocated in multiples of 1 MB.

Storage extents in 64-bit storage (above the bar) are allocated in multiples of 2 GB.

Each extent has an associated page pool extent (PPX) and page allocation map (PAM).

Examination of the SDSA extents shows several extents with large amounts of free space. For example, the extent beginning at 00700000 running through 0073FFFF has only 4 KB allocated and 252 KB free.

```
Extent list:      Start      End          Size     Free
                  00700000   0073FFFF     256K     252K
```

The DSA extent summary shows that the PPX for the extent at 00700000 is found at 09F0A100, and the associated PAM is found at 09F0A150. Examination of the PAM shows that only one page is allocated, and it belongs to the subpool with an ID of X'7A'.

```
  Start      End          Size   PPX_addr   Acc     DSA
  00700000   0073FFFF     256K   09F0A100    C     SDSA

  PPX.SDSA 09F0A100 Pagepool Extent Control Area

     0000  00506EC4 C6C8E2D4 D7D7E740 40404040  *.&>DFHSMPPX     *
     0010  E2C4E2C1 40404040 09A1BA68 071B3EA0  *SDSA    ........*
     0020  00040000 00700000 0073FFFF 071B5EE0  *................*
     0030  00000000 09F0A150 00000040 0710A268  *.....0.&;.. ..s.*
     0040  0003F000 00000000 00000000 00000000  *..0.............*

  PAM.SDSA 09F0A150 Page Allocation Map

     0000  00000000 00000000 00000000 00000000  *................*
     0010 -    002F LINES SAME AS PREVIOUS
     0030  00000000 0000007A 00000000 00000000  *................*
```

The domain subpool summary determines, for the SDSA, which subpool is associated with the ID of X'7A'. In this dump, 7A is the ID for subpool ZCTCTUA. Do not rely on the IDs being the same for multiple runs of CICS, because the IDs are assigned in the order in which the ADD_SUBPOOL is issued.

```
  ==SM: UDSA Summary (first part only)

     Size:                           512K
     Cushion size:                    64K
     Current free space:              56K  (10%)
   * Lwm free space:                  12K  ( 2%)
   * Hwm free space:                 276K  (53%)
     Largest free area:               56K
   * Times nostg returned:              0
   * Times request suspended:           0
     Current suspended:                 0
   * Hwm suspended:                     0
   * Times cushion released:            1
     Currently SOS:                   YES

  ==SM: SDSA Summary (first part only)

     Size:                          4352K
     Cushion size:                    64K
     Current free space:            2396K  (55%)
   * Lwm free space:                 760K  (17%)
   * Hwm free space:                2396K  (55%)
     Largest free area:              252K
   * Times nostg returned:              0
   * Times request suspended:           0
     Current suspended:                 0
   * Hwm suspended:                     0
   * Times cushion released:              0
     Currently SOS:                      NO
```

What to do next

1. Review the storage limits for your CICS system. See “Setting the limits for CICS storage” on page 76.
2. For an SOS condition in 24-bit storage, determine whether the **DSALIM** parameter is set as large as possible. See “Estimating, checking, and setting DSALIM” on page 78.
3. For an SOS condition in 31-bit storage, determine whether the **EDSALIM** parameter is set as large as possible. See “Estimating, checking, and setting EDSALIM” on page 79.
4. For an SOS condition in 64-bit storage, determine whether the z/OS **MEMLIMIT** parameter is set to an appropriate value. See “Estimating, checking, and setting MEMLIMIT” on page 81.
5. Review the use of options such as the maximum task specification (**MXT** parameter) and defining programs as resident, to keep down the overall storage requirement. Changing these settings might limit task throughput. You can also reduce a storage constraint below 16 MB by using programs that run above 16 MB. In addition, using the LPA reduces the amount of storage used in LDNUCRO by approximately 100 KB.
6. Consider the tuning possibilities of z/OS and other tuning possibilities outside CICS. Also consider ways of dividing your CICS region.
7. Consider enabling the CICS self-tuning mechanism, or fixing the size of one or more individual DSAs by using the appropriate SIT overrides. For instructions, see “Fixing short-on-storage conditions caused by subpool storage fragmentation” on page 90.

Related tasks

Avoiding short-on-storage conditions
To optimize your use of the CICS dynamic storage areas and their storage cushions, and help to avoid short-on-storage conditions, follow these principles.

Fixing short-on-storage conditions caused by subpool storage fragmentation
You might experience short-on-storage conditions in 24-bit storage or 31-bit storage despite increasing the **DSALIM** or **EDSALIM** limits, respectively. In this situation, you might need to enable the CICS self-tuning mechanism. It is also possible to fix the size of each individual DSA by using the corresponding SIT override.

Fixing short-on-storage conditions caused by subpool storage fragmentation
You might experience short-on-storage conditions in 24-bit storage or 31-bit storage despite increasing the **DSALIM** or **EDSALIM** limits, respectively. In this situation, you might need to enable the CICS self-tuning mechanism. It is also possible to fix the size of each individual DSA by using the corresponding SIT override.

About this task

Use the self-tuning mechanism and the SIT overrides only if increasing the **DSALIM** or **EDSALIM** limit does not completely resolve the short-on-storage problems.

Allocating into managed extents can result in a block of storage in an extent that is insufficient to satisfy a GETMAIN request. With the dynamic nature of the subpools and DSAs, this situation will probably resolve as the extent storage is reused. If you specify the initial DSA size using the SIT override for the affected DSA, CICS reserves contiguous extents up to the amount specified, and eliminates the blocks of storage.

Tip: Define MAPS as MAPS. If you defining MAPS as programs, they are loaded into LDRES rather than into LDNUC. LDRES is part of the SDSA and is more sensitive to fragmentation.

Procedure

1. You can add records to the local catalog to enable the CICS self-tuning mechanism for storage manager domain subpools.

For details of how to manipulate subpool records using the CICS-supplied utility program, DFHSMUTL, see Local catalog storage program (DFHSMUTL) .

2. You can fix the size of one or more individual DSAs by using the corresponding SIT overrides (**CDSASZE**, **UDSASZE**, **SDSASZE**, **RDSASZE**, **ECDSASZE**, **EUDSASZE**, **ESDSASZE**, and **ERDSASZE**).

   For more information about these overrides, see System initialization parameter descriptions and summary .

   To determine the values to use, follow this process:

   a) Collect DFH0STAT output for information showing storage use by each DSA during the intervals.

   b) Review the CICS statistics for several days.

      The statistics provide information that you can use to define the amount of storage used at a subpool and a DSA level. Extent usage is shown with the number of extents added and released.

      In addition to the DSA information provided in DFH0STAT, the results about each subpool are provided, including the DSA where it was allocated. If statistics are being gathered, end-of-day statistics only provide data since the last statistics collection.

Related tasks

Avoiding short-on-storage conditions
To optimize your use of the CICS dynamic storage areas and their storage cushions, and help to avoid short-on-storage conditions, follow these principles.

Analyzing short-on-storage conditions
Analysis of short-on-storage (SOS) problems begins by obtaining a dump when the system is in an SOS condition.

CICS subpools

In each dynamic storage area, storage is arranged in subpools. The CICS subpools are dynamically acquired as needed, a page at a time, from within the applicable dynamic storage area.

Most CICS subpools are in 31-bit (above-the-line) or 64-bit (above-the-bar) storage. The subpools in 24-bit (below-the-line) storage must be monitored more carefully because of the limited space available.

Individual subpools can be static or dynamic. Some subpools contain static CICS storage, which cannot be tuned. All the subpools are rounded up to a multiple of 4 KB in storage size. Include this rounding factor in any subpool sizing, or evaluation of storage size changes after tuning or other changes.

The CICS domain subpools statistics contain useful information about the size and use of the dynamic storage area subpools. The following topics list the subpools in each dynamic storage area and their use. You can use this information to identify the possible causes of excessive usage in individual subpools.

CICS subpools in the CDSA
The subpools in the CICS dynamic storage area (CDSA) are listed, together with the use of each one.

Table 5. CICS subpools in the CDSA

Subpool name	Description
AP_TCA24	Storage for the TCA when the task data location option is set to BELOW.
DFHAPD24	A general subpool for 24-bit application domain storage.
DFHTDSDS	Storage for real transient data SDSCIs, each of which contains a DCB that resides in 24-bit storage (below 16 MB).
DHPDPOOL	Storage for DCBs for partitioned data sets used by document handler domain.
FC_DCB	Storage for the DCBs for BDAM files. Each file that is defined requires 104 bytes.
FCCBELOW	Storage for real VSWA and data buffers for pre-reads. Each VSWA requires 120 bytes of storage. The maximum number of data buffers for pre-reads is given by: (number of strings) x (maximum record length) x (number of files).

Table 5. CICS subpools in the CDSA (continued)

Subpool name	Description
KESTK24	A single 2 KB 24-bit (below 16 MB) stack segment. This is a dummy stack segment that is shared by all tasks. Tasks that need to use 24-bit stack storage obtain an extension stack segment from the subpool KESTK24E.
KESTK24E	4 KB 24-bit (below 16 MB) extension stack segments obtained by tasks that need to use 24-bit stack storage. CICS preallocates a reserve pool of 24-bit extension stack segments that tasks can use if no other storage is available in the subpool.
LD_JFCB	Storage for the job file control blocks for the loader domain.
LDNRS	Storage for the CICS nucleus and macro tables that are RESIDENT. The CICS nucleus is approximately 192 KB and the size of the tables can be calculated. Programs are defined as EXECKEY (CICS) and link edited with RMODE(24) without the reentrant open.
LDNUC	Storage for the CICS nucleus and macro tables that are not RESIDENT. The CICS nucleus is approximately 192 KB and the size of the tables can be calculated. Programs are defined as EXECKEY (CICS) and link edited with RMODE(24) without the reentrant open.
SMCONTRL	Satisfies GETMAIN requests for control class storage.
SMSHARED	24-bit shared storage, for example RMI global work areas, EDF blocks for the life of the transaction being monitored, and other control blocks.
SMSHRC24	Used for many control blocks of SHARED_CICS24 class storage.
SMTP24	Storage for line and terminal I/O areas that cannot be located above 16 MB. The storage requirements depend on the amount of terminal and line traffic in the system. The subpool can be tuned by reducing the RAPOOL, RAMAX, TIOAL size, and number of MRO sessions.
SZSPFCAC	FEPI z/OS Communications Server ACB work areas.
TRUBELOW	Task-related user exit pool below 16 MB.
XMGEN24	General storage used by transaction manager.
ZCSETB24	Application control buffers below 16 MB.
ZCTCTUA	Storage for the TCTTE user area. It can be located in one of the following DSAs: SDSA, ECDSA, CDSA, or ESDSA. Its location is controlled by the system initialization parameter, **TCTUALOC**=ANY|BELOW and the system initialization parameter, **TCTUAKEY**=CICS|USER. The maximum size can be specified in USERAREALEN operand of the terminal definition. For more information about the terminal definition, see TERMINAL resources.

CICS subpools in the SDSA
The subpools in the shared dynamic storage area (SDSA) are listed, together with the use of each one.

Table 6. CICS subpools in the SDSA

Subpool name	Description
APECA	Storage for the event control areas.
DFHAPU24	A general subpool for application domain storage below 16 MB.
LDPGM	Storage for dynamically loaded application programs (RMODE (24)). The expected size of this subpool can be predicted from previous releases, and by taking LDPGMRO into account. The subpool size can be reduced by using 31-bit programs. Not reentrant.

Table 6. CICS subpools in the SDSA (continued)

Subpool name	Description
LDRES	Storage for resident application programs (RMODE (24). The expected size of this subpool can be predicted from previous releases, and by taking LDRESRO into account. The subpool size can be reduced by using 31-bit programs. Not reentrant.
OSCOBOL	Used for the allocation of the COBOL merged load list (MLL) control block and its extents. This subpool should never occupy more than its initial allocation of one page of storage.
SMSHRU24	Used for many control blocks of SHARED_USER24 class storage.
ZCTCTUA	Storage for the TCTTE user area. It can be located in one of the following DSAs: SDSA, ECDSA, CDSA, or ESDSA. Its location is controlled by the system initialization parameter, **TCTUALOC**=ANY|BELOW and the system initialization parameter, **TCTUAKEY**=CICS|USER. The maximum size can be specified in USERAREALEN operand of the terminal definition. For more information about the terminal definition, see TERMINAL resources.

CICS subpools in the RDSA
The subpools in the read-only dynamic storage area (RDSA) are listed, together with the use of each one.

Table 7. CICS subpools in the RDSA

Subpool name	Description
LDNRSRO	Storage for programs that are defined EXECKEY(CICS) that are RESIDENT, that were link edited REENTRANT and RMODE(24).
LDNUCRO	Storage for programs that are defined EXECKEY(CICS) that are not RESIDENT, that were link edited REENTRANT and RMODE(24).
LDPGMRO	Storage for programs that are defined EXECKEY(USER) that are not RESIDENT, that were link edited RMODE(24) and REENTRANT.
LDRESRO	Storage for programs that are defined EXECKEY(USER) and RESIDENT and were link edited REENTRANT and RMODE(24).

CICS subpools in the ECDSA
The subpools in the extended CICS dynamic storage area (ECDSA) are listed, together with the use of each one.

Table 8. CICS subpools in the ECDSA

Subpool name	Description
>LGJMC	Storage for the journal model resource entries for the log manager domain.
AITM_TAB	The autoinstall terminal model (AITM) table entry subpool (DFHAITDS).
AP_TCA31	Storage for the TCA when the task data location option is set to ANY.
AP_TXDEX	Storage for the application part of the TXD table.
APAID31	Storage for AIDs above the line.
APBMS	Storage used by BMS.
APCOMM31	Storage for COMMAREAs. The storage requirement depends on the size of COMMAREA specified and the number of concurrent users of the application.
APDWE	Storage for non-task deferred work elements.
APICE31	Storage for ICEs above the line.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	**Description**
APURD	Subpool containing unit of recovery descriptors (URDs) and nontask deferred work elements (DWEs).
ASYNC4K	4K buffers used by asynchronous operations in the sockets domain for non-IPIC protocols.
ASYNC64K	64K buffers used by asynchronous operations in the sockets domain for the IPIC protocol.
BAGENRAL	A general-purpose subpool for the business application manager domain.
BAOFBUSG	Buffer storage used by the business application manager domain.
BAOFT_ST	Storage used by activities in the business application manager domain.
BR_BFBE	Storage for the bridge facility block extension.
BR_BFNB	Storage for the bridge facility name block.
BR_BMB	Storage for the bridge message block.
BR_BSB	Storage for bridge start blocks.
BRGENRAL	General-purpose subpool used by the bridge.
BRNSBLK	Storage used for the bridge numberspace.
BRNSFBLK	Storage used for bridge files.
BRPC	Storage used for bridge primary clients.
BRVS	Storage used for bridge virtual terminals.
BRVSCA	Storage used for bridge virtual screen character attributes.
BRVSXA	Storage used for bridge virtual screen extended attributes.
CCNV_BCE	Storage for character conversion buffer chain elements.
CCNV_CCE	Storage for character conversion chain elements.
CCNV_TRT	Storage for character conversion translation tables. These tables are addressed by the conversion chain elements.
CCNVG_AN	Storage for character conversion anchor blocks.
COLARAY	Storage for web control block array storage.
CQCQ_AN	Storage for console queue management anchor blocks.
CQCQ_CB	Storage for console queue management command input buffers.
DBCTL	Subpool that contains the TIE blocks for RMI use, when called by the DBCTL task-related user exit program, DFHDBAT. The TIE is 120 bytes long, and appended to the TIE is the local task work area for this task-related user exit which is, for DFHDBAT, 668 bytes long. This subpool is present only when DBCTL is used. It can be tuned by limiting DBCTL threads or using maximum tasks (MXT) or transaction classes.
DBDBG	Storage for DBCTL global blocks.
DCTE_EXT	Storage for all extrapartition queue definitions.
DCTE_IND	Storage for all indirect queue definitions.
DCTE_INT	Storage for all intrapartition queue definitions.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
DCTE_REM	Storage for all remote queue definitions.
DDAPSESS	Storage for LDAP sessions state control blocks.
DDAPSRCH	Buffers for LDAP search results.
DDBROWSE	Storage for directory manager browse request tokens.
DDGENRAL	Storage for directory manager control blocks general information.
DDS_BFBE	Storage for directory manager directory elements for the BFBE table.
DDS_BFNB	Storage for directory manager directory elements for the BFNB table.
DDS_DCTE	Storage for directory manager directory elements for the DCTE table.
DDS_DHT1	Storage for directory manager directory elements for the DHT1 table.
DDS_DHT2	Storage for directory manager directory elements for the DHT2 table.
DDS_DSN	Storage for directory manager directory elements for the DSN table.
DDS_D2AC	Storage for directory manager directory elements for the D2AC table.
DDS_D2CS	Storage for directory manager directory elements for the D2CS table.
DDS_D2EN	Storage for directory manager directory elements for the D2EN table.
DDS_D2TN	Storage for directory manager directory elements for the D2TN table.
DDS_D2TT	Storage for directory manager directory elements for the D2TT table.
DDS_ECCS	Storage for directory manager directory elements for the ECCS table.
DDS_ECEV	Storage for directory manager directory elements for the ECEV table.
DDS_ECSC	Storage for directory manager directory elements for the ECSC table.
DDS_EPAD	Storage for directory manager directory elements for the EPAD table.
DDS_FCT	Storage for directory manager directory elements for the FCT table.
DDS_ISIA	Storage for directory manager directory elements for the ISIA table.
DDS_ISIN	Storage for directory manager directory elements for the ISIN table.
DDS_JVMD	Storage for directory manager directory elements for the JVMD table.
DDS_MLRL	Storage for directory manager directory elements for the MLRL table.
DDS_MLXT	Storage for directory manager directory elements for the MLXT table.
DDS_MQII	Storage for directory manager directory elements for the MQII table.
DDS_MQIN	Storage for directory manager directory elements for the MQIN table.
DDS_NQRN	Storage for directory manager directory elements for the NQRN table.
DDS_PIPL	Storage for directory manager directory elements for the PIPL table.
DDS_PPT	Storage for directory manager directory elements for the PPT table.
DDS_PTPO	Storage for directory manager directory elements for the PTPO table.
DDS_PTST	Storage for directory manager directory elements for the PTST table.
DDS_PTT	Storage for directory manager directory elements for the PTT table.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
DDS_REFE	Storage for directory manager directory elements for the REFE table.
DDS_RLBN	Storage for directory manager directory elements for the RLBN table.
DDS_RTXD	Storage for directory manager directory elements for the RTXD table.
DDS_SCAC	Storage for directory manager directory elements for the SCAC table.
DDS_SERV	Storage for directory manager directory elements for the SERV table.
DDS_SOCI	Storage for directory manager directory elements for the SOCI table.
DDS_SOSI	Storage for directory manager directory elements for the SOSI table.
DDS_TCL	Storage for directory manager directory elements for the TCL table.
DDS_TPNM	Storage for directory manager directory elements for the TPNM table.
DDS_TXD	Storage for directory manager directory elements for the TXD table.
DDS_USD1	Storage for directory manager directory elements for the USD1 table.
DDS_USD2	Storage for directory manager directory elements for the USD2 table.
DDS_USD3	Storage for directory manager directory elements for the USD3 table.
DDS_USD4	Storage for directory manager directory elements for the USD4 table.
DDS_WBST	Storage for directory manager directory elements for the WBST table.
DDS_WBUR	Storage for directory manager directory elements for the WBUR table.
DDS_WSRD	Storage for directory manager directory elements for the WSRD table.
DDS_WURS	Storage for directory manager directory elements for the WURS table.
DDS_W2AT	Storage for directory manager directory elements for the W2AT table.
DDS_W2RL	Storage for directory manager directory elements for the W2RL table.
DFHAPDAN	A general subpool for application domain storage above 16 MB but below 2 GB.
DFHD2CSB	Storage for control blocks representing DB2 threads created by the CICS/DB2 adapter.
DFHD2ENT	Storage for control blocks representing DB2ENTRY definitions.
DFHD2PKG	Storage for control blocks representing DB2 PACKAGESET definitions.
DFHD2TRN	Storage for control blocks representing DB2TRAN definitions.
DFHECCD	Storage for event capture data.
DFHECCS	Storage for event capture specification blocks.
DFHECDQE	Storage for event capture deferred filter queue elements.
DFHECEVB	Storage for event capture event binding blocks.
DFHECFP	Storage for event capture event filter predicate blocks.
DFHECSC	Storage for event capture system event calls.
DFHECSF	Storage for event capture system filter predicates.
DFHEPAC	Storage for event capture event adapter configuration data.
DFHMPGEN	Used for allocations of Managed Platform anchor block (MPA) and failed adapter (MPPFA) control blocks.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
DFHMPMOD	Used for allocations of Managed Platform model (MPMOD) control blocks.
DFHMPPPB	Used for allocations of Managed Platform policy (MPPPB) control blocks.
DFHTDG31	Transient data general storage and control blocks. The storage requirement depends on the number of buffers and strings, and on the control interval size specified.
DFHTDIOB	Intrapartition transient data input/output buffers. The storage requirement is given by the control interval size of the intrapartition transient data set multiplied by the number of buffers.
DFHTDWCB	Storage for the transient data wait elements.
DHCACHE	Storage for cached copies of document templates.
DHDBB	Storage for document bookmark blocks.
DHDCR	Storage for document control records.
DHDDB	Storage for document data.
DHDOA	Storage for document anchor blocks.
DHFSPATH	Storage for HFS path template extensions.
DHGENRAL	The general purpose subpool for the document manager domain.
DHSTB	Storage for document symbol tables.
DHTLPOOL	Storage for document handler template descriptors.
DLI	Subpool that contains the TIE blocks for RMI use, when called by the EXEC DL/I task-related user exit program, DFHEDP. The TIE is 120 bytes long, and appended to the TIE is the local task work area for this task-related user exit, which is, for DFHEDP, 4 bytes long. This subpool is present only when EXEC DL/I is used. It can be tuned by limiting DBCTL threads or using maximum tasks (MXT) or transaction classes.
DMSUBPOL	The domain manager subpool for general usage.
DP_GENRL	Storage for the control blocks for the DP domain.
DPLA	Storage for the anchor blocks for instore linked lists of debugging profiles.
DPLE	Storage for the elements in the instore linked lists of debugging profiles.
DPLP	Storage for the elements in the debug profile that is used for pattern matching.
DPTA	Storage for transaction instance state data that is required by the DP domain.
DS_STIMR	Storage for dispatcher domain STIMER tokens.
DS_TCB	Storage for dispatcher domain TCBs.
DS_VAR	The dispatcher domain variable length subpool.
DSBROWSE	Storage for dispatcher browse request tokens.
EC_GENRL	Storage for the control blocks for the EC domain.
EJMI	The enterprise bean method information.
EJOSGENS	The enterprise bean general subpool.
EJOSTSKS	The enterprise bean task subpool.
EJSPBFBC	Storage for web browser control blocks for enterprise beans.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
EJSPBVIC	Storage for enterprise bean control blocks.
EJSPCFBC	Storage for web browser control blocks for CorbaServers.
EJSPCFIC	Storage for control blocks for CorbaServers.
EJSPCOMM	Storage for anchor blocks for enterprise beans.
EJSPDFBC	Storage for web browser control blocks for deployed JAR files.
EJSPDFIC	Storage for control blocks for deployed JAR files.
EJSPGVNC	Storage for persistent storage for enterprise beans.
EJSPTVNC	Storage for transaction-related storage for enterprise beans.
EJSTGENS	Storage for control blocks for enterprise bean statistics.
EMBRB	Storage for event manager browse blocks.
EMEVA	Storage for the event manager event pool anchor.
EMEVB	Storage for event manager event blocks.
EMGENRAL	General-purpose subpool for event manager domain.
EP_GENRL	Storage for the control blocks for the EP domain.
EPADA	Storage for event processing adapter management.
EPADI	Storage for EP adapter name in EP adapter set.
EPADT	Storage for event processing adapter set management.
FC_ABOVE	Storage for real VSWA and data buffers for prereads. Each VSWA requires 120 bytes of storage. The maximum number of data buffers for prereads is given by: (number of strings) x (maximum record length) x (number of files)
FC_ACB	Storage for ACBs for VSAM files. Each VSAM file has one ACB, of 80 bytes.
FC_BDAM	Storage for BDAM file control blocks. Each BDAM file requires 96 bytes of storage.
FC_DSNAM	Storage for data set name blocks. Each file requires a data set name block, which uses 120 bytes of storage.
FC_FCPE	Storage for file control pool elements.
FC_FCPW	Storage for file control CFDT pool wait elements.
FC_FCUP	Storage for the file control CFDT unit of work pool block.
FC_FLAB	Storage for file control lasting access blocks.
FC_FLLB	Storage for file control lock locator blocks.
FC_FRAB	Storage for file request anchor blocks (FRABs). Each transaction that has issued a file control request has one FRAB. The FRAB is retained until the end of the task. There is a free chain of FRABs not currently in use.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
FC_FRTE	Storage for file request thread elements (FRTE). There is one FRTE for each active file control request per task. A file control request has a FRTE if it meets these conditions: • It has not yet stopped its VSAM thread. For example, a browse that has not yet issued an ENDBR. • It has updated a recoverable file and a sync point has not yet occurred. • It is holding READ-SET storage that must be freed in future. There is a free chain of FRTEs not currently in use.
FC_RPL	Storage for file control request parameter lists.
FC_SHRCT	Storage for file control SHRCTL blocks. There are eight of these blocks and each describes a VSAM LSR pool.
FC_VSAM	Storage for the file control table (FCT) entries for VSAM files.
FCB_256	File control buffers of length 256 bytes. These buffers are used by file control requests that are made against files with a maximum record length less than or equal to 256 bytes.
FCB_512	File control buffers of length 512 bytes. These buffers are used by file control requests that are made against files with a maximum record length between 256 + 1 bytes and 512 bytes.
FCB_1K	File control buffers of length 1 KB. These buffers are used by file control requests that are made against files with a maximum record length between 512 + 1 bytes and 1 KB.
FCB_2K	File control buffers of length 2 KB. These buffers are used by file control requests that are made against files with a maximum record length between 1 KB + 1 byte and 2 KB.
FCB_4K	File control buffers of length 4 KB. These buffers are used by file control requests that are made against files with a maximum record length between 2 KB + 1 byte and 4 KB.
FCB_8K	File control buffers of length 8 KB. These buffers are used by file control requests that are made against files with a maximum record length between 4 KB + 1 byte and 8 KB.
FCB_16K	File control buffers of length 16 KB. These buffers are used by file control requests that are made against files with a maximum record length between 8KB + 1 byte and 16 KB.
FCB_32K	File control buffers of length 32 KB. These buffers are used by file control requests that are made against files with a maximum record length between 16 KB + 1 byte and 32 KB.
FCB_64K	File control buffers of length 64 KB. These buffers are used by file control requests that are made against files with a maximum record length between 32 KB + 1 byte and 64 KB.
FCB_128K	File control buffers of length 128 KB. These buffers are used by file control requests that are made against files with a maximum record length between 64 KB + 1 byte and 128 KB.
FCB_256K	File control buffers of length 256 KB. These buffers are used by file control requests that are made against files with a maximum record length between 128 KB + 1 byte and 256 KB.
FCB_512K	File control buffers of length 512 KB. These buffers are used by file control requests that are made against files with a maximum record length between 256 KB + 1 byte and 512 KB.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
FCB_1M	File control buffers of length 1MB. These buffers are used by file control requests that are made against files with a maximum record length between 512 KB + 1 byte and 1 MB.
FCB_2M	File control buffers of length 2 MB. These buffers are used by file control requests that are made against files with a maximum record length between 1 MB + 1 byte and 2 MB.
FCB_4M	File control buffers of length 4 MB. These buffers are used by file control requests that are made against files with a maximum record length between 2 MB + 1 byte and 4 MB.
FCB_8M	File control buffers of length 8 MB. These buffers are used by file control requests that are made against files with a maximum record length between 4 MB + 1 byte and 8 MB.
FCB_16M	File control buffers of length 16 KB. These buffers are used by file control requests that are made against files with a maximum record length between 8 MB + 1 byte and 16 MB.
FCSTATIC	File control static storage.
ICUS	Storage for internal control element (ICE) secure extensions.
IE_GENRL	Storage for the control blocks for the IE domain.
IECCB	Storage for the conversation control blocks in the IE domain.
IECSB	Storage for the client state blocks in the IE domain.
IFGLUWID	The VSAM IFGLUWID area.
IIGENRAL	The IIOP domain general subpool.
IIMBR	The IIOP domain request model browse block.
IIMDB	The IIOP domain request model block.
IS_GENRL	Storage for the control blocks for the IS domain.
ISAQ	Storage for IS allocate queue elements.
ISCB	Storage for IS control blocks, used to record installed instances of IPCONNs.
ISQA	Storage for IS queue attach control blocks.
ISRD	Storage for IS remote delete requests.
ISSB	Storage for the IS session blocks, each of which is associated with an ISCB subpool.
ISSS	Storage for IS session sets.
KEANCHOR	Storage Manager domain anchors.
KESTK31	28 KB 31-bit (above the line) stack segments. There is one per MXT plus one for every dynamic system task that is running.
KESTK31E	8 KB 31-bit (above the line) extension stack segments. There is at least one for every ten tasks specified in the MXT limit.
KETASK	Storage for kernel task entries.
LD_CDE	Storage for loader domain dummy CDEs.
LD_CNTRL	Storage for loader domain general control information.
LD_LDBE	Storage for LDBE (loader browse element) control blocks for the loader domain.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
LD_LDWE	Storage for LDWE (loader suspend work element) control blocks for the loader domain.
LD_PLIBE	Storage for program library element storage for the loader domain.
LDENRS	Storage for the extended CICS nucleus, and 31-bit macro tables that are RESIDENT. The extended CICS nucleus is approximately 50 KB. Programs are defined with EXECKEY(CICS) and link edited RMODE(ANY) without the REENTRANT option.
LDENUC	Storage for the extended CICS nucleus and 31-bit macro tables that are not RESIDENT. The extended CICS nucleus is approximately 50 KB. Programs are defined with EXECKEY(CICS) and link edited RMODE(ANY) without the REENTRANT option.
LGBD	Storage for log stream name, journal name, and journal model browse tokens for the log manager domain.
LGGD	Storage for explicitly opened general logs for the log manager domain.
LGGENRAL	The general-purpose subpool for the log manager domain.
LGJI	Storage for journal name entries for the log manager domain.
LGSD	Storage for log stream data entries for the log manager domain.
LGUOW	Storage for unit-of-work data entries for the log manager domain.
LI_PLB	Storage for the language interface program language block. One is allocated for each program when control is first passed to it.
L2GENRAL	The log manager domain general subpool.
L2OFL2BL	Storage for logger block entries for the log manager domain.
L2OFL2BS	Storage for browseable stream objects for the log manager domain logger.
L2OFL2CH	Storage for chain objects for the log manager domain logger.
L2OFL2SR	Storage for stream objects for the log manager domain logger.
MDTTABLE	The MDT field attribute table for BMS maps sent through the CICS web interface.
ML_GENRL	General storage for the ML domain.
MN_CNTRL	Storage for monitoring control blocks general information.
MN_TIMAS	Storage for monitoring control blocks identity monitoring area.
MN_TMAS	Storage for monitoring control blocks transaction monitoring area.
MN_TRMAS	Storage for monitoring control blocks resource monitoring area.
MQM	WebSphere MQ communication storage.
MRO_QUEU	Used by the MRO work queue manager.
MROWORKE	Used by the MRO work queue manager elements.
NQEAS	Storage for NQ domain queue element areas.
NQGENRAL	A general subpool used by NQ domain.
NQPOOL	Storage for NQ domain enqueue pools.
NQRNAMES	Storage for NQRN directory entries.
OTGENRAL	The general subpool used by the OT domain.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
OTISINST	Storage for inflight state of OTS transactions.
OVERLAPD	Storage for overlap field merging.
PGCHCB	Storage for channel control blocks. This storage contains header information that describes a channel.
PGCPCB	Storage for the channel container pool control block. This storage contains header information that describes sets of containers.
PGCPCBCH	Storage for the chained container pool control block.
PGCRBB	Storage for browses of channel containers.
PGCRCB	Storage for channel container control blocks. This storage contains the header information for each container.
PGCSCB	Storage for channel container segments.
PGGENRAL	Storage for general purpose program manager domain subpools.
PGHMRSA	Storage for program handle manager cobol register save areas.
PGHTB	Storage for the program manager handle table block.
PGJVMCL	Storage for JVM class names.
PGLLE	Storage for program manager load list elements.
PGPGWE	Storage for program manager wait elements.
PGPPTE	Storage for program manager program definitions.
PGPTA	Storage for program manager transaction-related information.
PI_GENRL	General storage for the pipeline manager (PI) domain.
PI_POLCY	Currently not used.
PI_PRSER	Currently not used.
PINODEBL	Storage for pipeline objects.
PIPEINST	Storage for pipeline objects.
PITKDAT	Storage for pipeline token data for context token.
PITKPOOL	Storage for pipeline tokens.
PITXMAST	Storage for Web Services Atomic Transaction (WS-AT) main control block or PI domain transaction control block.
PR_TABLE	Storage for PTEs from the PRT.
PTTWSB	General storage for pool tokens.
RCLELEM	Storage for the web row-column element list storage.
RCTABLE	Web table storage.
RLGENRAL	The resource lifecycle general subpool.
RMGENRAL	The recovery manager general subpool.
RMOFRMLK	Storage for recovery manager link objects.
RMOFRMUW	Storage for recovery manager unit-of-work objects.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
ROWARAY	Web row array storage.
RS_FILEL	Region status (RS) domain file list storage.
RS_GENRL	Storage for the control blocks for the RS domain.
RUNTRAN	A transaction manager subpool for run transaction.
RUTKPOOL	A subpool for reusable token class.
RXGENRAL	A general subpool for RX domain.
RZGENRAL	A general subpool for request streams domain.
RZOFRSNR	Storage for request streams notification requests.
RZOFRSRG	Storage for request streams registration objects.
RZOFRZRS	Storage for request streams objects.
RZOFRZTR	Storage for request stream transports.
SHGENRAL	The general subpool for scheduler services domain.
SHOFSHRE	Storage for scheduler services request objects.
SJGENRAL	The general subpool for SJVM domain.
SJJ8TCB	Storage for TCBs in the SJVM domain.
SMSHRC31	Storage for many control blocks of the SHARED_CICS31 class.
SMTP	Line and terminal I/O areas. The storage requirements depend on the amount of terminal and line traffic in the system. The subpool can be tuned by reducing the RAPOOL, RAMAX, TIOAL size, and number of MRO sessions.
SOCKET	Storage for socket objects.
SOCKPOOL	Socket pool storage.
SOCKSSL	Storage for the SSL data related to a socket.
SOGENRAL	The sockets domain general subpool.
SOLTE	Storage for socket domain listener terminal entries.
SOSTE	Storage for socket domain socket terminal entries.
SOTBR	Storage for socket domain TCPIPSERVICE browse blocks.
SOTDB	Storage for socket domain TCPIPSERVICE blocks.
SOTKPOOL	Storage for socket domain socket tokens.
STSUBPOL	A statistics domain manager subpool.
SZSPFCCD	The FEPI connection control subpool.
SZSPFCCM	The FEPI common area subpool.
SZSPFCCV	The FEPI conversation control subpool.
SZSPFCDS	The FEPI device support subpool.
SZSPFCNB	The FEPI node initialization block subpool.
SZSPFCND	The FEPI node definition subpool.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
SZSPFCPD	The FEPI pool descriptor subpool.
SZSPFCPS	The FEPI property descriptor subpool.
SZSPFCRP	The FEPI request parameter list subpool.
SZSPFCRQ	The FEPI requests subpool.
SZSPFCSR	The FEPI surrogate subpool.
SZSPFCTD	The FEPI target descriptor subpool.
SZSPFCWE	The FEPI work element subpool.
SZSPVUDA	The FEPI data areas subpool.
TA_GENRL	Currently not used.
TASKASOC	Storage for sockets domain task association objects.
TD_TDCUB	Storage for all the transient data CI update control blocks.
TD_TDQUB	Storage for all the transient data queue update control blocks.
TD_TDUA	Storage for all the transient data UOW anchor control blocks.
TFUS	Storage for TCTTE secure extensions.
TIA_POOL	The timer domain anchor subpool.
TIQCPOOL	The timer domain quick cell subpool.
TSBRB	Storage for temporary storage (TS) browse blocks.
TSBUFFRS	Temporary storage I/O buffers. The storage requirement is given by: (TS control interval size) x (number of TS buffers). The use of temporary storage by application programs affects the size of a number of subpools associated with temporary storage control blocks.
TSGENRAL	The amount of storage used by the TSGENRAL subpool. The amount depends on the number of buffers and strings and the control interval size defined for the temporary storage data set.
TSICDATA	Storage for TS interval control elements.
TSMBR	Storage for temporary storage browse blocks.
TSMDB	Storage for temporary storage model blocks.
TSQAB	Storage for TS queue anchor blocks.
TSQOB	Storage for TS queue ownership blocks.
TSQUB	Storage for TS queue update blocks.
TSTSS	Storage for TS section descriptors.
TSTSX	Storage for TS auxiliary item descriptors.
TSW	Storage for TS wait queue elements.
UE_EPBPL	The subpool for the user exit program block (EPB).
USIDTBL	Storage for the attach security userid table entries (LUITs). See ISC/IRC attach time entry statistics for more information.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
WBGENRAL	The general subpool for CICS web support.
WBOUTBND	Storage for outbound HTTP buffers.
WBPATHN1	Storage for path node elements used for URI map storage for short path names.
WBPATHN2	Storage for path node elements used for URI map storage for long path names.
WBPATHUR	Storage used for URI map storage for URI path names.
WBRQB	Storage for web request objects.
WBS	Storage for inbound web session blocks used for the IPIC protocol.
WBURIMAP	Storage for URI mapping elements.
WBURIXT1	Storage for URI mapping element extensions (short).
WBURIXT2	Storage for URI mapping element extensions (long).
WBWRBR	Storage for web request browse blocks.
WBVHOST	Storage for URI virtual host elements.
WEB_STA	Web state-related storage.
WEBELEM	Storage for web output element lists.
WEBHTML	Storage for web HTML buffers.
WEBINB	Storage for web domain storage for incoming data.
WEB327B	Web domain 3270 buffer storage.
W2ATOMSE	Storage for Web 2.0 atom service elements.
W2ATOMX1	Storage for Web 2.0 atom service extensions.
W2ATOMX2	Storage for Web 2.0 atom service extensions.
W2GENRAL	The general-purpose subpool for the Web 2.0 domain.
XMGENRAL	The general-purpose subpool for the transaction manager.
XMTCLASS	Storage for the transaction manager tranclass definition.
XMTRANSN	Storage for transaction manager transactions; one for every transaction in the system.
XMTXDINS	The transaction manager transaction definition.
XMTXDSTA	The transaction manager transaction definition.
XMTXDTPN	The transaction manager transaction definition TPNAME storage.
ZC2RPL	Storage for the duplicate RPLs for active tasks. Each active task associated with a z/OS Communications Server terminal requires 304 bytes.
ZCBIMG	Storage for BIND images.
ZCBMSEXT	Storage for the BMS extensions for terminals. Subpool storage requirements are 48 bytes for each terminal, surrogate, ISC session, and console.
ZCBUF	Storage for the non-LU 6.2 buffer list.
ZCCCE	Storage for the console control elements. Each console requires 48 bytes.
ZCGENERL	The general-purpose subpool for terminal control.

Table 8. CICS subpools in the ECDSA (continued)

Subpool name	Description
ZCLUCBUF	Storage for the LU 6.2 SEND and RECEIVE buffer list.
ZCLUCEXT	Storage for the LU 6.2 extensions. The storage requirement is 224 bytes for each LU 6.2 session.
ZCNIBD	Storage for the NIB descriptors. Each terminal, surrogate, ISC session, and system definition requires 96 bytes of storage.
ZCNIBISC	Storage for the expanded NIB and response during OPNDST and CLSDST for ISC. Each concurrent logon and logoff requires 448 bytes of storage. The maximum number of concurrent requests is limited by the number of sessions. The storage can be tuned by reducing the number of sessions.
ZCNIBTRM	Storage for the expanded NIB during OPNDST and CLSDST for terminals. Each concurrent logon and logoff requires 192 bytes of storage. The maximum number of concurrent requests is limited by the number of terminals. The storage can be tuned by reducing the number of terminals.
ZCRAIA	Storage for the RECEIVE ANY I/O areas.
ZCRPL	Storage for the RPLs for active tasks. Each active task associated with a z/OS Communications Server terminal requires 152 bytes.
ZCSETB	Storage for application control buffers above 16 MB but below 2 GB.
ZCSKEL	Storage for the remote terminal entries. Each remote terminal definition requires 32 bytes of storage.
ZCSNEX	Storage for the TCTTE sign-on extensions. The storage requirement is 48 bytes for each terminal, surrogate, session, and console.
ZCTCME	Storage for the mode entries. Each mode entry requires 128 bytes of storage.
ZCTCSE	Storage for the system entries. Each system entry requires 192 bytes of storage.
ZCTCTTEL	Storage for the large terminal entries. 504 bytes of storage are required for every terminal, surrogate model, and ISC session defined.
ZCTCTTEM	Storage for the medium terminal entries. 400® bytes of storage are required for every IRC batch terminal.
ZCTCTTES	Storage for the small terminal entries. 368 bytes of storage are required for every MRO session and console.
ZCTPEXT	The TPE extension.
ZCTREST	The terminal control transaction restart subpool.
ZCTCTUA	Storage for the TCTTE user area. It can be located in one of the following DSAs: SDSA, ECDSA, CDSA, or ESDSA. Its location is controlled by the system initialization parameter, **TCTUALOC**=ANY|BELOW and the system initialization parameter, **TCTUAKEY**=CICS|USER. The maximum size can be specified in USERAREALEN operand of the terminal definition. For more information about the terminal definition, see TERMINAL resources.

CICS subpools in the ESDSA

The subpools in the extended shared dynamic storage area (ESDSA) are listed, together with the use of each one.

Table 9. CICS subpools in the ESDSA

Subpool name	**Description**
DFHAPUAN	A general subpool for application domain storage above 16 MB but below 2 GB.
IE_BUFF	The IE domain buffers that are used when processing inbound and outbound messages.
IIBUFFER	The IIOP domain buffer subpool.
IS_BUFF	Storage for the IS buffers that are used to hold the message data for an IS session block.
LDEPGM	Storage for extended (31-bit) dynamically-loaded application programs and programs defined EXECKEY(USER).
LDERES	Storage for extended (31-bit) resident application programs.
SJSCCHS	Storage for the Java Virtual Machine domain (SJ domain) class cache.
SJSJPTE	Storage for the SJ domain profile table entries.
SJSJTCB	Storage for the SJ domain TCB usage.
SJUSERKY	SJ domain user key storage.
SMSHRU31	Used for many control blocks of SHARED_USER31 class storage, RMI global work areas, EDF blocks for the life of the transaction being monitored, and other control blocks.
TGODR	Storage for the transaction group origin data record.
WEBINB	Inbound Web 3270 buffer storage.
ZCTCTUA	Storage for the TCTTE user area. It can be located in one of the following DSAs: SDSA, ECDSA, CDSA, or ESDSA. Its location is controlled by the system initialization parameter, **TCTUALOC**=ANY|BELOW and the system initialization parameter, **TCTUAKEY**=CICS|USER. The maximum size can be specified in USERAREALEN operand of the terminal definition. For more information about the terminal definition, see TERMINAL resources.

CICS subpools in the ERDSA

The subpools in the extended read-only dynamic storage area (ERDSA) are listed, together with the use of each one.

Table 10. CICS subpools in the ERDSA

Subpool name	**Description**
LDENRSRO	Storage for the extended CICS nucleus and 31-bit macro tables that are RESIDENT. The extended CICS nucleus is approximately 1850 KB. The contents of this subpool must be linked reentrant.
LDENUCRO	Storage for the extended CICS nucleus and 31-bit macro tables that are not RESIDENT. The extended CICS nucleus is approximately 1850 KB. The contents of this subpool must be linked reentrant.
LDEPGMRO	Storage for extended (31-bit) dynamically loaded application programs. The contents of this subpool must be linked reentrant.

Table 10. CICS subpools in the ERDSA (continued)

Subpool name	Description
LDERESRO	Storage for extended (31-bit) resident application programs. The contents of this subpool must be linked reentrant.

CICS subpools in the ETDSA
The subpools in the extended trusted dynamic storage area (ETDSA) are listed, together with the use of each one.

Table 11. CICS subpools in the ETDSA

Subpool name	Description
USGENRAL	The general purpose subpool for the user domain.
USRTMQUE	Storage for queue elements for users waiting for USRDELAY. Each queue element is 16 bytes.
USUDB	Storage for user data blocks. The storage requirement is 128 bytes for each unique user.
USXDPOOL	Storage for user domain transaction-related data. Each running transaction requires 32 bytes.
XSGENRAL	The general purpose subpool for the security domain.
XSXMPOOL	Storage for security domain transaction-related data. Each running transaction requires 56 bytes.

CICS subpools in the GCDSA
The subpools in the above-the-bar CICS dynamic storage area (GCDSA) are listed, together with the use of each one.

Table 12. CICS subpools in the GCDSA

Subpool name	Description
CPSM_64	Storage for CICSPlex SM API result sets in a System Management Single Server (SMSS) environment.
CQCQ_TR	Storage for the console queue processing trace table.
CQCQ_XT	Storage for the console queue transaction entry table.
DFHAPD64	A general subpool for 64-bit application domain storage.
DFHMPMDR	Used for allocations of Managed Platform model rules (MPMODR) control blocks.
DFHMPPMB	Used for allocations of Managed Platform policy modifier (MPPMB) control blocks.
DFHMPPRB	Used for allocations of Managed Platform policy rule (MPPRB) control blocks.
DFHMPTAS	Used for allocations of Managed Platform task lifetime storage (MPTAS) control blocks.
EP_64	Storage for control blocks for items in event capture queues, used in CICS event processing.
LD_APES	Active Program Element (APE) control blocks for the loader domain.
LD_CPES	Current Program Element (CPE) control blocks for the loader domain.
LD_CSECT	CSECT list storage for the loader domain.
ML64GNRL	Buffers for input and output for the z/OS XML System Services (XMLSS) parser.

Table 12. CICS subpools in the GCDSA (continued)

Subpool name	Description
MN_ADCS	Storage for association data control blocks.
MPSTAT	Storage for control blocks for any static data capture items defined for a policy rule with an event action.
PGCSDB	Storage for channel container segments, including segment headers.
PGPPTE64	Storage for application context data control blocks.
SMSHRC64	Used for many control blocks of SHARED_CICS64 class storage.
SOGNRL6	Storage for HTTPS data for the sockets domain.
TSDTN	Temporary storage (TS) digital tree nodes.
TSMAIN	Storage for main temporary storage.
TSMN0064	Fixed length elements for main temporary storage items that have lengths, including the header, less than or equal to 64 bytes. The header length is 8 bytes.
TSMN0128	128-byte fixed length elements for main temporary storage items.
TSMN0192	192-byte fixed length elements for main temporary storage items.
TSMN0256	256-byte fixed length elements for main temporary storage items.
TSMN0320	320-byte fixed length elements for main temporary storage items.
TSMN0384	384-byte fixed length elements for main temporary storage items.
TSMN0448	448-byte fixed length elements for main temporary storage items.
TSMN0512	512-byte fixed length elements for main temporary storage items.
TSMN0576	576-byte fixed length elements for main temporary storage items.
TSMN0640	640-byte fixed length elements for main temporary storage items.
TSMN0704	704-byte fixed length elements for main temporary storage items.
TSMN0768	768-byte fixed length elements for main temporary storage items.
TSMN0832	832-byte fixed length elements for main temporary storage items.
TSMN0896	896-byte fixed length elements for main temporary storage items.
TSMN0960	960-byte fixed length elements for main temporary storage items.
TSMN1024	1024-byte fixed length elements for main temporary storage items.
TSQUEUE	TS queue descriptors.
TSTSI	TS item descriptors.
WB64GNRL	Storage for HTTP data for the Web domain (WB).
WU_64	Storage for CMCI retained results and metadata.
XMGEN64	A general subpool for 64-bit storage.

CICS subpools in the GSDSA
The subpools in the above-the-bar shared dynamic storage area (GSDSA) are listed, together with the use of each one.

Table 13. CICS subpools in the GSDSA

Subpool name	Description
DFHAPU64	A general subpool for application domain storage above the bar.
SMSHRU64	Used for many control blocks of SHARED_USER64 class storage.

CICS kernel storage

CICS kernel storage consists of control blocks and data areas that CICS requires to manage system and user tasks throughout CICS execution. Most of this storage is allocated from the CICS DSAs. A small amount of this storage is allocated from MVS storage.

The kernel recognizes two types of task: static tasks and dynamic tasks. The kernel storage for static tasks is preallocated and is used for tasks controlled by the MXT mechanism. The storage for dynamic tasks is not preallocated and is used for tasks, such as system tasks, which are not controlled by the MXT value. Because the storage for dynamic tasks is not preallocated, the kernel might need to use a GETMAIN command to obtain the storage required to attach a dynamic task when the task is attached.

The number of static tasks depends on the current MXT value. There are MXT+1 static tasks. The storage for static tasks is always obtained by GETMAIN from the CICS DSAs. If MXT is lowered, the storage for an excess number of static tasks is freed again.

During early CICS initialization, the kernel allocates storage for eight dynamic tasks. This storage is obtained by GETMAIN from MVS and is always available for use by internal CICS tasks. All other storage for dynamic tasks is then allocated, as needed, from the CICS DSAs. Typically, when a dynamic task ends, its associated storage is freed.

The storage that CICS allocates during task initialization for a single task is the same for a static or dynamic task, as follows:

- A 1576-byte kernel task entry
- A 28K 31-bit stack

The allocated storage is all above the 16 MB line. CICS no longer allocates a 24-bit stack (below the line) for each task during task initialization.

In addition to the storage allocated at task initialization, the kernel also allocates pools of extension stack segments both above and below the 16 MB line.

- The size of each 31-bit extension stack segment (above the line) is 8 KB. Any task can use these extension stack segments if it overflows the 31-bit stack storage allocated to it. CICS preallocates a pool containing a number of 31-bit extension stack segments that is determined by dividing the current MXT value by 10.
- The size of each 24-bit extension stack segment (below the line) is 4 KB. Tasks obtain these extension stack segments whenever they require 24-bit stack storage. CICS preallocates a reserve pool of 24-bit extension stack segments that tasks can use if no other 24-bit stack storage is available.

When the kernel obtains storage using GETMAIN from the CICS DSAs, the following subpools are used:

KESTK24E in the CDSA
4 KB extension stack segments, 24-bit

KESTK31 in the ECDSA
28 KB stack segments, 31-bit

KESTK31E in the ECDSA
8 KB extension stack segments, 31-bit

KETASK in the ECDSA
1576-byte kernel task entries

64-bit MVS storage

64-bit MVS storage is available to the operating system to perform region-related services.

For information about 64-bit (above-the-bar) storage in an address space, see Using the 64-bit Address Space in the z/OS MVS Programming: Extended Addressability Guide.

If you run Java programs in a region, CICS uses the 64-bit JVM on z/OS. 64-bit MVS storage is allocated to each JVM that runs under the control of CICS.

For other CICS facilities that use 64-bit MVS storage, see "CICS facilities that use 64-bit storage" on page 82.

MVS storage below 2 GB

MVS storage below 2 GB is available to the operating system to perform region-related services in response to an operating system macro or SVC issued by the region.

For example, operating system components such as VSAM, DL/I, or DB2 issue MVS GETMAIN requests to obtain storage in which to build control blocks. These requests are met from MVS storage below 2 GB.

MVS storage is the amount of storage that remains after the dynamic storage areas and other CICS storage requirements are met. The size of MVS storage below 2 GB depends on MVS GETMAIN requirements during the execution of CICS. Opening files is the major contributor to usage of this area.

MVS storage below 2 GB is used to contain the following items:

- Control blocks and data areas that are required to open data sets, or for other operating system functions
- Program modules for the access method routines that are not already resident in the link pack area (LPA)
- Shared routines for the COBOL and PL/I programs

There are four major elements of virtual storage in MVS storage below 2 GB. Each storage area below 16 MB is duplicated above 16 MB.

- The common area below 16 MB
- The private area below 16 MB
- The extended common area above 16 MB
- The extended private area above 16 MB

Storage when CICS uses other products

The VSAM buffers and most of the VSAM file control blocks reside above 16 MB. The VSAM buffers might be for CICS data sets defined as using local shared resources (LSR) or nonshared resources (NSR). The VSAM LSR pool is built dynamically above 16 MB when the first file specified as using it is opened, and deleted when the last file using it is closed. Every opened data set requires some amount of storage in this area for such items as input/output blocks (IOBs) and channel programs.

Files that are defined as data tables use storage above 16 MB for records that are included in the table, and for the structures that allow them to be accessed.

Queued sequential access method (QSAM) files require some storage in this area. Transient data uses a separate buffer pool above 16 MB for each type of transient data queue. Storage is obtained from the buffer pool for transient data queue resources as they are installed. Transient data also uses a buffer pool above 16 MB where sections of extrapartition transient data queue definitions are copied for use by QSAM, when an extrapartition queue is being opened or closed.

CICS DBCTL uses DBCTL threads. DBCTL threads are specified in the CICS address space but they have storage requirements in the high private area of the CICS address space. If CICS uses DB2, MVS storage is allocated for each DB2 thread.

MVS storage limits

The physical placement of the MVS storage below 2 GB can be anywhere in the region, and might sometimes be above the CICS region. The region might expand into this MVS storage area, above the region, up to the IEALIMIT set by the installation or up to the default value. For more information about IEALIMIT, see z/OS MVS Installation Exits. This expansion occurs when operating system GETMAIN requests are issued, the MVS storage in the region is exhausted, and the requests are met from the MVS storage area above the region.

When both the MVS storage areas below 2 GB are exhausted, the GETMAIN request fails, causing abends or a bad return code if it is a conditional request.

The amount of MVS storage below 2 GB must be enough to satisfy the requests for storage during the entire execution of the CICS region. You must use caution; you never want to run out of MVS storage, but you also do not want to allocate too much.

The size of MVS storage below 2 GB is the storage that remains in the region after allowing for the storage required for the dynamic storage areas, the kernel storage areas, and the IMS/VS and DBRC module storage. It is important to specify the correct DSA sizes so that the required amount of MVS storage is available in the region.

Because of the dynamic nature of a CICS system, the demands on MVS storage varies through the day, that is, as the number of tasks increases or data sets are opened and closed. Also, because of this dynamic use of MVS storage, fragmentation occurs, and you must allocate additional storage to compensate for this.

The MVS common area
The MVS common area contains a number of nucleus, queue, link pack, common service, and storage areas.

The following areas comprise the MVS common area:

- Nucleus and extended nucleus
- System queue area (SQA and ESQA)
- Link pack areas (PLPA, MLPA, and CLPA)
- Common service areas (CSA and ECSA)
- Prefixed storage area (PSA).

All these elements of the common area, except the PSA, are duplicated above 16 MB.

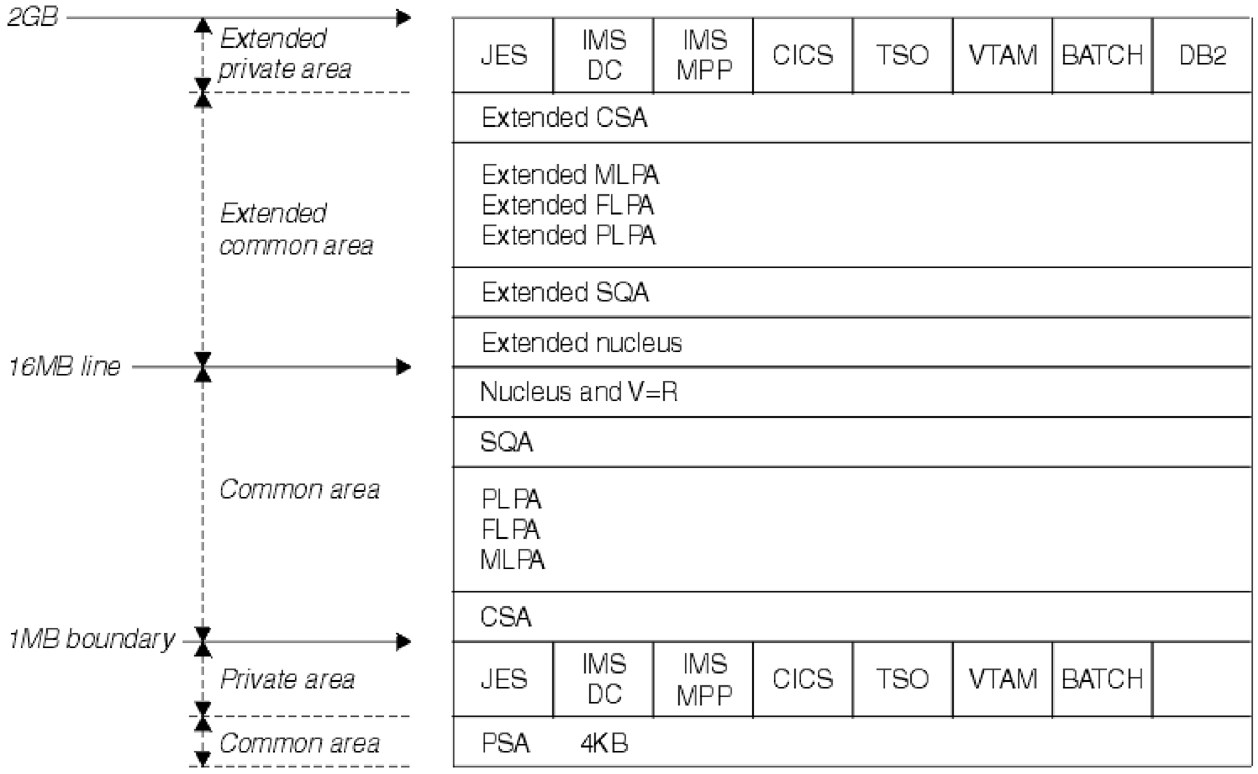


Figure 17. Virtual storage map

MVS nucleus and MVS extended nucleus
The MVS nucleus and MVS extended nucleus is a static area that contains the nucleus load module and extension to the nucleus. Although its size is variable depending on the configuration of an installation, it cannot change without a re-IPL of MVS.

The nucleus area below 16 MB does not include page frame table entries, and the size of the nucleus area is rounded up to a 4 KB boundary. In addition, the nucleus area is positioned at the top of the 16 MB map while the extended nucleus is positioned just above 16 MB.

System queue area (SQA) and extended system queue area (ESQA)
This area contains tables and queues relating to the entire system. Its contents are highly dependent on configuration and job requirements at an installation.

The total amount of virtual storage, number of private virtual storage address spaces, and size of the installation performance specification table are some of the factors that affect the system's use of SQA. The size of the initial allocation of SQA is rounded up to a 64 KB boundary, though SQA may expand into the common system area (CSA) in increments of 4 KB.

If the SQA is overallocated, the virtual storage is permanently wasted. If it is underallocated, it expands into CSA, if required. In a storage constrained system, it is better to be slightly underallocated. This can be determined by looking at the amount of free storage. If the extended SQA is underallocated, it expands into the extended CSA. When both the extended SQA and extended CSA are used up, the system allocates storage from SQA and CSA below the 16 MB line. The allocation of this storage could eventually lead to a system failure, so it is better to overallocate extended SQA and extended CSA.

Link pack area (LPA) and extended link pack area (ELPA)
The link pack area (LPA) contains all the common reentrant modules that are shared by the system.

The link pack area (LPA) can provide the following:

- Economy of real storage by sharing one copy of the modules
- Protection: LPA code cannot be overwritten, even by key 0 programs
- Reduced path length, because modules can be branched to.

It has been established that a 2 MB LPA is sufficient for MVS when using CICS with MRO or ISC, that is, the size of an unmodified LPA as shipped by IBM. If it is larger, there are load modules in the LPA that might be of no benefit to CICS. There might be SORT, COBOL, ISPF, and other modules that are benefiting batch and TSO users. You must evaluate whether the benefits you obtain are worth the virtual storage that they use. If modules are removed, check whether you need to increase the size of the regions they run in to accommodate them.

The pageable link pack area (PLPA) contains supervisor call routines (SVCs), access methods, and other read-only system programs, along with read-only re-enterable user programs selected by an installation to be shared among users of the system. Optional functions or devices selected by an installation during system generation add additional modules to the PLPA.

The modified link pack area (MLPA) contains modules that are an extension to the PLPA. The MLPA can be changed at IPL without requiring the create link pack area (CLPA) option at IPL to change modules in the PLPA.

Common service area (CSA) and extended common service area (ECSA)
The CSA and ECSA contain pageable system data areas that are addressable by all active virtual storage address spaces.

These service areas contain, for example, buffers or executable modules for IMS, ACF/SNA, and JES3. CSA and ECSA also contain control blocks that are used to define subsystems and provide working storage for areas such as TSO input/output control (TIOC), event notification facility (ENF), and message processing facility (MPF). When system configuration and activity increases, the storage requirements also increase.

CICS uses the ECSA for multiregion operation (MRO) to store control blocks only and not for data transfer. If cross-memory facilities are used, the ECSA usage is limited to the following amounts:

- 40 bytes per session if IRC (interregion communication) is open, irrespective of whether the resource is acquired and inservice, or released
- 4 KB per address space participating in MRO

In addition, the amount of storage used by CICS MRO for interregion buffers is detailed in the DFHIR3794 message issued to the CSMT destination at termination.

CICS also uses ECSA for IMS and shared data tables.

For static systems, the amount of unallocated CSA should be around 10% of the total allocated CSA; for dynamic systems, a value of 20% is optimal. Unlike the SQA, if CSA is depleted there is no scope for expansion and a re-IPL might be required.

Prefixed storage area (PSA)
The PSA contains processor-dependent status information such as program status words (PSWs). There is one PSA per processor; however, all of them map to virtual storage locations 0 KB to 4 KB as seen by that particular processor.

MVS treats this as a separate area; there is no PSA in the extended common area.

Private area and extended private area
The portion of the user private area in each virtual address space that is available to the user's application program is referred to as its *region*. Except for the 16 KB system region area, each storage area in the private area has a counterpart in the extended private area.

The private area contains the following areas:

- A local system queue area (LSQA)
- A scheduler work area (SWA)
- Subpools 229 and 230 (the requestor protect key area)
- A 16 KB system region area (used by the initiator)
- A user region for running programs and storing data.

See the virtual storage map for MVS in Figure 17 on page 113.

The private area user region can be any size up to the size of the entire private area (from the top end of the prefixed storage area (PSA) to the beginning, or bottom end, of the common service area (CSA)) **minus** the size of LSQA, SWA, subpools 229 and 230, and the system region: for example, 220 KB. It is recommended that the region is 420 KB less to allow for recovery termination management (RTM) processing.

The segment sizes are one megabyte, therefore CSA is rounded up to the nearest megabyte. The private area is in increments of one megabyte.

High private area
The area at the high end of the address space is not specifically used by CICS, but contains information and control blocks that the operating system needs to support the region and its requirements.

The high private area consists of four areas:

- LSQA
- SWA
- Subpool 229
- Subpool 230

The usual size of the high private area varies with the number of job control statements, messages to the system log, and number of opened data sets.

The total space used in this area is reported in the IEF374I message in the field labeled SYS=nnnnK at jobstep termination. A second SYS=nnnnK is issued, which refers to the high private area above 16 MB. This information is also reported in the sample statistics program, DFH0STAT.

You cannot reduce the size of this area, except possibly subpool 229. This subpool is where the z/OS Communications Server stores inbound messages when CICS does not have an open receive issued to the z/OS Communications Server. To determine whether this is happening, use CICS statistics obtained following CICS shutdown. Compare the maximum number of RPLs that are posted in the shutdown statistics with the RAPOOL value in the SIT. If these values are equal, subpool 229 is probably being used to stage messages, and the RAPOOL value should be increased.

In some situations, the way in which the storage in the high private area is used might cause an S80A abend. There are at least two considerations:

- The use of MVS subpools 229 and 230 by access methods such as SNA.

  SNA and VSAM might find insufficient storage for a request for subpools 229 and 230. Their requests are conditional and so should not cause an S80A abend of the job step (for example, CICS).
- The MVS operating system itself, relative to use of LSQA and SWA storage during job-step initiation.

  The MVS initiator's use of LSQA and SWA storage can vary, depending on whether CICS was started using an MVS START command, or started as a job step as part of already existing initiator and address space. Starting CICS with an MVS START command is better to minimize fragmentation in the space above the region boundary. If CICS is a job step initiated in a previously started initiator's address space, the way in which LSQA and SWA storage is allocated might reduce the apparently available virtual storage because of increased fragmentation.

Storage above the region boundary must be available for use by the MVS initiator (LSQA and SWA) and the access method (subpools 229 and 230).

Consider initiating CICS using an MVS START command, to minimize fragmentation of the space above your specified region size. The more effective use of the available storage might avoid S80A abends.

Your choice of sizes for the MVS nucleus, MVS common system area, and CICS region influences the amount of storage available for LSQA, SWA, and subpools 229 and 230. It is unlikely that the sizes and boundaries for the MVS nucleus and common system area can be changed easily. To create more space for the LSQA, SWA, and subpools 229 and 230, you might need to **decrease** the region size.

For more information about subpools and managing private storage allocation, see Virtual storage management in z/OS MVS Programming: Authorized Assembler Services Guide.

Local system queue area (LSQA)
This area generally contains the control blocks for storage and contents supervision. Depending on the release level of the operating system, it can contain subpools 233, 234, 235, 253, 254, and 255.

The total size of LSQA is difficult to calculate because it depends on the number of loaded programs, tasks, and the number and size of the other subpools in the address space. As a guideline, the LSQA area usually runs between 40 KB and 170 KB, depending on the complexity of the rest of the CICS address space.

The storage control blocks define the storage subpools in the private area, describing the free and allocated areas within those subpools. They can consist of such things as subpool queue elements (SPQEs), descriptor queue elements (DQEs), and free queue elements (FQEs).

The contents management control blocks define the tasks and programs in the address space, such as task control blocks (TCBs), the various forms of request blocks (RBs), contents directory elements (CDEs), and many more.

CICS DBCTL requires LSQA storage for DBCTL threads. Allow 9 KB for every DBCTL thread, up to the **MAXTHRED** value.

Scheduler work area (SWA)
The scheduler work area (SWA) is made up of subpools 236 and 237, which contain information about the job and step itself. Almost anything that appears in the job stream for the step creates some kind of control block here.

Generally, this area can be considered to increase with an increase in the number of DD statements. The distribution of storage in subpools 236 and 237 varies with the operating system release and whether dynamic allocation is used. The total amount of storage in these subpools starts at 100 to 150 KB, and increases by about 1 to 1.5 KB per allocated data set.

A subset of SWA control blocks can, optionally, reside above 16 MB. JES2 and JES3 have parameters that control this. If this needs to be done on an individual job basis, the SMF exit, IEFUJV, can be used.

Subpool 229
This subpool is used primarily for the staging of messages. JES uses this area for messages to be printed on the system log and JCL messages as well as SYSIN/SYSOUT buffers.

Generally, a value of 40 KB to 100 KB is acceptable, depending on the number of SYSIN and SYSOUT data sets and the number of messages in the system log.

Subpool 230
This subpool is used by the z/OS Communications Server for inbound message assembly for segmented messages. Data management keeps data extent blocks (DEBs) here for any opened data set.

Generally, the size of subpool 230 increases as the number of opened data sets increases. Starting with an initial value of 40 KB to 50 KB, allow 300 to 400 bytes per opened data set.

CICS DBCTL requires subpool 230 storage for DBCTL threads. Allow 3 KB for every DBCTL thread, up to the **MAXTHRED** value.

MVS storage above region
MVS storage above region is the storage that is left between the top of the region and the bottom of the high private area. Usually 200 KB to 300 KB of free storage is maintained to allow for use by the termination routines if these is an abend.

If this free storage is not enough for recovery termination management (RTM) processing, the address space might be terminated with a S40D abend that does not produce a dump.

This area can be very dynamic. As the high private area grows, it extends down into this area, and the CICS region can extend up into this area up to the value specified in **IEALIMIT**.

Splitting online systems: virtual storage

To increase the virtual storage available to a CICS system, you can split the system into two or more separate address spaces. Splitting a system can also provide higher availability, and you can use multiprocessor complexes to the best advantage because a system can then operate on each processor concurrently. Most CICS systems can be split.

To tune CICS to get more virtual storage, you must first tune MVS and then CICS. If, after you have tuned MVS common virtual storage, you still cannot run CICS in a single address space, you must then consider splitting the CICS workload into multiple address spaces. The new address spaces require more real storage, but the potential savings in virtual storage from splitting CICS regions are significant. You can split a CICS system by application function, by CICS function (such as a file owning or terminal owning region), or by a combination of the two functions.

Many installations find it convenient to split their CICS workloads into multiple independent address spaces, where the workload is easily definable and no resource sharing is required. If you can readily isolate application subsystems and their associated terminals, programs, and data sets, it is reasonable to split a single CICS address space into two or more independent address spaces. They become autonomous regions with no interactions.

If you can split a CICS system completely, with no communication required between the two parts, you reduce overheads and planning. If the new systems must share data, programs, or terminals, you can use CICS intercommunication. You can use IPIC (IP interconnectivity) connections, ISC over SNA (intersystem communication over SNA) connections, or MRO (multiregion operation) to connect CICS regions to each other. For descriptions of the CICS intercommunication methods and the facilities that are available with each method, such as transaction routing and function shipping, see Introduction to CICS intercommunication.

You can also consider creating additional copies of a CICS region, and using CICS intercommunication to provide transaction routing between them. If additional virtual storage is needed, it is reasonable, for example, to split the AOR into two or more additional CICS copies. When you split the system either partially or completely, you can reduce the amount of virtual storage needed for each region by removing any unused resident programs. Removing unused programs reduces the size of the relevant DSA.

CICS intercommunication uses additional processor cycles, and it can affect response time as well as processor time. The cost of intercommunication varies depending on the connection type (IPIC, MRO, or ISC over SNA), and on the intercommunication facilities that you use over that connection. For information about the performance considerations for different intercommunication methods and facilities, see "CICS MRO, ISC, and IPIC: performance and tuning" on page 138.

You might have to adjust certain parameters, such as **MXT**, when CICS systems are split. In an MRO system with function shipping, tasks of longer duration might also require further adjustment of **MXT** and other parameters (for example, file string numbers, virtual storage allocation).

If you plan to use MRO, consider sharing CICS code or application code using the MVS link pack area (LPA). Note that the LPA saves real storage, not virtual storage, and other non-CICS address spaces. Use of LPA for the eligible modules in CICS is controlled by the system initialization parameter **LPA=YES**, which tells CICS to search for the modules in the LPA. For further information about the use of the LPA, see "Using modules in the link pack area (LPA/ELPA)" on page 117.

Using modules in the link pack area (LPA/ELPA)

Some CICS management and user modules can be moved into the link pack area (LPA) or the extended link pack area (ELPA). For systems running multiple copies of CICS, this can allow those multiple copies to share the same set of CICS management code.

There are a number of benefits of placing code in the LPA or ELPA:

- The code is protected from possible corruption by user applications. Because the LPA or ELPA is in protected storage, it is virtually impossible to modify the contents of these programs.
- Performance can be improved and the demand for real storage reduced if you use the LPA or ELPA for program modules. If more than one copy of the same release of CICS is running in multiple address

spaces of the same processor, each address space requires access to the CICS nucleus modules. These modules may either be loaded into each of the address spaces or shared in the LPA or ELPA. If they are shared in the LPA or ELPA, this can reduce the working set and therefore, the demand for real storage (paging).

- You can decrease the storage requirement in the private area by judicious allocation of the unused storage in the LPA or ELPA created by rounding to the next segment.

Putting modules in the LPA or ELPA requires an IPL of the operating system. Maintenance requirements should also be considered. If test and production systems are sharing LPA or ELPA modules, you might want to run the test system without the LPA or ELPA modules when new maintenance is being tested.

The disadvantage of placing too many modules in the LPA (but not the ELPA) is that it can become excessively large. Because the boundary between the CSA and the private area is on a segment boundary, this means that the boundary may move down one megabyte. The size of the ELPA is not usually a problem.

Use the SMP/E USERMOD called LPAUMOD to select those modules that you want to use for the LPA. This indicates the modules that are eligible for LPA or ELPA. You can use this USERMOD to move the modules into your LPA library. All users with multiple CICS address spaces should put all eligible modules in the ELPA.

LPA=YES must be specified in the system initialization table (SIT). Specifying LPA=NO allows you to test a system with new versions of CICS programs (for example, a new release) before moving the code to the production system. The production system can then continue to use modules from the LPA while you are testing the new versions.

An additional control, the PRVMOD system initialization parameter, enables you to exclude particular modules explicitly from use in the LPA.

For information on installing modules in the LPA, see Installing CICS modules in the MVS link pack area.

Selecting aligned or unaligned maps

CICS maps that are used by basic mapping support (BMS) can be defined as aligned or unaligned. In aligned maps, the length field associated with a BMS data field in the BMS DSECT is always aligned on a halfword boundary. In unaligned maps, the length field follows on immediately from the preceding data field in the map DSECT. An aligned map is compiled with the AMAP option, and an unaligned one is compiled with the MAP option. A combination of aligned and unaligned maps can be used.

In unaligned maps, there is no guarantee that the length fields in the BMS DSECT are halfword-aligned. Some COBOL and PL/I Compilers, in this case, generate extra code in the program, copying the contents of any such length field to, or from, a halfword-aligned work area when its contents are referenced or changed.

Specifying map alignment increases the size of the BMS DSECT, at worst by one padding byte per map data field, and marginally increases the internal path length of BMS in processing the map. The best approach, therefore, is to use unaligned maps, except where the compiler being used would generate inefficient application program code.

In COBOL, an unaligned map generates an unsynchronized structure. In PL/I, an unaligned map generates a map DSECT definition as an unaligned structure. Correspondingly, aligned maps produce synchronized structures in COBOL and aligned structures in PL/I.

In CICS, BMS maps are always generated in groups (map sets). An entire map set must be defined as aligned or unaligned. Also, maps can be used by application programs written in various languages. In these cases, it is important to select the option that best suits the combination of programs and, if there is any requirement for both aligned and unaligned maps, select the ALIGNED option.

Avoid converting maps, for example, from aligned to unaligned, because changing the map DSECT also requires reassembly or recompilation of all application programs that reference it.

Map alignment is defined when maps are assembled. Aligned maps use the SYSPARM(A) option. The BMS=ALIGN/UNALIGN system initialization parameter defines which type of map is being used.

The map and map set alignment option can also be specified when maps and map sets are defined using the screen definition facility (SDF II) licensed program.

The importance of map alignment is demonstrated by inspecting programs that handle screens with many fields. Try recompiling the program when the BMS DSECT is generated first without, and then with, the map alignment option. If the program size, as indicated in the linkage edit map, drops significantly in the second case, use aligned maps where possible.

Defining programs as resident, nonresident, or transient

Programs, map sets, and partition sets can be defined as RESIDENT(NO|YES) and USAGE(NORMAL|TRANSIENT). Programs can be defined as RELOAD(NO|YES).

Any program defined in the CSD is loaded into the CDSA, RDSA, SDSA, ECDSA, ERDSA, or ESDSA on first usage. RELOAD(YES) programs cannot be shared or reused. A program with RELOAD(YES) defined is only removed following an explicit EXEC CICS FREEMAIN. USAGE(TRANSIENT) programs can be shared, but are deleted when the use count falls to zero. RESIDENT(NO) programs become eligible for deletion when the use count falls to zero. The CICS loader domain progressively deletes these programs as DSA storage becomes constrained, deleting first the programs that are used infrequently.

RESIDENT(YES) programs are not normally deleted. If NEWCOPY runs for any program, a new copy is loaded and used on the next reference and the old copy becomes eligible for deletion when its use count falls to zero.

On a CICS warm start, an initial free area for the various resident program subpools is allocated. The size of this area is based on the total lengths of all currently loaded resident programs as recorded during the preceding CICS shutdown. When a resident program is loaded, CICS attempts to fit it into the initial free area. If it does not fit, it is loaded outside the initial free area, and the space inside the initial free area remains deallocated until other (smaller) resident programs are loaded into it. This situation can occur if a resident program has increased its size since it was last loaded (before the last CICS shutdown). If the program in question is large, storage problems can occur because of the large amount of unused storage in the initial free area allocated for resident programs.

Because programs that are not in use are deleted on a least-recently-used (LRU) basis, define these programs as RESIDENT(NO) unless there are particular reasons to favor particular programs by keeping them permanently resident. Variations in program usage over time are automatically taken account of by the LRU algorithm.

Therefore, a much-used nonresident program is likely to remain resident anyway, while during periods of light usage, a resident program might be wasting the virtual storage it permanently occupies.

For programs written to run above the 16 MB line, specify EDSALIM large enough such that virtual storage is not a constraint.

If a program is large or frequently updated such that its size increases, consider defining it as non-resident and issuing a LOAD with the HOLD option as part of PLTPI processing.

You might define a program as RESIDENT for one of the following reasons:

- To avoid storage fragmentation, because all such programs are in a single block of storage (but not new copies of programs).
- For programs that deal with potential crises (for example, CEMT).
- Where there is heavy contention on the DFHRPL or dynamic program LIBRARYs. However, contention is usually handled by data set placement or other DASD tuning, or with use of MVS library lookaside to maintain program copies in an MVS dataspace.

Putting application programs above 16 MB

CICS keeps RMODE(ANY) application programs in the EDSA, which is in MVS extended virtual storage above 16 MB but below 2 GB (above the line). Work areas associated with the programs can also reside above the line.

It is possible to LINK or XCTL between 64-bit, 31-bit, and 24-bit addressing mode (AMODE) programs. You can convert programs to 31-bit or 64-bit addressing mode programs and move them above 16 MB but below 2 GB to the extended private area. Moving programs above 16 MB but below 2 GB frees that amount of virtual storage below 16 MB for other use.

See "Using modules in the link pack area (LPA/ELPA)" on page 117 for information about using programs from the LPA or extended link pack area (ELPA).

Using the ELPA is better than using the extended private area when multiple address spaces are employed, because the program is already loaded when CICS needs it, and real-storage usage is minimized.

When running a CICS system that has transaction isolation enabled, performance benefits can be gained by moving transactions and application programs above the line. Program work areas are then obtained from the EUDSA, which has a 1 MB page size, rather than the UDSA, which has a 4 KB page size. This facility is useful where there is demand for virtual storage up to the 16 MB line and there is sufficient real storage. Because the reason for using virtual storage above the line is to make the space below 16 MB available for other purposes, there is an overall increase in the demand for real storage when programs are moved above 16 MB but below 2 GB.

When a COMMAREA is passed between programs running in different addressing modes, the following restrictions apply:

- A COMMAREA passed from an AMODE(31) program to an AMODE(24) program must be able to be processed by the AMODE(24) program, therefore it must not contain 31-bit addresses.
- A COMMAREA passed from an AMODE(64) program to an AMODE(31) program must be able to be processed by the AMODE(31) program, therefore it must not contain 64-bit addresses.
- A COMMAREA passed from an AMODE(64) program to an AMODE(24) program must be able to be processed by the AMODE(24) program, therefore it must not contain 64-bit or 31-bit addresses.

Programs that are to reside above the 16 MB line must be link-edited with the AMODE(31),RMODE(ANY) options on the MODE statement of the link-edit.

Allocation of real storage when using transaction isolation

When transaction isolation is active, there is a cost in terms of real storage. If insufficient real storage is allocated, paging problems can result, which then affect performance. The cost depends on the number of subspaces in use in the system, and the size of the **EDSALIM** parameter.

Because the page size of the EUDSA is 1 MB, the value of **EDSALIM** is likely to be very large for a CICS system that has transaction isolation active. This virtual storage needs to be mapped with page and segment tables using real storage, so an increase in the real storage usage can occur. In addition to the real storage used to map the virtual storage for the **EDSALIM** value, subspaces also require real storage. For example:

- Each subspace requires 2.5 pages, where a page means a 4 KB page of real storage.
- Assuming that each transaction in the system requires a unique subspace, (transaction definition TASKDATAKEY(USER) and ISOLATE(YES)), real storage required is the **MXT** value x 2.5 pages.
- If each transaction in the system requires a page of storage in the EUDSA (1 MB page), a page table is required to map the storage. Real storage is the **MXT** value x 1 page.
- A further three pages are required, so the total of real storage is the **MXT** value x (1 + 2.5 pages) + 3 pages.
- All of this real storage is allocated from the ELSQA.

The figures for the real storage usage is in addition to that required for a CICS system that does not have transaction isolation active. The CICS requirement for real storage varies depending on the transaction load at any one time. As a guideline, each task in the system requires 9 KB of real storage. Multiply this number by the number of concurrent tasks that can be in the system at any one time (governed by the **MXT** system initialization parameter).

Limiting the expansion of subpool 229 using SNA pacing

Subpool 229 can be expanded if batch type terminals send data faster than a CICS transaction can process that data. The use of secondary to primary pacing, sometimes called inbound pacing, limits the amount of data queued in subpool 229 for any given batch terminal. The PACING parameter controls the flow of traffic from the network control program (NCP) to the terminal and does not affect the processor activity as such. The VPACING parameter controls the flow of traffic between the host and the NCP.

The VPACING parameter of the CICS APPL statement determines how many messages can be sent in a session to the z/OS Communications Server application program by another SNA logical unit without requiring that an acknowledgment (a pacing response) is returned. The host sends data path information units (PIUs) according to the definition of the VPACING parameter. The first PIU in a group carries a pacing indicator in the RH. When this PIU is processed by the NCP, the NCP sends a response to the host with the same pacing indicator set to request a new pacing group so that, for every *x* PIUs to a terminal and every *y* PIUs to a printer, the pacing response traffic must flow from the NCP to the host which, based on the volume of traffic, might cause a significant increase in host activity.

Normally, the VPACING parameter is implemented when a shortage of NCP buffers requires controlling the volume of flow between the host and the NCP. You can lessen the effect on the processor by increasing the VPACING parameter to a value that the NCP can tolerate.

The PACING parameter is required for most printers, to match the buffer capacity with the speed of printing the received data. Terminals do not normally require pacing unless there is a requirement to limit huge amounts of data to one LU, as is the case with some graphics applications. Use of pacing to terminals causes response time degradation. The combination of the PACING and VPACING parameters causes both response time degradation and increased processor activity, and increased network traffic.

Specify the PACING and VPACING parameters for all terminals to prevent a “runaway” transaction from flooding the SNA network with messages and requiring large amounts of buffer storage. If a transaction loops while issuing SEND commands to a terminal, IOBUF (CSA storage) and NCP buffers can fill up causing slowdowns and CSA shortage conditions.

Specify the PACING and VPACING parameters high enough so that normal data traffic can flow without being regulated, but excessive amounts of data are prevented from entering the network and impairing the normal flow of data.

For secondary to primary pacing, you must code in the following way:

- SSNDPAC=nonzero value in the LOGMODE entry pointed to by the secondary application program
- VPACING=nonzero value on the APPL definition for the secondary application.

The value used is coded on the VPACING parameter. If either of these values are zero, no pacing occurs.

Specify VPACING on the APPL statement defining the CICS region, and any nonzero value for the SSNDPAC parameter on the LU statement defining the batch device. Ensure that the device supports this form of pacing as specified in the component description manual for that device.

CICS storage protection facilities: Performance and tuning

The facilities that are related to storage protection are storage protection, transaction isolation, and command protection. These facilities protect storage from user application code.

Storage protection
Protects CICS code and control blocks from being overwritten accidentally by user applications.

Transaction isolation
Offers protection against transaction data being overwritten accidentally by other user transactions.

Command protection

Ensures that an application program does not pass storage to CICS using the EXEC CICS interface, which requires updating by CICS, although the application itself cannot update the storage.

Storage protection, transaction isolation, and command protection protect storage from user application code. They add no benefit to a region where no user code is executed; that is, a pure terminal-owning region (TOR) or a pure file-owning region (FOR) (where no distributed program link (DPL) requests are function-shipped).

Transaction isolation and applications

When using transaction isolation, it is necessary to *activate* pages of storage to the allocated subspace of the task. Before the storage is activated to the subspace, it is fetch protected so that the task cannot access the storage. After the storage is activated to the subspace allocated to the task, the task has read and write access to the storage. CICS must activate user storage to a subspace every time that the user task invokes a GETMAIN command to get a new page of user-key task-lifetime storage. Some performance cost is involved when activating storage to a subspace, so the activity should be kept to a minimum.

Storage below the 16 MB line is activated in multiples of 4 KB. Storage above the line is activated in multiples of 1 MB. So a user task that runs completely above the line is more likely to require only one activate operation.

Link edit your programs by using RMODE(ANY) and define them as DATALOCATION(ANY). All transactions should be defined as TASKDATALOC(ANY), thus reducing the number of storage activations.

When you need to obtain storage below the line, you can improve performance by obtaining all the storage in one GETMAIN request, rather than several smaller GETMAIN requests. This also minimizes the number of storage activate operations.

For more information, see MVS subspaces.

Tuning with Language Environment

When you run with Language Environment on CICS, there are several tuning actions you can take to optimize performance. If Language Environment is active in a CICS address space, the runtime libraries of the native language, such as COBOL or PL/I, are not needed. This means that CICS has a single interface to all the language run times.

For more information about Language Environment, see Programming languages and Language Environment.

Minimizing GETMAIN and FREEMAIN activity

One way to improve performance when you run programs with Language Environment is to reduce the number of GETMAIN and FREEMAIN requests required to manage the storage that Language Environment uses.

You can use the following system initialization parameters to minimize the number of GETMAIN and FREEMAIN requests that CICS performs on behalf of Language Environment:

- AUTODST
- RUWAPOOL

You can use these two options together in any combination.

To check the benefit of using these functions, run a CICS storage report to show the number of GETMAIN and FREEMAIN request in a region when either or both of the functions are active, and compare the results with previous runs.

AUTODST: Language Environment automatic storage tuning

You can optionally activate Language Environment's automatic storage tuning feature for CICS by setting the CICS system initialization parameter AUTODST to YES. When this function is active, Language

Environment monitors each main program execution, and notes if any additional storage had to be allocated for the program while it was active.

At the end of each program execution, if any additional storage had to be allocated, Language Environment retains this information. Next time the program is executed, Language Environment increases the initial storage allocation to include this extra storage. This process helps to minimize the number of GETMAIN and FREEMAIN requests that CICS has to perform.

Automatic storage tuning is particularly helpful for programs that issue many dynamic calls, as such programs can easily exceed their initial storage allocations. It also removes the need to tune storage manually for individual COBOL programs.

However, you should be aware that once Language Environment has increased the initial storage allocation for a program, it is never decreased. If a program execution requires an unusually large amount of storage, perhaps because the user has activated a seldom-used function of the program, this amount of storage is allocated for all subsequent executions of the program. So in rare cases, you can find that automatic storage tuning leads to an excessive allocation of storage for some programs.

You can alter the behavior of the automatic storage tuning mechanism using the Language Environment storage tuning user exit CEECSTX. The user exit can enable or disable automatic storage tuning for a particular program, and you might find this useful if you have an application whose storage needs vary greatly between different executions. It can also provide the starting values for initial storage allocation, and you can use it to limit the maximum amount of storage that Language Environment will allocate during the automatic storage tuning process.

If the CEECSTX user exit was previously used as your Language Environment storage tuning method, you might find that the automatic storage tuning mechanism provides the same function, without the user exit. You need to decide which mechanism to use as your main storage tuning method, because when you are running CICS with automatic storage tuning, the CEECSTX user exit has limited function. Automatic storage tuning operates by monitoring storage allocations, whereas the storage tuning user exit CEECSTX monitors the actual storage used by the user application program. Despite this, automatic storage tuning incurs less overhead than the tuning method based on the CEECSTX exit. Also, automatic storage tuning provides tuning for each initial program invoked by a transaction, while the CEECSTX exit provides tuning for only those programs contained in the table that the exit uses as its input. This means that automatic storage tuning can provide a greater benefit by tuning the storage used by more programs.

For more information about CEECSTX, see z/OS Language Environment Customization.

RUWAPOOL: Run-unit work area pools

The system pathlength increases when a CICS application invoked by Language Environment issues an **EXEC CICS LINK** request. Repeated **EXEC CICS LINK** calls to the same program invoked by Language Environment result in multiple GETMAIN and FREEMAIN requests for run-unit work areas (RUWAs).

Using the system initialization parameter RUWAPOOL(YES) results in the creation of a run-unit work area pool during task initialization. This pool is used to allocate RUWAs required by programs invoked by Language Environment. This reduces the number of GETMAIN and FREEMAIN requests in tasks that perform many **EXEC CICS LINKS** to programs invoked by Language Environment.

For more information about the **RUWAPOOL** system initialization parameter, see RUWAPOOL.

Language Environment run time options for AMODE (24) programs

The default Language Environment runtime options for CICS are ALL31(ON) and STACK(ANY). This means all programs that require Language Environment must be capable of addressing 31-bit storage, that is, must be AMODE(31), when Language Environment is enabled.

For AMODE(24) programs to run in a Language Environment-enabled CICS region, you can specify ALL31(OFF) and STACK(BELOW) for those programs that must run below the 16 MB line. However, if you change these options globally so that all programs use them, large amounts of storage will be allocated below 16 MB, which might cause a short-on-storage condition. When the ALL31(OFF) option is used, Language Environment acquires some control blocks, for example the RUWA, both above and below the

16 MB line, and so additional GETMAIN and FREEMAIN requests are needed to manage the duplicate control blocks.

You do not need to specify ALL31(OFF) as long as the program in question is the **initial** program invoked by a transaction, because Language Environment acquires storage for the enclave (program) in the correct addressing mode automatically. The exception is an AMODE(31) program that dynamically calls an AMODE(24) program. In that situation, the dynamically called AMODE(24) program needs to specify ALL31(OFF).

Using DLLs in C++

When each dynamic link library (DLL) is first loaded, the cost of initialization can be determined by the size of writable static area required by the DLL. Initialization costs can be reduced by removing unnecessary items from the writable static area.

When using DLLs, you should consider the following:

- Specifying the #pragma variable (x,NORENT). This places some read-only variables such as tables in the code area.
- Specifying #pragma strings(readonly). This works for C code whose default is that literal strings are modifiable. C++ already has literal strings as read only by default.
- Examine the prelinker map to determine the large areas. If you find, for example, @STATICC, you have unnamed writable static objects such as strings or static variables.

Minimizing the time Language Environment spends writing dump output to transient data queue CESE

The Language Environment runtime option TERMTHDACT controls the type and amount of diagnostic output produced by Language Environment for an unhandled error.

Using TERMTHDACT(DUMP), TERMTHDACT(TRACE), TERMTHDACT(UADUMP), or TERMTHDACT(UATRACE) can create a significant overhead in a production environment. These settings can cause large amounts of traceback and Language Environment dump data to be written to the CESE transient data queue.

If a traceback or CEEDUMP is not needed by the application environment, use TERMTHDACT(MSG) to eliminate the performance overhead of writing formatted CEEDUMPs to the CICS transient data queue CESE. If the traceback or CEEDUMP is required by the application, specify the CICSDDS option of TERMTHDACT to direct the Language Environment diagnostic output to the CICS dump data set, rather than to the CESE transient data queue.

Java applications: performance and tuning

You can improve the performance of your Java applications and the JVMs in which they run by analyzing and tuning your CICS regions.

For more information about improving the performance of your Java applications, see Improving Java performance.

For more information about using CICS statistics to manage and tune the Java workloads running in your CICS regions, see JVM server statistics.

MVS and DASD: performance and tuning

Tuning CICS for virtual storage under MVS depends on several elements: z/OS systems tuning, z/OS Communications Server SNA tuning, CICS tuning, and VSAM tuning. Because tuning is a top-down activity, ensure that you have already made a vigorous effort to tune z/OS before tuning CICS.

Your main effort to reduce virtual storage constraint and to get relief is concentrated on reducing the life of the various individual transactions; in other words, shortening task life. Upgrading your z Systems hardware can be a fast path to shortening task life:

- The installation of a faster processor can cause the current instructions to be executed faster and, therefore, reduce task life (internal response time), because more transactions can be processed in the same period.
- Additional real storage, if page-ins are frequently occurring (if there are more than 5 to 10 page-ins per second, CICS performance is affected), can reduce waits for the paging subsystem.
- Installing faster DASD can reduce the time spent waiting for I/O completion, and this shorter wait time for paging operations, data set index retrieval, or data set buffer retrieval can also reduce task life in the processor.

Look for the following indicators in your z/OS system to see if you have a problem with I/O specifically:

- Service level objectives are missed.
- Users complain about response times.
- I/O indicators show signs of stress, or you see high DEV DLY or USG for an important workload directly in Monitor III reports.

For more information, see the section about analyzing I/O activity in z/OS Resource Measurement Facility (RMF) Report Analysis.

MVS provides storage isolation for an MVS performance group, which allows you to reserve a specific range of real storage for the CICS address space and to control the page-rates for that address space based on the task control block (TCB) time absorbed by the CICS address space during execution.

You can isolate CICS data on DASD drives, strings, and channels to minimize the I/O contention suffered by CICS from other DASD activity in the system. Few CICS online systems generate enough I/O activity to affect the performance of CICS seriously if DASD is isolated in this manner.

So far (except when describing storage isolation and DASD sharing), we have concentrated on CICS systems that run a stand-alone single CICS address space. The sizes of all MVS address spaces are defined by the common requirements of the largest subsystem. If you want to combine the workload from two or more processors onto an MVS image, you must be aware of the virtual storage requirements of each of the subsystems that are to execute on the single-image processor. (For an overall description of virtual storage, see “CICS virtual storage” on page 69.) Review the virtual storage effects of combining the following kinds of workload on a single-image MVS system:

1. CICS and a large number (100 or more) of TSO users
2. CICS and a large IMS system
3. CICS and 5000 - 7500 SNA LU.

By its nature, CICS requires a large private region that might not be available when the common requirements of the large system on these other subsystems are satisfied. If, after tuning the operating system, SNA, VSAM, and CICS, you find that your address space requirements still exceed that available, you can split CICS using one of three options:

1. Multiregion option (MRO)
2. Intersystem communication (ISC)
3. Multiple independent address spaces.

Adding large new applications or making major increases in the size of your SNA network places large demands on virtual storage, and you must analyze them before implementing them in a production system. Careful analysis and system specification can avoid performance problems arising from the addition of new applications in a virtual-storage-constrained environment. If you have not made the necessary preparations, you typically become aware of problems associated with severe stress only after you have attempted to implement the large application or major change in your production system. Some of these symptoms are:

- Poor response times
- Short-on-storage
- Program compression

- Heavy paging activity
- Many well-tested applications suddenly abending with new symptoms
- S80A and S40D abends
- S822 abends
- Dramatic increase in I/O activity on the DFHRPL concatenation or dynamic LIBRARY concatenation.

The rest of this section describes techniques that you can use to improve the performance of CICS under MVS.

Networking and the z/OS Communications Server: performance and tuning

The performance of your SNA network and logical units (LUs) can be tuned in a number of different ways.

This section includes the following topics:

- https://ut-ilnx-r4.hursley.ibm.com/ts42_latest/help/topic/com.ibm.cics.ts.performance.doc/topics/dfht34d.html
- "Setting the size of the receive-any input areas" on page 128
- "Setting the size of the receive-any pool" on page 129
- "Using the MVS high performance option with SNA" on page 130
- "Adjusting the number of transmissions in SNA transaction flows" on page 131
- Using SNA chaining to segment large messages
- "Limiting the number of concurrent logon and logoff requests" on page 133
- "Adjusting the terminal scan delay" on page 134
- "Compressing output terminal data streams" on page 135
- Turning automatic installation of terminals

Setting the size of the terminal input and output area

The **IOAREALEN** attribute of a TYPETERM RDO resource definition specifies the size of the terminal input and output area that is to be passed to a transaction. The size of the TIOA can also be specified by the **TIOAL** parameter in the DFHTCT TYPE=REMOTE macro, if macro resource definition has been used.

The syntax for the **IOAREALEN** attribute in a TYPETERM RDO resource definition is *({0|value1},{0|value2})*. This setting is used only for the first input message for all transactions. One value defining the minimum size is used for non-SNA devices, while two values specifying both the minimum and maximum size are used for SNA devices.

If you specify *ATI(YES)*, you must specify an **IOAREALEN** value of at least one byte.

Effects

When *value1,0* is specified for **IOAREALEN**, *value1* is the minimum size of the terminal input/output area that is passed to an application program when a **RECEIVE** command is issued. If the size of the input message exceeds *value1*, the area passed to the application program is the size of the input message.

When *value1, value2* is specified, *value1* is the minimum size of the terminal input/output area that is passed to an application program when a **RECEIVE** command is issued. Whenever the size of the input message exceeds *value1*, CICS uses *value2*. If the input message size exceeds *value2*, the node abnormal condition program sends an exception response to the terminal.

Limitations

Real storage can be wasted if the **IOAREALEN** (*value1*) value, or the value for the **TIOAL** parameter in the DFHTCT TYPE=REMOTE macro, is too large for most terminal inputs in the network. However, if

IOAREALEN (*value1*) or **TIOAL** is smaller than most initial terminal inputs, excessive GETMAIN requests can occur, resulting in additional processor requirements, unless **IOAREALEN** (*value1*) or **TIOAL** is zero.

Suggestions

Set **IOAREALEN** (*value1*) or **TIOAL** to a value that is slightly larger than the average input message length for the terminal. The maximum value that can be specified for **IOAREALEN** or **TIOAL** is 32767 bytes.

If a value of nonzero is required, specify the most commonly encountered input message size. A multiple of 64 bytes minus 21 allows for SAA requirements and ensures good use of operating system pages.

For the z/OS Communications Server, you can specify two values if inbound chaining is used. The first value is the length of the normal chain size for the terminal and the second value is the maximum size of the chain. The length of the TIOA presented to the task depends on the message length and the size specified for the TIOA. See the following example:

```
Where x is any number of bytes, the following applies.

Without chain assembly:

If the TIOA size is specified as 20x
and the message length is        15x
then the TIOA acquired is        20x

If the TIOA size is specified as 20x
and the message length is        25x
then the TIOA acquired is        25x

With chain assembly:

If Value1 size is                20x
and Value2 size is               25x, then
if the length of a message is    15x
the TIOA acquired is             20x
and if the message length is     22x
the TIOA acquired is             25x
```

Figure 18. Message length and terminal input and output area length

Avoid specifying a *value1* that is too large, for example, by matching it to the size of the terminal display screen. This area is used only as input. If READ with SET is specified, the same pointer is used by applications for an output area.

Avoid specifying a *value1* that is too small, because extra processing time is required for chain assembly, or data is lost if inbound chaining is not used.

In general, a value of zero is best because it causes the optimum use of storage and eliminates the second GETMAIN request. If automatic transaction initiation (ATI) is used for that terminal, a minimum size of one byte is required.

The second value for SNA devices is used to prevent terminal streaming, and so make it slightly larger than the largest possible terminal input in the network. If a message larger than this second value is encountered, a negative response is returned to the terminal, and the terminal message is discarded.

Monitoring

RMF and NetView Performance Monitor (NPM) can be used to show storage usage and message size characteristics in the network.

Setting the size of the receive-any input areas

The system initialization parameter, **RAMAX**, specifies the size in bytes of the I/O area that is to be allocated for each SNA receive-any operation. You can use the **RAMAX** system initialization parameter in any networks that use the z/OS Communications Server SNA access method for LUs.

These storage areas are called receive-any input areas (RAIAs) and are used to receive the first terminal input for a transaction from the SNA. All input from SNA comes in request/response units (RUs).

Storage for the RAIAs, which is above the 16 MB line, is allocated by the CICS terminal control program during CICS initialization. This storage remains allocated for the entire execution of the CICS job step. The size of this storage is the product of the **RAPOOL** and **RAMAX** system initialization parameters.

Effects

SNA attempts to put any incoming RU into the initial receive-any input area, which has the size of **RAMAX**. If this area is not large enough, SNA creates a message indicating the problem and stating how many extra bytes are waiting that cannot be accommodated.

RAMAX is the largest size of any RU that CICS can take directly in the receive-any command. It is a limit against which CICS compares the indication from SNA of the overall size of the RU. If there is more, it is saved by SNA, and CICS gets the rest in a second request.

With a small **RAMAX**, you reduce the virtual storage taken up in RAIAs. However, you risk more processor usage in SNA tries again to get any data that could not fit into the RAIA.

For many purposes, the default **RAMAX** value of 256 bytes is adequate. If you know that many incoming RUs are larger than this value, you can always increase **RAMAX** to suit your system.

For individual terminals, there are separate parameters that determine how large an RU is going to be from these devices. It makes sense for **RAMAX** to be at least as large as the largest **SENDSIZE** attribute for frequently used terminals.

Limitations

Real storage can be wasted with a high **RAMAX** value. If the **RAMAX** value is set too low, extra processor time is needed to acquire additional buffers to receive the remaining data.

Suggestions

Set **RAMAX** with the size in bytes of the I/O area allocated for each receive-any request issued by CICS. The maximum value is 32767. Because most inputs are 256 bytes, this size is the default value specified.

Set **RAMAX** to be slightly larger than your CICS system input messages. If you know the message length distribution for your system, set the value to accommodate most of your input messages.

In any case, the size required for **RAMAX** need only take into account the first (or only) RU of a message. Thus, messages sent using SNA chaining do not require **RAMAX** to be set based on their overall chain length, but only on the size of the constituent RUs.

Do not specify a **RAMAX** value that is less than the RUSIZE (from the CINIT) for a pipeline terminal because pipelines cannot handle over-length data.

Receive-any input areas are taken from a fixed-length subpool of storage. A size of 2048 might appear to be adequate for two such areas to fit on one 4 KB page, but only 4048 bytes are available in each page, so only one area fits on one page. Defining a size of 2024 ensures that two areas, including page headers, fit on one page.

Monitoring

The size of RUs or chains in a network can be identified with an SNA line or buffer trace.

Setting the size of the receive-any pool

The **RAPOOL** system initialization parameter specifies the number of concurrent receive-any requests that CICS is to process from the z/OS Communications Server for SNA.

RAPOOL determines how many receive-any buffers there are at any time. Therefore, if the z/OS Communications Server for SNA has a lot of input simultaneously, it enables the z/OS Communications Server to put all the messages directly into CICS buffers rather than possibly having to store them elsewhere. The first operand *(value1)* is for non-HPO systems, the second operand (*value2*) is for HPO systems.

The HPO value for the non-HPO operand is derived according to the formula shown in RAPOOL. The second operand (*value2*) for HPO systems is used with minimal adjustment by the formula.

Effects

Initially, task input from a terminal or session is received by the SNA access method and is passed to CICS if CICS has a receive-any request outstanding.

For each receive-any request, an SNA request parameter list (RPL), a receive-any control element (RACE), and a receive-any input area (RAIA) are set aside. The RAIA value is specified by **RAMAX** (see "Setting the size of the receive-any input areas" on page 128 for RAIA considerations). The total area set aside for SNA receive-any operations is:

(maximum RAIA size + RACE size + RPL size) * RAPOOL

If *HPO=YES*, both RACE and RPL are above the 16 MB line.

In general, input messages up to the value specified in **RAPOOL** are all processed in one dispatch of the terminal control task. Because the processing of a receive-any request is a short operation, at times more messages than are specified in the **RAPOOL** value can be processed in one dispatch of terminal control. This situation happens when a receive-any request completes before the terminal control program has finished processing and there are additional messages from SNA.

The specified pool is used only for SNA receive-any processing of the first terminal message in a transaction or the first input to start a task. **RAPOOL** does not affect further inputs for conversational tasks or output. Additional inputs are processed with SNA receive-specific requests.

SNA posts the event control block (ECB) associated with the receive-any input area. CICS then moves the data to the terminal I/O area (TIOA) ready for task processing. The RAIA is then available for reuse.

The significance of **RAPOOL** depends on the environment of the CICS system. For example, if HPO is used then **RAPOOL** is significant.

Limitations

If the **RAPOOL** value is set too low, terminal messages might not be processed in the earliest dispatch of the terminal control program, causing transaction delays during high-activity periods. For example, if you use the default value and five terminal entries need to startup tasks, three tasks might be delayed for at least the time required to complete the SNA receive-any request and copy the data and RPL. In general, set no more than 5 to 10% of all receive-any processing at the **RAPOOL** ceiling, with none at the **RAPOOL** ceiling if there is sufficient storage.

If the **RAPOOL** value is set too high, excessive virtual storage might be used, but does not affect real storage because the storage is not page-fixed and is therefore paged out.

Suggestions

In some cases, it might be more economical for SNA to store the occasional peak of messages in its own areas rather than for CICS to have many RAIAs, which are unused most of the time.

Furthermore, there are situations where CICS reissues a receive-any request as soon as it finds one satisfied. It uses the same element over and over again in order to bring in any extra messages that are in SNA.

CICS maintains a z/OS Communications Server **VTAM RECEIVE ANY** for *n* of the RPLs, where *n* is either the RAPOOL value, or the MXT value minus the number of currently active tasks, whichever is the smaller.

Code **RAPOOL** with the number of fixed request parameter lists (RPLs) that you require. When it is not at the **MXT** value, CICS maintains a receive-any request for each of these RPLs. The number of RPLs that you require depends on the expected activity of the system, the average transaction lifetime, and the **MXT** specified.

The **RAPOOL** value you set depends on the number of sessions, the number of terminals, and the **ICVTSD** value (see "Adjusting the terminal scan delay" on page 134) in the system initialization table (SIT). Initially, for non-HPO systems, set **RAPOOL** to 1.5 times your peak *local* transaction rate per second plus the autoinstall rate. This value can then be adjusted by analyzing the CICS SNA statistics and by resetting the value to the maximum RPLs reached. The **RAPOOL** value does not include MRO sessions, so set this value to a low number in application-owning or file-owning regions (AORs or FORs).

For HPO systems, a small value (<= 5) is typically sufficient if specified through *value2* in the **RAPOOL** system initialization parameter. For example, `RAPOOL=20` is specified as either `RAPOOL=(20)` or `RAPOOL=(20,5)` to achieve the same effect.

Monitoring

The CICS SNA statistics contain values for the maximum number of RPLs posted on any one dispatch of the terminal control program, and the number of times the RPL maximum was reached. This maximum value can be greater than the **RAPOOL** value if the terminal control program is able to reuse an RPL during one dispatch. See Interpreting z/OS Communications Server statistics for more information.

Using the MVS high performance option with SNA

The MVS high performance option (HPO) can be used for processing SNA requests. The purpose of HPO is to reduce the transaction path length through the z/OS Communications Server.

The use of HPO and supervisor calls (SVCs) are specified in the system initialization table (SIT). If the default SVC numbers are acceptable, no tailoring of the system is required.

Effects

HPO bypasses some of the validating functions performed by MVS on I/O operations, and implements service request block (SRB) scheduling. This bypass shortens the instruction path length and allows some concurrent processing on MVS images for the z/OS Communications Server operations because of the SRB scheduling. This effect makes HPO useful in a multiprocessor environment, but not in a single processor environment.

Limitations

HPO requires CICS to be authorized. Some risks with MVS integrity are involved because a user-written module could be made to replace one of the CICS system initialization routines and run in authorized mode. This risk can be reduced by RACF protecting the CICS SDFHAUTH data set.

Use of HPO saves processor time, and does not increase real or virtual storage requirements or I/O contention. An expense of HPO might be the potential security exposure that arises because of a deficiency in validation.

Suggestions

All production systems with vetted applications can use HPO. It is application-transparent and introduces no function restrictions while providing a reduced pathlength through the z/OS Communications Server.

For z/OS Communications Server, the reduced validation does not induce any integrity loss for the messages.

Monitoring

There is no direct measurement of HPO. One method to check whether it is working is to take detailed measurements of processor usage with HPO turned on (SIT option) and with it turned off. Depending on the workload, you might not see much difference. Another way to check whether it is working is that you might see a small increase in the SRB scheduling time with HPO turned on.

RMF can give general information about processor usage. An SVC trace can show how HPO was used.

Take care when using HPO in a system that is being used for early testing of a new application or CICS code (a new release or PUT). Much of the pathlength reduction is achieved by bypassing control block verification code in the z/OS Communications Server. Untested code might possibly corrupt the control blocks that CICS passes to the z/OS Communications Server, and unvalidated applications can lead to security exposure.

Adjusting the number of transmissions in SNA transaction flows

Within CICS, the **MSGINTEG** and **ONEWTE** options can be used to control the communication requests and responses that are exchanged between the terminals in a network and the z/OS Communications Server and NCP communication programs. These options can be used in all CICS systems that use the Communications Server.

With resource definition online (RDO), protection can be specified in the PROFILE definition with the **MSGINTEG**, and **ONEWTE** options. The **MSGINTEG** option is used with SNA logical units (LU) only. See PROFILE resources for more information about defining a PROFILE resource.

Effects

One of the options in Systems Network Architecture (SNA) is whether the messages exchanged between CICS and a terminal are to be in definite or exception response mode. Definite response mode requires both the terminal and CICS to provide acknowledgment of message receipt from each other on a one-to-one basis.

SNA also ensures message delivery through synchronous data link control (SDLC), so definite response is not normally required. Specifying message integrity (**MSGINTEG**) causes the sessions for which it is specified to operate in definite response mode.

In normal cases, the session between CICS and a terminal operates in exception response mode.

You therefore have the following options:

- Not specifying **MSGINTEG**
- Specifying **MSGINTEG** (which asks for definite response to be forced)

In SNA, transactions are defined within brackets. A begin bracket (BB) command defines the start of a transaction, and an end bracket (EB) command defines the end of that transaction. Unless CICS knows ahead of time that a message is the last of a transaction, it must send an EB separate from the last message if a transaction terminates. The EB is an SNA command, and can be sent with the message, eliminating one required transmission to the terminal.

Specifying the one write operation (**ONEWTE**) option for a transaction implies that only one output message is to be sent to the terminal by that transaction, and allows CICS to send the EB along with that message. Only one output message is allowed if **ONEWTE** is specified and, if a second message is sent, the transaction is abended.

The second way to allow CICS to send the EB with a terminal message is to code the LAST option on the last terminal control or basic mapping support **SEND** command in a program. Multiple **SEND** commands can be used, but the LAST option must be coded for the final **SEND** in a program.

The third (and most common) way is to issue **SEND without WAIT** as the final terminal communication. The message is then sent as part of task termination.

Limitations

The **MSGINTEG** option causes additional transmissions to the terminal. Transactions remain in CICS for a longer period, and tie up virtual storage and access to resources, primarily enqueues. **MSGINTEG** is required if the transaction must know that the message was delivered.

When **MSGINTEG** is specified, the TIOA remains in storage until the response is received from the terminal. This option might increase the virtual storage requirements for the CICS region because of the longer duration of the storage needs.

Monitoring

You can monitor the use of the **MSGINTEG** and **ONEWTE** options from a Communications Server trace by examining the exchanges between terminals and CICS and, in particular, by examining the contents of the request/response header (RH).

Using SNA chaining to segment large messages

Systems Network Architecture (SNA) allows terminal messages to be chained, and lets large messages be split into smaller parts while still logically treating the multiple message as a single message. Chaining can be used in systems that use z/OS Communications Server SNA LUs of types that tolerate chaining.

Chaining characteristics are specified with the **SENDSIZE**, **BUILDCHAIN**, and **RECEIVESIZE** attributes.

The hardware requirements of each terminal normally dictate the input chain size and characteristics. The **BUILDCHAIN** and **RECEIVESIZE** attributes have default values that depend on device attributes. The size of an output chain is specified by the **SENDSIZE** attribute.

Effects

Because the network control program (NCP) also segments messages into 256 byte blocks for normal LU Type 0, 1, 2, and 3 devices, a **SENDSIZE** value of zero eliminates the processing effects of output chaining. A value of 0 or 1536 is required for local devices of this type.

If you specify the **SENDSIZE** attribute for intersystem communication (ISC) sessions, this attribute must match the **RECEIVESIZE** attribute in the other system. The **SENDSIZE** attribute or **TCT BUFFER** operand controls the size of the SNA element that is to be sent, and the **RECEIVESIZE** must match so that there is a corresponding buffer of the same size able to receive the element.

If you specify `BUILDCHAIN(YES)`, CICS assembles a complete chain of elements before passing them to an application. If you do not specify `BUILDCHAIN(YES)`, each individual RU is passed to an individual receive-any in the application. With SNA/3270, BMS does not work correctly if you do not specify `BUILDCHAIN(YES)`.

If you are dealing with large inbound elements that exceed a maximum of 32 KB, you cannot use the **BUILDCHAIN** attribute or **CHNASSY** operand. You must use multiple individual RUs, which extends the transaction life in the system.

Limitations

If you specify a low **SENDSIZE** value, this setting causes additional processing. Real and virtual storage are used to break the single logical message into multiple parts.

Chaining might be required for some terminal devices. Output chaining can cause flickering on display screens, which users might find disruptive. Chaining also causes additional I/O processing effects between the z/OS Communications Server and the NCP by requiring additional z/OS Communications Server subtasks and STARTIO operations. These effects are eliminated with applicable ACF/SNA releases by using the large message performance enhancement option (LMPEO).

Suggestions

The **RECEIVESIZE** value for IBM 3274-connected display terminals is 1024 and for IBM 3276-connected display terminals it is 2048. These values give good line characteristics while keeping processor usage to a minimum.

Monitoring

Use of chaining and chain size can be determined by examining a z/OS Communications Server trace. You can also use the CICS internal and auxiliary trace facilities, where the VIO ZCP trace shows the chain elements. Some network monitoring tools such as NetView Performance Monitor (NPM) give this data.

Limiting the number of concurrent logon and logoff requests

The **OPNDLIM** system initialization parameter defines the number of concurrent z/OS Communications Server logon and logoff requests that are to be processed by CICS. This parameter can be used in CICS systems that use the z/OS Communications Server as the terminal access method.

The **OPNDLIM** parameter can also be useful if there are times when all the user community tends to log on or log off at the same time, for example, during lunch breaks.

This parameter limits the number of concurrent logon OPNDST and logoff CLSDST requests. The smaller this value, the smaller the amount of storage that is required during the open and close process. For more information about this parameter, see OPNDLIM system initialization parameter.

Each concurrent logon and logoff requires storage in the CICS dynamic storage areas for the duration of that processing.

Effects

When logons occur automatically with either the CICS CONNECT=AUTO facility or the z/OS Communications Server LOGAPPL facility, large numbers of logons can occur at CICS startup or restart times.

The LOGAPPL facility offers two advantages if an automatic logon facility is required: it requires approximately 3500 bytes less storage in the z/OS Communications Server than the CONNECT=AUTO facility, and it logs terminals back on to CICS each time the device is activated to the z/OS Communications Server, rather than only at CICS initialization.

Limitations

If the value specified for **OPNDLIM** is too low, real and virtual storage requirements are reduced within CICS, and the z/OS Communications Server buffer requirements might be cut back, but session initialization and terminations take longer.

Suggestions

Use the default value initially and adjust if statistics indicate that too much storage is required in your environment or that the startup time is excessive.

Set **OPNDLIM** to a value not less than the number of logical units (LU) connected to any single z/OS Communications Server line.

Monitoring

Logon and logoff activities are not reported directly by CICS or any measurement tools, but can be analyzed using the information given in a z/OS Communications Server trace or z/OS Communications Server display command.

Adjusting the terminal scan delay

The terminal scan delay (**ICVTSD**) system initialization parameter determines the frequency with which CICS attempts to process terminal output requests.

The **ICVTSD** system initialization parameter is defined in units of milliseconds. Use the commands **CEMT** or **EXEC CICS SET SYSTEM SCANDELAY (nnnn)** to reset the value of **ICVTSD**.

In reasonably active systems, a nonzero **ICVTSD** virtually replaces ICV, because the time to the next terminal control table (TCT) full scan (non-SNA) or sending of output requests (SNA) is the principal influence on wait duration of the operating system.

The **ICVTSD** parameter can be used in all except very low-activity CICS systems.

In general, the **ICVTSD** value defines the time that the terminal control program must wait to process the following requests:

- Non-SNA LU I/O requests with `WAIT` specified
- Non-SNA output deferred until task termination
- Automatic transaction initiation (ATI) requests
- SNA LU management, including output request handling, in busy CICS systems with significant application task activity. This last case arises from the way that CICS scans active tasks.

On CICS non-SNA systems, the delay value specifies how long the terminal control program must wait after an application terminal request, before it carries out a TCT scan. The value controls batching and delay in the associated processing of terminal control requests. In a low-activity system, it controls the dispatching of the terminal control program.

Effects in SNA networks

In SNA networks, a low **ICVTSD** value does not cause full TCT scans, because the input from or output to SNA LU is processed from the activate queue chain, and only those terminal entries are scanned.

Request batching reduces processor time at the expense of longer response times. On CICS SNA systems, it influences how quickly the terminal control program completes SNA request processing, especially when the MVS high performance option (HPO) is being used.

With SNA LUs, CICS uses bracket protocol to indicate that the terminal is currently connected to a transaction. The bracket is started when the transaction is initiated, and ended when the transaction is terminated. Thus, there might be two outputs to the terminal per transaction: one for the data sent and one when the transaction terminates containing the end bracket. In fact, only one output is sent (except for **WRITE/SEND** with `WAIT` and definite response). CICS holds the output data until the next terminal control request or termination. It saves processor cycles and line utilization by sending the message and end bracket or change direction (if the next request was a READ/RECEIVE) together in the same output message (PIU). When the system gets busy, terminal control is dispatched less frequently and becomes more dependent upon the value specified in **ICVTSD**. Because CICS may not send the end bracket to SNA for an extended period, the life of a transaction can be extended. Storage is kept allocated for that task for longer periods, potentially increasing the amount of virtual storage required for the total CICS dynamic storage areas. Setting **ICVTSD** to zero can overcome this effect

Effects in non-SNA networks

ICVTSD is the major control on the frequency of full TCT scanning of non-SNA LUs. In active systems, a full scan is done approximately once every **ICVTSD** period. The average extra delay before sending an output message is about half this period.

In non-SNA networks, partial scans occur for other reasons, such as an input arriving from a LU, and any outputs for that line are processed at the same time. For that reason, a value of between 0.5 and one second is normally a reasonable setting for non-SNA networks.

CICS scans application tasks first, unless there is a scan driven by **ICVTSD**. In a highly used system, input and output messages might be unreasonably delayed if too large a **ICVTSD** value is specified.

Effects in all networks

The **ICVTSD** parameter can be changed in the system initialization table (SIT) or through JCL parameter overrides. If you have virtual storage constraint problems, reduce the value specified in **ICVTSD**. A value of zero causes the terminal control task to be dispatched most frequently. If you also have many non-SNA LUs, this value might increase the amount of nonproductive processor cycles. A value of 100—300 ms might be more appropriate for that situation. In a pure SNA environment, however, the processing effect is not significant, unless the average transaction has a short pathlength. Set **ICVTSD** to zero for a better response time and best virtual storage usage.

Limitations

In z/OS Communications Server (for SNA) systems, a low value adds the processing effect of scanning the activate queue TCTTE chain, which is normally a minor consideration. A high value in high-volume systems can increase task life and tie up resources owned by that task for a longer period, which can be a significant consideration.

A low, nonzero value of **ICVTSD** can cause CICS to be dispatched more frequently, which increases the processing effect of performance monitoring.

Suggestions

Set **ICVTSD** to a value less than the region exit time interval (ICV), which is also in the system initialization table. Use the value of zero in an environment that contains only SNA LUs and consoles, unless your workload consists of many short transactions.

Entering `ICVTSD=0` in an SNA LU-only environment is not recommended for a CICS workload consisting of low terminal activity but with high TASK activity. Periods of low terminal activity can lead to delays in CSTP being dispatched. Setting *ICVTSD=100-500* resolves this effect by causing CSTP to be dispatched regularly. For non-SNA systems, specify the value of zero only for small networks (1 - 30 terminals).

For almost all systems that are not “pure” SNA, set the range somewhere in the region of 100 ms to 1000 ms. **ICVTSD** can be varied from 300 - 1000 ms without a significant effect on the response time, but increasing the value decreases the processor activity effect. An **ICVTSD** larger than 1000 ms might not give any further improvement in processor usage, at a cost of longer response times.

If **ICVTSD** is reduced, and if there is ample processor resource, a small reduction in response time can be achieved. If you set the value below 250 ms, any improvement in response time is likely to seem negligible to the user and would have an increased effect on processor usage.

The absolute minimum level, for systems that are not “pure” SNA, is approximately 250 ms, Or, in high-performance, high-power systems that are “pure” SNA, the level is 100 ms.

Monitoring

Use RMF to monitor task duration and processor requirements. The dispatcher domain statistics reports the value of **ICVTSD**.

Compressing output terminal data streams

For output messages, CICS provides user exits with access to the entire output data stream. User code can be written to remove redundant characters from the data stream before the data stream is sent to the terminal.

For z/OS Communications Server for SNA devices, the global user exit used to compress terminal messages is **XZCOUT1**. For programming information, see SNA working-set module exits (XZCIN, XZCOUT, XZCOUT1, and XZIQUE).

This compression technique can produce a dramatic improvement in response times if the proportion of characters not needed is large, because telecommunication links are typically the slowest paths in the network.

Limitations

Some additional processor cycles are required to process the exit code, and the coding of the exit logic also requires some effort. Using a compression exit reduces the storage requirements of SNA and reduces line transmission time.

Suggestions

The simplest operation is to replace redundant characters, especially blanks, with a repeat-to-address sequence in the data stream for 3270-type devices.

Note: The repeat-to-address sequence is not handled quickly on some types of 3270 cluster controller. In some cases, alternatives can give superior performance. For example, instead of sending a repeat-to-address sequence for a series of blanks, consider sending an ERASE and then set-buffer-address sequences to skip over the blank areas. This method is satisfactory if nulls are acceptable in the buffer as an alternative to blanks.

Another technique for reducing the amount of data transmitted is to turn off any modified data tags on protected fields in an output data stream. This method eliminates the need for those characters to be transmitted back to the processor on the next input message, but review application dependencies on those fields before you try this approach.

There might be other opportunities for data compression in individual systems, but you need to investigate the design of those systems thoroughly before you can implement them.

Monitoring

The contents of output terminal data streams can be examined in an SNA trace.

Tuning automatic installation of terminals

During autoinstall processing, CICS obtains storage from the control subpool in the extended CICS dynamic storage area (ECDSA), to handle each autoinstall request.

The amount of virtual storage obtained is determined by the length of the CINIT request unit, which varies for different LU types. For a typical autoinstall request from an LU 6.2 terminal, the amount of dynamic virtual storage obtained is 120 -250 bytes.

The principal consumer of CICS resource in autoinstall processing is the autoinstall task (CATA) itself. If, for some reason, the autoinstall process is not proceeding at the rate expected during normal operations, there is a risk that the system could be filled with CATA transaction storage.

Maximum concurrent autoinstalls

The **AIQMAX** system initialization parameter codes the maximum number of devices that can be queued concurrently for autoinstall.

The **AIQMAX** value does not limit the total number of devices that can be autoinstalled.

The restart delay parameter

The **AIRDELAY** system initialization parameter specifies whether you want autoinstalled terminal definitions to be retained by CICS across a restart.

The value of the restart delay is specified as *hhmmss* and the default is `000700`, which is seven minutes. This delay means that if a terminal does not log on to CICS within seven minutes after an emergency restart, its terminal entry is scheduled for deletion.

Setting the restart delay to zero means that you do not want CICS to reinstall the autoinstalled terminal entries from the global catalog during emergency restart. In this case, CICS does not write the terminal entries to the catalog while the terminal is being autoinstalled. This setting can have positive performance effects on the following processes:

Autoinstall
: By eliminating the I/O activity, autoinstall has a shorter pathlength and becomes more processor-intensive. So, in general, the time taken to autoinstall a terminal is reduced. However, the response time of other tasks might increase slightly because CATA has a high priority and does not have to wait for as much I/O activity.

Emergency and warm restart
: When no autoinstalled terminal entries are cataloged, CICS has to restore fewer entries from the global catalog data set during emergency restart. Thus, if you have many autoinstalled terminals, the restart time can be improved when restart delay is set to zero.

Normal shutdown
: CICS deletes AI terminal entries from the global catalog data set during normal shutdown unless they were not cataloged (*AIRDELAY=0*) and the terminal has not been deleted. If the restart delay is set to zero, CICS has not cataloged terminal entries when they were autoinstalled, so they are not deleted. This setting can reduce normal shutdown time.

You must consider the risk of having some terminal users log on again because tracking has not completed, against the benefits introduced by setting the restart delay to zero. Because catchup takes only a few minutes, the chance of such a takeover occurring is typically small.

The delete delay parameter

The **AILDELAY** system initialization parameter lets you control how long an autoinstalled terminal entry remains available after the terminal has logged off. The default value of zero means that the terminal entry is scheduled for deletion as soon as the terminal is logged off. Otherwise, CICS schedules the deletion of the TCTTE as a timer task.

In general, setting the delete delay to a nonzero value can improve the performance of CICS when many autoinstalled terminals are logging on and off during the day. However, this setting does mean that unused autoinstalled terminal entry storage is not freed for use by other tasks until the delete delay interval has expired. This parameter provides an effective way of defining a terminal whose storage lifetime is somewhere between the lifetime of an autoinstalled terminal and a statically defined terminal.

The effect of setting the delete delay to a nonzero value can have different effects depending on the value of the restart delay:

Nonzero restart delay When the restart delay is nonzero, CICS catalogs autoinstalled terminal entries in the global catalog.

If the delete delay is nonzero as well, CICS retains the terminal entry so that it is reused when the terminal logs back on. This setting can eliminate the activities of:

- Deleting the terminal entry in virtual storage
- An I/O to the catalog and recovery log
- Rebuilding the terminal entry when the terminal logs on again.

Zero restart delay When the restart delay is zero, CICS does not catalog autoinstalled terminal entries in the global catalog whatever value is specified for the delete delay.

If the delete delay is nonzero, CICS retains the terminal entry so that it is reused when the terminal logs back on. This delay can save the processing effect of deleting the terminal entry in virtual storage and the rebuilding of the terminal entry when the terminal logs on again.

Effects

You can control the use of resource by autoinstall processing in three ways:

1. By using the transaction class limit to restrict the number of autoinstall tasks that can exist concurrently (see "Using transaction classes (MAXACTIVE) to control transactions" on page 59).
2. By using the CATA and CATD transactions to install and delete autoinstall terminals dynamically. If you have many devices autoinstalled, shutdown can fail due to the **MXT** system initialization parameter

being reached or CICS becoming short on storage. To prevent this possible cause of shutdown failure, consider putting the CATD transaction in a class of its own to limit the number of concurrent CATD transactions.

3. By specifying **AIQMAX** to limit the number of devices that can be queued for autoinstall. This setting protects against abnormal consumption of virtual storage by the autoinstall process, caused as a result of some other abnormal event.

   If this limit is reached, the **AIQMAX** system initialization parameter affects the LOGON and BIND processing by CICS. CICS requests z/OS Communications Server to stop passing LOGON and BIND requests to CICS. z/OS Communications Server holds such requests until CICS indicates that it can accept further LOGONs and BINDs (occurs when CICS has processed a queued autoinstall request).

Suggestions

If the autoinstall process is noticeably slowed down by the **AIQMAX** limit, raise it. If the CICS system shows signs of running out of storage, reduce the **AIQMAX** limit. If possible, set the **AIQMAX** system initialization parameter to a value higher than the value reached during normal operations.

Settings of (*restart delay=0*) and (*delete delay= hhmmss>0*) are the most efficient for processor and DASD utilization. However, this efficiency is gained at a cost of virtual storage, because the TCT entries are not deleted until the delay period expires.

A value of zero for both restart delay and delete delay is the best overall setting for many systems from an overall performance and virtual storage usage point of view.

If restart delay is greater than zero (cataloging active), the performance of autoinstall is affected by the definition of the global catalog (DFHGCD). The default buffer specifications used by VSAM might not be sufficient in a high activity system.

Because a considerable number of messages are sent to transient data during logon and logoff, consider the performance of these output destinations.

Monitoring

Monitor the autoinstall rate during normal operations by inspecting the autoinstall statistics regularly.

CICS MRO, ISC, and IPIC: performance and tuning

Multiregion operation (MRO), intersystem communication over SNA (ISC over SNA), and IP interconnectivity (IPIC) connections enable CICS systems to communicate and share resources with each other. Performance is influenced by the intercommunication facilities that you use with the connection and by your management of the connection.

These CICS intercommunication facilities are available using MRO, and ISC over SNA, and IPIC connections:

- Function shipping
- Distributed transaction processing
- Asynchronous processing
- Transaction routing
- Distributed program link

For descriptions of the CICS intercommunication methods and facilities, see Introduction to CICS intercommunication.

CICS ISC/IRC statistics show the frequency of use of intercommunication sessions and mirror transactions. The z/OS Communications Server SNA trace, an SVC trace, and RMF give additional information.

If each transaction makes a number of intercommunication requests, function shipping generally incurs the most processor usage. The number of requests per transaction that constitutes the break-even point depends on the nature of the requests.

Both distributed transaction processing (DTP) and asynchronous processing are, in many cases, the most efficient facilities for intercommunication because a variety of requests can be batched in one exchange. DTP, however, requires an application program specifically designed to use this facility.

Transaction routing, in most cases, involves one input and one output between systems, and the additional processor usage is minimal.

MRO

Multiregion operation (MRO), in general, causes less processor usage than intersystem communication (ISC) because the SVC pathlength is shorter than that through the multisystem networking facilities of SNA. CICS MRO provides a long-running mirror transaction and fastpath transformer program to further reduce processor usage.

Ensure that you have a sufficient number of MRO sessions defined between the CICS systems to take your expected traffic load. The increased cost in real and virtual storage is minimal, and task life is reduced, so the probable overall effect is to save storage. Examine the ISC/IRC statistics (see ISC/IRC system and mode entry statistics) to ensure that no allocates have been queued; also ensure that all sessions are being used. However, the definition of too many MRO sessions can unduly increase the processor time used to test their associated ECBs.

If you want only transaction routing with MRO, the processor usage is relatively small. The figure is release- and system-dependent (for example, it depends on whether you are using cross-memory hardware), but you can assume a total cost somewhere in the range of 15 - 30 KB instructions per message pair. This is a small proportion of most transactions, commonly 10% or less. The cost of MRO function shipping can be very much greater, because typically each transaction has many more inter-CICS flows. The cost depends greatly on the disposition of resources across the separate CICS systems.

MRO can affect response time as well as processor time. Delays occur in getting requests from one CICS system to the next. These delays arise because CICS terminal control in either CICS system has to detect any request sent from the other, and then has to process it. In addition, if you have a uniprocessor, MVS has to arrange dispatching of two CICS systems and that must imply extra WAIT/DISPATCH processor usage and delays.

Specify the system initialization parameter **`MROLRM=YES`** if you want to establish a long-running mirror task. This saves re-establishing communications with the mirror transaction if the application makes many function shipping requests in a unit of work.

When you use MRO, you can eliminate some processor usage for SVC processing with the use of MVS cross-memory services. Cross-memory services use the MVS common system area (CSA) storage for control blocks, not for data transfer, which can also be a benefit. Note, however, that MVS requires that an address space using cross-memory services be nonswappable.

ISC

For situations where ISC is used across MVS images, consider using XCF/MRO. CICS uses the MVS cross-system coupling facility (XCF) to support MRO links between MVS images for transaction routing, function shipping, and distributed program link. You can also use XCF/MRO for distributed transaction processing, if the LU6.1 protocol is adequate for your purpose. XCF/MRO consumes less processor resources than ISC.

You can prioritize ISC mirror transactions. The CSMI transaction is for data set requests, CSM1 is for communication with IMS systems, CSM2 is for interval control, CSM3 is for transient data and temporary storage, and CSM5 is for IMS DB requests. If one of these functions is particularly important, you can prioritize it over the rest. This prioritization is not effective with MRO because any attached mirror transaction services any MRO request while it is attached.

If ISC facilities tend to flood a system, you can control them with the SNA VPACING facility. Specifying multiple sessions (SNA parallel sessions) increases throughput by allowing multiple paths between the systems. With CICS, you can specify an SNA class of service (COS) table with LU6.2 sessions, which can prioritize ISC traffic in a network.

Interregion communication performance costs with MRO and ISC

Using the tables in these topics, you can compare the relative processing times of particular CICS API calls, and examine some of the other factors that affect overall processing times. These tables can help you make decisions concerning application design when you are considering performance. To calculate a time for a transaction, find the entries appropriate to your installation and application, and add their values together.

Before you work with these numbers, be aware of the following considerations:

- The cost per call is documented in 1 K or millisecond instruction counts taken from a tracing tool used internally by IBM. Each execution of an instruction has a count of 1. No weighting factor is added for instructions that use more machine cycles than others.
- Because the measurement consists of tracing a single transaction within the CICS region, any wait, for example a wait for I/O, results in a full MVS WAIT. This cost has been included in the numbers reported in this document. On a busy system the possibility of taking a full MVS WAIT is reduced because the dispatcher has a higher chance of finding more work to do.
- When judging performance, the numbers in this information should not be compared with those published previously, because a different methodology has been used.

Transaction routing performance costs

MRO XM	MRO XCF (through CTC)	MRO XCF (through CF)	ISC LU6.2
37.0	43.0	66.0	110.0

Function shipping performance costs (MROLRM=YES)

Type	MRO XM	MRO XCF (through CTC)	MRO XCF (through CF)
Initiate®/terminate environment	13.2	13.2	13.2
Each function shipping request	9.0	23.4	48.4
Sync point flow	9.0	23.4	48.4

Notes:

- These costs relate to CICS systems with long-running mirrors.
- ISC LU6.2 does not support **MROLRM=YES**.
- The cost of session allocation, initiation of the mirror transaction, stopping the mirror transaction, and session deallocation is included in the initiate/terminate environment.

For example, if you migrate from a local file access to MRO XM and request 6 function ships per transaction, the additional cost is calculated as follows:

13.2(Initiate/End) + (6(requests)*9.0(Request cost)) + 9.0(Sync point) = 76.2

Function shipping performance costs (MROLRM=NO)

Without long-running mirrors, each function ship read request incurs the cost of session allocation and mirror initialization and termination. However, the first change to a protected resource (for example, a READ UPDATE or a WRITE) causes the session and mirror to be held until a sync point.

MRO XM	MRO XCF (through CTC)	MRO XCF (through CF)	ISC LU6.2
21.4	35.0	59.9	115.0

IPIC

The CICS-supplied mirror program DFHMIRS is defined as a threadsafe program. For supported CICS facilities, over IPIC connections only, the remote CICS region uses a threadsafe mirror transaction and runs the request on an L8 open TCB whenever possible. Using open TCBs can reduce contention on the QR TCB and improve the overall throughput of a CICS region. However, DFHMIRS still needs to switch to the SO TCB when communicating with the TCP/IP stack to send and receive messages, so there is not necessarily a reduction in CPU consumption or in TCB switches for individual tasks.

For some applications, the performance benefits of using long-running mirrors can also be significant. IPIC supports the MIRRORLIFE attribute of the IPCONN, which can improve efficiency and provide performance benefits by specifying the lifetime of mirror tasks and the amount of time a session is held.

IPIC supports threadsafe processing for the LINK command between CICS TS 4.2 or later regions. If you are using a threadsafe program that makes DPL requests that are transmitted to another region using IPIC connections, you might benefit from improved performance by changing your dynamic routing program to be coded to threadsafe standards.

Function shipping file control, transient data, and temporary storage requests over an IPIC connection provides CICS application programs with the ability to run without regard to the location of the requested resources. Function shipping of file control and temporary storage requests using IPIC connections is threadsafe between CICS TS 4.2 or later regions. Function shipping of transient data requests using IPIC connections is threadsafe between CICS TS 5.1 or later regions. Any global user exit programs that are called in the remote CICS region for file control, transient data, and temporary storage requests must be enabled as threadsafe programs for the best performance.

For file control requests that are function shipped using IPIC connectivity, to gain the performance benefits of the open transaction environment, you must specify the system initialization parameter **`FCQRONLY=NO`** in the file-owning region.

Managing queues for intersystems sessions

When intersystems links are added to the system there is the possibility that they cannot respond adequately to transaction requests because the remote system is performing badly.

The poor performance can be due either to a long-term condition, such as lack of resource or overloading, or a temporary situation such as a memory dump being taken. In any case there is the danger that the problem can cause a long queue to form in the requesting system.

Mechanisms are provided in CICS for:

- Protection of the requesting system from using too many resources while transactions queue for the use of the intersystems sessions.
- Detection of problems in remote systems. CICS can issue messages to indicate a problem on an intersystems connection and the parameters control the criteria that are used to determine when a problem exists, or has gone away.

The two mechanisms are:

1. The QUEUELIMIT and MAXQTIME parameters on the connection resource definition.

   The QUEUELIMIT parameter limits the number of transactions which can be queued in allocate processing waiting for a session to become free. Any transactions which try to join a queue already at its limit are rejected.

   The MAXQTIME parameter is a control on the wait time of queued allocate requests that are waiting for free sessions on a connection that appears to be unresponsive. If the rate of processing of the queue indicates that a new allocate will take longer than the specified time to reach the head of the queue, the whole queue is purged.

2. The XZIQUE user exit, which is given control when an allocate request is about to be queued, or the first time it succeeds after a suspected problem. The XZIQUE exit can control the queue or you can use it to add more sophisticated controls of your own.

Both mechanisms produce the same effect on the application program which issued the allocate; a SYSIDERR condition is returned. Return codes are also provided to the dynamic routing program to indicate the state of the queue of allocate requests.

XZIQUE exit for managing MRO and APPC intersystem queues gives programming information about the XZIQUE exit and its relationship with the rest of CICS, including application programs and the dynamic routing program.

Relevant statistics

You can use connection statistics to detect problems in a CICS intersystem environment.

For each connection CICS records the following:

- The number of allocates queued for the connection, and the peak value of this number. (`Peak outstanding allocates` in the Connection statistics.)

  You can use this statistic to see how much queuing normally takes place on connections in your system. If there is occasionally a large queue you should consider controlling it. Are enough sessions defined? has more advice on setting the correct number of sessions for your connections.

For each of the queue control mechanisms, CICS records the following statistics for each connection:

- The number of allocates which were rejected due to the queue becoming too large
- The number of times the queue was purged because the throughput was too slow
- The number of allocates purged due to slow throughput.

Interpreting ISC/IRC system and mode entry statistics also contains an explanation of these statistics, and other connection statistics.

Ways of approaching the problem and recommendations

The queue limit mechanism can be used to control the number of tasks waiting for the use of an intersystems link.

You should use the control to ensure that even at its maximum length the queue does not use too many of the MXT slots in the system. You can also use the MAXACTIVE setting of a TRANCLASS definition if you can separate your transactions into classes that correspond to the remote regions they require.

To ensure free availability during normal running, provide a sufficient number of intersystems sessions. Session definitions do not occupy excessive storage, and the occupancy of transaction storage probably outweighs the extra storage for the session. The number of sessions should correspond to the peak number of transactions in the system which are likely to use the connection—you can see the maximum number of sessions being used from the terminal statistics for the connection. If all sessions are used, the connections statistics show the number of times allocates were queued compared with the total number of requests.

Even in a system that has no problems, there are significant variations in the numbers of transactions that are active at any time, and the actual peak number might be larger than the average over a few minutes at the peak time for your system. You should use the average rather than the actual peak; the queuing mechanism is intended to cope with short-term variations, and the existence of a queue for a short time is not a cause for concern.

The start of a queue is used by the queue limiting mechanism as a signal to start monitoring the response rate of the connection. If queues never form until there is a large problem, the detection mechanism is insensitive. If there are always queues in the system, you might experience false diagnosis.

You should set the queue limit to a number that is roughly the same size as the number of sessions—within the limits imposed by MXT if there are many connections whose cumulative queue capacity would reach MXT. In this latter case, design your own method—using ZXIQUE—of controlling queue lengths so that the allocation of queue slots to connections is more dynamic.

The MAXQTIME parameter can be set to reflect the maximum wait time expected of users for responses in case of potential problems. The MAXQTIME parameter should not be set at a low value in combination with a queue limit that is low, because this leads to a sensitive detection criterion.

Monitoring the settings

The number of allocates rejected by the queue control mechanism should be monitored. If there are too many, it may indicate a lack of resources to satisfy the demands on the system—or poor tuning.

The number of times the queue is purged should indicate the number of times a serious problem occurred on the remote system. If the purges do not happen when the remote system fails to respond, examine the setting of the MAXQTIME parameter—it may be too high, and insensitive. If the indication of a problem is too frequent and causes false alarms due to variations in response time of the remote system, the parameter may be too low, or the QUEUELIMIT value too low.

Using transaction classes DFHTCLSX and DFHTCLQ2 to control storage use

Use DFHTCLSX and DFHTCLQ2 in RDO group DFHISCT to control the amount of storage used by CICS to run the CLS1, CLS2, and CLQ2 transactions.

Effects

These tasks execute the activities needed to acquire an APPC conversation (CLS1/2), and to resynchronize units of work for MRO and APPC connections (CLQ2). Typically there are not many tasks, and they need no control. However, if your CICS system has many connection definitions, these connections might be acquired simultaneously as a result of initializing the system at startup, or as a result of a **`SET VTAM OPEN`** or **`SET IRC OPEN`** command.

Note: VTAM is now z/OS Communications Server.

How implemented

The system definitions are optional. Install resource group DFHISCT to activate them. As supplied, the **`MAXACTIVE`** parameter in the DFHTCLSX and DFHTCLQ2 is 25. This value gives sufficient control to prevent the system reaching a short-on-storage situation. Tasks CLS1 and CLS2 each require 12 KB of dynamic storage, and CLQ2 tasks require up to 17 KB. Do not set the purge threshold to a non-zero number and do not set the **`MAXACTIVE`** parameter to 0. Both values might prevent CICS from running tasks necessary to intersystems functions.

Do not set the **`MAXACTIVE`** value too low, because network delays or errors might cause one of the tasks in the TCLASS to wait and block the use of the TCLASS by succeeding transactions. Setting a low value can also extend shutdown time in a system with many connections.

Controlling the length of the terminal input/output area (SESSIONS IOAREALEN) for MRO sessions

For MRO function shipping, the SESSIONS definition attribute, **`IOAREALEN`**, is used. This attribute regulates the length of the terminal input/output area (TIOA) to be used for processing messages transmitted on the MRO link. These TIOAs are located above the 16 MB line.

The **`IOAREALEN`** value controls the length of the TIOA that is used to build a message transmitted to the other CICS system (that is, an outgoing message). You can specify two values (value1 and value2). Value1 specifies the initial size of the TIOA to be used in each session that is defined for the MRO connection. If the size of the message exceeds value1, CICS acquires a larger TIOA to accommodate the message. Only one value is required. However, if value2 is specified, CICS uses value2 whenever the size of the message exceeds value1.

A value of zero causes CICS to get a storage area exactly the size of the outgoing message, plus 600 bytes for CICS requirements. If the IOAREALEN value is not specified, it defaults to 4 KB.

Where useful

The **IOAREALEN** attribute can be used in the definition of sessions for either MRO transaction routing or function shipping. For MRO transaction routing, the value determines the initial size of the TIOA, whereas in the MRO function shipping environment, the value presents some tuning opportunities.

Limitations

If the **IOAREALEN** value is too large for most messages transmitted on your MRO link, real and virtual storage might be wasted. If IOAREALEN is smaller than most messages, or zero, excessive FREEMAIN and GETMAIN requests might occur, resulting in additional processor requirements.

Recommendations

For optimum storage and processor utilization, make **IOAREALEN** slightly larger than the length of the most commonly encountered formatted application data transmitted across the MRO link for which the sessions are defined.

For efficient operating system paging, add 600 bytes for CICS requirements and round up the total to a multiple of 64 bytes. A multiple of 64 bytes (or less) minus 600 bytes for CICS requirements ensures a good use of operating system pages.

How implemented

The TIOA size can be specified in the **IOAREALEN** attribute of the SESSIONS definition.

Batching requests (MROBTCH)

Certain events in a region can be accumulated in a batch before posting, until the number specified in the MROBTCH system initialization parameter is reached (or ICV times out).

Then, the region is started so that it can process the requests. The batching of MRO requests includes some non-MRO events such as:

- VSAM physical I/O completion
- Request completion carried out as a subtask on the CO TCB (mostly VSAM, and if SUBTSKS=1 is specified)
- DL/I request completion implemented through DBCTL

Strictly speaking, batching is applicable to a TCB rather than the region. MROBTCH is applied only to the "quasi-reentrant" mode TCB.

Effects of changing the default value of MROBTCH

Compared to no batching (MROBTCH=1, that is, the default), setting MROBTCH=n has the following effects:

- Up to [(n-1)*100/n]% saving in the processor usage for waiting and posting of that TCB. For example, for n=2, 50% savings might be achieved, for n=3, 66% savings, or for n=6, 83% savings.
- An average cost of (n+1)/2 times the average arrival time for each request batched.
- Increased response time might cause an increase in overall virtual storage usage as the average number of concurrent transactions increases.
- In heavily loaded systems at peak usage, some batching can happen as a natural consequence of queuing for a busy resource. Using a low MROBTCH value greater than one might then decrease any difference between peak and off-peak response times.

Setting MROBTCH higher than 6 is not recommended as the decreasing additional processor saving is unlikely to be worth the further increased response time.

You require a relatively low value of MROBTCH for ICV to maintain reasonable response time during periods of low utilization.

Setting a suitable batch value

Depending on the amount of response time degradation you can afford, you can set MROBTCH to different values. Use the CICS-SM perspective of the CICS Explorer (**Operations** > **Regions view** > **Region attributes** > **MRO Batch requests**) or use **`EXEC CICS SET SYSTEM MROBATCH`** to arrive at a suitable batch value for a given workload.

For programming information about the **`EXEC CICS`** system programming commands, see System commands.

During slow periods, the ICV unconditionally dispatches the region, even if the batch is not complete and provides a minimum delay. In this case, set ICV to 500 milliseconds in each region.

Extending the life of mirror transactions (MROLRM and MROFSE)

The MROLRM system initialization parameter can have a significant effect on the performance of a workload in an MRO function shipping environment.

Setting `MROLRM=NO` causes the mirror to be attached and detached for each function-shipped request until the first request for a recoverable resource or a file control start browse is received. After such a request is received, the mirror remains attached to the session until the calling transaction reaches syncpoint.

Setting `MROLRM=YES` in a region receiving function shipping requests causes a mirror transaction to remain attached to the MRO session from first request until the calling transaction reaches syncpoint. This option causes system-dependent effects, as follows:

- Some systems show significant improvements in processor utilization per transaction. They are likely to be systems with a significant percentage of inquiry transactions, each with multiple VSAM calls, or transactions with many reads followed by a few updates.
- Some systems show no performance difference. Workloads using IMS, or transactions that make a lot of use of VSAM-update or browse-activity, may fall into this category.
- Some systems could be degraded because there is an extra flow at syncpoint. An example of this would be a system with a very simple inquiry transaction workload.

In general, setting `MROLRM=YES` is recommended.

Setting `MROFSE=YES` in the front-end region prevents the mirror task in the back-end region from being terminated after syncpoint. The mirror task in the back-end region will only be terminated when the front-end task terminates.

Use of `MROFSE=YES` in the front-end region is not recommended when long-running tasks may be used to function-ship requests. This is because a SEND session will be unavailable for allocation to other tasks when unused. It might also prevent the connection from being released when contact has been lost with the back-end region, until the task terminates or issues a function-ship request.

Controlling the deletion of shipped terminal definitions (DSHIPINT and DSHIPIDL)

In a transaction routing environment, terminal definitions can be shipped from a terminal-owning region (TOR) to an application-owning region (AOR).

A shipped terminal definition in an AOR becomes redundant when:

- The terminal user logs off.
- The terminal user stops using transactions which route to the AOR.
- The TOR on which the user is signed on is shut down.
- The TOR is restarted without recovering autoinstalled terminal definitions, and the autoinstall user program DFHZATDX assigns a new set of terminal IDs to the same set of terminals.

Shipped terminal definitions which have become redundant can be deleted. Long-lasting shipped terminal definitions do not generally cause storage problems because of the relatively small amounts of storage which they occupy. However, there are other considerations, such as security, which might require that redundant shipped terminal definitions are not permitted to persist in an AOR.

The CICS-supplied transaction CRMF periodically scans the shipped terminal definitions in the AOR and flags shipped terminal definitions which it has determined to be redundant. If any redundant definitions have been identified, the CICS-supplied transaction CRMD is invoked to delete them. This processing is referred to as the CICS timeout delete mechanism.

The system initialization parameters **DSHIPINT** and **DSHIPIDL** control the amount of time for which a redundant shipped terminal definition is allowed to survive and the frequency at which shipped terminal definitions are tested for redundancy.

Effects

The **DSHIPIDL** system initialization parameter determines the period of time for which a shipped terminal definition is permitted to remain inactive before being flagged for deletion.

The **DSHIPINT** system initialization parameter determines the time interval between invocations of the CRMF transaction. CRMF examines all shipped terminal definitions to determine which of them have been idle for longer than the time interval specified by **DSHIPIDL**. If CRMF identifies any redundant terminal definitions, it invokes CRMD to delete them.

Where useful

The CRMF/CRMD processing is most effective in a transaction routing environment in which there may be shipped terminal definitions in an AOR which remain idle for considerable lengths of time.

Implementation

The maximum length of time for which a shipped terminal definition may remain idle before it can be flagged for deletion is specified by the CICS system initialization parameter **DSHIPIDL**. The interval between scans to test for idle definitions is specified by the CICS system initialization parameter **DSHIPINT**.

Both these parameters can be adjusted. Note that the revised interval to the next invocation of the timeout delete mechanism starts from the time the command is issued, not from the time it was last invoked, nor from the time of CICS startup.

Monitoring

The CICS terminal autoinstall statistics provide information on the current setting of the **DSHIPINT** and **DSHIPIDL** parameters, the number of shipped terminal definitions built and deleted, and the idle time of the shipped terminal definitions.

Limitations

The DSHIPINT value is the dominant factor in determining how long an idle shipped terminal definition survives before being deleted

After CRMF/CRMD processing has deleted a shipped terminal definition, the terminal definition must be reshipped when the terminal user next routes a transaction from the TOR to the AOR. DSHIPIDL values must not be set low enough to cause shipped terminal definitions to be frequently deleted between transactions. Such processing could incur CPU processing costs, not just for the deletion of the shipped terminal definition, but also for the subsequent reinstallation when the next transaction is routed.

Consider that a large value chosen for DSHIPINT influences the length of time that a shipped terminal definition survives. The period of time for which a shipped terminal definition remains idle before deletion is extended by an average of half of the DSHIPINT value. This occurs because a terminal, after it has exceeded the limit for idle terminals set by the DSHIPIDL parameter, must wait (for half of the DSHIPINT interval) before CRMF is scheduled to identify the terminal definition as idle and flag it for CRMD to delete. When the DSHIPINT interval is significantly longer than the DSHIPIDL interval (which is the case if the default values of 120000 for DSHIPINT and 020000 for DSHIPIDL are accepted), DSHIPINT becomes the dominant factor in determining how long an idle shipped terminal definition survives before being deleted.

Recommendations

Do not assign too low a value to DSHIPIDL. The storage occupied by the shipped terminal definitions is not normally a concern, so the default value, which specifies a maximum idle time of 2 hours is reasonable, unless other concerns (such as security) suggest that it should be shorter.

Decide whether you want to delete idle shipped terminal definitions incrementally or altogether. CRMF processing in itself causes negligible CPU overhead, so a low value for DSHIPINT may therefore be specified at little cost, if a sensible value for DSHIPIDL has been chosen. Specifying a low value for DSHIPINT so that CRMF runs relatively frequently could mean that idle terminal definitions are identified in smaller batches, so that CRMD processing required to delete them is spread out over time.

A higher value for DSHIPINT, especially if the default value of 12 hours is accepted, may mean that CRMF identifies a considerable number of idle terminal definitions, so that a larger burst of CPU is required for the CRMD processing. To ensure that this type of processing occurs during periods of low activity in the CICS region, the INQUIRE/SET/PERFORM DELETSHIPPED commands are available to help you schedule when the CRMF transaction will be invoked.

CICS VSAM and file control: Performance and tuning

This section describes performance tuning issues related to VSAM and file control.

VSAM tuning: General objectives

Tuning consists of providing a satisfactory level of service from a system at an acceptable cost. A satisfactory service for VSAM is likely to be obtained by providing adequate buffers to minimize physical I/O, and allowing several operations concurrently on the data sets.

The costs of assigning additional buffers and providing for concurrent operations on data sets are the additional virtual and real storage that is required for the buffers and control blocks.

Several factors influence the performance of VSAM data sets, including whether to use local shared resources or nonshared resources. If you compress CICS data sets to free storage and improve performance, you must do the compression while CICS is not running. To avoid shutting down CICS, use LIBRARY resources to easily take data sets offline for compression without affecting continuous availability. For details, see Using dynamic program LIBRARY resources.

A distinction is made between files and data sets:

- A *file* means a view of a data set as defined by an installed CICS file resource definition and a VSAM ACB.
- A *data set* means a VSAM sphere, including the base cluster with any associated alternate index paths.

Local shared resources (LSR) or nonshared resources (NSR)

You must decide for each file whether to use local shared resources (LSR) or nonshared resources (NSR) for its VSAM buffers and strings.

All files opened for access to a particular VSAM data set must typically use the same resource type.

Access to VSAM control intervals (CIs)

An important difference between LSR and NSR is in concurrent access to VSAM control intervals (CIs):

- In LSR, there is only one copy of a CI in storage; the second of the requests must queue until the first operation completes. LSR permits several read operations to share access to the same buffer.
- NSR allows multiple copies of a CI in storage. You can have one (and only one) string updating a CI and other strings reading different copies of the same CI.

However, updates require exclusive use of the buffer and must queue until a previous update or previous reads have completed; reads must wait for any update to finish. It is possible, therefore, that transactions with concurrent browse and update operations that run successfully with NSR might, with LSR, encounter a deadlock as the second operation waits unsuccessfully for the first to complete.

The CICS monitoring facility provides performance data for the exclusive control wait time for each user task. The performance data field 426, FCXCWTT, in the DFHFILE group, shows the elapsed time in which the task waited for exclusive control of a VSAM control interval.

Size of control intervals (CIs)

The size of the data set CIs is not a parameter specified to CICS, and is defined through VSAM AMS. However, it can have a significant performance effect on a CICS system that provides access to the control interval.

In general, direct I/O runs slightly more quickly when the data CI is small, whereas sequential I/O is quicker when the data CI is large. With NSR files, it is possible to get a good compromise by using a small data CI but also assigning extra buffers, which leads to chained and overlapped sequential I/O. However, all the extra data buffers get assigned to the first string doing sequential I/O.

VSAM functions most efficiently when its control areas are the maximum size. Set the data CI larger than the index CI. Thus, typical CI sizes for data are 4 KB to 12 KB, and for index, 1 KB to 2 KB.

In general, specify the size of the data CI for a file, but allow VSAM to select the appropriate index CI to match. An exception is if key compression turns out to be less efficient than VSAM expects it to be. In this case, VSAM might select too small an index CI size. You might find an unusually high rate of control area (CA) splits occurring with poor use of DASD space. If this problem is suspected, specify a larger index CI.

With LSR, there might be a benefit in standardizing the CI sizes, because this standardization allows more sharing of buffers between files and allows a lower total number of buffers. Conversely, there might be a benefit in giving a file unique CI sizes to prevent it from competing for buffers with other files that use the same pool.

Try to keep CI sizes at 512-bytes, 1 KB, 2 KB, or any multiple of 4 KB. Avoid unusual CI sizes like 26 KB or 30 KB. A CI size of 26 KB does not mean that physical block size is 26 KB; the physical block size is most likely to be 2 KB in this case because it is device-dependent.

Considerations for ESDS files

There are some special performance considerations when choosing a **STRINGS** value for an ESDS file.

If an ESDS is used as an add-only file (that is, it is used only in write mode to add records to the end of the file), a string number of 1 is suggested. Any string number greater than 1 can significantly affect performance, because of exclusive control conflicts that occur when more than one task attempts to write to the ESDS at the same time.

If an ESDS is used for both writing and reading, with writing, say, being 80% of the activity, it is better to define two file definitions, using one file for writing and the other for reading.

Number of buffers

Some important differences exist between LSR and NSR in the way that VSAM allocates and shares the buffers.

The set of buffers of one size in an LSR pool is called a subpool. You use up to 255 separate LSR pools for file control files. You also must decide how to distribute the data sets across the LSR pools. CICS provides separate LSR buffer pools for data and index records. If only data buffers are specified, only one set of buffers is built and used for both data and index records. The number of buffers for each subpool is controlled by the DATA and INDEX parameters of the LSRPOOL definition. You can specify precise numbers or have CICS calculate the numbers.

NSR files or data sets have their own set of buffers and control blocks. Enough buffers must be provided for each file to support the concurrent accesses specified in the STRINGS parameter for the file. VSAM enforces this requirement for NSR. NSR is not supported for transactions that use transaction isolation. File control commands using NSR files are not threadsafe.

For more information, see “Number of buffers and strings for LSR and NSR” on page 150.

Number of strings

The next decision to make is the number of concurrent accesses to be supported for each file and for each LSR pool.

You must specify VSAM strings. A string is a request to a VSAM data set requiring positioning within the data set. Each string specified results in a number of VSAM control blocks (including a placeholder) being built.

When deciding on the number of strings for a particular file, consider the maximum number of concurrent tasks. Because CICS command level does not allow more than one request to be outstanding against a particular data set from a particular task, there is no point in allowing strings for more concurrent requests.

For more information, see “Number of buffers and strings for LSR and NSR” on page 150.

Effects

LSR has significant advantages, by providing the following effects:

- More efficient use of virtual storage because buffers and strings are shared.
- Better performance because of better buffer lookaside, which can reduce I/O operations.
- Better read integrity because there is only one copy of a CI in storage.
- Self-tuning because more buffers are allocated to busy files and frequently referenced index control intervals are kept in buffers.
- Use of synchronous file requests and a UPAD exit. CA and CI splits for LSR files do not cause either the subtask or main task to wait. VSAM takes the UPAD exit while waiting for physical I/O, and processing continues for other CICS work during the CA/CI split.

  File control requests for NSR files are done asynchronously, however, and still cause the CICS main task or subtask to stop during a split.
- Support for transaction isolation.

NSR can provide the following effects:

- Specific tuning in favor of a particular data set
- Better performance for sequential operations.

Suggestions

Use LSR for all VSAM data sets except where you have one of the following situations:

- A file is active but there is no opportunity for lookaside because, for example, the file is large.
- High performance is required by the allocation of extra index buffers.
- Fast sequential browse or mass insert is required by the allocation of extra data buffers.
- Control area (CA) splits are expected for a file, and extra data buffers are to be allocated to speed up the CA splits.

If you have only one LSR pool, a particular data set cannot be isolated from others using the same pool when it is competing for strings. It can only be isolated when it is competing for buffers by specifying unique CI sizes. In general, you get more self-tuning effects by running with one large pool. It is possible to isolate busy files from the remainder or give additional buffers to a group of high performance files by using several pools. It is also possible that a highly active file has more successful buffer lookaside and less I/O if it is set up as the only file in an LSR subpool rather than using NSR. Also the use of multiple pools eases the restriction of 255 strings for each pool.

Limitations

All files with the same base data set, except read-only files with DSNSHARING(MODIFYREQS) specified in the file definition, must use either the same LSR pool, or all use NSR.

SERVREQ=REUSE files cannot use LSR.

Number of buffers and strings for LSR and NSR
The number of buffers and strings may affect your decision to use either LSR or NSR for each file.

Number of buffers for LSR and NSR

Some important differences exist between LSR and NSR in the way that VSAM allocates and shares the buffers:

LSR

Allowing CICS to calculate the LSR parameters is easy but it incurs additional processing to build the pool, when the first file that needs the LSR pool is opened. Consider the following factors if you allow CICS to calculate an LSR pool:

- CICS must read the VSAM catalog for every file that is specified to use the pool.
- The processing is increased if the data sets involved are migrated at the time that CICS performs the calculation. To enable CICS to read the VSAM catalog for each data set associated with the LSR pool, each data set must be recalled.
- Not only can a single recall cause a significant delay for the task that caused the recall, but it is a synchronous operation that delays other activities that CICS is running under the same TCB.

  You can avoid these delays by designing your SMS storage classes and migration policies to avoid CICS data sets being migrated. See z/OS DFSMShsm Storage Administration for information about setting data set migration criteria.

  CICS outputs an information message, DFHFC0989, when a recall is necessary, advising you that the consequent delay is not an error situation.
- An LSR pool calculated by CICS cannot be fine-tuned by specifying actual sizes for each buffer.
- In LSR, there is no preallocation of buffers to strings, or to particular files or data sets. When VSAM must reuse a buffer, it picks the buffer that has been referenced least recently. Strings are always shared across all data sets. Before issuing a read to disk when using LSR, VSAM first scans the buffers to check if the control interval it requires is already in storage. If so, it might not have to issue the read. This buffer lookaside can reduce I/O significantly.
- LSR files share a common pool of buffers and a common pool of strings, that is, control blocks supporting the I/O operations. Other control blocks define the file and are unique to each file or data set.

  When changing the size of an LSR pool, refer to the CICS statistics before and after the change is made. These statistics show whether the proportion of VSAM reads satisfied by buffer lookaside is changed or not.

  It is better for data and index buffers to be kept separate. If you define LSRPOOLs, you can define separate data and index buffers with the LSRPOOL definition. If you allow CICS to build the LSRPOOL, the data and index buffers will not be separate.

  Take care to include buffers of the correct size. If no buffers of the required size are present, VSAM uses the next larger buffer size.

NSR

- Enough buffers must be provided for each file to support the concurrent accesses specified in the **`STRINGS`** parameter for the file. In fact, VSAM enforces this requirement for NSR.
- Specify the number of data and index buffers for NSR using the **`DATABUFFERS`** and **`INDEXBUFFERS`** parameters of the file definition. It is important to specify sufficient index buffers. If a KSDS consists of just one control area and, therefore, just one index CI, the minimum index buffers equal to

STRINGS is sufficient. But when a KSDS is larger than this value, at least one extra index buffer must be specified so that at least the top-level index buffer is shared by all strings. Further index buffers reduce index I/O to some extent.

- Set **DATABUFFERS** to the minimum at STRINGS + 1, unless the aim is to enable overlapped and chained I/O in sequential operations or it is necessary to provide the extra buffers to speed up CA splits.
- When the file is an alternate index path to a base, the same **INDEXBUFFERS** (if the base is a KSDS) and **DATABUFFERS** settings are used for alternate index and base buffers (see "CICS calculation of LSR pool parameters" on page 154). In NSR, the minimum number of data buffers is STRNO + 1, and the minimum index buffers (for KSDSs and alternate index paths) is STRNO. One data and one index buffer are preallocated to each string, and one data buffer is kept in reserve for CA splits. If there are extra data buffers, these buffers are assigned to the first sequential operation; they can also be used to speed VSAM CA splits by permitting chained I/O operations. If there are extra index buffers, they are shared between the strings and are used to hold high-level index records, thus providing an opportunity for saving physical I/O.

Note: Always design and program transactions to avoid deadlocks. For further information, see Transaction deadlocks.

Number of strings

VSAM requires one or more strings for each concurrent file operation. For nonupdate requests (for example, a READ or BROWSE), an access using a base needs one string. An access using an alternate index needs two strings (one to hold position on the alternate index and one to hold position on the base data set). For update requests where no upgrade set is involved, a base still needs one string, and a path two strings. For update requests where an upgrade set is involved, a base needs 1+n strings and a path needs 2+n strings, where n is the number of members in the upgrade set. VSAM needs one string per upgrade set member to hold position. For each concurrent request, VSAM can reuse the n strings required for upgrade set processing because the upgrade set is updated serially.

A simple operation such as direct reading frees the string or strings immediately. However, a read for update, mass insert, or browse request retains the string or strings until a corresponding update, unlock, or end browse request is performed.

The interpretation of the **STRNO** parameter by CICS and by VSAM differs depending upon the context:

- The equivalent **STRINGS** parameter of the LSR pool definition (LSRPOOL) has the same meaning as the **STRNO** parameter in the VSAM BLDVRP macro; that is, the absolute number of strings to be allocated to the resource pool. Unless an LSR pool contains only base data sets, the number of concurrent requests that can be handled is less than the **STRINGS** value specified.
- The equivalent **STRINGS** parameter of the file definition has the same meaning as the **STRNO** parameter in the VSAM ACB for NSR files. That is, the actual number of concurrent outstanding VSAM requests that can be handled. When alternate index paths or upgrade sets are used, the actual number of strings that VSAM allocates to support these paths or upgrade sets can be greater than the **STRINGS** value specified.

For LSR, it is possible to specify the precise numbers of strings, or to have CICS calculate the numbers. The number specified in the LSR pool definition is the actual number of strings in the pool. If CICS calculates the number of strings, it derives the pool **STRINGS** from the RDO file definition. It interprets this pool, like with NSR, as the actual number of concurrent requests.

You must decide how many concurrent read, browse, update, mass insert requests, and so on, you must support.

If access to a file is read only with no browsing, there is no need to have many strings; just one might be sufficient. While a read operation only holds the VSAM string for the duration of the request, it might need to wait for the completion of an update operation on the same CI.

In general, where some browsing or updates are used, set **STRINGS** to 2 or 3 initially and check CICS file statistics regularly to see the proportion of wait-on strings encountered. Wait-on strings of up to 5% of file accesses would typically be considered acceptable. Do not try, with NSR files, to keep wait-on strings permanently zero.

CICS manages string usage for both files and LSR pools. For each file, whether it uses LSR or NSR, CICS limits the number of concurrent VSAM requests to the STRINGS= specified in the file definition. For each LSR pool, CICS also prevents more requests being concurrently made to VSAM than can be handled by the strings in the pool. If additional strings are required for upgrade-set processing at update time, CICS anticipates this requirement by reserving the additional strings at read-for-update time. If there are not enough file or LSR pool strings available, the requesting task waits until they are freed.

The CICS monitoring facility provides performance data for the VSAM string wait time for each user task. The performance data field 427, FCVSWTT, in the DFHFILE group, shows the elapsed time in which the task waited for a VSAM string. The CICS LSR pool statistics give information about the number of strings, the number of requests that waited for strings, and the maximum number of strings that were active at one time.

When deciding on the number of strings for a particular file, consider the maximum number of concurrent tasks. Because CICS command level does not allow more than one request to be outstanding against a particular data set from a particular task, there is no point in allowing strings for more concurrent requests.

If you want to distribute your strings across tasks of different types, the transaction classes can also be useful. You can use transaction class limits to control the transactions issuing the separate types of VSAM request, and for limiting the number of task types that can use VSAM strings, leaving a subset of strings available for other uses.

All placeholder control blocks must contain a field long enough for the largest key associated with any of the data sets sharing the pool. Assigning one inactive file that has a large key (primary or alternate) into an LSR pool with many strings might use excessive storage.

VSAM specifications for LSR
Define VSAM buffer allocations and string settings for LSR. Specify the resource percentile and the maximum key length for LSR.

Defining VSAM buffer allocations for LSR

For files that use local shared resources (LSR), the number of buffers to be used is not specified explicitly by file. The files share the buffers of appropriate sizes in the LSR pool. The number of buffers in the pool can either be specified explicitly using the **BUFFERS** parameter in the file definition on the CICS system definition data set (CSD), or you can leave it to CICS to calculate.

Use the **BUFFERS** parameter in CICS systems that use VSAM LSR files in CICS file control. It allows for exact definition of specific buffers for the LSR pool. The number of buffers can have a significant effect on performance. The use of many buffers can permit multiple concurrent operations, if there are the corresponding number of VSAM strings. It can also increase the chance of successful buffer lookaside with the resulting reduction in physical I/O operations.

The optimum buffer allocation involves a trade-off between increasing the I/O saving due to lookaside and increasing the real storage requirement. This optimum is different for buffers used for indexes and buffers used for data. The optimum buffer allocation for LSR is likely to be less than the buffer allocation for the same files using NSR.

The effects of these parameters can be monitored through transaction response times and data set and paging I/O rates. The settings influence both file and LSRPOOL statistics. The CICS file statistics show data set activity of the VSAM data sets. The VSAM catalog and RMF can show data set activity, I/O contention, space usage, and control interval (CI) size.

Note: From z/OS V2.2, VSAM provides a dynamic buffer addition capability that will allocate extra buffers for an LSR pool when no buffer is available for a given VSAM request. For CICS, it is preferable to retry the request rather than allow immediate expansion of an LSR pool, so dynamic buffer addition is not enabled for CICS LSR pools.

CICS provides metrics in both statistics and monitoring that can be used to tune LSR buffer allocations. Refer to the statistics for files for the number of times that tasks were queued as a result of buffer waits. Refer to exception monitoring data to identify which transactions are forced to wait and for how long.

Defining VSAM string settings for LSR

The **STRINGS** parameter is used to determine the number of strings and the number of concurrent operations possible against the LSR pool, assuming that there are buffers available. The **STRINGS** parameter can be used in CICS systems with VSAM data sets.

The number of strings is defined by the **STRNO** parameter in the file definition on the CSD, which limits the concurrent activity for that particular file.

The **STRINGS** parameter relating to files using LSR has the following effects:

- It specifies the number of concurrent requests that can be made against that specific file.
- It is used by CICS to calculate the number of strings and buffers for the LSR pool.
- It is used as the **STRINGS** value for the VSAM LSR pool.
- It is used by CICS to limit requests to the pools to prevent a VSAM short-on-strings condition (note that CICS calculates the number of strings required per request).
- A number greater than 1 can adversely affect performance for ESDS files used exclusively in write mode. With a string number greater than 1, the cost of resolving exclusive control conflicts is greater than the cost of waiting for a string. Each time exclusive control is returned, a GETMAIN is issued for a message area, followed by a second call to VSAM to obtain the owner of the control interval.

A maximum of 255 strings is allowed per pool. The effects of the **STRINGS** parameter can be seen in changes to the response times for each file entry. The CICS LSR pool statistics give information about the number of strings, the number of requests that waited for strings, and the maximum number of strings that were active at one time. The CICS performance data field 427, FCVSWTT, in the DFHFILE group, shows the elapsed time in which each user task waited for a VSAM string.

Examination of the string numbers in the CICS statistics shows that there is a two-level check on string numbers available: one at the data set level (see File control statistics in DFHSTUP reports), and one at the shared resource pool level (see LSR pool statistics in DFHSTUP reports).

Specifying the maximum key length for LSR

The **KEYLENGTH** parameter in the file definition in the CSD, or the **MAXKEYLENGTH** parameter in the LSR pool definition, specifies the size of the largest key to be used in an LSR pool. The **KEYLENGTH** parameter can be used in CICS systems with VSAM data sets. Specify the maximum key length explicitly using the **KEYLENGTH** parameter in the file definition on the CSD. Or leave it to CICS to determine the maximum key length from the VSAM catalog.

The **KEYLENGTH** parameter causes the placeholder control blocks to be built with space for the largest key that can be used with the LSR pool. Too small a specified **KEYLENGTH** prevents requests for files that have a longer key length. Set the key length so it is always as large as, or larger than, the largest key for files using the LSR pool.

Specifying the resource percentile for LSR

The **SHARELIMIT** parameter in the LSR pool definition specifies the percentage of the buffers and strings that CICS applies to the value that it calculates. The **SHARELIMIT** parameter can be used in CICS systems with VSAM data sets. The **SHARELIMIT** parameter is specified in the LSR pool definition.

The **SHARELIMIT** parameter is ignored if both the **BUFFERS** and the **STRINGS** parameters are specified for the pool. **SHARELIMIT** can be applied only to files that are allocated at initialization of the LSR pool, when the first file in the pool is opened. Therefore, it is always wise to specify the decimal **STRINGS** and **BUFFERS** for an LSR pool.

CICS calculation of LSR pool parameters
If you have not specified LSR parameters for a pool, CICS calculates the buffers and strings required for you.

To do this calculation, CICS scans all the installed file resource definitions for files specified to use the pool. For each file, it uses the following values:

- From the CICS file resource definitions:
  - The number of strings, as specified on the **STRINGS** parameter
- From the VSAM catalog:
  - The levels of index for each of these files
  - The control interval (CI) sizes
  - The keylengths for the base, the path (if it is accessed through an alternate index path), and upgrade set alternate index.

If you have specified only buffers or only strings, CICS performs the calculation for the buffers and strings you have not specified.

The following information helps you calculate the buffers required. A particular file might require more than one buffer size. For each file, CICS determines the buffer sizes required for the following components:

- The data component
- The index component, if it is a KSDS
- The data and index components for the alternate index, if it is an alternate index path
- The data and index components for each alternate index in the upgrade set, if any

The number of buffers for each file is calculated as follows:

- For data components for base and alternate index = (STRINGS= in the file resource definition entry) + 1
- For index components for base and alternate index = (STRINGS= in the file resource definition entry) + (the number of levels in the index) – 1
- For data and index components for each alternate index in the upgrade set, one buffer each

When this calculation has been done for all the files that use the pool, the total number of buffers for each size is further calculated as follows:

- The number is reduced to either 50% or the percentage specified in the **SHARELIMIT** in the LSRPOOL definition. The **SHARELIMIT** parameter takes precedence.
- If necessary, the number is increased to a minimum of three buffers.
- The number is rounded up to the nearest 4 KB boundary.

To calculate the number of strings, CICS determines the number of strings required to handle concurrent requests for each file as the sum of the following values:

- **STRINGS** parameter value for the base
- **STRINGS** parameter value for the alternate index (if it is an alternate index path)
- *n* strings if there is an upgrade set (where *n* is the number of members in the upgrade set).

Note: If the LSR pool is calculated by CICS and the data sets have been archived by hierarchical storage manager (HSM), when the first file that needs the LSR pool is opened, the startup time of a CICS system can be considerably lengthened because the data sets are needed one by one. CICS obtains the necessary catalog information, but it does not open the database. Therefore the database is still effectively archived. This problem recurs when the region is started again, and remains until the data set has been opened.

When the strings have been accumulated for all files, the total number of buffers is further calculated as follows:

- The total is reduced to either 50% or the percentage specified in the **SHARELIMIT** parameter in the LSRPOOL definition. The **SHARELIMIT** parameter takes precedence.
- The total is reduced to 255 (the maximum number of strings allowed for a pool by VSAM).
- The total is increased to the largest specified **STRINGS** value for a particular file.

The parameters calculated by CICS are shown in the CICS statistics.

Note: From z/OS V2.2, VSAM provides a dynamic buffer addition capability that will allocate extra buffers for an LSR pool when no buffer is available for a given VSAM request. For CICS, it is preferable to retry the request rather than allow immediate expansion of an LSR pool, so dynamic buffer addition is not enabled for CICS LSR pools.

CICS provides metrics in both statistics and monitoring that can be used to tune LSR buffer allocations. Refer to the statistics for files for the number of times that tasks were queued as a result of buffer waits. Refer to exception monitoring data to identify which transactions are forced to wait and for how long.

Switching data sets from RLS mode to LSR mode

There might be occasions when you must switch a data set from RLS mode to non-RLS mode (for example, to read-only LSR mode during a batch update). This switch could lead to the LSR pools that are not explicitly defined, and which CICS builds using default values, not having sufficient resources to support files switched to LSR mode after the pool has been built.

To avoid files failing to open because of the lack of adequate resources, you can specify that CICS includes files opened in RLS mode when it is calculating the size of an LSR pool using default values. To specify the inclusion of files defined with RLSACCESS(YES) in an LSR pool that is being built using values that CICS calculates, specify RLSTOLSR=YES for this system initialization parameter (RLSTOLSR=NO is the default)

See RLSTOLSR for more information about this parameter.

Data set name sharing

Data set name (DSN) sharing is the default for all VSAM data sets. It is specified as MACRF=DSN in the VSAM ACB. It causes VSAM to create a single control block structure for the strings and buffers required by all the files that relate to the same base data set cluster, whether as a path or direct to the base. VSAM makes the connection at open time of the second and subsequent files. Only if DSN sharing is specified does VSAM realize that it is processing the same data set.

This single structure offers the following benefits:

- It provides VSAM update integrity for multiple access control blocks (ACB) updating one VSAM data set.
- It allows the use of VSAM share options 1 or 2, while still permitting multiple update blocks within the CICS region.
- It saves virtual storage.

DSN sharing is the default for files using both NSR and LSR. The only exception to this default is made when opening a file that has been specified as read-only (*READ=YES* or *BROWSE=YES*) and with *DSNSHARING(MODIFYREQS)* in the file resource definition. CICS provides this option so that a file (represented by an installed file resource definition) can be isolated from other users of that same data set in a different LSR pool or in NSR by suppressing DSN sharing. CICS ignores this parameter for files with update, add, or delete options because VSAM would not then be able to provide update integrity if two file control file entries were updating the same data set concurrently.

The **NSRGROUP** parameter is associated with DSN sharing. It is used to group file resource definitions that are to refer to the same VSAM base data set. NSRGROUP=*name* does not affect data sets that use LSR.

When the first member of a group of DSN-sharing NSR files is opened, CICS must specify to VSAM the total number of strings to be allocated for all file entries in the group, with the **BSTRNO** value in the ACB. VSAM builds its control block structure at this time regardless of whether the first data set to be opened is a path or a base. CICS calculates the value of **BSTRNO** used at the time of the open by adding the STRINGS values in all the files that share the same **NSRGROUP** parameter.

If you do not provide the **NSRGROUP** parameter, the VSAM control block structure can be built with insufficient strings for later processing. Avoid this structure for performance reasons. In such a case, VSAM invokes the dynamic string addition feature to provide the extra control blocks for the strings as they are required, and the extra storage is not released until the end of the CICS run.

Alternate index considerations

For each alternate index defined with the **UPGRADE** attribute, VSAM upgrades the alternate index automatically when the base cluster is updated.

For NSR, VSAM uses a special set of buffers associated with the base cluster. This set consists of two data buffers and one index buffer, which are used serially for each alternate index associated with a base cluster. It is not possible to tune this part of the VSAM operation.

For LSR, VSAM uses buffers from the appropriate subpool.

Take care when specifying to VSAM that an alternate index is in the upgrade set. Whenever a new record is added, an existing record deleted, or a record updated with a changed attribute key, VSAM updates the alternate index in the upgrade set. This update involves extra processing and extra I/O operations.

Situations that cause extra physical I/O

Some situations that can lead to many physical I/O operations, thus affecting both response times and associated processor path lengths, are as follows:

- When a KSDS is defined with SHROPT of 4, all direct reads cause a refresh of both index and data buffers (to ensure the latest copy).
- Any sequence leading to CICS issuing ENDREQ invalidates all data buffers associated with the operation. This situation might occur when you end a get-update (without the following update), a browse (even a start browse with a no-record-found response), a mass-insert, or any get-locate from a program. If the operation is not explicitly ended by the program, CICS ends the operation at sync point or end of task.
- If there are more data buffers than strings, a start browse causes at least half the buffers to participate immediately in chained I/O. If the browse is short, the additional I/O is unnecessary.

Other VSAM definition parameters

Select free space parameters with care, because these parameters can help reduce the number of control interval (CI) and control area (CA) splits. Where records are inserted all over a VSAM data set, it is appropriate to include free space in each CI. Where the inserts are clumped, free space in each CA is required. If all the inserts take place at just a few positions in the file, allow VSAM to split the CA, and it is not necessary to specify any free space at all.

Adding records to the end of a VSAM data set does not cause CI or CA splits. Adding sequential records to anywhere but the end causes splits. An empty file with a low-value dummy key tends to reduce splits; a high-value key increases the number of splits.

VSAM specifications for NSR
Defining VSAM string settings for NSR and defining VSAM buffer allocations for NSR.

Defining VSAM buffer allocations for NSR

For files using nonshared resources (NSR), the **INDEXBUFFERS** and **DATABUFFERS** parameters define VSAM index buffers and data buffers.

The **INDEXBUFFERS** and **DATABUFFERS** parameters are defined in the file definition on the CSD. They correspond exactly to VSAM ACB parameters: **INDEXBUFFERS** is the number of index buffers, **DATABUFFERS** is the number of data buffers.

- Effects

  The number of buffers can have a significant effect on performance. The use of many buffers can permit multiple concurrent operations (if there are the corresponding number of VSAM strings) and efficient sequential operations and control area (CA) splits. Providing extra buffers for high-level index records can reduce physical I/O operations.

  Buffer allocations above the 16 MB line represent a significant part of the virtual storage requirement of most CICS systems.

- Limitations

  These parameters can be overridden by VSAM if they are insufficient for the strings specified for the VSAM data set. The maximum specification is 255. A specification greater than this value is automatically reduced to 255. Never override VSAM strings and buffers by specifying the **AMP** attribute on the DD statement.

- Limitations

  The effects of these parameters can be monitored through transaction response times and data set and paging I/O rates. The CICS file statistics show data set activity to VSAM data sets. The VSAM catalog and RMF can show data set activity, I/O contention, space usage, and control interval (CI) size.

Defining VSAM string settings for NSR

The **STRINGS** parameter is used to determine the number of concurrent operations possible against the file, and against the VSAM base cluster to which the file relates.

Use the **STRINGS** parameter in CICS systems that use VSAM NSR files in CICS file control.

The number of strings is defined by the **STRINGS** parameter in the CICS file definition on the CSD. It corresponds to the VSAM parameter in the ACB, except where a base file is opened as the first for a VSAM data set. In this case, the CICS -accumulated **BSTRNO** value is used as the **STRNO** value for the ACB.

- Effects

  The **STRINGS** parameter for files using NSR has the following effects:

  - It specifies the number of concurrent asynchronous requests that can be made against that specific file.
  - It is used as the **STRINGS** value in the VSAM ACB.
  - It is used, with the **BASE** parameter, to calculate the VSAM **BSTRNO** value.
  - A number greater than 1 can adversely affect performance for ESDS files used exclusively in write mode. With a string number greater than 1, the cost of invalidating the buffers for each of the strings is greater than the cost of waiting for the string, and there can be a significant increase in the number of VSAM EXCP requests.

  Strings represent a significant part of the virtual storage requirement of most CICS systems. With CICS, this storage is above the 16 MB line.

- Limitations

  A maximum of 255 strings can be used as the **STRNO** or **BSTRNO** values in the ACB.

- Monitoring

  The effects of the **STRINGS** parameter can be seen in changes to response times. The CICS performance data field 427, FCVSWTT, in the DFHFILE group, shows the elapsed time in which each user task waited for a VSAM string. The CICS LSR pool statistics give information about the number of strings, the number of requests that waited for strings, and the maximum number of strings that were active at one time. RMF can show I/O contention in the DASD subsystem.

Using VSAM subtasking

The optional concurrent (CO) mode TCB is used for processes that can safely run in parallel with other CICS activity such as VSAM requests. The SIT keyword **SUBTSKS** has been defined to have numeric values

(0 and 1) to specify whether there is to be a CO TCB. The system initialization parameter, **SUBTSKS=1**, defines that subtasking is to be used.

Subtasking is useful with CICS systems that use VSAM.

Only use subtasking in a multiprocessing system in a region that is limited by a single processor, but has spare capacity on other processors in the MVS image. If used in other circumstances, it can cause throughput degradation because of the dispatching of multiple tasks.

Effects

The objective of subtasks is to increase the maximum throughput of a single CICS system on multiprocessors. However, the intertask communication increases total processor utilization.

When I/O is done on subtasks, any extended response time which would cause the CICS region to stop, such as control interval (CI) or control area (CA) splitting in NSR pools, causes only the additional TCB to stop. This effect might allow more throughput in a region that has many CA splits in its file, but has to be assessed cautiously regarding the extra processing associated with using the subtask.

When the **SUBTSKS=1** system initialization parameter has been specified, the following subtasks effects are seen:

- All non-RLS VSAM file control WRITE requests to KSDS are subtasked.
- All other file control requests are never subtasked.
- Auxiliary temporary storage or intrapartition transient data requests are subtasked.
- Resource security checking requests are subtasked when the CICS main TCB (quasi-reentrant mode) exceeds approximately 70% activity.

Limitations

Subtasking can improve throughput only in multiprocessor MVS images, because additional processor cycles are required to run the extra subtask. For that reason, we do not recommend the use of this facility on uniprocessors (UP). Use it only for a region that reaches the maximum capacity of one processor in a complex system that has spare processor capacity, or has NSR files that undergo frequent CI or CA splitting.

Regions that do not contain significant amounts of VSAM data set activity (particularly update activity) do not gain from VSAM subtasking.

Application task elapsed time might increase or decrease because of conflict between subtasking processing and better use of multiprocessors. Task-related DSA occupancy increases or decreases proportionately.

Suggestions

Specify **SUBTSKS=1** only when the CICS system is run on an MVS image with two or more processors, and the peak processor utilization due to the CICS main TCB in a region exceeds about 70% of one processor, and a significant amount of I/O activity within the CICS address space is eligible for subtasking.

In this environment, the capacity of a second processor can be used to perform the I/O scheduling activity for VSAM data sets, auxiliary temporary storage, and intrapartition transient data.

The maximum system throughput of this CICS region can be increased by using the I/O subtask, but at the expense of some additional processing for communication between the subtask and the MVS task under which the transaction processing is performed. This additional processing is seldom justified unless the CICS region has reached or is approaching its throughput limit.

A TOR that is largely or exclusively routing transactions to one or more AORs has little I/O that is eligible for subtasking. It is not, therefore, a good candidate for subtasking.

An AOR is a good candidate only if a significant amount of VSAM I/O is performed within the AOR rather than being function-shipped to an FOR.

Consider subtasking for a busy FOR that often has a significant amount of VSAM I/O (but remember that DL/I processing of VSAM data sets is *not* subtasked).

VSAM subtasking for threadsafe applications using local VSAM LSR or RLS, with FCQRONLY=NO set in the SIT, is not normally recommended. Performance benefits are greater for threadsafe file control applications, by using multiple L8 or L9 TCBs.

Monitoring

CICS dispatcher domain statistics include information about the modes of TCB listed in Dispatcher TCB Modes report.

CMF data and CICS trace are fully available.

Using data tables

Data tables enable you to build, maintain, and have rapid access to data records contained in tables held in virtual storage above the 16 MB line. Therefore, they can provide a substantial performance benefit by reducing DASD I/O and path length resources. The path length to retrieve a record from a data table is shorter than the path length to retrieve a record that is already in a VSAM buffer.

You can define data tables using either the DEFINE FILE command of the CED*x* transaction or the DFHCSDUP utility program.

Effects

Using data tables has the following effects:

- After the initial data table load operation, DASD I/O can be eliminated for all user-maintained and for read-only CICS-maintained data tables (CMTs).
- Reductions in DASD I/O for CMTs are dependent on the READ/WRITE ratio. This ratio is the number of READ to WRITE calls that are experienced on the source data set, before the data table implementation. These reductions also depend on the data table READ-hit ratio: the number of READ calls that are satisfied by the table, compared with the number of requests that go against the source data set.
- CICS file control processor consumption can be reduced by up to 70%. This reduction is dependent on the file design and activity, and is given here as a general guideline only. Actual results vary from installation to installation.

For CMTs, CICS ensures the synchronization of source data set and data table changes. When a file is recoverable, the necessary synchronization is already implemented by the existing record locking. When the file is unrecoverable, there is no CICS record locking and the note string position (NSP) mechanism is used instead for all update requests. This action might have a small performance impact of additional VSAM ENDREQ requests in some instances.

Suggestions

Data tables are defined by two RDO parameters of the file definition, **TABLE** and **MAXNUMRECS** . No other changes are required.

Begin by selecting only one or two candidates. You might want to start with a CMT to simplify recovery considerations.

Select a CMT with a high READ to WRITE ratio. This information can be found in the CICS LSRPOOL statistics (see topicpage LSR pool statistics) by running a VSAM LISTCAT job.

Use READ INTO, because READ SET incurs slightly more internal processing.

Monitor your real storage consumption. If your system is already real-storage constrained, having large data tables could increase your page-in rates, and in turn could adversely affect CICS system performance. Use your normal performance tools such as RMF to look at real storage and paging rates.

Select files that have a high proportion of full keyed direct reads as CMT candidates.

Files that have a large proportion of update activity that does not require to be recovered across a restart would be better suited for user-maintained data tables.

User-maintained data tables can use the global user exit XDTRD to both modify and select records. This action could allow the user-maintained data table to contain only the information relevant to the application.

If storage isolation is specified, you must allow for the extra storage needed by the data tables to prevent CICS incurring increased paging.

Try to avoid the situation where two open files, one defined as a CMT and the other as a VSAM file, refer to the same underlying VSAM sphere (for example, both refer to the same data set name). In this situation, the VSAM file is treated almost as if it were a CMT, meaning that it gets both the advantages and disadvantages of a CMT. The advantage is much faster read and browse processing from the table created for the other file.

The disadvantages for the performance of the VSAM file are as follows:

- Updates must update both the file and the table.
- If the VSAM file refers to a path rather than to the base (that is, it uses alternate keys) it loses the advantage of fast reads.
- Requests for the VSAM file are always switched to the QR task control block (TCB) and are not processed on an open TCB.

Monitoring

Performance statistics are gathered to assess the effectiveness of the data table. They are in addition to the statistics available through the standard CICS file statistics.

The following information is recorded:

- The number of attempts to read from the table
- The number of unsuccessful read attempts
- The number of bytes allocated to the data table
- The number of records loaded into the data table
- The number of attempts to add to the table
- The number of records rejected by a user exit when they were being added to the table either during loading or through the API
- The number of attempts to add a record that failed due to the table being full (already at its maximum number of records)
- The number of attempts to update table records through rewrite requests.
- The number of attempts to delete records from the table
- The highest value that the number of records in the table has reached since it was last opened.

There are circumstances in which apparent discrepancies in the statistics might be seen, caused, for example, by the existence of in-flight updates.

Using coupling facility data tables

The API used to store and retrieve the data from a coupling facility data table (CFDT) is based on the file control API used for user-maintained data tables.

A CFDT is similar in many ways to a shared user-maintained data table. For information about shared data tables, see Introduction to shared data tables.

A CFDT is defined to a CICS region using a FILE definition with the following parameters:

- **`TABLE(CF)`**
- **`MAXNUMRECS(NOLIMIT`***number*(1 through 99999999))

- **CFDTPOOL**(*pool_name*)
- **TABLENAME**(*name*)
- **UPDATEMODEL(CONTENTION|LOCKING)**
- **LOAD(NO|YES)**

MAXNUMRECS specifies the maximum number of records that CFDT can hold.

The first CICS region to open the CFDT determines the attributes for the file. Once opened successfully, these attributes remain associated with the CFDT through the data in the coupling facility list structure. Unless this table or coupling facility list structure is deleted or altered by a CFDT server operator command, the attributes persist even after CICS and CFDT server restarts. Other CICS regions attempting to open the CFDT must have a consistent definition of the CFDT, for example using the same update model.

The CFDT server controls the coupling facility list structure and the data tables held in this structure. The parameters documented in Coupling facility data table server parameters describe how initial structure size, structure element size, and entry-to-element ratio can be specified.

The data, unlike a UMT, is not kept in a dataspace in an MVS image and controlled by a CICS region, but kept in a coupling facility list structure. Control is shared between CFDT server regions. A CICS region requesting access to a CFDT communicates with a CFDT server region running in the same MVS image, using the MVS authorized cross-memory (AXM) server environment. The same technique is used by CICS temporary storage servers.

CFDTs are useful for informal shared data. Uses could include a sysplex-wide shared scratchpad, look-up tables of telephone numbers, and creating a subset of customers from a customer list. Compared with existing methods of sharing data of this kind, such as shared data tables, shared temporary storage or RLS files, CFDTs offer some distinct advantages:

- If the data is frequently accessed for modification, CFDT provides superior performance compared with function-shipped UMT requests, or using an RLS file
- CFDT-held data can be recoverable within a CICS transaction. Recovery of the structure is not supported, but the CFDT record is recoverable in the event of a unit of work failure, a CICS region failure, a CFDT server failure, or an MVS failure (that is, updates made by units of work that were in-flight at the time of the failure are backed out). Such recoverability is not provided by shared temporary storage.

Locking model and contention model

There are two models of coupling facility data table, a contention model or locking model.

Locking model. Records held in a coupling facility list structure are marked as locked by updating the adjunct area associated with the coupling facility list structure element that holds the data. Locking a record requires an additional coupling facility access to set the lock, having determined on the first access that the data was not already locked.

If, however, there is an update conflict, a number of extra coupling facility accesses are needed, as described in the following sequence of events:

1. The request that encounters lock contention is initially rejected.
2. The requester modifies the locked record adjunct area to express an interest in it. This area is a second extra coupling facility access for the lock waiter.
3. The lock owner has the update rejected because the record adjunct area has been modified, requiring the CICS region to read and try the update again. This results in two extra coupling facility accesses.
4. The lock owner sends a lock release notification message. If the lock was requested by a different server, this results in a coupling facility access to write a notification message to the other server and a coupling facility access to read it on the other side.

Contention model. The contention update model uses the entry version number to track changes. The entry version number is changed each time the record is updated. This change allows an update request to check that the record has not been altered since its copy of the record was acquired.

When an update conflict occurs, additional coupling facility accesses are needed:

- The request that detects that the record has changed is initially rejected and a CHANGED response is sent.
- The application receiving the response has to decide whether to try the request again.

Using the contention model, an exception condition (CHANGED) notifies an application that a rewrite following a read for update, or a delete following a read for update, needs to be tried again because the copy of the record in the table has been updated by another task before the rewrite or delete could be performed. The contention model does not lock a record, but uses the version number of the table entry for the record to check that it has not been altered. If the version of this record on rewrite or delete is not the same as when the original read for update was performed, the CHANGED condition is returned.

The locking model causes records to be locked following a read for update request so that multiple updates cannot occur.

A contention model CFDT is unrecoverable. A locking model CFDT can be recoverable or unrecoverable. For an unrecoverable locking model, CFDT locks are held until a read for update sequence is completed by a rewrite, a delete or an unlock request, but not until the next syncpoint. Changes are not backed out if a unit of work fails. In the recoverable case, locks are held until syncpoint, and the CFDT record is recoverable in the event of a unit of work failure, CICS region failure, CFDT server failure, or MVS failure.

The relative cost of using update models and recovery is related to the amount of coupling facility accesses needed to support a request. Contention requires the least number of accesses, but if the data is changed, additional programming and coupling facility accesses would be needed to handle this condition. Locking requires more coupling facility accesses, but does mean that a request does not need to be tried again, whereas repeat tries can be required when using the contention model. Recovery also requires further coupling facility accesses, because the recovery data is kept in the coupling facility list structure.

The following table shows the amount of coupling facility accesses needed to support the CFDT request types by update model.

Table 14. Coupling facility access by request type and update model

Request description	**Contention**	**Locking**	**Recoverable**
Open, Close	3	3	6
Read, Point	1	1	1
Write new record	1	1	2
Read for Update	1	2	2
Unlock	0	1	1
Rewrite	1	1	3
Delete	1	1	2
Delete by key	1	2	3
Syncpoint	0	0	3
Lock WAIT	0	2	2
Lock POST	0	2	2
Cross-system POST	0	2 per waiting server	2 per waiting server

For a description of how to define a coupling facility data table (CFDT), and start a coupling facility data table server, see Defining a coupling facility data table pool.

Effects

In a test that compared the use of a CFDT with a function-shipped UMT between 2 CICS regions running on different MVS members of a sysplex, it was found that overall CPU utilization was reduced by over 40% by using CFDTs. Some general observations that might be useful are as follows:

- Access to CFDT records of 4094 bytes or less (4096 K or 4 K including 2 bytes of prefix data) are handled as synchronous coupling facility requests by the CFDT server. Requests for records of greater than 4 K bytes are made asynchronously. These asynchronous accesses cost a little more in CPU usage and response time. In a benchmark test comparing the same transaction rates (337 per second) but different record sizes, the less than 4 K CFDT workload took 41.7% less CPU than the UMT equivalent. The greater than 4 K CFDT workload took 41.1% less CPU with no measurable degradation of response time.
- Using the contention model requires the least coupling facility accesses but because the CHANGED condition needs to be handled and might need to be tried again, maximum benefit is derived when there are few CHANGED conditions. These occurrences are reported in the CICS statistics which follow.
- If the CFDT records are 63 bytes or less in length, the record data is stored in the entry adjunct area of the coupling facility list structure, which gives improved performance when using the contention update mode.
- Using the locking model with recovery is the most costly mode of CFDT operation. Not only does this require more coupling facility accesses, but the CFDT server is also acting as a resource manager, coordinating the committal of updates with the requesting CICS region. In a benchmark test involving the READ/UPDATE and REWRITE of CFDT records at a transaction rate of 168 per second, there was no significant difference in CPU utilization between transactions using contention and locking CFDTs. However, if the CFDT was defined as recoverable, the CPU utilization of the same transactions increased by approximately 15%.

Suggestions

Choose an appropriate use of a CFDT. For example, for cross-system, recoverable scratchpad storage, where shared TS does not give the required functionality, or VSAM RLS incurs too much processing.

A large file requires a large amount of coupling facility storage to contain it. Smaller files are better CFDT candidates (unless your application is written to control the number of records held in a CFDT).

The additional cost of using a locking model compared with a contention model is not great. Considering that using the contention model might need application changes if you are using an existing program, locking is probably the best choice of update model for your CFDT. If coupling facility accesses are critical to you, they are minimized by the contention model.

Recovery costs slightly more in CPU usage and in coupling facility utilization.

Allow for expansion when sizing the CFDT. The amount of coupling facility storage a structure occupies can be increased dynamically up to the maximum defined in the associated coupling facility resource management (CFRM) policy with a **`SETXCF ALTER`** command. The **`MAXTABLES`** value defined to the CFDT server allows for expansion. Therefore, consider setting it to a value higher than your initial requirements. If a CFDT does become full, its capacity can be increased using the CFDT operator command SET `TABLE=name,MAXRECS=n`.

Monitor the utilization of the CFDT regularly both through CICS and CFDT statistics and RMF. Check that the size of the structure is reasonable for the amount of data it contains. A maximum used of 80% is a reasonable target. Define a maximum coupling facility list structure size in the CFRM policy definition greater than the initial allocation size specified by the **`POOLSIZE`** parameter in the CFDT server startup parameters. This setting enables you to enlarge the structure dynamically with a **`SETXCF ALTER`** command if the structure does fill, in extraordinary circumstances.

Ensure that the AXMPGANY storage pool is large enough. This pool can be increased by increasing the REGION size for the CFDT server. Insufficient AXMPGANY storage might lead to 80A abends in the CFDT server.

Monitoring

Both CICS and the CFDT server produce statistics records. These records are described in Coupling Facility Data Table Pools report in Reference.

The CICS file statistics report the various requests by type issued against each CFDT. They also report if the CFDT becomes full, the highest number of records held and a Changed Response/Lock Wait count. This last item can be used to determine for a contention CFDT how many times the CHANGED condition was returned. For a locking CFDT, this count reports how many times requests were made to wait because the requested record was already locked.

For more information, see Data Tables reports.

Coupling facility data table statistics

The coupling facility data table (CFDT) server reports comprehensive statistics on both the coupling facility list structure it uses and the data tables it supports. It also reports on the storage that is used within the CFDT region by its AXM routines (the AXMPGLOW and AXMPGANY areas). This data can be written to SMF and can also be produced automatically at regular intervals, or by operator commands to the job log of the CFDT server.

The CFDT statistics are calculated from information that is returned by recent coupling facility requests. If the relevant information was not accessed recently by the current server, the statistics are not necessarily accurate. The number of tables and the number of lists are updated each time that the server opens or closes a table, but at other times they might not be updated. The element and entry counts are updated on successful completion of most types of coupling facility access request.

The following code is an example of coupling facility statistics that are produced by a CFDT server:

```
DFHCF0432I Table pool statistics for coupling facility list structure DFH
CFLS_PERFCFT2:
 Structure:    Size  Max size Elem size   Tables:   Current   Highest
             12288K    30208K       256                   4         4
 Lists:       Total    In use  Max used   Control      Data
                137        41        41        37         4
               100%       30%       30%       27%        3%
 Entries:     Total    In use  Max used      Free  Min free   Reserve
               3837      2010      2010      1827      1827       191
               100%       52%       52%       48%       48%        5%
 Elements:    Total    In use  Max used      Free  Min free   Reserve
              38691     12434     12434     26257     26257      1934
               100%       32%       32%       68%       68%        5%
```

This example shows the amount of space that is currently used in a coupling facility list structure (Size) and the maximum size (Max size) defined for the structure. The structure size can be increased by using a **SETXCF ALTER** command. The number of lists that are defined is determined by the **MAXTABLES** parameter for the CFDT server. In this example, the structure can support up to 100 data tables (and 37 lists for control information).

Each list entry comprises a fixed-length section for entry controls and a variable number of data elements. The size of these elements is fixed when the structure is first allocated in the coupling facility, and is specified to the CFDT server by the **ELEMSIZE** parameter. The allocation of coupling facility space between entry controls and elements is altered automatically and dynamically by the CFDT server to improve space utilization if necessary.

The reserve space is used to ensure that rewrites and server internal operations can still function if a structure fills with user data.

The amount of storage that is used with the CFDT region to support AXM requests is also reported. For example:

```
AXMPG0004I Usage statistics for storage page pool AXMPGANY:
     Size    In Use  Max Used      Free  Min Free
   30852K      636K      672K    30216K    30180K
     100%        2%        2%       98%       98%
               Gets     Frees   Retries     Fails
               3122      3098         0         0
AXMPG0004I Usage statistics for storage page pool AXMPGLOW:
      Size    In Use  Max Used      Free  Min Free
      440K       12K       12K      428K      428K
      100%        3%        3%       97%       97%
                Gets     Frees   Retries     Fails
                   3         0         0         0
```

The CFDT server uses storage in its own region for AXMPGANY and AXMPGLOW storage pools. AXMPGANY accounts for most of the available storage above 16 MB in the CFDT region. The AXMPGLOW refers to 24 bit addressed storage (below 16 MB) and accounts for only 5% of this storage in the CFDT region. The CFDT server has a small requirement for such storage.

Local shared resources (LSR) or nonshared resources (NSR)

You must decide for each file whether to use local shared resources (LSR) or nonshared resources (NSR) for its VSAM buffers and strings.

All files opened for access to a particular VSAM data set must typically use the same resource type.

Access to VSAM control intervals (CIs)

An important difference between LSR and NSR is in concurrent access to VSAM control intervals (CIs):

- In LSR, there is only one copy of a CI in storage; the second of the requests must queue until the first operation completes. LSR permits several read operations to share access to the same buffer.
- NSR allows multiple copies of a CI in storage. You can have one (and only one) string updating a CI and other strings reading different copies of the same CI.

However, updates require exclusive use of the buffer and must queue until a previous update or previous reads have completed; reads must wait for any update to finish. It is possible, therefore, that transactions with concurrent browse and update operations that run successfully with NSR might, with LSR, encounter a deadlock as the second operation waits unsuccessfully for the first to complete.

Size of control intervals (CIs)

The size of the data set CIs is not a parameter specified to CICS, and is defined through VSAM AMS. However, it can have a significant performance effect on a CICS system that provides access to the control interval.

In general, direct I/O runs slightly more quickly when the data CI is small, whereas sequential I/O is quicker when the data CI is large. With NSR files, it is possible to get a good compromise by using a small data CI but also assigning extra buffers, which leads to chained and overlapped sequential I/O. However, all the extra data buffers get assigned to the first string doing sequential I/O.

VSAM functions most efficiently when its control areas are the maximum size. Set the data CI larger than the index CI. Thus, typical CI sizes for data are 4 KB to 12 KB, and for index, 1 KB to 2 KB.

In general, specify the size of the data CI for a file, but allow VSAM to select the appropriate index CI to match. An exception is if key compression turns out to be less efficient than VSAM expects it to be. In this case, VSAM might select too small an index CI size. You might find an unusually high rate of control area (CA) splits occurring with poor use of DASD space. If this problem is suspected, specify a larger index CI.

With LSR, there might be a benefit in standardizing the CI sizes, because this standardization allows more sharing of buffers between files and allows a smaller total number of buffers. Conversely, there might be a benefit in giving a file unique CI sizes to prevent it from competing for buffers with other files that use the same pool.

Try to keep CI sizes at 512 bytes, 1 KB, 2 KB, or any multiple of 4 KB. Avoid unusual CI sizes like 26 KB or 30 KB. A CI size of 26 KB does not mean that physical block size is 26 KB; the physical block size is most likely to be 2 KB in this case because it is device-dependent.

Number of buffers for LSR and NSR

Some important differences exist between LSR and NSR in the way that VSAM allocates and shares the buffers:

LSR

The set of buffers of one size in an LSR pool is called a *subpool*. You use up to 255 separate LSR pools for file control files. You also must decide how to distribute the data sets across the LSR pools. CICS provides separate LSR buffer pools for data and index records. If only data buffers are specified, only one set of buffers is built and used for both data and index records. The number of buffers for each subpool is controlled by the **DATA** and **INDEX** parameters of the LSRPOOL definition. You can specify precise numbers or have CICS calculate the numbers.

Allowing CICS to calculate the LSR parameters is easy but it incurs additional processing to build the pool, when the first file that needs the LSR pool is opened. Consider the following factors if you allow CICS to calculate an LSR pool:

- CICS must read the VSAM catalog for every file that is specified to use the pool.
- The processing is increased if the data sets involved are migrated at the time that CICS performs the calculation. To enable CICS to read the VSAM catalog for each data set associated with the LSR pool, each data set must be recalled.
- Not only can a single recall cause a significant delay for the task that caused the recall, but it is a synchronous operation that delays other activities that CICS is running under the same TCB.

  You can avoid these delays by designing your SMS storage classes and migration policies to avoid CICS data sets being migrated. See z/OS DFSMShsm Storage Administration for information about setting data set migration criteria.

  CICS outputs an information message, DFHFC0989, when a recall is necessary, advising you that the consequent delay is not an error situation.
- An LSR pool calculated by CICS cannot be fine-tuned by specifying actual sizes for each buffer.
- In LSR, there is no preallocation of buffers to strings, or to particular files or data sets. When VSAM must reuse a buffer, it picks the buffer that has been referenced least recently. Strings are always shared across all data sets. Before issuing a read to disk when using LSR, VSAM first scans the buffers to check if the control interval it requires is already in storage. If so, it might not have to issue the read. This buffer lookaside can reduce I/O significantly.
- LSR files share a common pool of buffers and a common pool of strings, that is, control blocks supporting the I/O operations. Other control blocks define the file and are unique to each file or data set.

  When changing the size of an LSR pool, refer to the CICS statistics before and after the change is made. These statistics show whether the proportion of VSAM reads satisfied by buffer lookaside is changed or not.

  In general, you would expect to benefit more by having extra index buffers for lookaside, and less by having extra data buffers. This benefit is a further reason for standardizing LSR data and index CI sizes, so that one subpool does not have a mix of index and data CIs in it.

  Because data and index buffers are specified separately with the LSRPOOL definition, there is no requirement to use CI size to differentiate between data and index values.

  Take care to include buffers of the correct size. If no buffers of the required size are present, VSAM uses the next larger buffer size.

NSR

- Enough buffers must be provided for each file to support the concurrent accesses specified in the **STRINGS** parameter for the file. In fact, VSAM enforces this requirement for NSR.
- Specify the number of data and index buffers for NSR using the **DATABUFFERS** and **INDEXBUFFERS** parameters of the file definition. It is important to specify sufficient index buffers. If a KSDS consists of just one control area and, therefore, just one index CI, the minimum index buffers equal to **STRINGS** is sufficient. But when a KSDS is larger than this value, at least one extra index buffer must be specified so that at least the top-level index buffer is shared by all strings. Further index buffers reduce index I/O to some extent.

- Set **DATABUFFERS** to the minimum at STRINGS + 1, unless the aim is to enable overlapped and chained I/O in sequential operations or it is necessary to provide the extra buffers to speed up CA splits.
- When the file is an alternate index path to a base, the same **INDEXBUFFERS** (if the base is a KSDS) and **DATABUFFERS** settings are used for alternate index and base buffers (see "CICS calculation of LSR pool parameters" on page 154). In NSR, the minimum number of data buffers is STRNO + 1, and the minimum index buffers (for KSDSs and alternate index paths) is STRNO. One data and one index buffer are preallocated to each string, and one data buffer is kept in reserve for CA splits. If there are extra data buffers, these buffers are assigned to the first sequential operation; they can also be used to speed VSAM CA splits by permitting chained I/O operations. If there are extra index buffers, they are shared between the strings and are used to hold high-level index records, thus providing an opportunity for saving physical I/O.
- NSR files or data sets have their own set of buffers and control blocks.

Note: NSR is not supported for transactions that use transaction isolation. File control commands using NSR files are not threadsafe.

Always design and program transactions to avoid deadlocks. For further information, see Transaction deadlocks.

Number of strings

The next decision to make is the number of concurrent accesses to be supported for each file and for each LSR pool.

You must specify VSAM strings. A string is a request to a VSAM data set requiring positioning within the data set. Each string specified results in a number of VSAM control blocks (including a placeholder) being built.

VSAM requires one or more strings for each concurrent file operation. For nonupdate requests (for example, a READ or BROWSE), an access using a base needs one string. An access using an alternate index needs two strings (one to hold position on the alternate index and one to hold position on the base data set). For update requests where no upgrade set is involved, a base still needs one string, and a path two strings. For update requests where an upgrade set is involved, a base needs 1+n strings and a path needs 2+n strings, where n is the number of members in the upgrade set. VSAM needs one string per upgrade set member to hold position. For each concurrent request, VSAM can reuse the n strings required for upgrade set processing because the upgrade set is updated serially.

A simple operation such as direct reading frees the string or strings immediately. However, a read for update, mass insert, or browse request retains the string or strings until a corresponding update, unlock, or end browse request is performed.

The interpretation of the **STRNO** parameter by CICS and by VSAM differs depending upon the context:

- The equivalent **STRINGS** parameter of the LSR pool definition (LSRPOOL) has the same meaning as the **STRNO** parameter in the VSAM BLDVRP macro; that is, the absolute number of strings to be allocated to the resource pool. Unless an LSR pool contains only base data sets, the number of concurrent requests that can be handled is less than the **STRINGS** value specified.
- The equivalent **STRINGS** parameter of the file definition has the same meaning as the **STRNO** parameter in the VSAM ACB for NSR files. That is, the actual number of concurrent outstanding VSAM requests that can be handled. When alternate index paths or upgrade sets are used, the actual number of strings that VSAM allocates to support these paths or upgrade sets can be greater than the **STRINGS** value specified.

For LSR, it is possible to specify the precise numbers of strings, or to have CICS calculate the numbers. The number specified in the LSR pool definition is the actual number of strings in the pool. If CICS calculates the number of strings, it derives the pool **STRINGS** from the RDO file definition. It interprets this pool, like with NSR, as the actual number of concurrent requests.

You must decide how many concurrent read, browse, update, mass insert requests, and so on, you must support.

If access to a file is read only with no browsing, there is no need to have many strings; just one might be sufficient. While a read operation only holds the VSAM string for the duration of the request, it might need to wait for the completion of an update operation on the same CI.

In general, where some browsing or updates are used, set **STRINGS** to 2 or 3 initially and check CICS file statistics regularly to see the proportion of wait-on strings encountered. Wait-on strings of up to 5% of file accesses would typically be considered acceptable. Do not try, with NSR files, to keep wait-on strings permanently zero.

CICS manages string usage for both files and LSR pools. For each file, whether it uses LSR or NSR, CICS limits the number of concurrent VSAM requests to the STRINGS= specified in the file definition. For each LSR pool, CICS also prevents more requests being concurrently made to VSAM than can be handled by the strings in the pool. If additional strings are required for upgrade-set processing at update time, CICS anticipates this requirement by reserving the additional strings at read-for-update time. If there are not enough file or LSR pool strings available, the requesting task waits until they are freed. The CICS statistics give details of the string waits.

When deciding on the number of strings for a particular file, consider the maximum number of concurrent tasks. Because CICS command level does not allow more than one request to be outstanding against a particular data set from a particular task, there is no point in allowing strings for more concurrent requests.

If you want to distribute your strings across tasks of different types, the transaction classes can also be useful. You can use transaction class limits to control the transactions issuing the separate types of VSAM request, and for limiting the number of task types that can use VSAM strings, leaving a subset of strings available for other uses.

All placeholder control blocks must contain a field long enough for the largest key associated with any of the data sets sharing the pool. Assigning one inactive file that has a large key (primary or alternate) into an LSR pool with many strings might use excessive storage.

Considerations for ESDS files

There are some special performance considerations when choosing a **STRINGS** value for an ESDS file.

If an ESDS is used as an add-only file (that is, it is used only in write mode to add records to the end of the file), a string number of 1 is suggested. Any string number greater than 1 can significantly affect performance, because of exclusive control conflicts that occur when more than one task attempts to write to the ESDS at the same time.

If an ESDS is used for both writing and reading, with writing, say, being 80% of the activity, it is better to define two file definitions, using one file for writing and the other for reading.

Effects

LSR has significant advantages, by providing the following effects:

- More efficient use of virtual storage because buffers and strings are shared.
- Better performance because of better buffer lookaside, which can reduce I/O operations.
- Better read integrity because there is only one copy of a CI in storage.
- Self-tuning because more buffers are allocated to busy files and frequently referenced index control intervals are kept in buffers.
- Use of synchronous file requests and a UPAD exit. CA and CI splits for LSR files do not cause either the subtask or main task to wait. VSAM takes the UPAD exit while waiting for physical I/O, and processing continues for other CICS work during the CA/CI split.

  File control requests for NSR files are done asynchronously, however, and still cause the CICS main task or subtask to stop during a split.
- Support for transaction isolation.

NSR can provide the following effects:

- Specific tuning in favor of a particular data set
- Better performance for sequential operations.

Suggestions

Use LSR for all VSAM data sets except where you have one of the following situations:

- A file is active but there is no opportunity for lookaside because, for example, the file is large.
- High performance is required by the allocation of extra index buffers.
- Fast sequential browse or mass insert is required by the allocation of extra data buffers.
- Control area (CA) splits are expected for a file, and extra data buffers are to be allocated to speed up the CA splits.

If you have only one LSR pool, a particular data set cannot be isolated from others using the same pool when it is competing for strings. It can only be isolated when it is competing for buffers by specifying unique CI sizes. In general, you get more self-tuning effects by running with one large pool. It is possible to isolate busy files from the remainder or give additional buffers to a group of high performance files by using several pools. It is also possible that a highly active file has more successful buffer lookaside and less I/O if it is set up as the only file in an LSR subpool rather than using NSR. Also the use of multiple pools eases the restriction of 255 strings for each pool.

Limitations

All files with the same base data set, except read-only files with DSNSHARING(MODIFYREQS) specified in the file definition, must use either the same LSR pool, or all use NSR.

SERVREQ=REUSE files cannot use LSR.

Coupling facility data tables

The CPU instruction data provided here was obtained using a 9672-R55 system.

Two tables are provided:

- The first for record lengths that result in synchronous coupling facility accesses (less than 4K)
- The second for record lengths that result in asynchronous coupling facility accesses (greater than 4K).

Note that asynchronous requests take more CPU time to process. The response times are also slightly longer than for synchronous requests. CPU instructions per API call for record lengths less than 4 K are as follows:

API CALL	**CONTENTION**	**LOCKING**	**RECOVERABLE**
READ	11.8	11.8	11.8
READ/UPDATE	12.0	22.2	22.4
REWRITE	19.5	24.0	33.0
WRITE	8.0	8.0	13.0
DELETE	7.0	11.0	16.5

CPU instructions per API call for record lengths greater than 4 K are as follows:

API CALL	**CONTENTION**	**LOCKING**	**RECOVERABLE**
READ	15.3	15.3	15.3
READ/UPDATE	15.0	25.7	25.9
REWRITE	23.0	27.5	36.5

API CALL	CONTENTION	LOCKING	RECOVERABLE
WRITE	11.5	11.5	16.5
DELETE	10.5	14.5	20.0

Using VSAM record-level sharing

VSAM record-level sharing (RLS) is a VSAM data set access mode, introduced in DFSMS, and supported by CICS. RLS enables VSAM data to be shared, with full update capability, between many applications running in many CICS regions. With RLS, CICS regions that share VSAM data sets can reside in one or more MVS images within an MVS sysplex.

RLS also provides some benefits when data sets are shared between CICS regions and batch jobs.

RLS involves the use of the following components:

A VSAM server, subsystem SMSVSAM

This subsystem runs in its own address space to provide the RLS support required by CICS application owning regions (AORs) and batch jobs, within each MVS image in a Parallel Sysplex® environment.

The CICS interface with SMSVSAM is through an access control block (ACB), and CICS registers with this ACB to open the connection. Unlike the DB2 and DBCTL database manager subsystems, which require user action to open the connections, if you specify **RLS=YES** as a system initialization parameter, CICS registers with the SMSVSAM control ACB automatically during CICS initialization.

A CICS region must open the control ACB to register with SMSVSAM before it can open any file ACB in RLS mode. Each normal file ACB remains the interface for file access requests.

Sharing-control data sets

VSAM requires a number of these data sets for RLS control. The VSAM sharing control data sets are logically partitioned, linear data sets. They can be defined with secondary extents, but all the extents for each data set must be on the same volume.

Define at least three sharing-control data sets. VSAM requires two active data sets for use in duplexing mode, and a third data set as a spare in case one of the active data sets fails.

For more information about sharing-control data sets, and for a JCL example to define them, see z/OS DFSMSdfp Storage Administration.

Common buffer pools and control blocks

For data sets accessed in non-RLS mode, VSAM control blocks and buffers (local shared resources (LSR) pools) are located in each CICS address space. They are thus not available to batch programs, and not even to another CICS region.

With RLS, all the control blocks and buffers are allocated in an associated data space of the SMSVSAM server. This structure provides one large buffer pool for each MVS image, which can be shared by all CICS regions that are connected to the SMSVSAM server, and also by batch programs. Buffers in this data space are created and freed automatically.

DFSMS provides the **`RLS_MAX_POOL_SIZE`** parameter that you can specify in the IGDSMS*xx* SYS1.PARMLIB member. There are no other tuning parameters for RLS as there are with LSR pools. Management of the RLS buffers is fully automatic.

Using RLS with entry-sequenced data sets (ESDS) can have a negative effect on the availability of the data set when you are adding records using multiple tasks from multiple regions. This is because adding a record requires an exclusive add-to-end lock in order to perform the write. If a CICS region fails while writing to an ESDS, the data set might be locked until the CICS region is restarted.

To use RLS access mode with CICS files, do the following tasks:

1. Define the required sharing control data sets.
2. Specify the **`RLS_MAX_POOL_SIZE`** parameter in the IGDSMS*xx* SYS1.PARMLIB member.
3. Ensure that the SMSVSAM server is started in the MVS image for which you want RLS support.

4. Specify the system initialization parameter **RLS=YES**. This parameter enables CICS to register automatically with the SMSVSAM server by opening the control ACB during CICS initialization. RLS support cannot be enabled dynamically later if you start CICS with RLS=NO.
5. Ensure that the data sets you plan to use in RLS-access mode are defined, using Access Method Services (AMS), with the required recovery attributes using the **LOG** and **LOGSTREAMID** parameters on the IDCAMS DEFINE statements. If you use an existing data set that was defined without these attributes, redefine the data set with these attributes specified.
6. Specify **RLSACCESS(YES)** on the file resource definition.

CICS can use three different modes to access a VSAM file. These are non-shared resources (NSR) mode, local shared resources (LSR) mode, and record-level sharing (RLS) mode. (CICS does not support VSAM global shared resources (GSR) access mode.) The mode of access is not a property of the data set itself, it is a property of the way that the data set is opened. This means that a given data set can be opened by a user in NSR mode at one time, and RLS mode at another. The term non-RLS mode is used as a generic term to refer to the NSR or LSR access modes supported by CICS. Mixed-mode operation means a data set that is opened in RLS mode and a non-RLS mode concurrently, by different users.

Although data sets can be open in different modes at different times, all the data sets within a VSAM sphere must normally be opened in the same mode. A sphere is the collection of all the components—the base, index, any alternate indexes, and alternate index paths—associated with a given VSAM base data set. However, VSAM does permit mixed-mode operations on a sphere by different applications, subject to some CICS restrictions.

Effects

The tests and measurements described were carried out using RLS with key-sequenced data sets (KSDS). As described earlier in this topic, RLS is not suggested for use with entry-sequenced data sets (ESDS), as it can cause problems with performance and availability when you are adding records.

There is an increase in CPU costs when using RLS compared with function-shipping to a file-owning region (FOR) using MRO. When measuring CPU usage using the standard DSW workload, the following comparisons were seen:

- Switching from local file access to function-shipping across MRO cross-memory (XM) connections incurred an increase of 7.02 ms per transaction in a single CPC.
- Switching from MRO XM to RLS incurred an increase of 8.20 ms per transaction in a single CPC.
- Switching from XCF/MRO to RLS using two CPUs produced a *reduction* of 2.39 ms per transaction.
- Switching from RLS using one CPC to RLS using two CPUs there was no appreciable difference.

In terms of response times, the performance measurements showed that:

- Function-shipping with MRO XM is better than RLS, but this choice restricts function-shipping to within one MVS image, and prevents full exploitation of a Parallel Sysplex with multiple MVS images or multiple CPUs.
- RLS is better than function-shipping with XCF/MRO, when the FOR is running in a different MVS image from the AOR.

However, performance measurements on their own do not tell the whole story, and do not take account of other factors; for example:

- Because more applications need to share the same VSAM data, the load increases on the single FOR to a point where the FOR can become a throughput bottleneck. The FOR is restricted, because of the CICS internal architecture, to the use of a single TCB for user tasks, which means that a CICS region generally does not use multiple CPUs
- Session management becomes more difficult as more AORs connect to the FOR.

These negative aspects of using an FOR are resolved by using RLS, which provides the scalability lacking in a FOR.

Monitoring

Using RLS-access mode for VSAM files involves SMSVSAM as well as the CICS region issuing the file control requests. This choice means monitoring the performance of both CICS and SMSVSAM to get the full picture, using a combination of CICS performance monitoring data and SMF Type 42 records written by SMSVSAM:

CICS monitoring
For RLS access, CICS writes performance class records to SMF containing:

- RLS CPU time on the SMSVSAM SRB
- RLS wait time

SMSVSAM SMF data
SMSVSAM writes Type 42 records, subtypes 15, 16, 17, 18, and 19, providing information about coupling facility cache sets, structures, locking statistics, CPU usage, and so on. This information can be analyzed using RMF III post processing reports.

The following code is an example of the JCL that you can use to obtain a report of SMSVSAM data:

```
//RMFCF     JOB (accounting_information),MSGCLASS=A,MSGLEVEL=(1,1),CLASS=A
//STEP1     EXEC PGM=IFASMFDP
//DUMPIN    DD DSN=SYS1.MV2A.MANA,DISP=SHR
//DUMPOUT   DD DSN=&&SMF,UNIT=SYSDA,
//          DISP=(NEW,PASS),SPACE=(CYL,(10,10))
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
  INDD(DUMPIN,OPTIONS(DUMP))
  OUTDD(DUMPOUT,TYPE=000:255))
//POST      EXEC PGM=ERBRMFPP,REGION=0M
//MFPINPUT  DD DSN=&&SMF,DISP=(OLD,PASS)
//SYSUDUMP  DD SYSOUT=A
//SYSOUT    DD SYSOUT=A
//SYSPRINT  DD SYSOUT=A
//MFPMSGDS  DD SYSOUT=A
//SYSIN     DD *
 NOSUMMARY
 SYSRPTS(CF)
 SYSOUT(A)
 REPORTS(XCF)
/*
```

CICS file control statistics contain the typical information about the numbers of file control requests issued in the CICS region. They also identify which files are accessed in RLS mode, and provide counts of RLS timeouts and EXCP counts for RLS files. They do not contain any information about the SMSVSAM server, or its buffer usage, or its accesses to the coupling facility.

Threadsafe file control applications

By default, CICS forces file control commands issued by threadsafe applications to run on the QR TCB. If you change the system initialization parameter **FCQRONLY** to specify NO, file control commands for local VSAM LSR or RLS files can run on an L8 or L9 TCB.

Using threadsafe file control can result in significant throughput improvements in CICS regions that have multiple processors available. Tasks currently running on an L8 or L9 TCB do not switch back to the QR TCB when the file control command is issued, but continue to run on the L8 or L9 TCB. These tasks benefit from greater concurrency and increased task throughput. Processor reduction and faster throughput is noticeable for threadsafe applications that combine file control commands with DB2 or WebSphere MQ requests.

To benefit from threadsafe file control, applications must meet the following requirements:

- The program resource must be defined with CONCURRENCY(THREADSAFE) or CONCURRENCY(REQUIRED).
- The file control commands that are issued must be to a local VSAM LSR or RLS file.

- The system initialization parameter **FCQRONLY=NO** must be specified for the CICS region where the file control commands run. **FCQRONLY=YES** is the default.

Threadsafe file control benefits CICS regions where the files are defined as local to the CICS region and are either VSAM LSR or RLS. From a file control perspective, in CICS regions with a mix of file types, consider specifying the system initialization parameter **FCQRONLY=NO**. Then define programs that access local VSAM LSR or RLS files with CONCURRENCY(THREADSAFE) and programs that access other file types with CONCURRENCY(QUASIRENT). If the files in a CICS region are not local VSAM LSR or RLS, use the default system initialization parameter **FCQRONLY=YES**.

Function shipped requests to file-owning regions (FORs)

If you function ship file control requests from application-owning regions (AORs) to file-owning regions (FORs), choose your setting for **FCQRONLY** as follows:

- For FORs at CICS TS 4.2 or later that use IP interconnectivity (IPIC) connections over TCP/IP, specify **FCQRONLY=NO** to optimize performance for those connections.
- For FORs that use MRO links or ISC over SNA connections, specify **FCQRONLY=YES** to optimize performance for those connections. Also use **FCQRONLY=YES** for all FORs earlier than CICS TS 4.2.

If an AOR function ships all its file control requests to FORs and has no local files, you can use the default **FCQRONLY=YES** for the AOR, because the region does not benefit from threadsafe file control. For AORs that have some local files, choose the setting for **FCQRONLY** depending on the file types in the region.

File control API costs

For read operations, the VSAM I/O cost is not included because the need to access DASD depends on the workload. For the read operation to complete, both the index and data must be accessed. If the index or data are not in a buffer, an I/O operation is required for each level of index and one for the data.

The relative number of instructions, in 1K instruction counts, for the I/O for each file type is as follows:

- 9.5 for a key-sequenced data set (KSDS)
- 9.5 for an entry-sequenced data set (ESDS)
- 8.2 for a relative record data set (RRDS)

READ

KSDS	ESDS	RRDS	Data Table (CMT)
3.0	2.4	2.2	First: 1.5 Subsequent: 1.1

READ UPDATE

Recoverable and nonrecoverable files are included in the READ UPDATE cost:

Table 15. Nonrecoverable files

KSDS	ESDS	RRDS
3.1	2.3	2.2

A recoverable READ UPDATE puts the before image into the log buffer which, if not subsequently written to primary storage, is written out before the REWRITE is completed.

KSDS	ESDS	RRDS
5.5	4.3	4.2

REWRITE

Recoverable and nonrecoverable files are included in the REWRITE cost. Every REWRITE has a data VSAM I/O associated with it.

Table 16. Nonrecoverable files

KSDS	ESDS	RRDS
10.2	10.1	10.1

A REWRITE of a recoverable file requires that the log buffer that containing the before image is written out. If the buffer has not already been written out since the READ UPDATE, the cost of writing the log buffer is incurred. When the before image has been hardened, the VSAM I/O takes place. At the end of the transaction, there are additional costs involved in sync pointing if recoverable resources were updated. See “Sync pointing” on page 178.

KSDS	ESDS	RRDS
10.4	10.3	10.3

WRITE

The cost for WRITE includes nonrecoverable files and recoverable files. Every WRITE has a data VSAM I/O associated with it. The index needs to be written only when a control area split occurs.

Table 17. Nonrecoverable files

KSDS	ESDS	RRDS
12.9	11.1	10.9

Every WRITE has a hidden READ associated with it to ensure that the record is not already present in the file. This under-the-cover READ could incur the cost of I/Os if the index, the data, or both are not in the buffer. Each WRITE to a recoverable file requires that the log buffer containing the data image has been written out before the VSAM I/O takes place.

At the end of the transaction, there are additional costs involved in sync pointing if recoverable resources were updated. See “Sync pointing” on page 178.

Table 18. Recoverable files

KSDS	ESDS	RRDS
14.9	13.1	12.9

DELETE

You cannot delete from an ESDS record file.

Table 19. Nonrecoverable files

KSDS	RRDS
12.5	11.5

At the end of the transaction, additional costs are involved in sync pointing if recoverable resources were updated. See “Sync pointing” on page 178.

Table 20. Recoverable files

KSDS	RRDS
14.5	13.5

Browsing

STARTBR	READNEXT	READPREV	RESETBR	ENDBR
3.1	1.5	1.6	2.6	1.4

UNLOCK

The path length for **EXEC CICS UNLOCK** is 0.7.

Database management for performance

You can tune a number of aspects of database management in order to improve performance.

Setting DBCTL parameters

A number of parameters are required to assist with DBCTL performance. These include **MINTHRD** and **MAXTHRD**, which are specified in the DRA startup table (DFSPZP) and DEDB parameters (**CNBA**, **FPBUF**, and **FPBOF**), which are defined during DBCTL system generation or at DBCTL initialization.

For more information about the DBCTL parameters and tuning a CICS-DBCTL system, see Specifying numbers of threads and DEDB performance and tuning considerations.

Tuning the CICS DB2 attachment facility

The CICS DB2 attachment facility provides a multithread connection to DB2. The DB2CONN, DB2ENTRY, and DB2TRAN definitions of the CICS DB2 attachment facility define the authorization and access attributes on a transaction and transaction group basis. You can optimize performance between CICS and DB2 by adjusting the transaction class limits, MXT system parameters of CICS, and the THREADWAIT, TCBLIMIT, THREADLIMIT, and PRIORITY attributes of DB2CONN and DB2ENTRY.

A number of topics provide more information about the CICS DB2 attachment and performance considerations:

- Defining the CICS DB2 connection explains the recommendations for defining the CICS DB2 connection for optimum performance.
- How threads are created, used, and terminated explains threads and the use of the THREADWAIT, TCBLIMIT, and THREADLIMIT parameters with DB2.
- Application design and development considerations for CICS Db2 has recommendations for application design.
- Tuning a CICS application that accesses DB2 has recommendations for tuning CICS DB2 applications.

In summary, the objectives in tuning the CICS attachment facility are to:

- Optimize the number of threads in the connection.

  The total number of threads in the connection, and the number of threads for each dedicated entry and the pool must be optimized. A larger number of threads than is needed requires additional processor time to dispatch the TCBs and additional storage for plans, data, and control blocks. If an insufficient number of threads is defined, response time increases.
- Optimize the assignment and reuse of threads.

  Reusing threads avoids the thread creation and termination process, including plan allocation and authorization checks. Thread creation and termination represent a significant part of the processing time for a simple transaction. Thread reuse can be measured using CICS DB2 statistics.

Limit conversational transactions either through transaction classes or by using a dedicated DB2ENTRY (THREADLIMIT greater than 0) with THREADWAIT=YES specified. Otherwise, they tie up the pool. Do not allow conversational transactions to use the pool.

- For pool and entry threads, choose the priority assigned to the subtask thread TCBs, using the PRIORITY parameter.

  The **PRIORITY** parameter controls the priority of the CICS open L8 thread TCBs relative to the CICS main TCB (QR TCB). There are three options: PRIORITY=HIGH, PRIORITY=LOW, and PRIORITY=EQUAL.

  When PRIORITY=HIGH is specified, transactions run at a higher priority than CICS, saving virtual storage, releasing locks, and avoiding other transactions deadlocking or timing out. However, if all threads are specified with PRIORITY=HIGH, CICS itself might be at too low a priority, so for example, a complex SQL call could spend a long time in DB2, and the CICS TCB might not be dispatched.

  Set PRIORITY=HIGH for your transactions with the highest weighted average number of SQL calls. The highest weighted average is equal to the number of SQL calls per transaction multiplied by the frequency of transaction. Set PRIORITY=LOW or EQUAL for other transactions. If the CPU usage per call is high, you should not set PRIORITY=HIGH.
- Choose the best authorization strategy to avoid or minimize the process of signon by each thread.
- Minimize the number of DB2ENTRYs. Use wildcarding and dynamic plan selection where relevant to combine appropriate transactions in an entry. Allow low use transactions to default to the pool. However, it should be noted that defining transaction IDs using wildcard characters removes the ability to collect CICS DB2 statistics on a per transaction basis as statistics are collected for each DB2ENTRY which will now represent a group of transactions.

For information about tuning DB2 tables and the DB2 subsystem, and for general considerations when tuning a DB2 application, see Managing Db2 performance in Db2 for z/OS product documentation.

Selecting authorization IDs for performance and maintenance

A process that connects to or signs on to DB2 must provide one or more DB2 short identifiers, called authorization IDs, that can be used for security checking in the DB2 address space. Every process must provide a primary authorization ID, and it can optionally provide one or more secondary authorization IDs. CICS transactions that acquire a thread into DB2 are considered as processes, and must provide authorization IDs.

tells you how to choose and set up the authorization IDs that a CICS transaction passes to DB2 when the thread used by the transaction signs on to DB2. The authorization IDs for a transaction are determined by attributes in the resource definition for the thread that the transaction uses. For entry threads, this is the DB2ENTRY definition, and for pool threads or command threads, this is the DB2CONN definition.

When choosing the type of authorization ID that a CICS transaction will use, take into account performance and maintenance considerations.

Performance considerations for authorization IDs

From the point of view of performance, choosing one of the options USERID, OPID, TERM, TX or GROUP on the AUTHTYPE attribute means that any CICS transaction using a DB2 thread is likely to have a different authorization ID from the last transaction that used the thread. This causes sign-on processing to occur. Choosing the SIGN option, or using the AUTHID attribute instead of the AUTHTYPE attribute, means that CICS transactions will have the same authorization ID. If the transactions using a thread have the same authorization ID, sign-on processing can be bypassed.

However, although the options USERID, OPID, TERM, TX or GROUP have disadvantages for performance, they make DB2 security checking more granular. For example, if a transaction's thread is defined with AUTHTYPE(USERID), DB2 security checking uses the CICS user ID of the individual that is using the transaction. If a transaction's thread is defined with AUTHTYPE(SIGN), DB2 security checking uses the SIGNID that has been defined for the whole CICS region, so DB2 only checks that the CICS region is permitted to access DB2 resources. If you do use one of the options that gives the same authorization ID for all transactions, use CICS transaction-attach security to restrict access to transactions (see Controlling users' access to DB2-related CICS transactions).

An alternative solution for plans is to use a GRANT command in DB2 to give EXECUTE authority on a plan to PUBLIC, because this also causes sign-on processing to be bypassed. DB2 ignores the changed authorization ID. This is not quite as efficient as using a constant authorization ID and transaction id, because some processing still takes place in the CICS DB2 attachment facility. Security considerations for your DB2 subsystem could prevent the use of this solution, as it allows no security checking for the plan within DB2.

Maintenance considerations for authorization IDs

From the point of view of maintenance, when you use the options USERID, OPID, TERM, TX or GROUP for authorization IDs, you need to grant permissions in DB2 to a greater number of authorization IDs. For example, if a CICS transaction executes a plan in DB2, and the transaction's thread is defined with AUTHTYPE(USERID), you need to grant permission to use the plan in DB2 to all the CICS user IDs of individuals who can use the transaction. If you use the SIGN option, or use the AUTHID attribute instead of the AUTHTYPE attribute, you need to grant permissions to fewer authorization IDs.

However, as already mentioned, using a limited range of authorization IDs makes DB2's own security checking less granular. If your priority is security, but you are concerned about high levels of maintenance in your DB2 system, a possible solution is to set up secondary authorization IDs for CICS users. Providing secondary authorization IDs for CICS transactions tells you how to do this. You can create a RACF group, and connect your CICS users to this RACF group. Use the GROUP attribute of the DB2ENTRY definition for the thread used by the transaction, so that the RACF group is one of the secondary IDs that is passed to DB2. Then grant DB2 permissions to the RACF group. To remove a CICS user's DB2 permissions, disconnect them from the RACF group. If you use this solution, DB2's security checking can ensure that individual CICS users are authorized to access resources within DB2, but you do not have to specifically grant permission to each CICS user ID.

Logging

Because logging costs contain some of the variable costs incurred by synchronous accesses to the coupling facility, they are documented here in terms of milliseconds of CPU time.

When looking at the cost of accessing recoverable resources, the cost of writing the log buffer to primary storage has been separated from the API cost. FORCE and NOFORCE are the two types of write operations to the system log buffer.

- The FORCE operation requests that the log buffer is written out and is made non-volatile. The transaction that made this request is suspended until the process completes. The log is not written out immediately but is deferred using an internal algorithm. The first forced write to the log sets the clock ticking for the deferred log flush. Subsequent transactions requesting log forces will put their data in the buffer and suspend until the original deferred time has expired. This permits buffering of log requests and it means that the cost of writing the log buffer is shared between many transactions.
- The NOFORCE operation puts the data into the log buffer, which is written to primary storage when a FORCE operation is requested or the buffer becomes full.

The cost of writing a log buffer varies, depending on which of the following situations applies:

- The write is synchronous to the coupling facility
- The write is asynchronous to the coupling facility
- A staging data set is being used
- DASD-only logging is being used

Synchronous writes to the coupling facility

Writes of less than 4 K in size are generally synchronous. A synchronous write uses a special instruction that accesses the coupling facility directly. The instruction lasts for as long as it takes to access the coupling facility and return. This access time, known as the *CF Service Time*, depends on both the speed of the coupling facility and the speed of the link to it. CF Service Times can be monitored using RMF III, as shown in Figure 21 on page 184. For synchronous writes, the CPU cost of the access changes as the CF Service Time changes; this is not true of asynchronous writes.

Asynchronous writes to the CF
Asynchronous writes do not use the same instruction used by synchronous writes. A CICS task that does an asynchronous log write gives up control to another task, and the operation is completed by the logger address space.

For more information about logging, see "CICS logging and journaling: Performance and tuning" on page 178.

Sync pointing

The sync point cost needs to be factored into the overall transaction cost. The amount of work at sync point varies according to the number of different types of resource managers involved during the unit of work (UOW). Therefore, the cost can vary.

Typically, a sync point calls all the resource managers that have been involved during the UOW. These might have to place data in the log buffer before it is written out. For example, recoverable transient data (TD) defers putting data into the log buffer until a sync point. Recovery manager itself puts commit records into the log buffer and requests a forced write. For these reasons it is difficult to give a precise cost for a sync point, but the following information should be used as a guide:

A sync point can be split as follows:

Part	Value
Basic cost	5.0
Put commit records in the log buffer	2.0
For each RM used in UOW	2.5
Write log buffer	See "Logging" on page 177

This table shows sync point costs, in 1K instruction units, for local resources only. If distributed resources are updated, communication costs must added.

If no recoverable resources have been updated, the only cost is the transaction termination cost:

Transaction cost	Assembler	COBOL
Termination	6.2	10.0

Note: The transaction initialization cost is calculated from the start of transaction attach to the start of the CICS application code. If recoverable resources have been updated, the sync pointing cost must be added to the termination cost.

CICS logging and journaling: Performance and tuning

Individual CICS log streams can use either coupling facility log structures or the CICS log-manager-supported DASD-only option of the MVS system logger. You can tune the performance of the log manager in a number of ways.

For more information about the types of storage used by CICS log streams, see Defining the logger environment for CICS.

For information about how you can define each log stream (based on its usage) when you use coupling facility log structures, see Coupling facility or DASD-only?. For information about the relative performance of coupling facility and DASD-only log streams, see "Logging" on page 177.

If you use a coupling facility, you can use a standalone model. Alternatively, you can use the integrated coupling migration facility (ICMF) to provide the services of a coupling facility in a logical partition (LPAR).

This means that the coupling facility and MVS are not failure-independent, thereby requiring the use of staging data sets.

For additional advice and examples relating to performance and tuning for logging, see the following documents and subtopics:

- The IBM Redbooks publication Systems Programmer's Guide to: z/OS System Logger, SG24-6898. This document provides a thorough explanation of the z/OS System Logger, and explains how it should be set up for optimum performance with CICS and other exploiters.
- The IBM Redpaper Performance Considerations and Measurements for CICS and System Logger, REDP-3768. This document, which was written in support of the Redbook publications, supplies additional guidance on the interactions between CICS and z/OS System Logger, provides examples of different CICS and System Logger configurations, and demonstrates the tuning process.
- The IBM support document Useful CICS Logger information. This document provides links to two presentations dealing with performance evaluation and troubleshooting for CICS and z/OS System Logger.
- Defining a couple data set for system logger in z/OS Management Facility Configuration Guide.
- Examples of using the IXCMIAPU utility in z/OS MVS Setting Up a Sysplex.

The CICS log manager

The CICS log manager provides facilities for the creation, control, and retrieval of journals when CICS is running. Journals are intended to record, in chronological order, any information that you might later need to reconstruct data or events. For example, you can create journals to act as audit trails; to record database updates, additions, and deletions for backup purposes; or to track transaction activity in the system.

The CICS log manager controls all logging and journaling using services provided by the MVS system logger. The CICS log manager supports:

- The CICS system log
- Forward recovery logs
- Auto-journals for file control and terminal control operations
- User journals

The MVS system logger provides:

- Media management and archiving
- Log data availability through direct, and sequential, access to log records.

Log stream storage

A log stream is a sequence of data blocks, with each log stream identified by its own log stream identifier —the log stream name (LSN). The CICS system log, forward recovery logs, and user journals map onto specific MVS log streams. CICS forward recovery logs and user journals are referred to as general logs, to distinguish them from system logs.

Each log stream is a sequence of blocks of data, which the CICS log manager internally partitions over three different types of storage:

1. Primary storage, which holds the most recent records written to the log stream. Primary storage can consist of either:
   - A structure within a coupling facility. (The use of a coupling facility allows CICS regions in different MVS images to share the same general log streams.) Log data written to the coupling facility is also copied to either a data space or a staging data set.
   - A data space in the same MVS image as the system logger. Log data written to the data space is also copied to a staging data set.

2. Auxiliary storage—when the primary storage for a log stream becomes full, the older records automatically spill into auxiliary storage, which consists of data sets managed by the storage management subsystem (SMS). Each log stream, identified by its log stream name (LSN), is written to its own log data sets.
3. Tertiary storage—a form of archive storage, used as specified in your hierarchical storage manager (HSM) policy. Optionally, older records can be migrated to tertiary storage, which can be either DASD data sets or tape volumes.

Figure 19 on page 180 and Figure 20 on page 181 show the types of storage used by the CICS system logger.

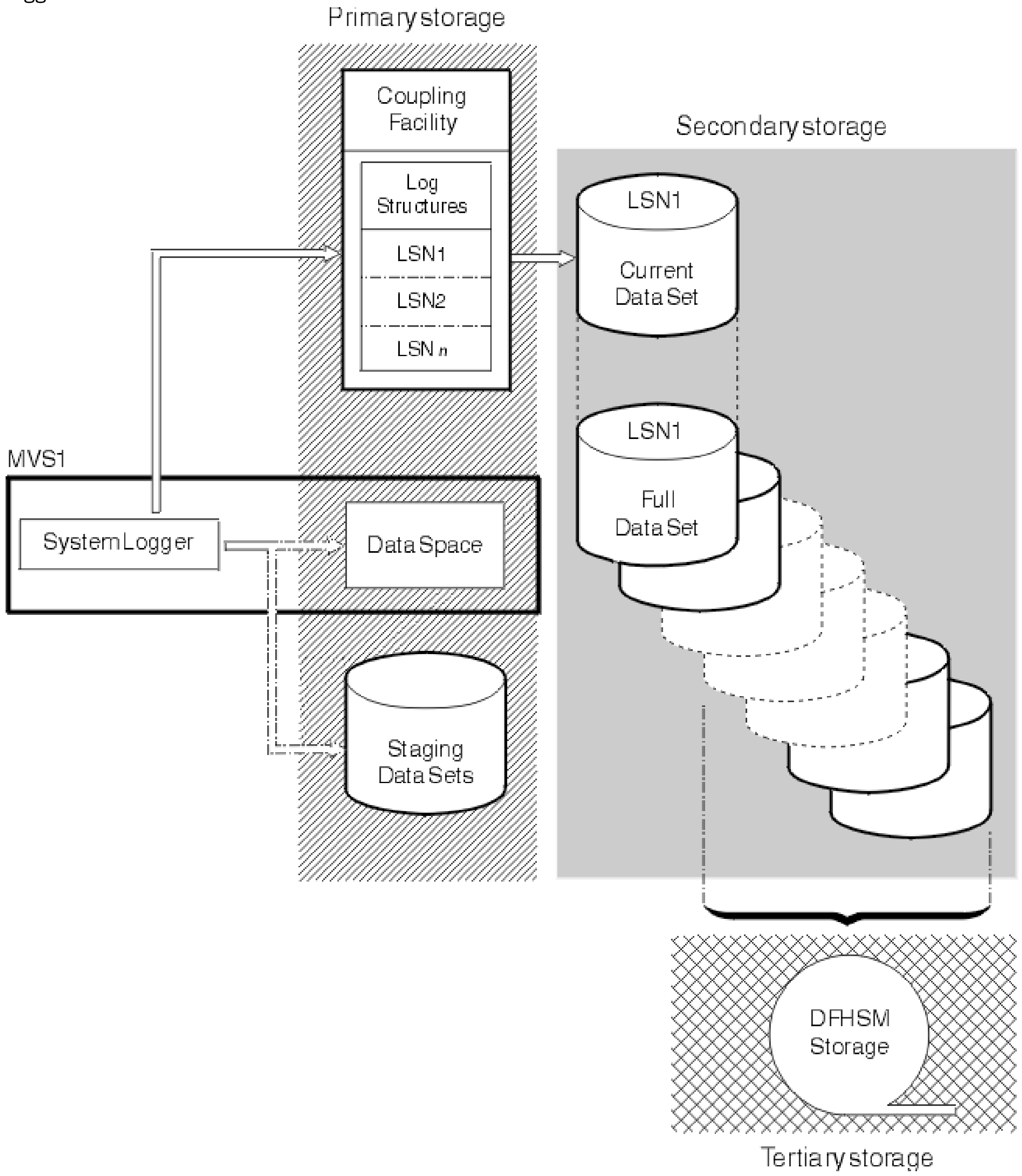


Figure 19. The types of storage used by the MVS system logger

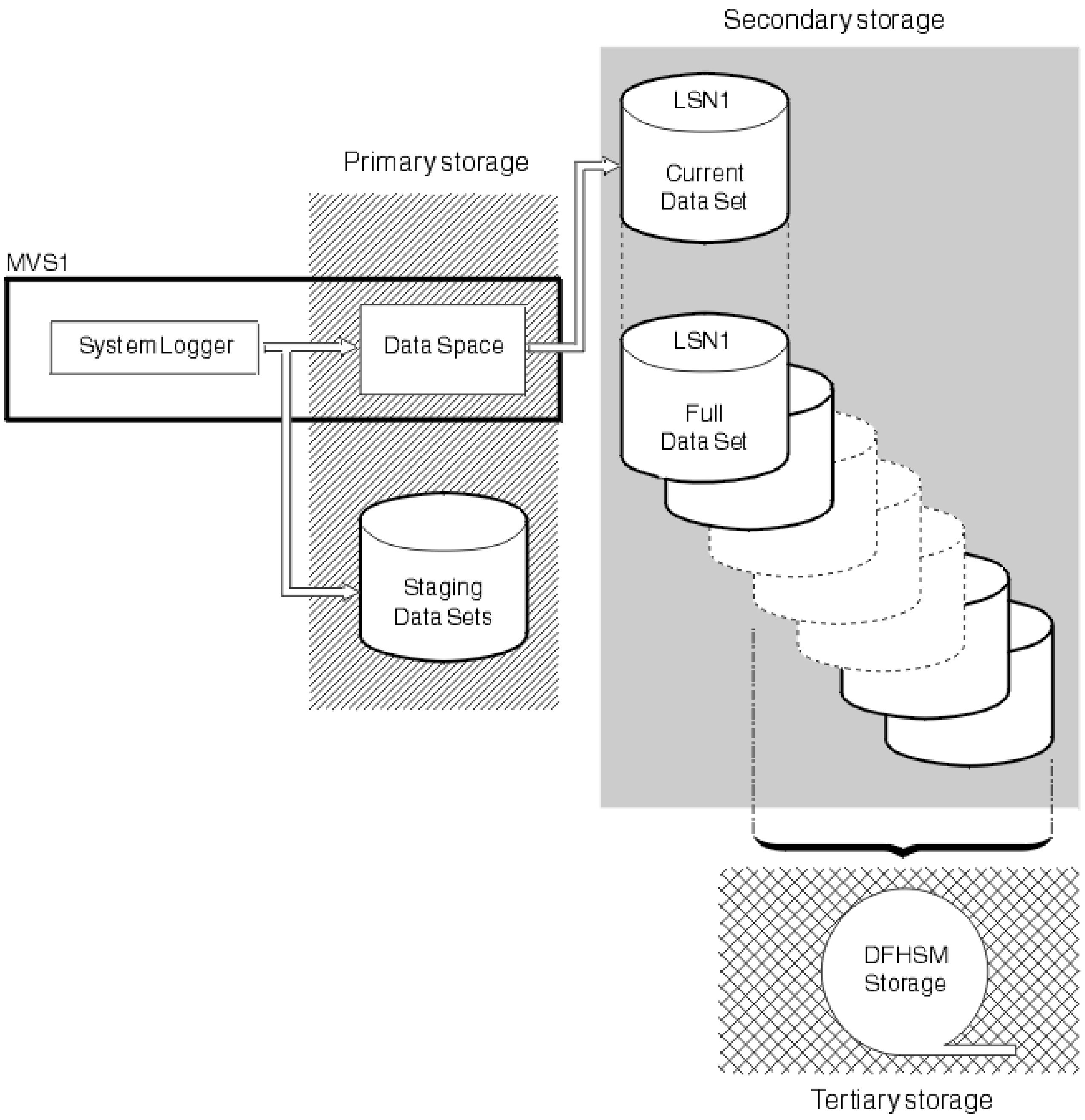


Figure 20. The types of storage used by the MVS system logger

Journal records

Journal records are written to a log stream either directly from a user application program or from a CICS management program on behalf of a user application.

Journal records can be written from a user application using the WRITE JOURNALNAME API command. You enable or disable a journal from an application program with the SET JOURNALNAME SPI command.

Access to journaled data in log streams is provided through an MVS subsystem interface (SSI), LOGR. Your existing user programs can read the general log streams, providing you specify the **SUBSYS** parameter and supporting options on the DD for log streams in your batch job JCL. If you specify the LOGR subsystem name on the **SUBSYS** parameter, LOGR can intercept data set open and read requests at the SSI and convert them into log stream accesses.

Depending on the options specified on the **SUBSYS** parameter, general log stream journal records are presented in one of two ways:

- In the record format used at CICS/ESA 4.1 and earlier, for compatibility with older utilities (selected by the COMPAT41 option)
- In the CICS Transaction Server for z/OS format for newer or upgraded utilities that needed to access log record information.

CICS system log records are available only in the CICS Transaction Server for z/OS format, so you must ensure that any utilities that handled system log records in releases before CICS Transaction Server for z/OS are converted to handle this format.

Journal records can be read offline by user-written programs. You can generate the DSECTs that such programs require by including certain statements in the program code:

- For records in the CICS Transaction Server for z/OS format on general logs, offline user-written programs can map journal records by including an INCLUDE DFHLGGFD statement. This statement generates the assembler version of the DSECT.
- For records formatted with the COMPAT41 option, offline user-written programs can map journal records by issuing the DFHJCR CICSYST=YES statement, which results in the DFHJCRDS DSECT being included in the program.

  The generated DSECT is the same as the DSECT that is obtained for CICS programs by the COPY DFHJCRDS statement. The only difference is that the fields are not preceded by a CICS storage accounting area. The DSECT is intended to map journal records directly in the block, rather than in a CICS storage area.

Monitoring the logger environment

CICS collects statistics on the data written to each journal and log stream; this data can be used to analyze the activity of a single region. However, because general log streams can be shared across multiple MVS images, it can be more useful to examine the statistics generated by MVS.

About this task

The MVS system logger writes SMF Type 88 records containing statistics for each connected log stream. MVS supplies in SYS1.SAMPLIB a sample reporting program, IXGRPT1, that you can use as supplied, or modify to meet your requirements. Alternatively, you can use some other SMF reporting program. For information about the SMF Type 88 records and the sample reporting program, see z/OS MVS System Management Facilities (SMF).

The main events to monitor routinely are as follows:

- For coupling facility log streams, the number of "structure full" events
- For DASD-only log streams, the number of "staging data set full" events.

If these events occur frequently, this indicates that the logger cannot write data to auxiliary storage quickly enough to keep up with incoming data, which causes CICS to wait before it can write more data.

Procedure

1. Consider the following solutions to resolve problems that occur as a result of event-full conditions:
   a) Increase the size of primary storage (that is, the size of the coupling facility structure or, for a DASD-only log stream, the size of the staging data set), in order to smooth out spikes in logger load.
   b) Reduce the data written to the log stream by not merging so many journals or forward recovery logs on to the same stream.
   c) Reduce the **`HIGHOFFLOAD`** threshold percentage, the point at which the system logger begins offloading data from primary storage to offload data sets.
   d) Review the size of the offload data sets. Offload data sets must be large enough to avoid too many “DASD shifts”—that is, new data set allocations. Aim for no more than one DASD shift per hour. You can monitor the number of DASD shifts using the SMF88EDS record.

e) Examine device I/O statistics for possible contention on the I/O subsystem used for offload data sets.

f) Use faster DASD devices.

The best CICS system logs performance is achieved when CICS can delete log tail data that is no longer needed before it is written to auxiliary storage by the MVS system logger. To monitor that this is being achieved, your reporting program can examine the values in the SMF88SIB and SMF88SAB SMF Type 88 records, which provide helpful information relating to log data.

SMF88SIB
: Data deleted from primary storage without first being written to DASD offload data sets. For a system log stream, this value is normally high in relation to the value of SMF88SAB. For a general log stream, this value is normally zero.

SMF88SAB
: Data deleted from primary storage after being written to DASD offload data sets. For a system log stream, this value is normally low in relation to the value of SMF88SIB. For a general log stream, this value is normally high.

Note: In any SMF interval, the total number of bytes deleted from primary storage (SMF88SIB plus SMF88SAB) might not match the total number of bytes written to auxiliary storage. Data is only written to offload data sets and then deleted from primary storage when the **HIGHOFFLOAD** threshold limit is reached.

2. If the SMF88SAB record frequently contains high values for a CICS system log:

   a) Check that RETPD=*dddd* is not specified on the MVS definition of the log stream. For information about the MVS **RETPD** parameter, see Managing auxiliary storage.

   b) Check that no long-running transactions are making recoverable updates without syncpointing.

   c) Consider increasing the size of primary storage.

   d) Consider increasing the **HIGHOFFLOAD** threshold value.

   e) Consider reducing the value of the **AKPFREQ** system initialization parameter.

Writing data to the coupling facility: Performance considerations

At the application design level you must consider that the average block size written to the coupling facility affects the performance of the CICS log manager.

When the average block size of data being written to the coupling facility is less than 4 KB, the write request is processed synchronously. The operation is synchronous to CICS, as is the instruction used to access the coupling facility, in that it runs for as long as it takes to place the data in the structure. For this reason, it is unwise to mix fast processors with slow coupling facilities. If the access time to a particular coupling facility remains constant, then for synchronous accesses, the faster the processor the more processor cycles are used by the request.

When the average block size of data being written to the coupling facility is greater than 4 KB, the write request is processed asynchronously; the CICS task gives up control and the MVS system logger posts the event control block (ECB) when the write request has been satisfied. This can result in an asynchronous request taking longer to complete than a synchronous one.

Synchronous requests might be changed by the subsystem into asynchronous requests if necessary—for example, if the subchannel is busy. Changed requests show on an RMF III report as CHNGD. Figure 21 on page 184 shows an extract from an RMF report showing the numbers of synchronous and asynchronous writes to a coupling facility structure. The report gives the system name, the total number of requests, and the average number of requests per second. For each type of request, it gives the number of requests, the percentage of all requests that this number represents, the average service time, and the standard deviation.

```
STRUCTURE NAME = LOG_FV_001        TYPE = LIST
          # REQ    -------------- REQUESTS -------------
SYSTEM    TOTAL             #     % OF  -SERV TIME(MIC)-
NAME      AVG/SEC           REQ    ALL     AVG   STD_DEV

MV2A      15549    SYNC     15K  95.3%   476.1    339.6
          27.87    ASYNC   721    4.6%  3839.0   1307.3
                   CHNGD    12    0.1%  INCLUDED IN ASYNC
```

Figure 21. RMF report showing numbers of synchronous and asynchronous writes to a coupling facility

Note: This applies only to log streams that use coupling facility structures.

Defining the number of log streams: Performance considerations

Coupling facility space is divided into structures by the coupling facility resource management (CFRM) policy; the maximum is 255 structures. Multiple log streams can use the same structure. Ensure that log streams used by applications that write similar sized data records share the same structure. The reasons for this relate to the values defined in the **AVGBUFSIZE** and **MAXBUFSIZE** parameters on the structure definition.

Generally, the more log streams per structure, the more difficult it is to tune the various parameters that affect the efficiency and performance of the CICS log manager.

When a coupling facility structure is defined, it is divided into two areas: one holds list entries, and the other holds list elements.

List elements are units of logged data and are either 256-bytes or 512-bytes long. List entries are index pointers to the list elements. There is one list entry per log record. There is at least one element per log record.

If you define **MAXBUFSIZE** with a value greater than 65276, data is written in 512-byte elements. If you define **MAXBUFSIZE** with a value less than, or equal to, 65276, data is written in 256-byte elements. The maximum value for this parameter is 65532.

The proportion of the areas occupied by the list entries and the list elements is determined by a ratio calculated as follows:

```
AVGBUFSIZE / element size
```

The resulting ratio represents the ratio, *nn*: 1, where *nn* represents element storage, and 1 represents entry storage. This is subject to a minimum of 1:1.

This ratio has performance significance because it can be inappropriate for a combination of many different applications with different logging requirements and behavior.

Element/entry ratio and the number of log streams per structure

AVGBUFSIZE is set at the structure level and dictates the ratio for the whole structure. If many applications write significantly differing amounts of data to their log streams at significantly differing intervals, some applications might experience unexpected DASD offloading, incurring increased processor usage.

The DASD offloading is unexpected because the log stream might not yet have reached the **HIGHOFFLOAD** threshold. Generally, the greater the number of log streams per structure, the greater the chance that the element/entry ratio is inappropriate for certain applications that use the log streams.

Each log record places an entry in the list entry area of the structure, and the data is loaded as one or more elements in the list element area. If the list entry area exceeds 90% of its capacity, all log streams are offloaded to DASD. DASD offloading commences regardless of the current utilization of the log stream, and continues until an amount of data equal to the difference between the **HIGHOFFLOAD** threshold and the **LOWOFFLOAD** threshold has been offloaded.

For example, the list entry area might exceed 90% of its capacity while log stream A is only 50% used. The **HIGHOFFLOAD** threshold is 80% and the **LOWOFFLOAD** threshold is 60%. Even though log stream

A has not reached its **HIGHOFFLOAD** threshold, or even the **LOWOFFLOAD** threshold, data is offloaded until 20% of the log stream has been offloaded. This is the difference between 80% and 60%. After the offloading operation has completed, log stream A is at 30% utilization (50% minus 20%).

Thus, the log stream used by an application that issues few journal write requests might be offloaded to DASD because of frequent journal write requests by other applications that are using other log streams in the same structure.

However, if multiple log streams share the same structure, a situation where list entry storage reaches 90% utilization occurs only where all the log streams have a similar amount of logging activity.

Dynamic repartitioning and the frequency of DASD offloading

The space in a coupling facility structure is dynamically partitioned between all the log streams connected to the structure. As more log streams connect, DASD offloading might occur more often.

Whenever a log stream connects to, or disconnects from, a coupling facility structure, the structure undergoes dynamic repartitioning. This means that the space in the structure is partitioned between all the log streams connected to the structure. As more log streams connect, less space is allocated to each log stream. The result can be a higher frequency of DASD offloading, because reduced log stream space means that the log stream **HIGHOFFLOAD** threshold percentages are reached more often.

A value of 64000 for **MAXBUFSIZE** is appropriate for most environments.

If **MAXBUFSIZE** is set to greater than 65276, the element size is 512 bytes. With a 512-byte element, space might be unused and therefore wasted because of padding to the end of the last element for the log record. This situation is less likely when records are larger and systems are busier.

AVGBUFSIZE and **MAXBUFSIZE** are parameters for use in the IXCMIAPU program, which you run to define coupling facility structures. For more information, see Administrative data utility in z/OS MVS Setting Up a Sysplex.

The following facilities are available to monitor the data traffic to log streams on structures, and from log streams to DASD:

- The CICS log stream statistics. These provide a range of statistical information including a value for average bytes written per write, which you can calculate by dividing the total bytes value by the total writes value. This can help you to tune the value for **AVGBUFSIZE**.
- Statistics provided by RMF, including a value 'elements per entry', which you can calculate by dividing the total number of elements value by the total number of entries value. You can check the activity in element units on the log stream. RMF also informs you of the proportion of requests, per structure, that have been processed synchronously and asynchronously. You can isolate structures that hold synchronously processed log stream requests from those that hold asynchronously processed log stream requests.
- SMF88 records. These provide a range of statistical information, including the number of bytes offloaded.

LOWOFFLOAD and HIGHOFFLOAD parameters on log stream definition

Data from a log stream can be offloaded to DASD data sets when the log stream use (in the coupling facility or the staging data set) reaches its **HIGHOFFLOAD** limit. The amount of data offloaded is determined by using the **LOWOFFLOAD** limit.

The **HIGHOFFLOAD** limit is specified when the log stream is defined.

This information is relevant if you are using log streams that use coupling facility structures. However, much of the guidance also applies to DASD-only log streams.

For more information about DASD-only log streams, see “DASD-only logging” on page 189.

For a system log, all records that have been marked for deletion are physically deleted; if, after this has been done, the **LOWOFFLOAD** limit has not been reached, the oldest active records are offloaded to DASD until **LOWOFFLOAD** is reached. For a general log, the oldest data is offloaded to DASD until the **LOWOFFLOAD** limit is reached.

There are also situations where offloading of data from the log stream data set occurs although the **HIGHOFFLOAD** threshold (and **LOWOFFLOAD** threshold in some circumstances) of the log stream has not been reached:

- When the **HIGHOFFLOAD** threshold is reached in the staging data set. If the size of the staging data set is proportionally smaller than the log stream, the **HIGHOFFLOAD** threshold is reached on the staging data set before it is reached on the log stream data set.
- When the list entry area of the log stream reaches 90% of its capacity.

In these situations, the amount of data offloaded from the log stream is determined as follows:

```
(Current utilization or HIGHOFFLOAD, whichever is the greater) - LOWOFFLOAD
```

This is the percentage of the log stream data set that is offloaded.

HIGHOFFLOAD and **LOWOFFLOAD** are parameters for use in the IXCMIAPU program that you run to define log stream models and explicitly named individual log streams. For more information, see Administrative data utility in z/OS MVS Setting Up a Sysplex.

SMF88 records and RMF provide a range of statistical information that helps you in the tuning of these parameters.

The primary system log

When an activity keypoint happens, CICS deletes the tail of the primary system log, DFHLOG. This means that data for completed units of work older than the previous activity keypoint is deleted. Data for each incomplete unit of work older than the previous activity keypoint is moved onto the secondary system log, DFHSHUNT, provided that the UOW has done no logging in the current activity keypoint interval.

To minimize the frequency of DASD offloading, try to ensure that system log data produced during the current activity keypoint interval, plus data not deleted at the previous activity keypoint, is always in the coupling facility structure. To avoid offloading this data to DASD, you can use these settings:

- Set **HIGHOFFLOAD** to 80.
- Minimize the amount of log data produced between activity keypoints by specifying a low value on the **AKPFREQ** parameter, for example, a value of 4000.
- Ensure that the value of **LOWOFFLOAD** is greater than the space required for the sum of:
  1. The system log data generated during one complete activity keypoint interval
  2. The system log data generated (between sync points) by your longest-running transaction.

  Use one of the following formulas to calculate a value for **LOWOFFLOAD**:

  ```
  LOWOFFLOAD = ((trandur * 90) / (akpintvl + trandur)) + 10
  [where RETPD=0 is specified]
  ```

  or

  ```
  LOWOFFLOAD = (trandur * 90) / (akpintvl + trandur)
  [where RETPD=dddd is specified]
  ```

  where:

  – `akpintvl` is the interval between activity keypoints. It varies according to workload and its calculation is based on peak workload activity, as follows:

    ```
    akpintvl = AKPFREQ / ((N1 * R1) + (N2 * R2*) + (Nn * Rn))
    ```

    where:

    - `N1, N2 ...` Nn is the transaction rate for each transaction (trans/sec)
    - `R1, R2 ...` Rn is the number of log records written by each transaction

– `trandur` is the execution time (between sync points) of the longest-running transaction that runs as part of the normal workload.

  If this duration is longer than the `akpintvl` value, you can either:

  - Increase the value of **AKPFREQ**, thus increasing the value of `akpintvl` (providing this does not result in an unacceptably large coupling facility structure size).
  - Change the application logic to cause more frequent sync points.
  - Calculate a structure size based on a shorter transaction duration, and accept that DASD offloading occurs when the long-running transaction is used.

  A good empirical range for the DFHLOG **LOWOFFLOAD** parameter value is between 40% and 60%. A value that is too low can result in physical offloading of log data from primary to auxiliary storage after the MVS Logger offload process has completed physical deletion of any unwanted log data during offload processing. Conversely, too high a value might mean that subsequent offload processing occurs more frequently, as less space is freed up from primary storage during an offload operation.

  If the results of the calculation from the formula do not lie within the range of 40% to 60%, it might be that your workload has unusual values for `trandur` or `akpintvl`.

  Review log stream definition values (such as **LOWOFFLOAD**) after analysis of information such as statistics from MVS logger SMF 88 records.

General logs

The recommendations for forward recovery logs and user journals are different to those for the system log. There is no requirement here to retain logged data in the coupling facility structure. Rather, due to the typical use of such data, you might only need a small structure and offload the data rapidly to DASD. If so, default **HIGHOFFLOAD** to 80 and **LOWOFFLOAD** to 0.

Tuning the size of staging data sets

MVS keeps a second copy of data written to the coupling facility in a data space, for use when rebuilding a coupling facility in the event of an error. This is satisfactory as long as the coupling facility is failure-independent (in a separate CPC and non-volatile) from MVS.

Where the coupling facility is in the same CPC, or uses volatile storage, the MVS system logger supports staging data sets for copies of log stream data that would otherwise be vulnerable to failures that impact both the coupling facility and the MVS images.

Elements (groups of log records) are written to staging data sets in blocks of 4 KB (not in 256-byte or 512-byte units as for log stream data sets).

Use the following formulas to help you tune the size of your staging data sets:

```
staging data set size= (NR * AVGBUFSIZE rounded up to next unit of 4096)
```

where NR is the number of records to fill the coupling facility structure. This can be calculated as follows:

```
NR = coupling facility structure size / (AVGBUFSIZE rounded up to next element)
```

Ensure that the coupling facility structure and staging data set can hold the same number of records. Staging data sets are subject to the same offloading thresholds as log streams are. It is sensible, therefore, to ensure as far as possible that offloading activity will be at the same frequency.

It is generally better to overestimate, rather than underestimate, staging data set size. To calculate staging data set size to accommodate the maximum number of records (where there is one record per element), use the following formulas:

Where element size is 512-bytes:

```
maximum staging data set size = 8 * coupling facility structure size
```

Where element size is 256-bytes:

```
maximum staging data set size = 16 * coupling facility structure size
```

Investigate using DASD FastWrite facilities with a view to storing data in the DASD cache, as opposed to writing it directly to the staging data set. This also enables a faster retrieval of data should it be required. Be aware, however, that if you fill the cache, data is also then written out to the staging data set whenever data is written to the cache.

The activity keypoint frequency (AKPFREQ)

The activity keypoint frequency value, AKPFREQ, specifies the number of write requests to the CICS system log stream output buffer required before CICS writes an activity keypoint. A keypoint is a snapshot of inflight tasks in the system at that time.

During emergency restart, CICS needs to read back for records for only those tasks that are identified in a keypoint. CICS reads the system log backward until the first activity keypoint is encountered (which is the last activity keypoint taken).

Taking a keypoint imposes an overhead on the running system:

- If you set AKPFREQ too high, such that the keypoint frequency is too low, writing keypoints slows the system for only a short time.
- If you set AKPFREQ too low, such that the keypoint frequency is too high, the emergency restart time might be short, but you also incur increased processing, because more activity keypoints are processed.

It is advisable to set AKPFREQ to the default value of 4000. With an optimum setting of AKPFREQ, the whole of the system log can remain in the coupling facility.

Increasing the AKPFREQ value increases the amount of primary storage required for the system log. Decreasing the AKPFREQ value has the following effects:

- Restart time might be reduced.
- The amount of primary storage required for the system log decreases.
- Task wait time and processor cycles tend to increase.
- Paging might increase.

The last two effects can affect system performance, but not significantly.

If you set the AKPFREQ value to zero, emergency restart takes longer. In this situation, CICS cannot perform log tail deletion until shutdown, by which time the system log spills to auxiliary storage. Because there are no activity keypoints, CICS needs to read the whole of the system log, so it needs to retrieve the spilled system log from DASD offload data sets.

Activity keypoint frequency is determined by the AKPFREQ system initialization parameter. You can alter **AKPFREQ** while CICS is running by using the **`CEMT SET SYSTEM AKP(value)`** command.

The CICS log stream global statistics include information about the activity keypoint frequency. See Logstream reports for more information.

A message, DFHRM0205, is written to the CSMT transient data destination each time that a keypoint is taken.

AKPFREQ and MRO

In an MRO environment, the session allocation algorithm selects the lowest-numbered free session for use by the next task to run. Consequently, if many sessions have been defined (perhaps to cope with peak workload requirements), the higher-numbered sessions are less likely to be used frequently during quieter periods.

In an MRO environment, CICS implements the "implicit forget" process, an optimization of the two-phase commit. This means that when the mirror transaction at the remote end of an MRO connection completes any end-of-task processing, all information relating to the task is deleted when any new flow on that session arrives. This flow is usually the first flow for the next task or transaction allocated to run on the session as a result of the MRO session allocation algorithm.

Short-term variations in the arrival rate of transactions means that some mirror transactions waiting to process an implicit forget can persist for some time. This is particularly the case where such mirror transactions have been allocated to high-numbered sessions during a peak period, now passed, of transaction arrival rate.

The keypoint program uses an appreciable amount of processor capacity in processing persisting units of work such as those relating to mirror transactions waiting to process an implicit forget. This is exacerbated when the AKPFREQ value is low.

An optimum setting of AKPFREQ allows many of these persistent units of work to complete during normal transaction processing activity. This minimizes the processor processing used by the keypoint program. For this reason, you must be cautious when reducing the value of AKPFREQ to less than the default value.

The log defer interval (LGDFINT)

The **LGDFINT** system initialization parameter specifies the log defer interval used by CICS log manager when determining how long to delay a forced journal write request before starting the MVS system logger.

The value is specified in milliseconds. Performance evaluations of typical CICS transaction workloads have shown that a value of 5 milliseconds gives the best balance between response time and central processor cost.

CICS performance can be adversely affected by a change to the log defer interval value. Too high a value delays CICS transaction throughput due to the additional wait before starting the MVS system logger.

An example of a scenario where a reduction in the log defer interval might be beneficial to CICS transaction throughout would be where many forced log writes are being issued, and little concurrent task activity is occurring. Such tasks will spend considerable amounts of their elapsed time waiting for the log defer period to expire. In such a situation, there is limited advantage in delaying a call to the MVS system logger to write out a log buffer, since few other log records are added to the buffer during the delay period.

Although the range of possible values for the log defer interval is from 0 to 65535 milliseconds, the default of 5 milliseconds is considered to be the correct interval when setting the parameter in most cases.

A log defer interval value of less than 5 milliseconds reduces the delay in CICS log manager before starting the IXGWRITE macro. This might improve the transaction response time, but increases processor cost for the system because CICS has fewer journal requests into a given call to the MVS system logger, and so must start the IXGWRITE macro more often.

Conversely, increasing the log defer interval value to greater than 5 milliseconds increases the transaction response time, because CICS increases the delay period before starting the IXGWRITE macro. However, more transactions can write their own log data in to the same log buffer before it is written to the MVS system logger, and hence the total processor cost of driving IXGWRITE calls is reduced.

The log defer interval is determined by the **LGDFINT** system initialization parameter. **LGDFINT** can be altered with the **CEMT SET SYSTEM[LOGDEFER(value)]** command while CICS is running.

The CICS log stream global statistics capture information about the log defer interval. See Logstream reports for more information.

DASD-only logging

The primary storage used by a DASD-only log stream consists of a data space owned by the MVS logger and staging data sets. You can tune for DASD-only logging to improve performance.

No data is written to coupling facility structures. In its use of staging data sets, a DASD-only log stream is similar to a coupling facility log stream defined with DUPLEX(YES) COND(NO).

When the staging data set reaches its **HIGHOFFLOAD** limit, data is either deleted or offloaded until the **LOWOFFLOAD** limit is reached.

The following principles apply to DASD-only log streams as much as to coupling facility log streams:

- Size system logs so that system log data produced during the current activity keypoint interval, plus data not deleted at the previous activity keypoint, is retained in primary storage
- For the system log, avoid “staging data set full” conditions and offloading to auxiliary storage.

The basic principles of sizing the staging data set for a DASD-only log stream are the same as for sizing a staging data set for a coupling facility log stream, as described in “Tuning the size of staging data sets” on page 187. Take the values that you obtain as a starting point, and monitor your logger environment to adjust the size of the staging data set.

Use the following formula to calculate a starting point for the size of the staging data set for the system log. The formula calculates the value to be specified on the **STG_SIZE** parameter of the log stream definition; that is, the size is expressed as a number of 4 KB blocks.

```
Staging
DS size [No. of 4K blocks] = (AKP duration) * No. of log writes per second
                                               for system log
where:
AKP duration = (CICS TS 390 AKPFREQ) / (No. of buffer puts per second)
```

The values for the number of log writes per second and buffer puts per second can be taken from your CICS statistics. In CICS Transaction Server releases, the log stream statistics fields collect these statistics as "write requests" (LGSWRITES) and "buffer appends" (LGSBUFAPP), and you can divide the totals by the number of seconds in your statistics interval.

If you want to make a more accurate estimate for the size of the staging data set, consult the following documents:

- The IBM Redpaper Performance Considerations and Measurements for CICS and System Logger, REDP-3768. This document supplies guidance on the interactions between CICS and z/OS System Logger, provides examples of different CICS and System Logger configurations, and demonstrates the tuning process.
- The IBM Redbooks publication Systems Programmer's Guide to: z/OS System Logger, SG24-6898. This document explains how to obtain and use an IXGRPT1 report to estimate the size of a staging data set for a DASD-only log stream. (IXGRPT1 is a sample program provided with z/OS.)

CICS temporary storage: Performance and tuning

CICS temporary storage is intended for short-lived data. An application can write data to temporary storage as a series of numbered items in a temporary storage queue. CICS also creates some temporary storage queues for its own use. Temporary storage is heavily used in many CICS systems.

The ways in which you can tune the use of CICS temporary storage depend on the locations of the temporary storage available to the CICS region. Temporary storage can be main storage in the CICS region, auxiliary storage in a VSAM data set, or shared temporary storage pools in a z/OS coupling facility. The temporary storage can be associated with the local CICS region or a remote queue-owning region (QOR). For an overview of the locations for temporary storage, see “CICS temporary storage: overview” on page 191.

For main temporary storage, you can monitor the use of storage and use the **TSMAINLIMIT** system initialization parameter to set a suitable limit. For more information about tuning main temporary storage, see “Main temporary storage: monitoring and tuning” on page 193.

For auxiliary temporary storage, you must balance several factors when you set up the VSAM data set and when you are tuning the use of CICS temporary storage. The following factors affect the performance of auxiliary temporary storage:

- The control interval size for the data set

- The number of VSAM buffers in the CICS region
- The number of VSAM strings for I/O to the data set

For more information about tuning auxiliary temporary storage, see "Auxiliary temporary storage: monitoring and tuning" on page 195.

Consider setting up shared temporary storage pools to improve availability and support dynamic transaction routing. Shared temporary storage pools require temporary storage servers (typically one server in each z/OS image in the sysplex), but they have a number of advantages:

- No storage is used in the CICS region for the shared temporary storage pools.
- Shared temporary storage pools do not cause inter-transaction affinities. Local temporary storage queues in main or auxiliary storage can cause inter-transaction affinities, where affected transactions must run in the same region to access the queue. Inter-transaction affinities can affect performance by limiting the scope for workload routing across AORs in a sysplex.
- Compared to remote queue-owning regions, access to temporary storage queues in shared temporary storage pools in a coupling facility is quicker.
- If you use more than one temporary storage server for each pool, availability is better than it is for a remote queue-owning region. If one temporary storage server or z/OS image fails, transactions can be dynamically routed to another application-owning region on a different z/OS image.

CICS temporary storage: overview

You can set up temporary storage for a CICS region in three locations: main storage, auxiliary storage, or shared temporary storage pools in a z/OS coupling facility.

Main storage
Main temporary storage is in 64-bit (above-the-bar) storage in the CICS region. You use the **TSMAINLIMIT** system initialization parameter to specify the amount of storage that is available to temporary storage queues.

You can use local main storage in the CICS region where the applications run, or you can function ship temporary storage requests to a remote queue-owning region (QOR).

Auxiliary storage
Auxiliary temporary storage is in a nonindexed VSAM data set named DFHTEMP. You define the available space and any additional extents when you set up this data set. Some 31-bit (above-the-line) storage is used in the CICS region for VSAM buffers to make control intervals available from the VSAM data set. You use the **TS** system initialization parameter to set the number of buffers. Like main temporary storage, auxiliary temporary storage can be associated with the local CICS region or a remote queue-owning region.

Shared temporary storage pools in a z/OS coupling facility
Shared temporary storage pools (TS pools) are in a z/OS coupling facility managed by a temporary storage data sharing server (TS server). Each pool corresponds to a list structure in the coupling facility. You specify the size of each temporary storage pool using the coupling facility resource manager (CFRM) policy definition utility in z/OS. Shared temporary storage pools do not use any storage in the CICS region, and applications access them directly from the local CICS region.

When applications use the WRITEQ TS and READQ TS commands to access temporary storage queues, the requests are processed by the CICS temporary storage domain, which creates temporary storage queues in the appropriate storage location and places the data in them. Any task can retrieve the data using the symbolic name of the temporary storage queue. The CICS temporary storage domain can process multiple requests concurrently, but it serializes requests made for the same temporary storage queue, and the queue is locked for the duration of each request.

You use TSMODEL resource definitions to set up models that CICS uses to create temporary storage queues. Each model specifies the following attributes for temporary storage queues with names that match the model:

- The location of the temporary storage where the queue must be stored

- Whether the temporary storage is associated with the local CICS region or a remote CICS region, such as a queue-owning region
- Whether the queue is deleted automatically by CICS, if it remains unused for a period of time and is not deleted by an application
- Whether the queue is recoverable

Table 21 on page 192 summarizes the storage usage and the features that you can select for temporary storage queues in each location.

Table 21. Features of temporary storage locations

Temporary storage location	**Storage type**	**Automatic queue deletion**	**Recovery**
Main storage	64-bit storage in CICS region	Available	Not available
Auxiliary storage	VSAM data set, plus 31-bit storage in CICS region for buffers	Available for non-recoverable queues	Available
Shared temporary storage pool	z/OS coupling facility	Available	CICS recovery is not available, but the queues are persistent (they are not affected by a CICS restart)

CICS also creates some temporary storage queues for its own use. These queues can be in main temporary storage or auxiliary temporary storage. For example, CICS uses temporary storage for the following purposes:

- Basic mapping support (BMS) paging and routing
- Caching of messages
- Interval control
- The CICS execution diagnostic facility (EDF)
- Local queueing for MRO, ISC, and IPIC while the target system is unavailable

When you view the temporary storage queues in your CICS system, queues with names that start with these characters are CICS queues: **, $$, X'FA' through X'FF', CEBR, and DF.

Automatic deletion of temporary storage queues

CICS can automatically delete nonrecoverable temporary storage queues that have not been referenced recently. To use this feature, you set suitable expiry intervals in the temporary storage models (TSMODEL resource definitions).

Automatic deletion frees storage occupied by temporary storage queues that were not deleted by applications and that are no longer required.

The expiry interval for a temporary storage model applies to the temporary storage queues that are associated with that model. Temporary storage queues use the expiry interval that exists for the TSMODEL resource definition at the time that the queue is created.

By default, the expiry interval is zero, that is, no expiry interval applies to the temporary storage queues. Such queues are never eligible for automatic deletion.

You can set an expiry interval in minutes, up to a maximum of 900,000 minutes (that is 15,000 hours). CICS uses the value rounded up to the nearest multiple of 10 minutes. The interval count begins after each use of the temporary storage queue. If the queue is not used again before the expiry interval is reached, the queue becomes eligible for CICS to delete it automatically. When at least one nonzero expiry interval in at least one TSMODEL resource definition exists, CICS starts to scan the CICS region regularly to find eligible queues. The CICS clean up task scans the temporary storage queues in the CICS region and deletes the queues that are eligible for automatic deletion.

Expiry intervals apply to temporary storage queues in the following locations:

- Main temporary storage in the local CICS region.
- Nonrecoverable auxiliary temporary storage (DFHTEMP data set) associated with the local CICS region.
- Queues in shared temporary storage pools.

Expiry intervals do not apply to the following types of temporary storage queue, so CICS never deletes them automatically:

- Queues in auxiliary temporary storage that are defined as recoverable.
- Queues in a remote CICS region. To make CICS delete remote temporary storage queues, specify an expiry interval in a suitable TSMODEL resource definition in the region that owns the queues.
- Queues that CICS creates for its own use.
- Queues that do not match any temporary storage model.

If you change the expiry interval in a TSMODEL resource definition, existing temporary storage queues that match the model are not affected. Those queues continue to use the expiry interval that applied when they were created. If all the TSMODEL resource definitions with a nonzero expiry interval are deleted from a CICS region, CICS stops scanning for expired temporary storage queues.

When the CICS clean up task performs a scan, it issues message DFHTS1605. This message shows the number of temporary storage queues that were scanned and the number that were deleted. If the clean up task ends abnormally, it issues message DFHTS0001, and does not run again until CICS is restarted.

The CICS clean up task cannot delete temporary storage queues if the system has reached TSMAINLIMIT (see TSMAINLIMIT system initialization parameter) and there was an attempt to write to a TS queue such that a TS request lock is held. In this situation, the DFHTS1605 message reports that 0 queues were deleted.

Automatic deletion for TST users

If your CICS region still uses a temporary storage table (TST), which can be used in combination with TSMODEL resource definitions, the TST might include a TSAGE parameter. TSAGE specifies an aging limit in days, up to 512 days, for temporary storage queues. If the TST includes a nonzero TSAGE and there is an emergency restart of CICS, CICS deletes temporary storage queues that were not referenced during the specified interval. The TSAGE parameter does not cause automatic deletion of queues at any other time.

Main temporary storage: monitoring and tuning

You can monitor and control the amount of storage in the CICS region that is used by temporary storage queues.

About this task

From CICS TS for z/OS, Version 5.1, main temporary storage is located in 64-bit storage, so the available space is greater than in earlier CICS releases. Main temporary storage does not require VSAM I/O activity or communication with a temporary storage server. However, temporary storage queues in main temporary storage are not recoverable.

The CICS temporary storage statistics show information about the use of main temporary storage. You can also use CICSPlex SM or CICS commands to see the amount of main temporary storage in use, and the current limit. When 75% or more of the maximum allowed storage is in use, CICS issues messages about this situation.

You use the **`TSMAINLIMIT`** system initialization parameter to specify the amount of storage in the CICS region that is available for temporary storage queues to use. You can specify an amount of storage in the range 1 - 32768 MB (32 GB).

However, you must also check the setting for the z/OS parameter **MEMLIMIT**.

Procedure

1. Specify expiry intervals in your temporary storage models. When you specify an expiry interval, CICS can automatically delete temporary storage queues that match the models if they are not deleted by applications.

   For more information about expiry intervals, see Automatic deletion of temporary storage queues.
2. Use CICSPlex SM, CICS commands, or CICS statistics to monitor the amount of main temporary storage in use.
   - The CICS temporary storage global and summary statistics show the number of times that main temporary storage use reached the limit set by **TSMAINLIMIT**, and the peak amount of virtual storage that was used for data in main temporary storage.
   - The TEMPSTORAGE resource shows the storage in use compared to the maximum allowed limit.
3. Look out for messages from CICS about high usage of main temporary storage.
   - CICS issues message DFHTS1601 when 75% or more of the maximum allowed storage is in use.
   - CICS issues message DFHTS1602 if an application attempts to write an item of data that would make the main temporary storage in use exceed the maximum allowed limit (the **TSMAINLIMIT** value). In this situation, applications cannot write to temporary storage queues in main temporary storage until space becomes available.

   If either of these messages are issued, try to delete old temporary storage queues or increase the **TSMAINLIMIT** setting, as described in the following steps. CICS issues message DFHTS1604 when usage falls below 70% of the maximum allowed.
4. Before you change the **TSMAINLIMIT** setting, check your current setting for the z/OS parameter **MEMLIMIT**.

   The amount of storage that you make available for temporary storage queues must not be greater than 25% of the **MEMLIMIT** value. For information about the **MEMLIMIT** value for CICS and instructions to check the value of **MEMLIMIT** that currently applies to the CICS region, see "Estimating, checking, and setting MEMLIMIT" on page 81.
5. Optional: To change the amount of storage available for temporary storage queues, change the **TSMAINLIMIT** setting.

   You can change the **TSMAINLIMIT** setting in a running CICS system.
   - If you increase the **TSMAINLIMIT** setting and the new value is greater than 25% of the value of **MEMLIMIT**, **TSMAINLIMIT** remains unchanged and message DFHTS1607 is issued.
   - If you decrease the **TSMAINLIMIT** setting, CICS attempts to maintain at least 25% free space in allowed storage above current utilization, so that temporary storage write requests do not reach the **TSMAINLIMIT** value too rapidly. The value is set as follows:
     - If there is currently less than 25% free space, **TSMAINLIMIT** remains unchanged. Message DFHTS1606 is issued.
     - If at least 25% of the new limit will be free space, the setting is decreased to the value that you choose.
     - If less than 25% of the new limit would be free space, setting is decreased to the current utilization plus 33% of that utilization.

   If the value of **TSMAINLIMIT** is changed, CICS issues message DFHTS1603, which shows the new setting.

Results

The following table shows the cost of main storage. In this example, *n* represents the number of items in the queue before it is deleted.

Table 22. The cost of main storage

WRITEQ	REWRITE	READQ	DELETEQ
1.0	0.8	0.8	0.71 + 0.23 x *n*

Auxiliary temporary storage: monitoring and tuning

The performance of auxiliary temporary storage is influenced by the characteristics of the VSAM data set DFHTEMP that you set up for temporary storage. It is also affected by the number of VSAM buffers and strings that you specify for the CICS region.

About this task

The CICS temporary storage statistics show information about the use of auxiliary temporary storage, the use of buffers and strings, and I/O activity. For additional information about data set performance, use RMF or the VSAM catalog.

The cost approximations for auxiliary TS queues do not include any VSAM I/O cost. A VSAM I/O costs approximately 11.5K instructions and occurs in the following situations:

- When attempting to write an item that does not fit in any buffer
- When reading an item that is not in the buffer
- When reading a control interval from DASD with no available buffer space, if the least recently used buffer must first be written out.

Therefore, under certain circumstances, a READQ could incur the cost of two VSAM I/Os.

Procedure

The following actions can influence the performance of auxiliary temporary storage:

- You specify a control interval (CI) size when you set up the VSAM data set DFHTEMP. When the use of temporary storage by applications or by CICS changes in your CICS region, verify that the control interval size is still suitable.

  If you write items larger than the control interval size to a temporary storage queue in auxiliary storage, CICS processes the items, but performance might degrade.

  For information about the control interval size, see The control interval size.
- For more efficient use of DASD space, you can specify secondary extents when you set up the VSAM data set DFHTEMP.

  CICS uses secondary extents if there are no control intervals remaining in DFHTEMP with sufficient space for new data. You can define a temporary storage data set with a primary extent large enough for normal activity, and with secondary extents for exceptional circumstances.

  For instructions to define additional extents, see Defining data sets with multiple extents and volumes.
- To help ensure that space is not wasted in auxiliary temporary storage, specify expiry intervals in your temporary storage models for nonrecoverable queues.

  Expiry intervals make CICS automatically delete any temporary storage queues that might not be deleted by applications. When an unused queue is deleted from a DFHTEMP control interval, CICS can move the remaining records to the start of the control interval, and use the space for new data. Efficient deletion of old queues can reduce the time required to locate a control interval with free space, and reduce the need to use secondary extents.

  For more information about expiry intervals, see “Automatic deletion of temporary storage queues” on page 192.
- If you specify the system initialization parameter **SUBTSKS=1**, CICS runs temporary storage VSAM requests on the concurrent (CO) mode TCB, which could increase throughput.

- You use the **TS** system initialization parameter to specify the numbers of VSAM buffers and strings for auxiliary temporary storage in the CICS region. If auxiliary temporary storage is heavily used in the CICS region, you might want to experiment with adjusting these numbers.

  Increasing the numbers of buffers and strings can reduce task waits and VSAM I/O requests, but it also increases storage use in the CICS region.

Recoverable and nonrecoverable TS queues

The cost of temporary storage is different for the recoverable TS queue and the nonrecoverable TS queue.

The main difference between the cost of accessing recoverable and nonrecoverable TS queues is incurred at sync point time. For recoverable queues, the following events occur at sync point time:

- The VSAM I/O cost is incurred if any control interval has been used during the unit of work, and has not already reached DASD.
- The new DASD control interval addresses are put in the log buffer. The cost for recovery manager to do this is about 2000 instructions.
- A forced log write is requested and the sync point completes when the log buffer has been written to primary storage.

In each table, *n* represents the number of items in the queue before it is deleted.

Table 23. Recoverable TS queue

WRITEQ	**REWRITE**	**READQ**	**DELETEQ**
1.4	1.9	1.0	0.87 + 0.18 * *n*

Table 24. Nonrecoverable TS queue

WRITEQ	**REWRITE**	**READQ**	**DELETEQ**
1.3	1.8	1.0	0.75 + 0.18 * *n*

CICS transient data (TD) facility: Performance and tuning

Transient data (TD) is used in many circumstances within CICS, and various options can affect the performance of this facility.

The circumstances in which transient data is used include:

- Servicing requests made by user tasks, for example, a request to build a queue of data for later processing.
- Servicing requests from CICS, primarily to write messages to system queues for printing. Transient data should, therefore, be set up at your installation to capture these CICS messages.
- Managing the DASD space holding the intrapartition data.
- Initiating tasks based on queue trigger level specification and on records written to an intrapartition destination.
- Requesting logging for recovery as specified in your CICS transient data definitions.
- Passing extrapartition requests to the operating system access method for processing.

Limitations

Application requirements might dictate a lower trigger level, or physical or logical recovery, but these facilities increase processor requirements. Real and virtual storage requirements might be increased, particularly if several buffers are specified.

Implementation

Transient data performance is affected by the **TRIGGERLEVEL** and **RECOVSTATUS** operands in the transient data resource definitions that have been installed.

Recommendations

The following suggestions might help to reduce waits during QSAM processing:

- Avoid specifying a physical printer.
- Use single extent data sets whenever possible to eliminate waits resulting from the end of extent processing.
- Avoid placing data sets on volumes that are subject to frequent or long duration RESERVE activity.
- Avoid placing many heavily-used data sets on the same volume.
- Choose BUFNO and BLKSIZE such that the rate at which CICS writes or reads data is less than the rate at which data can be transferred to or from the volume; for example, avoid BUFNO=1 for unblocked records whenever possible.
- Choose an efficient BLKSIZE for the device employed such that at least three blocks can be accommodated on each track.

Monitoring

The CICS statistics show transient data performance. CICS transient data statistics can be used to determine the number of records written or read. Application knowledge is required to determine the way in which the lengths of variable length records are distributed. CICS transient data statistics also show the peak size of each intrapartition transient data queue during the statistics interval. RMF or the VSAM catalog shows data set performance.

Recovery options

Recovery can affect the length of time for which a transient data record is enqueued.

You can specify one of three options:

- *No recovery*. If you specify no recovery, there is no logging, and no enqueuing for protecting resources.
- *Physical recovery*. Specify physical recovery when you need to restore the intrapartition queue to the status that it had immediately before a system failure. The main performance consideration is that there is no deferred transient data processing, which means that automatic task initiation might occur instantaneously. Records that have been written can be read by another task immediately. Control intervals (CIs) are released as soon as they have been exhausted. For every WRITEQ TD request, the CI buffer is written to the VSAM data set.

  Note: All other resources that offer recovery within CICS provide only logical recovery. Using backout in an abend situation would exclude your physically recoverable and nonrecoverable transient data from the backout.
- *Logical recovery*. Specify logical recovery when you want to restore the queues to the status that they had before execution of the failing task (when the system failed or when the task ended abnormally). Thus, logical recovery works in the same way as recovery defined for other recoverable resources such as file control and temporary storage.

In summary, physical recovery ensures that records are restored in the case of a system failure, while logical recovery also ensures integrity of records in the case of a task failure, and ties up the applicable transient data records for the length of a task that enqueues on them.

Up to 32767 buffers and 255 strings can be specified for a transient data set, with serial processing only through a destination.

Specifying a higher trigger level on a destination causes a smaller number of tasks to be initiated from that destination. Transient data can participate in file subtasking if SUBTSKS=1 is specified in the SIT (see "Using VSAM subtasking" on page 157).

Nonrecoverable TD queue

A nonrecoverable TD queue has costs associated with it.

WRITEQ	**READQ**	**DELETEQ**
1.5	1.3	1.3

Note:

The main difference between nonrecoverable and logically recoverable TD queues occurs at sync point time. At sync point, the new TD queue addresses are put in the log buffer and a forced log write is requested. The cost to put the data in the buffer is 2 K. The cost of writing the log buffer to the coupling facility is described in "Using coupling facility data tables" on page 160.

Logically recoverable TD queue

A logically recoverable TD queue has costs associated with it.

WRITEQ	**READQ**	**DELETEQ**
First: 2.8 Subsequent:1.5	First: 2.4 Subsequent:1.4	1.1

Notes:

The main difference between nonrecoverable and logically recoverable TD queues occurs at sync point time. At sync point, the new TD queue addresses are put in the log buffer and a forced log write is requested. The cost to put the data in the buffer is 2 K. The cost of writing the log buffer to the coupling facility is described in "Using coupling facility data tables" on page 160.

Physically recoverable TD queue

Physically recoverable WRITEQ requests involve forcing a VSAM I/O and forcing a log write to the coupling facility (CF) for every request.

WRITEQ	**READQ**	**DELETEQ**
19.7	First: 9.3 Subsequent:8.8	8.7

Intrapartition transient data considerations

The approximations for nonrecoverable and logically recoverable intrapartition transient data queues do not include any VSAM I/O cost.

A VSAM I/O operation costs approximately 11.5 K and occurs in the following situations:

- When attempting to write an item that will not fit in any buffer.
- When reading an item that is not in the buffer.
- When reading a control interval from DASD and there is no available buffer space. If this situation occurs, the least recently used buffer must first be written out. Therefore, under certain circumstances, a READQ could incur the cost of two VSAM I/O operations.

Multiple VSAM buffers

When you use multiple buffers and strings for intrapartition transient data (TD) support, this can remove the possible constraint in transient data caused by the use of a single system-wide buffer (and string). You can use statistics to tune the system with regard to transient data usage.

If requests have to be queued, they are queued serially by transient data destination. Typically, a request has to be queued if the control interval it requires is in use, or if one or more previous requests for the same queue or destination are already waiting. Under these conditions, the servicing of requests for other queues or destinations can continue.

The use of multiple buffers also increases the likelihood that the control interval required by a particular request is already available in a buffer. This can lead to a significant reduction in the number of real input/output requests (VSAM requests) that have to be performed. However, VSAM requests are always executed whenever their use is dictated by the requirements of physical and logical recovery.

The number of buffers that CICS allocates for transient data is specified by the **TD** system initialization parameter. The default is three.

The provision of multiple buffers allows CICS to retain copies (or potential copies) of several VSAM control intervals (CIs) in storage. Several transient data requests to different queues can then be serviced concurrently using different buffers. Requests are serialized by queue name, not globally. Multiple buffers also allow the number of VSAM requests to the TD data set to be reduced by increasing the likelihood that the CI required is already in storage and making it less likely that a buffer must be flushed to accommodate new data. VSAM requests are still issued when required by recovery considerations.

The benefits of multiple buffers depend on the pattern and extent of usage of intrapartition transient data in an installation. For most installations, the default specification (three buffers) should be sufficient. Where the usage of transient data is extensive, it is worthwhile to experiment with larger numbers of buffers. The buffer statistics give sufficient information to help determine a suitable allocation. In general, the aim of the tuning should be to minimize the number of times a task must wait because no buffers are available to hold the required data.

In the tuning process, there is a trade-off between improving transient data performance and increased storage requirements. Specifying a large number of buffers might decrease transient data I/O and improve concurrency, but might also lead to inefficient usage of real storage. Also, if there is a large number of buffers and a small number of queues, internal buffer searches per queue may take longer.

The buffers are obtained from the ECDSA during initialization.

Multiple VSAM strings

As far as concurrent input/output operations with CICS are concerned, the transient data (TD) programs issue VSAM requests whenever real input/output is required between the buffers and the VSAM TD data sets. The use of multiple VSAM strings enables multiple VSAM requests to be executed concurrently, which in turn leads to faster servicing of the buffers.

VSAM requests are queued whenever the number of concurrent requests exceeds the number of available strings. Constraints caused by this be relieved by increasing the number of available strings, up to a maximum of 255. The limit of 255 on the number of strings should be taken into consideration when choosing the number of buffers. If the number of buffers is more than the number of strings, the potential for string waits increases.

The number of VSAM strings that CICS allocates for TD is specified by the **TD** system initialization parameter. The CICS default is 3.

Logical recovery

Logging and enqueuing occur with logical recovery transactions (including dynamic backout of the failing task's activity on the transient data queue). Logical recovery is generally used when a group of records have to be processed together for any reason, or when other recoverable resources are to be processed in the same task.

During processing of the transient data request, the destination queue entry is enqueued from the first request, for either input or output, or both (if the queue is to be deleted), until the end of the UOW. This means that none of the other tasks can access the queue for the same purpose during that period of time, thus maintaining the integrity of the queue's status.

At the end of the UOW (sync point or task completion), sync point processing takes place and the queue entry is logged. Any purge requests are processed (during the UOW, a purge only marks the queue ready for purging). The empty control intervals are released for general transient data use. Any trigger levels reached during the UOW cause automatic task initiation to take place for those queues that have a trigger level greater than zero. The buffer is written out to the VSAM data set as necessary.

The DEQUEUE function on the queue entry occurs, releasing the queue for either input or output processing by other tasks. Records written by a task can then be read by another task.

Logging activity

With *physical* recovery, the queue entry is logged after each READQ, WRITEQ, and DELETEQ command, and at an activity keypoint time (including the warm keypoint).

With *logical* recovery, the queue entry is logged at sync point and at activity keypoint time (including the warm keypoint).

Secondary extents for intrapartition transient data

During initialization of intrapartition transient data, CICS initializes a VSAM empty intrapartition data set by formatting control intervals until the first extent of the data set is filled. Additional control intervals are formatted as required if the data set has been defined with multiple extents.

The use of secondary extents allows more efficient use of DASD space. You can define an intrapartition data set with primary extents large enough for normal activity, and with secondary extents for exceptional circumstances, such as unexpected peaks in activity.

It follows that you can reduce or eliminate the channel and arm contention that is likely to occur because of heavy use of intrapartition transient data.

Extrapartition transient data considerations

Extrapartition destinations are, in practice, sequential data sets where CICS uses the QSAM PUT LOCATE or PUT MOVE commands.

The main performance factor to note is the possibility of operating system waits; that is, the complete CICS region waits for the I/O completion. A lengthy wait can occur for one of the following reasons:

- No buffer space is available.
- Secondary space is allocation.
- Volume (extent) switching is available.
- The data set has been opened or closed dynamically.
- A forced end of the volume has been caused by the application.
- The data set is defined on a physical printer and the printer has run out of paper.
- A RESERVE command has been issued for another data set on the same volume.

Therefore, try to eliminate or minimize the occurrences of CICS region waits by:

- Having sufficient buffering and blocking of the output data set
- Avoiding volume switching by initially allocating sufficient space
- Avoiding dynamic OPEN or CLOSE actions during peak periods.

An alternative method of implementing sequential data sets is to employ a CICS user journal. Table 25 on page 200 summarizes the differences between these two methods.

Table 25. Extrapartition transient data versus user journal

Extrapartition TD	**User Journal**
Region (CICS) may wait	Task waits
Buffer location: In MVS storage	Buffer location: In DSA
Number of buffers: 1 - 32767	2 buffers
Input or output	Both input and output, but tasks may wait

Table 25. Extrapartition transient data versus user journal (continued)

Extrapartition TD	**User Journal**
Accessible by multiple tasks	• Accessible for output by multiple tasks • Accessible for input by single task under exclusive control

The approximate calculations for performance costs in extrapartition TD queues do not include any I/O cost. An I/O operation for a physically sequential file costs approximately 7 K and occurs in the following situations:

- When attempting to write an item that does not fit in any buffer.
- When reading an item that is not in the buffer.
- When reading data from DASD and there is no available buffer space. If this situation occurs, the least recently used buffer must first be written out.

Therefore, under certain circumstances, a READQ could incur the cost of two I/O operations.

Extrapartition TD queues are nonrecoverable.

WRITEQ	**READQ**
1.2	1.0

Indirect destinations

To avoid specifying extrapartition data sets for the CICS-required entries (such as CSMT and CSSL) in CSD definitions for TD queues, you are recommended to use indirect destinations for combining the output of several destinations to a single destination. This saves storage space and internal management overheads.

Long indirect chains can, however, cause significant paging to occur.

Global CICS enqueue and dequeue: Performance and tuning

Global CICS enqueue and dequeue extends the CICS application programming interface to provide an enqueue mechanism that serializes access to a named resource across a specified set of CICS regions contained within a sysplex.

Because global CICS enqueue and dequeue eliminates a significant cause of inter-transaction affinity, it enables better exploitation of parallel sysplex, providing better performance, capacity, and availability. It also reduces the need to provide inter-transaction affinity rules to dynamic routing mechanisms such as CICSPlex SM, thus reducing the system management cost of exploiting parallel sysplex.

CICS uses z/OS global resource serialization to provide sysplex-wide protection for the resources that participate in global CICS enqueue and dequeue. For more information on z/OS global resource serialization, see z/OS MVS Planning: Global Resource Serialization.

Implementation

You use an ENQMODEL resource definition to define each named resource for which the **`EXEC CICS ENQ`** and **`EXEC CICS DEQ`** commands have a sysplex-wide scope. The CICS regions that need to use sysplex-wide enqueue or dequeue function must all have the required ENQMODEL resources defined and installed. The recommended way to ensure this is for the CICS regions to share a CSD, and for the initialization group lists to include the same ENQMODEL groups. For more information on creating ENQMODEL resource definitions, see ENQMODEL resources.

For applications where the resource name is configured dynamically, so is not known in advance, you can use the enqueue EXEC interface program exits XNQEREQ and XNQEREQC to supply characters at the start of the resource name that match a suitable ENQMODEL resource definition. For more information on these user exits, see Enqueue EXEC interface program exits XNQEREQ and XNQEREQC.

When the **EXEC CICS ENQ** and **EXEC CICS DEQ** commands are issued for a resource, CICS checks for a matching installed ENQMODEL definition. If there is a matching ENQMODEL resource that specifies an enqueue scope, CICS passes the information to z/OS global resource serialization to manage the enqueue. z/OS global resource serialization provides sysplex-wide protection of the resource.

z/OS global resource serialization includes resource name lists (RNLs) that specify the scope of resources. RNL processing can cause the scope of resources to change from the scope that was specified in the ENQMODEL resource definition in CICS.

The default in z/OS is that global resource serialization searches the appropriate RNL for enqueue and dequeue requests, and uses the RNL to determine the scope of the resource. However, the default in CICS, as specified by the **NQRNL** system initialization parameter, is that all enqueue and dequeue requests specify **RNL=NO** and so are excluded from RNL processing. This action means that global resource serialization only uses the scope specified in the ENQMODEL resource definition in CICS, but it also means that the enqueue request is ignored by alternative serialization products, which impacts protection of the resource to systems outside the current global resource serialization environment that are using alternative serialization products. If you want z/OS global resource serialization to use RNL processing for enqueue and dequeue requests from CICS, specify the system initialization parameter **NQRNL=YES** for the CICS regions where RNL processing should be performed.

For more information on RNL processing for global resource serialization, see z/OS MVS Planning: Global Resource Serialization.

Recommendations

z/OS global resource serialization combines systems into a global resource serialization complex. One or more systems are connected to each other in a ring configuration (GRS=RING) or connected to a coupling facility lock structure in a star configuration (GRS=STAR). When global resource serialization is initialized as a star configuration, all the information about resource serialization is held in the ISGLOCK coupling facility structure. Global resource serialization accesses the coupling facility when a requestor issues an enqueue or dequeue instruction on a global names resource.

Note: Use GRS=RING with caution as this configuration can result in serious performance constraints. For performance reasons, in a sysplex of greater than two MVS images use a global resource serialization star configuration.

The performance impact can be for many reasons, but primarily it is due to the delay in having the request complete the ring. A large number of MVS images in the ring combined with a large value for RESMIL causes delays in the request completing the ring. The enqueue request cannot be granted until the request returns to the originating MVS image. Use a value of 0, or no greater than 1, for RESMIL in the GRSCNF member of SYS1.PARMLIB.

CICS monitoring facility: Performance and tuning

The CICS monitoring facility collects data about the performance of all user-supplied and CICS-supplied transactions during online processing for later offline analysis. Monitoring data is useful for performance, tuning, and for charging your users for the resources they use. The records produced by CICS monitoring are of the MVS System Management type 110 and they are written to an SMF data set.

Introduction to CICS monitoring has information about the different types of monitoring data.

In terms of performance, collecting performance class data can be a significant overhead. The overhead is likely to be about 5% to 10%, but depends on the workload. MVS address space or RMF data can be gathered whether or not the CICS monitoring facility is active, to give an indication of the performance overhead incurred when using the CICS monitoring facility. CICS Monitoring Domain statistics show the number of monitoring records produced of each type.

If you do not need accounting information because other billing processes exist, and you have other means of gathering any performance data required, do not use the CICS monitoring facility to collect performance class data. Do not collect exception class data if you do not require it.

Recording of monitoring data incurs an overhead, but, to tune a system, both performance and exception information might be required. If tuning is not a daily process, the CICS monitoring facility might not need to be run all the time. When tuning, run the CICS monitoring facility during peak volume times because at those times performance problems typically occur.

To help reduce the overhead, data compression for monitoring records is set as the default. If overuse of the SMF data set is a potential problem, consider excluding fields from monitoring records.

Controlling CICS monitoring explains how to set CICS monitoring facility options using system initialization parameters and how to change these options while CICS is running.

CICS trace: performance and tuning

The CICS tracing, handled by the CICS trace domain, records all requests that application programs make to CICS for various services. The storage and processing requirements depend on the number of trace entries that are recorded. Using CICS trace increases processing requirements considerably. Not using CICS trace, however, reduces the amount of problem determination information that is available for the CICS region.

CICS does not provide a direct measurement of processor use caused by tracing. RMF can show the processing and storage requirements. Auxiliary trace, where trace entries are written to auxiliary storage, has an additional cost because of the I/O operations. Although two buffers are used for auxiliary trace, even if the I/O can be overlapped, the I/O rate is quite large for a busy system.

You can control the amount of tracing that is done in a CICS region. You can limit the transactions or components that are traced, and the levels of trace data that are captured for them. You can set these options at CICS startup by using CICS system initialization parameters, or while CICS is running by using CICS interfaces. For information about defining the tracing that is done in the CICS region, see CICS trace.

CICS always performs exception tracing when it detects an exception condition, so you always have first failure data capture regardless of the limits that you set for CICS trace. In a production region, for example, you might want to set tracing options so that exception traces are written to auxiliary storage, but no other tracing is carried out. For instructions describing how to do this, see CICS exception tracing.

The trace data produced by CICS trace has a number of possible destinations. Any combination of any of these destinations can be active at any time:

- The internal trace table
- The auxiliary trace data sets
- The MVS generalized trace facility (GTF) data sets
- The JVM server trace file in z/OS Unix System Services

Also, when a transaction dump is produced, CICS copies the internal trace table to produce the transaction dump trace table. For information about selecting trace destinations, see Setting trace destinations and tracing status.

Internal trace table: storage use

Every CICS region must always have an internal trace table. The internal trace table is used as a buffer for the other trace destinations. If no trace destinations at all are currently started, CICS still writes exception trace entries to the internal trace table to provide first failure data capture.

You use the **TRTABSZ** system initialization parameter to specify the size of the internal trace table at CICS startup. The minimum size of the internal trace table is 16 KB, and the maximum size is 1 GB. The default size is 12288 KB (12 MB). The trace table should be large enough to contain the entries needed for debugging purposes.

CICS obtains MVS 64-bit (above-the-bar) storage (outside the CICS DSAs) for the internal trace table.

If you change the size of the internal trace table, check your current setting for the z/OS parameter **MEMLIMIT**. **MEMLIMIT** limits the amount of 64-bit storage that the CICS address space can use. Your setting for **TRTABSZ** must remain within **MEMLIMIT**, and you must also allow for other use of 64-bit storage in the CICS region.

For information about the **MEMLIMIT** value for CICS, and instructions to check the value of **MEMLIMIT** that currently applies to the CICS region, see Estimating, checking, and setting MEMLIMIT in Improving performance. For further information about **MEMLIMIT** in z/OS, see Limiting the use of memory objects in the z/OS MVS Programming: Extended Addressability Guide.

Transaction dump trace table: storage use

When a transaction dump is produced, CICS copies the current internal trace table to produce the transaction dump trace table. CICS obtains MVS storage in 64-bit (above-the-bar) storage for the transaction dump trace table when a transaction dump is taken.

You use the **TRTRANSZ** system initialization parameter to specify the size of the transaction dump trace table. The minimum size is 16 KB, and the default size is 1024 KB.

Before CICS TS for z/OS, Version 4 Release 2, the transaction dump trace table was in 31-bit (above-the-line) storage. If you specified a small size for the transaction dump trace table at that time because of concerns about the availability of 31-bit storage, consider reviewing your **TRTRANSZ** value to provide a larger transaction dump trace table now that 64-bit storage is used.

Because the transaction dump trace table is in 64-bit storage, check your current setting for the z/OS parameter **MEMLIMIT** when you set the size of the trace table.

Auxiliary trace data sets: storage use

The auxiliary trace data sets are CICS-owned BSAM data sets on disk or tape. You must create the data sets before you start CICS; you cannot define them while CICS is running. For instructions to set up the auxiliary trace data sets, see Setting up auxiliary trace data sets.

When you start auxiliary trace, either at CICS startup or while CICS is running, two 4 KB buffers for the CICS auxiliary trace data sets are allocated from MVS storage in the 31-bit (above-the-line) storage of the CICS region. MVS storage is not included in the CICS DSAs. The buffers are freed if you stop auxiliary trace, but they are not freed when you pause auxiliary trace or switch between the auxiliary trace data sets.

GTF data sets: storage use

The GTF buffer is allocated in 64-bit storage.

JVM server trace: storage use

Setting the tracing level for the SJ component to 3, 4 or 5 results in increased processing requirements in all JVM servers in the region. For more information see Activating and managing tracing for JVM servers in Troubleshooting and support.

CICS security: Performance and tuning

CICS provides an interface for an external security manager (ESM), such as RACF, for three types of security: transaction, resource, and command security.

Effects

Transaction security verifies the authorization of an operator to run a transaction. Resource security limits access to data sets, transactions, transient data destinations, programs, temporary storage records, and journals. Command security is used to limit access to specific commands and applies to special system programming commands; for example, **EXEC CICS INQUIRE**, **SET**, **PERFORM**, **DISCARD**, and **COLLECT**. Transactions that are defined with CMDSEC=YES must have an associated user.

Limitations

Protecting transactions, resources, or commands unnecessarily increases both processor cycles, and real and virtual storage requirements.

Recommendations

Because transaction security is enforced by CICS, it is suggested that the use of both resource security and command security should be kept to the minimum. The assumption is that, if operators have access to a particular transaction, they therefore have access to the appropriate resources.

Implementation

Resource security is defined with the **RESSEC(YES)** attribute in the TRANSACTION definition. Command security is defined with the **CMDSEC(YES)** attribute in the TRANSACTION definition.

Monitoring

No direct measurement of the overhead of CICS security is given. RMF shows overall processor usage.

For more information, see RACF facilities.

Tuning for VERIFY TOKEN and SIGNON TOKEN

For best performance, ensure a sufficient number of open TCBs and define programs as threadsafe.

VERIFY TOKEN and **SIGNON TOKEN** requests run on an open TCB if possible. If the **VERIFY TOKEN** or **SIGNON TOKEN** is issued on an open TCB, it runs the request on this TCB. If the **VERIFY TOKEN** or **SIGNON TOKEN** is not issued on an open TCB, it switches to an open TCB if one is available, otherwise it switches to the Resource Owning (RO) TCB.

For best performance, set the **MAXOPENTCBS** system initialization parameter to a high enough value to allow sufficient open TCBs for the workload, and define programs that use **VERIFY TOKEN** or **SIGNON TOKEN** as threadsafe.

CICS startup and shutdown time: Performance and tuning

If you want to reduce the amount of time required for CICS startup and normal shutdown, the areas to check include the startup procedures and autoinstall.

The IBM Redbooks publication, IBM z Systems Mean Time to Recovery Best Practices, SG24-7816, contains information about how to customize CICS to minimize startup and shutdown time.

The following topics describe how to improve performance for CICS startup and shutdown.

Improving startup procedure

Because various configurations are possible with CICS, different aspects of the startup might require attention.

Procedure

You can define and tune aspects to improve startup performance. For more information about the CICS startup procedures and CICS system initialization, see CICS startup.

1. Define the following items:
   a) The global and local catalogs
   b) The CICS system definition (CSD) data set

   c) The temporary storage data sets or transient data intrapartition data sets

   For details on how to define each data set, see Defining data sets.

2. When defining your terminals, pay attention to the position of group names within the GRPLIST. If the group containing the TYPETERMs is last, all the storage used for building the terminal definitions is held until the TYPETERMs are known. This might cause your system to go short on storage.

   Groups in the GRPLIST in the system initialization table (SIT) are processed sequentially. Place the groups containing the model TERMINAL definitions followed by their TYPETERMs in the GRPLIST before the user transactions and programs. This process minimizes the virtual storage that is tied up while CICS is processing the installation of the terminals.

   Note: All terminals are installed, even surrogate terminal control table (TCT) entries for MRO.

   You must ensure that the DFHVTAM group precedes any TERMINAL or TYPETERM definition in your GRPLIST. The DFHVTAM group is contained in the DFHLIST group list, so adding DFHLIST first to your GRPLIST ensures that the condition is met. If you do not add DFHLIST, the programs used to build the TCT are loaded for each terminal, thus slowing initial and cold starts.

   Do not have more than 100 entries in any group defined in the CSD. If you have too many entries, this might cause unnecessary overhead during processing, and make maintenance of the group more difficult.

3. Enure that changing the **START** parameter does not change the default for any facilities that your users do not want to have auto-started. Any facility that you might want to override can be coded in the **PARM** on the EXEC statement, or all of them can be overridden by specifying by specifying the ALL option for the **START** parameter.
4. If you do not intend to use CICS web support or the Secure Sockets Layer, ensure that TCPIP=NO is specified in the SIT. If TCPIP=YES is specified, the Sockets domain task control block is activated.
5. Tune the VSAM parameters of the local and global catalogs to suit your installation:

   a) Control interval (CI) sizes should be changed for optimum data and DASD sizes (see "Local shared resources (LSR) or nonshared resources (NSR)" on page 147 for more information). In most cases 2KB index CI, and 8 KB or 16 KB data CI, are suitable sizes.

   b) You can you specify the **BUFNI** and **BUFND** parameters in your JCL for the global catalog data set with the **AMP** parameter, rather than using **BUFSPACE**.

   c) Alter the number of index buffers by coding the number of strings plus the number of index set records in the index. The number of records in the index set can be calculated from IDCAMS LISTCAT information as follows:

   - T = total number of index records (index REC-TOTAL)
   - D = data control interval size (data CISIZE)
   - C = data control intervals per control area (data CI/CA)
   - H = data high-used relative byte address (data HURBA)

   d) The number of index set records can then be computed. The calculation is really the number of used control areas. The number of sequence set records must be the same as the number of used CAs.

   - *The number of sequence set records: S = H / (D X C)*
   - *The number of index set records: I = T - S*

   Do not spend time trying to tune free space as it has no effect.

   You can obtain the number of index levels by using the IDCAMS LISTCAT command against a GCD after CICS has been shut down. Because a cold start mainly uses sequential processing, it should not require any extra buffers in addition to the buffers automatically allocated when CICS opens the file.

6. Consider whether to use the recovery manager utility program DFHRMUTL. On cold and initial starts, CICS normally deletes all the resource definition records from the global catalog. You can save the time taken to delete resource definition records by using the recovery manager utility program, DFHRMUTL. For more information, see Recovery manager utility (DFHRMUTL).

- Before a cold start, run DFHRMUTL with `SET_AUTO_START=AUTOCOLD,COLD_COPY` as input parameters. This creates a copy of the global catalog data set that contains only those records needed for a cold start. If the return code from this job step is normal, you can replace the original global catalog with the new copy (taking an archive of the original catalog if you want). An example of the JCL is provided with the description of DFHRMUTL.
- Before an initial start, run DFHRMUTL with `SET_AUTO_START=AUTOINIT,COLD_COPY` as input parameters, and follow the same procedure to use the resulting catalog.

7. Allocate your DATA and INDEX data sets on different units, if possible.
8. Consider the use of autoinstalled terminals as a way of improving cold start, even if you do not expect any storage savings. On startup, fewer terminals are installed, reducing the startup time.
9. Set the **RAPOOL** system initialization parameter to a value that allows faster autoinstall rates. For more information, see "Setting the size of the receive-any pool" on page 129.
10. Specify the buffer, string, and key length parameters in the LSR pool definition. Setting these parameters reduces the time taken to build the LSR pool, and also reduces the open time for the first file to use the pool.

    If you have defined performance groups for the CICS system, ensure that all steps preceding the CICS step are also in the same performance group or, at least, have a high enough dispatching priority so as not to delay their execution.

    The use of DISP=(...,PASS) on any non-VSAM data set used in steps preceding CICS reduces allocation time the next time the data sets are needed. If you do not use PASS on the DD statement, this causes the subsequent allocation of these data sets to go back through the catalog, which is a time-consuming process.

    If possible, have one VSAM user catalog with all of the CICS VSAM data sets and use a STEPCAT DD statement to reduce the catalog search time.

    Keep the number of libraries defined by DFHRPL to a minimum. One large library requires less time to perform the LLACOPY than many smaller libraries. Similar consideration should be applied to any dynamic LIBRARY resources installed at startup.You can use the shared modules in the link pack area (LPA) to help reduce the time required to load the CICS nucleus modules. For advice on how to install CICS modules in the LPA, see Installing CICS modules in the MVS link pack area in Installing.

    CICS does not load programs at startup time for resident programs. The storage area is reserved, but the program is loaded on the first access through program control for that program. This process speeds up the startup. The correct way to find a particular program or table in storage is to use the program-control LOAD facility to find the address of the program or table. If it is the first access, using the LOAD facility physically loads the program into its predefined storage location .

    The use of a program list table post initialization (PLTPI) task to load these programs is one possible technique, but you must bear in mind that the CICS system is not operational until the PLTPI processing is complete, so you should not load every program. Load only what is necessary, or the startup time might increase.

Autoinstall performance

You might want to increase the number of buffers to improve autoinstall performance. Increasing the number of buffers can stop the high-level index being read for each autoinstall.

If you have many terminals autoinstalled, shutdown can fail due to the value of the **MXT** system initialization parameter being reached or CICS becoming short on storage. To prevent this possible cause of shutdown failure, consider putting the CATD transaction in a class of its own to limit the number of concurrent CATD transactions. Also, the **AIQMAX** parameter can be specified to limit the number of devices that can be queued for autoinstall. This parameter protects against abnormal consumption of virtual storage by the autoinstall or delete process, caused as a result of some other abnormal event.

If the CATD transaction limit is reached, the **AIQMAX** system initialization parameter affects the LOGON, LOGOFF, and BIND processing by CICS. CICS requests the z/OS Communications Server to stop passing such requests to CICS. The z/OS Communications Server holds the requests until CICS indicates that it can accept further commands.

This occurs when CICS has processed a queued autoinstall request.

MVS automatic restart management

You can use the MVS automatic restart manager (ARM) to implement a sysplex-wide integrated automatic restart mechanism. A sysplex can use ARM and z/OS Communications Server persistent sessions spread across many terminal-owning regions (TORs) in a generic resource set.

Automatic restart management (ARM) is a sysplex-wide integrated restart mechanism that performs the following tasks:

- Restarts MVS subsystems in place if they abend (or if notified of a stall condition by a monitor program)
- Restarts all the elements of a workload (for example, CICS TORs, application-owning regions (AORs), file-owning regions (FORs), and DB2) on another MVS image after an MVS failure
- Restarts a failed MVS image

ARM and z/OS Communications Server persistent sessions provide good recovery times in the event of a TOR failure, and the TOR restart is reduced because only a fraction of the network must be rebuilt. You can log on to the generic resource while the failed TOR restarts.

ARM provides faster restart by providing surveillance and automatic restart. The need for operator-initiated restarts, or other automatic restart packages, are eliminated. For more information about MVS automatic restart management, see Implementing MVS automatic restart management in Installing and z/OS MVS Setting Up a Sysplex.

CICS business transaction services: Performance and tuning

Business transaction services (BTS) introduced a business transaction model to CICS.

Effects

You can use BTS to create a type of program that controls the flow of many separate CICS transactions so that these individual transactions become a single business transaction.

Recommendations

A BTS transaction can comprise many separate CICS transactions and also can span a considerable execution time, so there are no specific performance recommendations for BTS transactions. However, some general observations can be useful.

Implementation

To support BTS functionality, CICS keeps data in new types of data sets: the local request queue (DFHLRQ) and a BTS repository. The local request queue data set stores pending BTS requests. Each CICS region has its own data set. The local request queue data set is a recoverable VSAM key-sequenced data set (KSDS). Tune it for best performance like a VSAM KSDS.

You can have one or more BTS repositories. A BTS repository is normally a VSAM KSDS and holds state data for processes, activities, containers, events, and timers. A BTS repository is associated with a process through the PROCESSTYPE definition. If the activities of a BTS process are to be dispatched on more than one CICS region, their BTS repositories must be shared between those regions. The repository can be either of the following file types:

- A VSAM KSDS file that is owned by a file-owning region and defined as REMOTE in participating regions
- A VSAM RLS file that is shared between the participating regions

To support the execution of the BTS processes, CICS runs one or many transactions. A BTS process consists of one or more activities. Each activity runs as a series of CICS transaction executions. If an activity becomes dormant, for example, it is waiting for an event, the activity restarts after that event occurs, and a new CICS transaction is started, even if this is a continuation of the business transaction. You might see many executions of the transaction identifier specified in a process or activity definition in the CICS statistics for a single BTS transaction. The application program that is run when an activity is executed is not necessarily the one that is defined in the transaction definition. In BTS, the Process or Activity definition in application programs can specify a different program to run.

The number of transactions run and the number and type of file accesses to the BTS repository, depend on how you choose to use BTS services. To see this information for your applications, examine the CICS statistics reports. Be aware that containers are stored in the BTS repository. Ensure that the repository is large enough to contain all the active BTS data. A good way to do this is to use scaling, based on a test system.

You can use monitor data, DFHCBTS, to collect information on activities within processes.

Managing workloads

Workload management in a sysplex is provided by the z/OS Workload Manager (WLM) and by CICSPlex SM workload management.

The z/OS Workload Manager

The z/OS Workload Manager provides automatic and dynamic balancing of system resources (central processors and storage) across a sysplex.

The z/OS Workload Manager balances system resources by:

- Adopting a goal-oriented approach
- Gathering real time data from the subsystems that reflect performance at an individual task level
- Monitoring z/OS- and subsystem-level delays and waits that contribute to overall task execution times
- Dynamically managing the resources of the sysplex, using the performance goals, and the real time performance and delay data, as inputs to system resource management algorithms.

This resource management is particularly significant in a sysplex environment, but is also of value to subsystems running in a single z/OS image.

Note: If you use CICSPlex SM to control dynamic routing in a CICSplex, you can base its actions on the CICS response time goals of the CICS transactions as defined to the z/OS Workload Manager. See Dynamic routing with CICSPlex SM.

The z/OS Workload Manager provides the following benefits:

- Improved performance through z/OS resource management. Improvement can depend on many factors, for example:
  - System hardware configuration
  - How the system is partitioned
  - Whether CICS subsystems are single or multiregion
  - The spread of types of applications or tasks performed, and the diversity of their profile of operation
  - The extent to which the sysplex workload changes dynamically.
- Improved efficiency of typical z/OS sysplexes through improved overall capacity and increased work throughput.
- Simplified z/OS tuning. Systems that have an operating signature that makes it difficult or time consuming to attain or maintain optimal tuning by current means can benefit the most.

The main benefit is that you do not need to continually monitor and tune CICS to achieve optimum performance. You can set your workload objectives in the service definition, then the workload component of z/OS manages the resources and the workload to achieve your objectives.

The z/OS Workload Manager produces performance reports that you can use to establish reasonable performance goals and for capacity planning.

The CICS function for z/OS workload management incurs negligible impact on CICS storage.

CICS support for the z/OS Workload Manager is initialized automatically during CICS startup. All CICS regions (and other z/OS subsystems) running on a z/OS image with z/OS workload management are subject to the effects of the Workload Manager.

User-written resource managers and other non-CICS code that is attached to CICS through the RMI should be modified to provide z/OS Workload Manager support, if workload management is to work correctly for CICS-based tasks which cross the RMI into such areas.

The IBM Redbooks Publication System Programmer's Guide to: Workload Manager, SG24-6472-03, gives a broad understanding of the Workload Manager component of the z/OS system. It covers basic aspects of WLM together with the new functions available in the z/OS release up to z/OS 1.7. The book provides a discussion on how to create WLM policies based on business goals and the types of transactions you run in your systems.

Terms used in z/OS workload management

The following terms are used in the description of z/OS workload management.

classification rule
A rule used by the workload manager component of z/OS to assign a service class.

service class
A group of work that has the same service goals or performance objectives, resource requirements, or availability requirements. For workload management, a service goal and, optionally, a resource group is assigned to a service class.

service definition
An explicit definition of all the workloads and processing capacity in a sysplex. A service definition includes service policies, workloads, service classes, resource groups, and classification rules.

service policy
A set of performance goals for all z/OS images using z/OS workload management in a sysplex. There can be only one active service policy for a sysplex, and all subsystems in goal mode within that sysplex process towards that policy. However, you can create several service policies, and switch between them to cater for the different needs of different processing periods.

workload
A group of service classes.

Span of z/OS Workload Manager operation

The z/OS Workload Manager operates across a sysplex. There can be only one active service policy for all z/OS images running in a sysplex.

All CICS regions (and other z/OS subsystems) running on a z/OS image with z/OS workload management active are subject to the effects of workload management.

If the CICS workload involves non-CICS resource managers, such as DB2 and DBCTL, CICS passes information through the resource manager interface (RMI) to enable the z/OS Workload Manager to relate the part of the workload within the non-CICS resource managers to the part of the workload within CICS.

The CICS interface modules that handle the communication between a task-related user exit and the resource manager are usually referred to as the resource manager interface (RMI) or the task-related user exit (TRUE) interface.

Performance goals for CICS regions

You can define performance goals, such as response times, for CICS (and other z/OS subsystems that comprise your workload).

You can define goals for:

- Individual CICS regions

- Groups of transactions running under CICS
- Individual transactions running under CICS
- Transactions associated with individual userids
- Transactions associated with individual LU names.

To define the performance goals for CICS regions, allocate each CICS job a service class and then specify target response times for the service class. Typically, production regions and test regions are placed in different service classes, because response times for production regions are more critical than for test regions.

Workload management also collects performance and delay data, which can be used by reporting and monitoring products, such as the Resource Measurement Facility (RMF), Tivoli Decision Support for z/OS, or vendor products.

The service level administrator defines your installation's performance goals, and monitoring data, based on business needs and current performance. The complete definition of workloads and performance goals is called a *service definition*. You may already have this kind of information in a service level agreement (SLA).

Defining classification rules for your CICS workload

Classification rules determine how to associate incoming work with a service class. Optionally, the classification rules can assign incoming work to a report class, for grouping report data.

There is one set of classification rules for each service definition. The classification rules apply to every service policy in the service definition; so there is one set of rules for the sysplex.

You should use classification rules for every service class defined in your service definition. For more information, see Defining Classification Rules in z/OS MVS Planning: Workload Management.

Classification rules categorize work into service classes and, optionally, report classes, based on work qualifiers. You set up classification rules for each z/OS subsystem type that uses workload management. The work qualifiers that CICS can use (and which identify CICS work requests to the z/OS Workload Manager) are:

CT
Connection type

CTG
Connection type group

LU
LU name

LUG
LU name group

SI
Subsystem instance (generic applid)

SIG
Subsystem instance group

TC
Transaction class

TCG
Transaction class group

TN
Transaction identifier

TNG
Transaction identifier group

UI
Userid

UIG
Userid group.

Note:

1. Typically, work is classified in the region in which it arrives in CICS. For example, work originating from a user terminal is typically classified in a terminal-owning region. Web requests are typically classified in a listener region. Work originating in an application-owning region is classified in that region. Where a work request is passed between CICS regions, the transaction is not reclassified in each region. Instead, the original classification is passed with the transaction from region to region.
2. You can use group qualifiers to specify groups of transaction IDs or user IDs; for example, GRPACICS could specify a group of CICS transaction IDs, which you could specify in classification rules by TNG GRPACICS. Using group qualifiers is a much better method of specifying classification rules than classifying each transaction separately.
3. The WLM service class token is not supported over a z/OS Communications Server LU62 link because there is only one set of rules for the sysplex. LU62 links can be outside the z/OS SYSPLEX and WLM could not access the information.

You can use classification groups to group disparate work under the same work qualifier—if, for example, you want to assign it to the same service class.

You can set up a hierarchy of classification rules. When CICS receives a transaction, the z/OS Workload Manager searches the classification rules for a matching qualifier and its service class or report class. Because a piece of work can have more than one work qualifier associated with it, it may match more than one classification rule. Therefore, the order in which you specify the classification rules determines which service classes are assigned.

You are recommended to keep classification rules simple.

Defining service classes

Service classes are categories of work, within a workload, to which you can assign performance goals.

You can create service classes for groups of work with similar:

- Performance goals

  You can assign the following performance goals to the service classes:

  Response time
  You can define an average response time (the amount of time required to complete the work) or a response time with percentile (a percentage of work to be completed in the specified amount of time).

  Discretionary
  You can specify that the goal is discretionary for any work for which you do not have specific goals.

  Velocity
  For work not related to transactions, such as batch jobs and started tasks. For CICS regions started as started tasks, a velocity goal applies only during start-up.

  Note:

  1. For service classes for CICS transactions, you cannot define velocity performance goals, discretionary goals, or multiple performance periods.
  2. For service classes for CICS regions, you cannot define multiple performance periods.

- Business importance to the installation

  You can assign an importance to a service class, so that one service class goal is recognized as more important than other service class goals. There are five levels of importance, numbered, from highest to lowest, 1 to 5.

You can also create service classes for started tasks and JES, and can assign resource groups to those service classes. You can use such service classes to manage the workload associated with CICS as it starts up, but before CICS transaction-related work begins. (Note that when you define CICS in this way, the address space name is specified as TN, for the task or JES “transaction” name.)

There is a default service class, called SYSOTHER. It is used for CICS transactions for which z/OS workload management cannot find a matching service class in the classification rules—for example, if the couple data set becomes unavailable.

For RMF to provide meaningful Workload Activity Report data it is suggested that you use the following guidelines when defining the service classes for CICS transactions. In the same service class:

1. Do not mix CICS-supplied transactions with user transactions
2. Do not mix routed with non-routed transactions
3. Do not mix conversational with pseudo-conversational transactions
4. Do not mix long-running and short-running transactions.

Matching CICS performance parameters to service policies

You must ensure that the CICS performance parameters are compatible with the Workload Manager service policies used for the CICS workload.

In general, you should define CICS performance objectives to the z/OS Workload Manager first, and observe the effect on CICS performance. Once the z/OS Workload Manager definitions are working correctly, you can then consider tuning the CICS parameters to further enhance CICS performance. However, you should use CICS performance parameters as little as possible.

Performance attributes that you might consider using are:

- Transaction priority, passed on dynamic transaction routing.

  You should take care when choosing the priority to assign to each transaction. Although you can specify transaction priorities from 1 to 255, you should avoid using a large number of closely spaced values. You will get as much benefit if you use a small number of widely spaced values.

  The priority assigned by the CICS dispatcher must be compatible with the performance parameters defined to the z/OS Workload Manager.
- Maximum number of concurrent user tasks for the CICS region.
- Maximum number of concurrent tasks in each transaction class.
- Maximum number of sessions between CICS regions.

CICSPlex SM workload management

CICSPlex SM workload management directs work requests to a target region that is selected using one of four routing algorithms.

The queue algorithm (QUEUE)
: CICSPlex SM routes work requests initiated in the requesting region to the most suitable target region in the designated set of target regions.

The link neutral queue algorithm (LNQUEUE)
: The link neutral queue algorithm corresponds to the queue algorithm, except that the type of connection between the routing and target region is not considered.

The goal algorithm (GOAL)
: CICSPlex SM routes work requests to the target region that is best able to meet the goals that have been predefined using the z/OS Workload Manager.

The link neutral goal algorithm (LNGOAL)
: The link neutral goal algorithm corresponds to the goal algorithm, except that the type of connection between the routing and target region is not considered.

For more information, see Workload routing.

The CICSPlex SM dynamic routing program EYU9XLOP is invoked to route work requests to the selected target region. EYU9XLOP supports both workload routing and workload separation. You define to

CICSPlex SM which requesting, routing, and target regions in the CICSplex can participate in dynamic routing, and any affinities that govern the target regions to which particular work requests must be routed. The output from the CICS Interdependency Analyzer can be used directly by CICSPlex SM. For information about the CICS Interdependency Analyzer, see the CICS Interdependency Analyzer for z/OS, and the IBM Redbooks publication IBM CICS Interdependency Analyzer.

There are no special requirements for using CICSPlex SM workload management, which supports both the distributed routing and dynamic routing models of CICS. Workload management of the following types of requests is supported:

- Dynamic transaction routing
- Dynamic DPL
- Start requests
- BTS activities
- 3270 link requests

CICSPlex SM workload management offers the following benefits:

- A dynamic routing program to make more intelligent routing decisions; for example, based on workload goals.
- Improved CICS support for z/OS goal-oriented workload management.
- Easier access to a global temporary storage owning region in the z/OS sysplex environment. This avoids inter-transaction affinity that can occur with the use of local temporary storage queues.
- Intelligent routing (through CICSPlex SM) in a CICSPlex that has at least one requesting region linked to multiple target regions.

For information about setting up and using CICSPlex SM workload management, see Managing workloads through CICSPlex SM and Introduction to workload management.

Improving event processing performance

CICS Event Processing (EP) consists of three main components: capturing an event, dispatching the EP adapter, and emitting the event by running the EP adapter. For each component, different factors influence performance.

Factors when capturing an event

For CICS performance associated with running EP, consider the following factors:

- Processor usage is negligible with EP started and no event binding file installed.
- Processor usage is negligible with EP started and event binding files installed, but when the capture point is not defined in any <locationFilter> (the capture point and application command predicates) element in any of the Event Capture Specifications.
- For primary predicate matching, use of the attribute **`filterOperator="EQ"`** ('Equals' specified in the filtering predicate) can result in improved performance, compared to other **`filterOperator`** values. An optimization is used when **`filterOperator="EQ"`** is specified.

  Note: The CICS event binding editor uses an asterisk (*) to show the primary predicate for the capture point selected.
- If you are using <dataCapture> elements (**Emitted Business Information** items mapped in the **Information Sources** part of the capture specification) to capture specific parts of the data associated with a capture point, keep the number of <dataCapture> elements reasonably low. Otherwise, performance can be adversely affected, because a separate container is created for each data capture component. For example, if bytes 0 - 5, 10 - 15, 20 - 25, 30 - 35 of a record associated with an **`EXEC CICS WRITE`** command are to be captured, it is more efficient to define a single <dataCapture> element to capture bytes 0 - 35 than to define four <dataCapture> elements for each of the separate data areas.

- Where possible, do not include unnecessary filter predicates. For example, it is often not necessary to filter on both the transaction ID and the current program name.
- Put the filter predicates that exclude the most unwanted events before the filter predicates that exclude fewer unwanted events.
- Where possible, do not use filter predicates on zoned decimal or packed decimal data fields that might contain unusable data.

CICS statistics in the EVENTBINDING GLOBAL STATISTICS category show the Total Event Filter operations. For more information, see EVENTBINDING statistics.

Factors when dispatching the EP adapter

EP runs multiple dispatcher tasks, which run on L8 TCBs, to service the emitting of events. These dispatcher tasks count toward the limit set by CICS for the number of L8 and L9 mode open TCBs. CICS sets this limit automatically using the formula `(2 * MXT value) + 32`. So that EP does not monopolize the L8 TCBs, the maximum number of L8 TCBs used for EP dispatchers is limited to one third of the open TCB limit. CICS statistics in the EVENTPROCESS STATISTICS category show the peak event capture queue and the peak dispatcher tasks so that you can monitor the use of L8 TCBs by EP. For more information, see EVENTPROCESS statistics.

Adapters can be either attached or linked depending on the adapters configuration. Attaching a task for each event increases processor usage; using the attach method means that the event dispatcher task does not wait for the event to be emitted to continue processing the event queue.

When a transaction ID or user ID is specified in the CICS event binding editor adapter section, the adapter is attached as a separate task. When a transaction ID or user ID is not specified, the adapter is linked to from the dispatcher task. However, the HTTP EP adapter is always attached.

Factors when emitting events by running the EP adapters

The cost of running the EP adapters depends on the EP adapter that is being used:

TSQ
The TSQ EP adapter imposes the least cost in terms of CPU consumed.

WebSphere MQ
Each event has an `MQPUT1` call for the WebSphere MQ EP adapter.

Assured event emission: You need the capability and reliability of assured event emissions to build business-critical application extensions based on events. However, assuring event emission (using synchronous emission mode) transfers the cost of event emission from an asynchronous event processing task to the application thread. Therefore, assured event emission could have an adverse affect on application response time, similar to adding an MQPUT for the event to the application itself. Because of the effect on response time, consider carefully which events demand the added level of reliability that is achieved through synchronous emission mode. Asynchronous emission mode is the better choice when it is less important that an event is occasionally lost (between the time an event is captured but not yet emitted); for example, for mail applications when an event is infrequently lost. Specifying synchronous emission mode only where it is needed ensures that your business is making the best use of assured event emission with the least impact on performance.

HTTP
Each event has a `WEB OPEN`, `WEB CONVERSE`, and a `WEB CLOSE` call for the HTTP EP adapter. By default, CICS closes the client HTTP connection after the event is emitted, and a new connection is opened for the next event. CICS can keep client HTTP connections open after events are emitted, so that they can be reused for subsequent event emissions, and you can save the processor overhead for reopening the connections. To keep connections open, specify the SOCKETCLOSE attribute in the URIMAP resource that the HTTP EP adapter uses for the connection. Choose an appropriate expiry time for the connections based on your emission rate.

A CEPH task is attached for each event emitted using the HTTP adapter. The CEPH transaction has a **`DTIMEOUT`** value of 5 seconds so that the HTTP EP adapter tasks timeout if a connection cannot be established with the event server within 5 seconds. The CEPH task then emits the event to an HTTP 1.1 compliant server. CICS provides a PROFILE for the CEPH transaction called DFHECEPH. This profile has an **RTIMOUT** value of 5 seconds so that the HTTP EP adapter tasks timeout if the event server does not respond within 5 seconds. You can copy this profile and the CEPH transaction if you want to change them.

- If the emission rate is high compared to the HTTP 1.1 compliant server or network response time, these CEPH tasks can flood the CICS system resulting in the MXT limit being reached. You must copy the CEPH transaction and then assign your transaction to a transaction class to avoid the MXT limit being reached. The transaction class must have a **MAXACTIVE** value low enough to avoid reaching the MXT limit, and a **PURGETHRESH** value set to a non-zero value.
- Any CEPH tasks that are purged when the **PURGETHRESH** limit is reached do not emit their events. Any tasks purged when the **PURGETHRESH** limit is reached result in the following message being written to the CSMT transient data destination: `DFHAC2036 Transaction CEPH has failed with abend AKCC`.

Note: When you use the copied profile or transaction, you must change your adapter configuration in the Event binding editor to run the HTTP adapter using this new transaction.

Transaction start

The transaction start EP adapter starts a new CICS task. Therefore, processor usage increases overall because of starting the transaction that is driven as a result of the CICS event. Also, each data capture field is made available to a started transaction in a container. A large number of these CICS tasks can impose a significant increase in processor usage on the adapter.

CICS statistics in the EVENTPROCESS STATISTICS category show the number of the following events:

- Events dispatched to the WebSphere MQ EP adapter
- Events dispatched to the HTTP EP adapter
- Events dispatched to the Transaction EP adapter
- Events dispatched to the TSQ EP adapter
- Events dispatched to the Custom EP adapter

For more information, see EVENTPROCESS statistics.

CICS region storage for event processing

Event processing uses 64-bit (above-the-bar) storage in the CICS region. Your use of event processing therefore influences the value that you choose for the z/OS **MEMLIMIT** parameter that applies to the CICS region.

If you use the temporary storage queue (TSQ) adapter and select main temporary storage, this data is also in 64-bit storage.

For information about the **MEMLIMIT** value for CICS, and instructions to check the value of **MEMLIMIT** that currently applies to the CICS region, see Estimating, checking, and setting MEMLIMIT in Improving performance. For further information about **MEMLIMIT** in z/OS, see Limiting the use of memory objects in the z/OS MVS Programming: Extended Addressability Guide.

Improving DBCTL performance

You can tune your CICS-DBCTL setup to make efficient use of resources to help you reach performance objectives.

Performance parameters in CICS

System design considerations for CICS with DBCTL are similar to the design considerations that applied to local DL/I. For example, do not allow excessive database accesses or updates in a single UOW. However, some system design considerations are specific to CICS with DBCTL.

The fact that DBCTL is structured to have one TCB per thread is an additional consideration for CICS. This allows more concurrent processing, but you need to specify minimum and maximum numbers of threads that are consistent with the needs of your system. For more information, see "Specifying numbers of threads" on page 217.

The storage specified in CICS system initialization parameters DSALIM and EDSALIM is used for different resources in a CICS-DBCTL environment.

- DSALIM is used to specify the upper limit of the total amount of storage within which CICS can allocate the individual DSAs below the 16 MB line.
- EDSALIM is used to specify the upper limit of the total amount of storage within which CICS can allocate the individual EDSAs above 16 MB but below 2 GB.

Local uses DSA storage for PSB and DMB pools, but with DBCTL, these blocks are stored outside CICS. Instead, you need to allow for the storage DBCTL needs in CICS for DRA code when specifying DSALIM and EDSALIM. This storage is allocated in the CICS region, but not from DSA or EDSA storage. For information about specifying DSALIM and EDSALIM, see CICS dynamic storage areas..

Using single-phase commit

CICS can use single-phase commit instead of two-phase commit when, for a specific UOW, DBCTL is the only recoverable resource used. Using single-phase commit in these circumstances improves CICS performance with DBCTL by eliminating unnecessary logging, cutting restart time, decreasing transaction cost, and improving response time in both CICS and DBCTL. For information on using single-phase commit, see Increasing efficiency: single-update and read-only protocols.

Performance parameters in IMS

From an IMS point of view, tuning DBCTL is much like tuning an IMS system.

Additional considerations are DRA threads, described in "Specifying numbers of threads" on page 217, and DEDBs, described in "DEDB performance and tuning considerations" on page 220.

Response time: assigning job dispatching priorities

To minimize response times, assign a higher dispatching priority to the CICS address space than to the DBCTL address spaces (DBCTL, DLISAS, and DBRC).

Although CICS can be regarded as a "front end" to DBCTL, you must be aware that CICS must also manage the network and the application environment for non-DLI transactions such as DB2 or VSAM. This means that CICS has different CPU requirements from other front ends to DBCTL, such as a BMP or an MPP. For example, when a CICS transaction is waiting for a response to a DBCTL request, CICS dispatches other CICS transactions.

If IRLM is assigned a priority of *n*, CICS should have a priority of *n*-1, DBRC a priority of *n*-2, and DBCTL and DLISAS a priority of *n*-3.

For further guidance on assigning priorities, see System administration in IMS product documentation.

Specifying numbers of threads

The DRA startup parameters, MINTHRD and MAXTHRD, specify the minimum and maximum numbers of threads that can process DBCTL DL/I or DEDB requests. The MINTHRD and MAXTHRD parameters are specified in the DRA startup table (DFSPZP).

See Defining the IMS DRA startup parameter table for more information on DRA startup parameters.

The IMS system generation parameter, MAXREGN, specifies the number of regions (or threads), to be allocated at startup, that DBCTL can handle for all connected CICS systems and BMPs. The number can increase dynamically, to a limit of 999, as required.

The number you specify for MAXREGN should be no less than the sum of the MINTHRD parameters specified for active CICS systems, and for BMPs.

In Figure 22 on page 219, the following threads are in use: one from BMPA, one from BMPB, five from CICSA and three from CICSB, making a total of 10 threads. A MAXREGN of 10 has therefore been specified for DBCTLA.

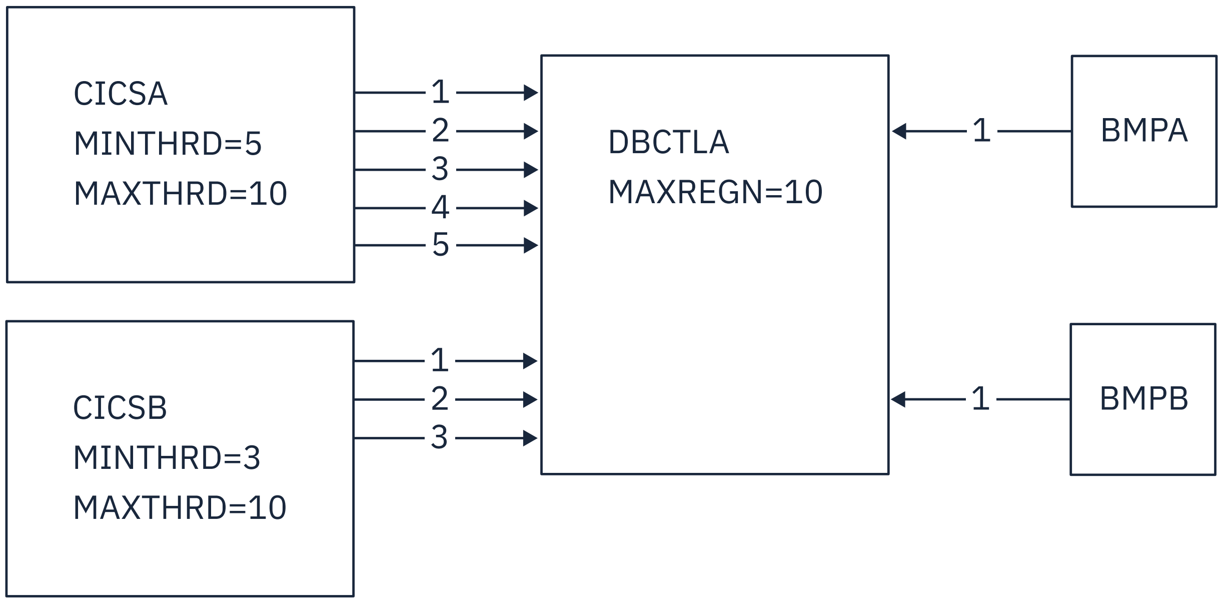


Figure 22. Interaction of MAXREGN, MINTHRD, and MAXTHRD

MAXTHRD can be used in DBCTL systems to ensure that, at peak loads, additional threads can be built in addition to those already allocated as a result of MINTHRD, thus avoiding waiting for threads. The maximum number of threads you can specify in DBCTL is 999. The default is 1 or the number defined by MINTHRD, whichever is the highest. MAXTHRD controls the maximum number of tasks for which this CICS system can have PSBs scheduled in DBCTL. Any requests to schedule a PSB when the MAXTHRD limit is reached is queued by the DRA. One thread is equivalent to one MVS TCB, thus giving more concurrency on multiprocessors. There is a storage allocation of about 9 KB per thread in the local system queue area (LSQA) below the 16 MB line. Because these threads are available for the duration of the DBCTL connection, there is no pathlength overhead for collapsing and reallocating thread related storage, and throughput should, therefore, be faster. The number of threads that you specify must be large enough for your system's needs, but if you specify a number that exceeds those needs, this will have an adverse effect on the performance of the DRA. If you specify a minimum thread value that is higher than your system's actual minimum activity, this will tie up threads unnecessarily, preventing DBCTL from allocating them to other CICS systems or BMPs. If you specify a minimum thread value that is too low, this can also affect performance; if the level of thread activity falls, this could cause the DRA to release threads down to the minimum value. These threads would then have to be reestablished if the thread requests increased again.

The number you specify for MAXTHRD should reflect what you consider to be the peak load of DBCTL threads needed. The number of threads you specify will affect performance. The larger the number you have preallocated, the more storage is needed. However, if threads are preallocated, the time needed to allocate them on demand is saved, thus improving response time and throughput. So, if your system is storage constrained, specify a lower value for MINTHRD, and use MAXTHRD as a "safety valve". If response time and throughput are more important than storage requirements, specify a higher number for MINTHRD so that more threads are ready to be used.

After the MINTHRD limit is exceeded, threads continue to be built up to the MAXTHRD limit but, because each thread's control blocks are allocated during PSB scheduling, the pathlength is greater for the tasks running after the MINTHRD limit has been reached.

Also bear DBCTL thread activity in mind when specifying the MXT system initialization parameter. You use MXT to specify the maximum number of tasks that CICS will allow to exist at any time. With DBCTL, MXT should be enough to allow for the number specified in MINTHRD, plus the number you need for "standard" CICS tasks. With DB2, there is no minimum number of threads. See Setting the maximum task specification (MXT) for general help on MXT.

To help you decide on the optimum values for minimum and maximum numbers of DBCTL threads, monitor thread usage and IMS task throughput (to see if tasks are being delayed), and IMS I/O rates. For details of thread statistics produced, including maximum and minimum thread usage, see DBCTL statistics. See DBCTL data returned to IMS log for details of data produced for monitoring IMS I/O rates. You can also use CICS auxiliary trace to check for queueing for threads and PSBs.

DEDB performance and tuning considerations

If you use DEDBs, you must define the characteristics and usage of the IMS DEDB buffer pool. You do this by specifying parameters both in the CICS region and the IMS (DBCTL) region. The DBCTL DEDB parameters are useful when tuning a CICS/DBCTL DEDB fastpath environment. DBBF and DBFX are parameters defined during DBCTL system generation or at DBCTL initialization. CNBA, FPBUF, and FPBOF are defined in the DRA startup table (DFSPZP).

All parameters that you need to during IMS system definition or execution, including DRA startup parameters, are described in Defining the IMS DRA startup parameter table) .

The main concerns in defining DEDB buffer pools are the total number of buffers in the IMS region, and how they are shared by CICS threads. You use the following IMS FPCTRL parameters to define the number of buffers:

- DBBF: total number of buffers
- DBFX: number of buffers used exclusively by the DEDB system.

The number remaining when you subtract the value specified for DBFX from the value specified for DBBF is the number of buffers available for the needs of CICS threads; for this example, It is a fixed number is assumed for DBFX. DBBF must, therefore, be large enough to accommodate all batch message processing programs (BMPs) and CICS systems that you want to connect to this DBCTL system.

When a CICS thread connects to IMS, its DEDB buffer requirements are specified using a normal buffer allocation (NBA) parameter. For a CICS system, there are two NBA parameters in the DRA startup table:

1. CNBA buffers needed for the CICS system. This is taken from the total specified in DBBF.
2. FPBUF buffers to be given to each CICS thread. This is taken from the number specified in CNBA. FPBUF is used for each thread that requests DEDB resources, and so should be large enough to handle the requirements of any application that can run in the CICS system.

A CICS system might fail to connect to DBCTL if its CNBA value is more than that available from DBBF. An application might receive schedule failure if the FPBUF value is more than that available from CNBA. The FPBUF value is used when an application tries to schedule a PSB that contains DEDBs.

When a CICS system has successfully connected to DBCTL, and the application has successfully scheduled a PSB containing DEDBs, the DRA startup parameter FPBOF becomes relevant. FPBOF specifies the number of overflow buffers each thread will get if it exceeds FPBUF. These buffers are not taken from CNBA. Instead, they are buffers that are *serially* shared by all CICS applications or other dependent regions that are currently exceeding their NBA allocation.

Because overflow buffer allocation (OBA) usage is serialized, thread performance can be affected by NBA and OBA specifications. If FPBUF is too small, more applications need to use OBA, which may cause delays due to contention. If both NBA and OBA are too small, the application fails. If FPBUF is too large, this affects the number of threads that can concurrently access DEDB resources, and increases the number of schedule failures.

In a CICS-DBCTL environment, the main performance concern is the trade-off between speed and concurrent access. The size of this trade-off is dictated by the kind of applications you are running in the CICS system. If the applications have approximately the same NBA requirements, there is no trade-off. You can specify a FPBUF large enough to never need OBA. This speeds access and there is no waste of buffers in CNBA, thus enabling a larger number of concurrent threads using DEDBs. The more the buffer requirements of your applications vary, the greater the trade-off. If you want to maintain speed of access (because OBAs are not being used) but decrease concurrent access, you should increase the value of FPBUF. If you prefer to maintain concurrent access, do not increase the value of FPBUF. However, speed of access will decrease because this and possibly other threads will need to use the OBA function.

For information on specifying the parameters CNBA, FPBOF, and FPBUF, see Defining the IMS DRA startup parameter table. For further guidance on DEDB buffer specification and tuning, see sections on DEDBs in Database administration in IMS product documentation and System administration in IMS product documentation.

Monitoring data at the transaction level is returned to CICS by DBCTL at schedule end and transaction termination. This data includes DEDB statistics. To obtain the monitoring data, two event monitoring points (EMPs) must be added to your CICS monitoring control table (MCT).

Exploiting Open Transaction Environment (OTE)

The CICS-DBCTL interface can be defined as threadsafe and CICS can run the CICS-DBCTL task-related user exit (TRUE) on an L8 open task control block (open TCB).

The open transaction environment (OTE) is supported in IMS version 12 with PTFs for APAR PM29194 applied and in IMS version 13 with PTFs for APAR PM29195 applied. Later releases of IMS require no PTFs. To activate IMS to use OTE, parameter OPENTHRD=CCTL needs to be specified in the DRA startup table (DFSPZP). If specified, then during connect processing CICS enables the CICS-DBCTL TRUE as an OPENAPI TRUE.

An open API TRUE is run on an L8 open TCB, which is dedicated for use by the calling CICS task. Running an application on an open TCB improves throughput and performance by reducing the use of the QR TCB. Threadsafe CICS applications that run on an L8 open TCB and use threadsafe CICS-DBCTL commands now avoid up to four TCB switches for each call to IMS. For more information about CICS IMS applications and the OTE, see Enabling CICS IMS applications to use the open transaction environment (OTE) through threadsafe programming.

If you do not specify OPENTHRD=CCTL, then CICS runs the CICS-DBCTL TRUE on the QR TCB and IMSDRA TCBs will be utilised.

When using IMS Version 12 or later with OPENTHRD=CCTL, you must change the way you calculate CICS and IMSprocessor times, for more information see, DBCTL monitoring data returned to CICS in Monitoring.

Using DEDBs

Using DEDBs can provide performance improvements in a number of areas, including a reduction in path length, parallel processing capability, less I/O processing, and a reduced logging overhead.

- Reduced path length
  - DEDBs use Media Manager for more efficient control interval (CI) processing, which can reduce pathlength.
  - DEDBs have their own resource manager, which means:
    - Less interaction with whichever lock manager you are using (PI or the IRLM), provided you are not using block level sharing.
    - Simplified buffer handling (and reduced pathlength) because DEDBs have their own buffer pool.
- Parallel processing

  DEDB writes are not done during the life of the transactions but are kept in buffers. Actual update operations are delayed until a synchronization point and are done by asynchronous processing using output threads in the control region. The output thread runs as a service request block (SRB): a separate dispatchable MVS task. You can specify up to 255 output threads. This means that:

  - The CICS task can be freed earlier
  - Parallel processing is increased and throughput on multiprocessors is improved.
- Less I/O

  The cost of I/O per SDEP segment inserted can be very low because SDEP segments are gathered in one buffer and are written out only when it is full. This means that many transactions can "share the cost" of SDEP CI writes to a DEDB. SDEPs should have larger CIs to reduce I/Os.
- Reduced logging overhead

  DEDB log buffers are written to OLDS only when they are full. This means less I/O than would be needed with full function databases.

High speed sequential processing (HSSP)

Using DBCTL enables you to use high speed sequential processing (HSSP). HSSP is useful with applications that do large scale sequential updates to DEDBs, which may require an image copy after the DEDBs are updated. Using HSSP provides the following major benefits:

- DEDB processing time can be improved by using the IBM 3990 Storage Control Model 3 Fast Write capability and the IBM 3990 Storage Control Model 3 Sequential Mode for both READs and WRITEs.
- You can take an HSSP image copy during a sequential update job. This avoids having to make a subsequent sequential pass through the DEDB areas to take an image copy.
- HSSP reduces elapsed DEDB processing time by using private buffer pools and optimizing locking.
- Only a minimum amount of log data is written to the IMS system log when you request an HSSP image copy. This reduces the large amount of logging that such large scale sequential runs usually involve.

For further guidance on HSSP, see Database administration in IMS product documentation.

IMS asynchronous database buffer purge facility

IMS includes the asynchronous database buffer purge facility.

At syncpoint time, when database buffers are to be flushed, buffers that are to be written to different devices are written concurrently, rather than serially, as in earlier releases of IMS. For further guidance, see System administration in IMS product documentation.

The asynchronous database buffer purge facility should improve response time for transactions that update databases on multiple devices in a single UOW.

Virtual storage usage

CICS regions that previously used local DL/I can obtain considerable virtual storage constraint relief because the following storage areas reside in the DBCTL address spaces: all DL/I and DBRC code and control blocks, OSAM and VSAM buffer pools and related control blocks, PSB, DMB, and ENQ pools.

However, DBCTL requires some MVS CSA storage, which can lower the maximum available region size in the MVS system. See System administration in IMS product documentation for details of CSA and other DBCTL storage requirements.

Improved throughput on multiprocessors

You can obtain throughput improvements on multiprocessors when using IMS Version 12 or later by using the CICS open transaction environment (OTE), providing that the application code is threadsafe.

You can obtain further performance improvements by using DEDBs instead of full-function databases.

Improving intersystem performance

There are a number of ways that you can improve aspects of CICS performance in a multi-system environment.

"Intersystem session queue management" on page 223 describes methods for controlling the length of intersystem queues.

"Efficient deletion of shipped terminal definitions" on page 225 describes how to delete redundant shipped terminal definitions from AORs and intermediate systems.

Intersystem session queue management

This section describes how to control the number of queued requests for sessions on intersystem links (allocate queues).

Note: This section describes how to control queues for sessions on established connections. The specialized subject of using local queuing for function-shipped **`EXEC CICS START NOCHECK`** requests is described in Local queuing of START commands.

Overview of session queue management

In a perfect intercommunication environment, queues would never occur because work flow would be evenly distributed over time, and there would be enough intersystem sessions available to handle the maximum number of requests arriving at any one time.

However, in the real world this is not the case, and, with peaks and troughs in the workload, queues do occur: queues come and go in response to the workload. The situation to avoid is an unacceptably high level of queuing that causes a bottleneck in the work flow between interconnected CICS regions, and which leads to performance problems for the terminal end-user as throughput slows down or stops. This abnormal and unexpected queuing should be prevented, or dealt with when it occurs: a "normal" or optimized level of queuing can be tolerated.

For example, function shipping requests between CICS application-owning regions and connected file-owning regions can be queued in the issuing region while waiting for free sessions. Provided a file-owning region deals with requests in a responsive manner, and outstanding requests are removed from the queue at an acceptable rate, then all is well. But if a file-owning region is unresponsive, the queue can become so long and occupy so much storage that the performance of connected application-owning regions is severely impaired. Further, the impaired performance of the application-owning region can spread to other regions. This condition is sometimes referred to as “sympathy sickness”, although it should more properly be described as intersystem queuing, which, if not controlled, can lead to performance degradation across more than one region.

Managing allocate queues

There are three methods for managing allocate queues.

Using resource definitions to manage your queues

You can specify the QUEUELIMIT and MAXQTIME options on the CONNECTION and IPCONN resource definitions for intersystem links that have simple control requirements; for example, links that carry noncritical traffic.

QUEUELIMIT defines the maximum number of allocate requests that CICS is to queue while waiting for free sessions on the connection.

MAXQTIME defines the approximate time for which allocate requests will queue for free sessions on a connection that is unresponsive. MAXQTIME is used only if a queue limit is specified on QUEUELIMIT, and if that limit is reached.

When an allocate request is received that causes the QUEUELIMIT value to be exceeded, CICS calculates whether the rate of processing of the queue will allow the new request to be processed in the maximum queuing time. If the request is not processed, CICS purges the queue. No further queuing takes place until the connection has freed a session. At this point, queuing begins again.

When CICS purges an allocate request because the QUEUELIMIT and MAXQTIME settings are exceeded, the SYSIDERR condition is returned to the application program.

For information about the QUEUELIMIT and MAXQTIME attributes, see CONNECTION attributes and IPCONN attributes.

Using the NOQUEUE option

A further method of controlling *explicit* allocate requests is to specify the NOQUEUE|NOSUSPEND option of the **`EXEC CICS ALLOCATE`** command.

However, while this enables you to control specific requests, it takes no account of the state of the queue at the time the requests are issued. And it is of no use in controlling *implicit* allocate requests (where the session request is instigated by, for example, a function shipping request). For programming information about API options, see ALLOCATE (APPC) .

Using the XISQUE and XZIQUE global user exits

You can control the queuing of allocate requests through a global user exit program, which provides more flexibility than setting a queue limit on the connection. Use XISQUE to manage IPIC queues and XZIQUE to manage MRO and APPC queues.

With the XISQUE and XZIQUE exits, you can quickly detect queuing problems (bottlenecks). Both exits enable allocate requests to be queued or rejected, depending on the length of the queue. You can use XISQUE and XZIQUE to stop and then reestablish a connection that has a bottleneck.

The XZIQUE exit extends the function that the XISCONA exit provides for MRO and APPC connections. XISCONA is called for function shipping and DPL requests only, including function shipped EXEC CICS START requests used for asynchronous processing. XZIQUE is called for transaction routing, asynchronous processing, and distributed transaction processing requests, in addition to function shipping and DPL. Compared with the XISCONA exit, XZIQUE receives more detailed information on which to base its action. For information about the relationship between XISCONA and XZIQUE, see Interaction with the XISCONA exit.

Uses of a queuing global user exit program

When the exit is enabled, your XZIQUE or XISQUE global user exit program can check on the state of the allocate queue for a particular connection in the local system.

Information is passed to the exit program in a parameter list that is structured to provide data about nonspecific allocate requests or requests for specific modegroups, depending on the session request. If you are using the XZIQUE exit, nonspecific allocate requests are for MRO, LU6.1, and APPC sessions that do not specify a modegroup.

Using the information passed in the parameter list, your global user exit program selects the system action to take:

- Queue the allocate request. This action is possible only if the queue limit has not been reached.
- Reject the allocate request.
- Reject this allocate request and purge all queued requests for the connection.
- Reject this allocate request and purge all queued requests for the modegroup.

Your exit program might base its action on one of the following criteria: Exit XISQUE

- The length of the allocate queue.
- Whether the number of queued requests has reached the limit set by the QUEUELIMIT option. If the queue limit has not been reached, you might decide to queue the request.
- The rate at which sessions are being allocated on the connection. If the queue limit has been reached but session allocation is acceptably quick, you might decide to reject only the current request. If the queue limit has been reached and session allocation is unacceptably slow, you might decide to purge the whole queue.

For details of the information passed in the XISQUE parameter list, and advice about designing and coding an XISQUE exit program, see the programming information in XISQUE exit for managing IPIC intersystem queues.

For details of the information passed in the XZIQUE parameter list, and advice about designing and coding an XZIQUE exit program, see the programming information in XZIQUE exit for managing MRO and APPC intersystem queues.

Efficient deletion of shipped terminal definitions

This chapter describes how CICS deletes redundant shipped terminal definitions.

It contains the following topics:

- “Overview of how shipped terminals are deleted” on page 225
- “Implementing timeout delete” on page 226
- “Tuning the performance of timeout delete” on page 227

Overview of how shipped terminals are deleted

In a transaction routing environment, terminal definitions can be shipped from a terminal-owning region (TOR) to an application-owning region (AOR) when they are first needed, rather than being statically defined in the AOR.

The "terminal" could be an APPC device or system. In this case, the shipped definition would be of an APPC connection.

Shipped definitions can become redundant in the following situations:

- A terminal user logs off.
- A terminal user stops using remote transactions.
- The TOR is shut down.

- The TOR is restarted, autoinstalled terminal definitions are not recovered, and the autoinstall user program, DFHZATDX, assigns a new set of termids to the same set of terminals.

At some stage, redundant definitions must be deleted from the AOR (and from any intermediate systems between the TOR and AOR). For brevity, we shall refer to AORs and intermediate systems collectively as *back-end systems*. This is particularly necessary in the last situation in the previous list, to prevent a possible mismatch between termids in the TOR and the back-end systems.

CICS method of deleting redundant shipped definitions consists of two parts:

- Selective deletion
- A timeout delete mechanism.

Selective deletion

Each time a terminal definition is installed, CICS creates a unique “instance token” and stores it within the definition.

Thus, if the definition is shipped to another region, the value of the token is shipped too. All transaction routing attach requests pass the token within the function management header (FMH). If, during attach processing, an existing shipped definition is found in the remote region, it is used *only if the token in the shipped definition matches that passed by the TOR*. Otherwise, it is deleted and an up-to-date definition shipped.

The timeout delete mechanism

You can use the timeout delete mechanism in your back-end systems, to delete shipped definitions that have not been used for transaction routing for a defined period Its purpose is to ensure that shipped definitions remain installed only while they are in use.

Note: Shipped definitions are not deleted if there is an automatic initiate descriptor (AID) associated with the terminal.

Timeout delete gives you flexible control over shipped definitions. CICS allows you to:

- Stipulate the minimum time a shipped definition must remain installed before being eligible for deletion
- Stipulate the time interval between invocations of the mechanism
- Reset these times online
- Cause the timeout delete mechanism to be invoked immediately.

The parameters that control the mechanism allow you to arrange for a “tidy-up” operation to take place when the system is least busy.

Implementing timeout delete

To use timeout delete in a CICS Transaction Server for z/OS system to which terminals are shipped, you specify two system initialization parameters.

DSHIPIDL={020000|hhmmss}

Specifies the minimum time, in hours, minutes, and seconds, that an *inactive* shipped terminal definition must remain installed in this region. When the CICS timeout delete mechanism is invoked, only those shipped definitions that have been inactive for longer than the specified time are deleted.

You can use this parameter in a transaction routing environment, on the application-owning and intermediate regions, to prevent terminal definitions having to be reshipped because they have been deleted prematurely.

hhmmss

Specify a 1 to 6 digit number in the range 0-995959. Numbers that have fewer than six digits are padded with leading zeros.

DSHIPINT={120000|0|hhmmss}

Specifies the interval between invocations of the CICS timeout delete mechanism. The timeout delete mechanism removes any shipped terminal definitions that have not been used for longer than the time specified by the DSHIPIDL parameter.

You can use this parameter in a transaction routing environment, on the application-owning and intermediate regions, to control:

- How often the timeout delete mechanism is invoked.
- The approximate time of day at which a mass delete operation is to take place, relative to CICS startup.

0

The timeout delete mechanism is not invoked. You might set this value in a terminal-owning region, or if you are not using shipped definitions.

hhmmss

Specify a 1 to 6 digit number in the range 1-995959. Numbers that have fewer than six digits are padded with leading zeros.

For details of how to specify system initialization parameters, see CICS system initialization.

After CICS startup you can examine the current settings of DSHIPIDL and DSHIPINT. For flexible control over when mass delete operations take place, you can reset the interval until the next invocation of the timeout delete mechanism. (The revised interval starts *from the time the command is issued*, **not** from the time the remote delete mechanism was last invoked, nor from CICS startup.) Alternatively, you can invoke the timeout delete mechanism.

Tuning the performance of timeout delete

A careful choice of DSHIPINT and DSHIPIDL settings results in a minimal number of mass deletions of shipped definitions, and a scheduling of those that do take place for times when your system is lightly loaded.

Conversely, a poor choice of settings could result in unnecessary mass delete operations. Here are some suggestions for coding DSHIPINT and DSHIPIDL:

DSHIPIDL

In setting this value, you must consider the length of the work periods during which remote users access resources on this system. Do they access the system intermittently, all day? Or is their work concentrated into intensive, shorter periods?

By setting too low a value, you could cause definitions to be deleted and reshipped unnecessarily. It is also possible that you could cause automatic transaction initiation (ATI) requests to fail with the “terminal not known” condition. This condition occurs when an ATI request names a terminal that is not defined to this system. Usually, the terminal is not defined because it is owned by a remote system, you are using shippable terminals, and no prior transaction routing has taken place from it. By allowing temporarily inactive shipped definitions too short a life, you could increase the number of calls to the XALTENF and XICTENF global user exits that deal with the “terminal not known” condition.

DSHIPINT

You can use this value to control the time of day at which your mass delete operations take place.

For example, if you usually warm-start CICS at 7 a.m., you could set DSHIPINT to 150000, so that the timeout delete mechanism is invoked at 10 p.m., when few users are accessing the system.

Attention: If CICS is recycled, perhaps because of a failure, the timeout delete interval is reset. Continuing the previous example, if CICS is recycled at 8:00 p.m., the timeout delete mechanism will be invoked at 11:00 a.m. the following day (15 hours from the time of CICS initialization). In these circumstances, you could use the SET DELETSHIPPED and PERFORM DELETSHIPPED commands to accurately control when a timeout delete takes place.

CICS provides statistics to help you tune the DFHIPIDL and DFHIPINT parameters. The statistics are available online, and are mapped by the DFHA04DS DSECT. For details of the statistics provided, see Autoinstall statistics .

Improving FEPI performance

You cannot tune FEPI itself because it is already optimized for speed of response. However, you can manage the performance of FEPI applications.

FEPI runs under a separate CICS task control block (TCB) and CICS permits only one application program to issue a FEPI command at a time. This is a major influence on FEPI performance. Although many application programs can have FEPI commands being processed at any time, only one application can issue a FEPI command.

In a lightly loaded system, CICS does not run FEPI until a command is issued. Therefore, performance is impacted by the overhead of starting up the TCB so that the FEPI command can be processed. In a heavily loaded system, this overhead is not present, because the TCB is already active processing earlier FEPI commands.

FEPI tries to minimize this overhead by issuing timer requests that ensure that the TCB is not inactive for more than one second.

There are three main principles that can be used in FEPI applications to provide the best performance:

- Minimize the number of commands. Each FEPI command generates a CICS WAIT, even if no network transmission is involved.
- Keep data transmission to a minimum.
- Avoid disconnecting sessions.

Application development techniques are described in FEPI application techniques for performance.

For FEPI system programming, you can reduce the number of commands by using lists of resources on a command where possible. However, if you use a list that results in a z/OS Communications Server operation, the following conditions might occur:

- Flood z/OS Communications Server by requesting too many operations at once
- Flood the back-end system with requests for session initiation
- Flood the front-end system with started begin- or end-session transactions.

So you must carefully evaluate the benefits of using lists.

Using CICS monitoring

CICS monitoring data can help with performance tuning and resource planning for applications that use FEPI.

By default, CICS performance class monitoring records include the following data about the user task:

- The number and type of requests made to FEPI
- The time spent waiting for requests to FEPI to complete
- The number of requests to FEPI that are timed out.

For detailed information about the FEPI-related fields in performance class monitoring records, see Performance class data: listing of data fields.

Using statistics data

CICS statistics can help with performance tuning and resource planning for applications that use FEPI.

The standard CICS statistics reports contain data about usage of:

- FEPI pools

- FEPI connections
- FEPI targets.

To obtain the current statistics for a FEPI pool, connection, or target, a utility program can issue an **`EXEC CICS COLLECT STATISTICS`** command. For example, the command `EXEC CICS COLLECT STATISTICS SET(pointer) POOL(GRPD)` returns the current statistics for the 'GRPD' pool. To map the returned statistics, your utility program should include the appropriate CICS-supplied copybook:

DFHA22DS
FEPI pool statistics

DFHA23DS
FEPI connection statistics

DFHA24DS
FEPI target statistics.

The copybooks are supplied in COBOL, PL/I, and assembler language.

To cause all FEPI statistics to be written immediately to the SMF statistics data set, you can use either the EXEC CICS or the CEMT version of the **`PERFORM STATISTICS RECORD FEPI`** command. For details of the **`CEMT COLLECT STATISTICS`** and **`PERFORM STATISTICS RECORD`** commands, see CEMT - main terminal; for programming information about the equivalent **`EXEC CICS`** commands, see the Introduction to System programming commands.

To format and print FEPI-related statistics in the DFHSTATS data set, you can use the CICS-supplied utility program, DFHSTUP. To print only the FEPI statistics, specify the command parameter SELECT TYPE=FEPI. For information about how to use the DFHSTUP program, see Statistics utility program (DFHSTUP). For detailed information about fields in the FEPI statistics records, see FEPI statistics.

Improving Java performance

You can take various actions to improve the performance of Java applications and the JVMs in which they run.

About this task

In addition to fine-tuning CICS itself, you can further improve the performance of Java applications in the following ways:

- Ensuring that the Java applications are well written
- Tuning the Java Runtime Environment (JVM)
- Tuning the language in which the JVM runs

Procedure

1. Determine the performance goals for your Java workload.

   Some of the most common goals include minimizing processor usage or application response times. After you decide on the goal, you can tune the Java environment.
2. Analyze your Java applications to ensure that they are running efficiently and do not generate too much garbage.

   IBM has tools that can help you to analyze Java applications to improve the efficiency and performance of particular methods and the application as a whole.
3. Tune the JVM server.

   You can use statistics and IBM tools to analyze the storage settings, garbage collection, task waits, and other information to tune the performance of the JVM.
4. Tune the Language Environment enclave in which a JVM runs.

   JVMs use MVS storage, obtained by calls to MVS Language Environment services. You can modify the runtime options for Language Environment to tune the storage that is allocated by MVS.

5. Optional: If you use the z/OS shared library region to share DLLs between JVMs in different CICS regions, you can tune the storage settings.

Determining performance goals for your Java workload

Tuning CICS JVMs to achieve the best overall performance for a given application workload involves several different factors. You must decide what the preferred performance characteristics of your Java workload are. When you establish these characteristics, you can determine what parameters to change and how to change them.

The following performance goals for Java workloads are most common:

Minimum overall processor usage
: This goal prioritizes the most efficient use of the available processor resource. If a workload is tuned to achieve this goal, the total use of the processor across the entire workload is minimized, but individual tasks might experience high processor consumption. Tuning for the minimum overall processor usage involves specifying large storage heap sizes for your JVMs to minimize the number of garbage collections.

Minimum application response times
: This goal prioritizes ensuring that an application task returns to the caller as rapidly as possible. This goal might be especially relevant if there are Service Level Agreements to be achieved. If a workload is tuned to achieve this goal, applications respond consistently and quickly, though a higher processor usage might occur for garbage collections. Tuning for minimum application response times involves keeping the heap size small and possibly using the gencon garbage collection policy.

Minimum JVM storage heap size
: This goal prioritizes reducing the amount of storage used by JVMs. You can reduce the amount of storage that is used in the JVM, by reducing the JVM heap size.

  Note: Reducing the JVM heap size might result in more frequent garbage collection events.

Other factors can affect the response times of your applications. The most significant of these is the Just In Time (JIT) compiler. The JIT compiler optimizes your application code dynamically at run time and provides many benefits, but it requires a certain amount of processor resource to do this.

Analyzing Java applications using IBM Health Center

To improve the performance of a Java application, you can use IBM Health Center to analyze the application. This tool provides recommendations to help you improve the performance and efficiency of your application.

About this task

IBM Health Center is available in the IBM Support Assistant Workbench. These free tools are available to download from IBM as described in the Getting Started guide for IBM Health Center. Try to run the application in a JVM on its own. If you are running a mixed workload in a JVM server, it might be more difficult to analyze a particular application.

Procedure

1. Add the required connection options to the JVM profile of the JVM server.

   The IBM Health Center documentation describes what options you must add to connect to the JVM from the tool.
2. Start IBM Health Center and connect it to your running JVM.

   IBM Health Center reports JVM activity in real time so wait a few moments for it to monitor the JVM.
3. Select the **Profiling** link to profile the application.

   You can check the time spent in different methods. Check the methods with the highest usage to look for any potential problems.

Tip: The **Analysis and Recommendations** tab can identify particular methods that might be good candidates for optimization.

4. Select the **Locking** link to check for locking contentions in the application.

   If the Java workload is unable to use all the available processor, locking might be the cause. Locking in the application can reduce the amount of parallel threads that can run.

5. Select the **Garbage Collection** link to check the heap usage and garbage collection.

   The **Garbage Collection** tab can tell you how much heap is being used and how often the JVM pauses to perform garbage collection.

   a) Check the proportion of time spent in garbage collection.

      This information is presented in the Summary section. If the time spent in garbage collection is more than 2%, you might need to adjust your garbage collection.

   b) Check the pause time for garbage collection.

      If the pause time is more than 10 milliseconds, the garbage collection might be having an effect on application response times.

   c) Divide the rate of garbage collection by the number of transactions to find out approximately how much garbage is produced by each transaction.

      If the amount of garbage seems high for the application, you might have to investigate the application further.

What to do next

After you have analyzed the application, you can tune the Java environment for your Java workloads.

Garbage collection and heap expansion

Garbage collection and heap expansion are an essential part of the operation of a JVM. The frequency of garbage collection in a JVM is affected by the amount of garbage, or objects, created by the applications that run in the JVM.

Allocation failures

When a JVM runs out of space in the storage heap and is unable to allocate any more objects (an allocation failure), a garbage collection is triggered. The Garbage Collector cleans up objects in the storage heap that are no longer being referenced by applications and frees some of the space. Garbage collection stops all other processes from running in the JVM for the duration of the garbage collection cycle, so time spent on garbage collection is time that is not being used to run applications. For a detailed explanation of the JVM garbage collection process, see Generational Concurrent Garbage Collector and z/OS User Guide for IBM SDK, Java Technology Edition, Version 8.

When a garbage collection is triggered by an allocation failure, but the garbage collection does not free enough space, the Garbage Collector expands the storage heap. During heap expansion, the Garbage Collector takes storage from the maximum amount of storage reserved for the heap (the amount specified by the `-Xmx` option), and adds it to the active part of the heap (which began as the size specified by the `-Xms` option). Heap expansion does not increase the amount of storage required for the JVM, because the maximum amount of storage specified by the `-Xmx` option has already been allocated to the JVM at startup. If the value of the `-Xms` option provides sufficient storage in the active part of the heap for your applications, the Garbage Collector does not have to carry out heap expansion at all.

At some point during the lifetime of the JVM, the Garbage Collector stops expanding the storage heap, because the heap has reached a state where the Garbage Collector is satisfied with the frequency of garbage collection and the amount of space freed by the process. The Garbage Collector does not aim to eliminate allocation failures, so some garbage collection can still be triggered by allocation failures after the Garbage Collector has stopped expanding the storage heap. Depending on your performance goals, you might consider this frequency of garbage collection to be excessive.

Garbage collection options

You can use different policies for garbage collection that make trade-offs between throughput of the application and the overall system, and the pause times that are caused by garbage collection. Garbage collection is controlled by the `-Xgcpolicy` option:

`-Xgcpolicy:optthruput`
This policy delivers high throughput to applications but at the cost of occasional pauses, when garbage collection occurs.

`-Xgcpolicy:gencon`
This policy helps to minimize the time that is spent in any garbage collection pause. Use this garbage collection policy with JVM servers. You can check which policy is being used by the JVM server by inquiring on the JVMSERVER resource. The JVM server statistics have fields that tell you how many major and minor garbage collection events occur and what processor time is spent on garbage collection.

When you use this policy, it is also worth considering the `-Xgc:concurrentScavenge` setting - which is not a default setting - if your system has a large heap and is response-time sensitive. In these situations it can help to reduce garbage collection pause times. For more information, see -Xgc:concurrentScavenge.

`-XX:+HeapManagementMXBeanCompatibility`
This policy is set by default if you have APAR PI87181 applied and are using Java 8 SR5 and above. The policy ensures consistent garbage collection statistics with previous levels of Java. For more information, see -XX:[+|-]HeapManagementMXBeanCompatibility.

`-XX:-HeapManagementMXBeanCompatibility`
You can choose to opt in to using this policy. The policy enables the default heap changes in Java 8 SR5 and above; however in some cases, the garbage collection statistics might indicate the heap usage to be greater than the maximum heap size. For more information, see -XX:[+|-]HeapManagementMXBeanCompatibility.

You can change the garbage collection policy by updating the JVM profile. For details of all the garbage collection options, see .

Improving JVM server performance

To improve the performance of applications that run in a JVM server, you can tune different parts of the environment, including the garbage collection and the size of the heap.

About this task

CICS provides statistics reports on the JVM server, which include details of how long tasks wait for threads, heap sizes, frequency of garbage collection, and processor usage. You can also use additional IBM tools that monitor and analyze the JVM directly to tune JVM servers and help with problem diagnosis. You can use the statistics to check that the JVM is performing efficiently, particularly that the heap sizes are appropriate and garbage collection is optimized.

Procedure

1. Check the amount of processor time that is used by the JVM server.

   Dispatcher statistics can tell you how much processor time the T8 TCBs are using. JVM server statistics tell you how long the JVM is spending in garbage collection and how many garbage collections occurred. Application response times and processor usage can be adversely affected by the JVM garbage collection.
2. Ensure that there is enough available storage capacity in the CICS address space. The CICS address space contains the Language Environment heap size that is required by the JVM server.
3. Tune the garbage collection and heap in the JVM.

A small heap can lead to very frequent garbage collections, but too large a heap can lead to inefficient use of MVS storage. You can use IBM Health Center to visualize and tune garbage collection and adjust the heap accordingly.

What to do next

For more detailed analysis of memory usage and heap sizes, you can use the Memory Analyzer tool in IBM Support Assistant to analyze Java heap memory using system dump or heap dump snapshots of a Java process.

To start one or more JVM servers in a CICS region, you must ensure that enough storage capacity is available for the JVM to use, excluding any storage capacity that is allocated to CICS.

Examining processor usage by JVM servers

You can use the CICS monitoring facility to monitor the processor time that is used by transactions running in a JVM server. All threads in a JVM server run on T8 TCBs.

About this task

You can use the DFH$MOLS utility to print the SMF records or use a tool such as CICS Performance Analyzer to analyze the SMF records.

Procedure

1. Turn on monitoring in the CICS region to collect the performance class of monitoring data.
2. Check the performance data group DFHTASK.

   In particular, you can look at the following fields:

Field ID	Field name	Description
283	MAXTTDLY	The elapsed time for which the user task waited to obtain a T8 TCB, because the CICS region reached the limit of available threads. The thread limit is 2000 for each CICS region and each JVM server can have up to 256 threads.
400	T8CPUT	The processor time during which the user task was dispatched by the CICS dispatcher domain on a CICS T8 mode TCB. When a thread is allocated a T8 TCB, that same TCB remains associated with the thread until the processing completes.
401	JVMTHDWT	The elapsed time that the user task waited to obtain a JVM server thread because the CICS system had reached the thread limit for a JVM server in the CICS region. This does not apply to Liberty JVM servers.

3. To improve processor usage, reduce or eliminate the use of tracing where possible.
   a) In a production environment, consider running your CICS region with the CICS main system trace flag set off.

      Having this flag on significantly increases the processor cost of running a Java program. You can set the flag off by initializing CICS with SYSTR=OFF, or by using the CETR transaction.
   b) Ensure that you activate JVM trace only for special transactions.

      JVM tracing can produce large amounts of output in a very short time, and increases the processor cost. For more information about controlling JVM tracing, see Diagnostics for Java.
4. Do not use the `USEROUTPUTCLASS` option in JVM profiles in a production environment.

   Specifying this option has a negative effect on the performance of JVMs. The `USEROUTPUTCLASS` option enables developers using the same CICS region to separate JVM output, and direct it to a suitable destination, but it involves the building and invocation of additional class instances.

Calculating storage requirements for JVM servers

To start one or more JVM servers in a CICS region, you must ensure that there is enough free storage available for each JVM to use. CICS and other products that are running in the same region might require a considerable amount of z/OS storage. CICS allocations such as DSALIM and EDSALIM affect storage availability and might be over-allocated compared to the peak requirements.

About this task

The storage that is required for a JVM server does not come from CICS DSA storage. Some is managed by the Language Environment handling requests such as **`malloc()`** issued by C code, and some are managed directly by the JVM that uses z/OS storage management requests such as **`IARV64`**. The Language Environment uses z/OS storage services. The release of the JVM in use has a bearing on whether Language Environment or z/OS manage certain types of storage. For example, the storage for the Java Heap might be managed by Language Environment in one JVM release but managed by z/OS in another.

There are a few user configuration options that directly specify the amount of storage for a JVM. For example, the Java command line option **`-Xmx3G`** requests 3 GB of 64-bit storage for the Java Heap.

The amount of storage that is used outside of the Java Heap is a function of what the JVM server is asked to handle in terms of the workload and Java classes that are run. This might be a considerable amount. A JVM server might be configured to handle anything from a relatively simple to a complex workload. An example of a complex workload is when a CICS Liberty JVM server is started. Even a simple workload might require many internal JVM processes that can require a considerable amount of storage. The dynamic nature of this type of storage makes an accurate estimate impossible.

JVM server dynamic storage requirements

The first JVM that starts causes the amount of storage to be defined for the **`USS SHRLIBRGNSIZE`** parameter to be allocated within 31-bit ELSQA storage. This reduces the amount of available 31-bit user region storage. The **`D OMVS,L`** command output shows the value of **`SHRLIBRGNSIZE`**.

4K of 24-bit storage is required for each JVM thread, and a single JVM server might legitimately start many more than 100 threads even before you consider the number of CICS managed JVM server threads defined by THREADLIMIT. In addition, UNIX System Services require 256 K of contiguous 24-bit storage during the process of creating a new thread.

A single JVM server might use several 100 MB of 31-bit user region storage, for example, the JIT compiler alone might use up to 128 MB in z/OS subpool 1 or 2. On top of the JIT usage, you must add other dynamic requirements of the JVM and all products in the region that interact with the JVM. The Java command line option **`-Xcompressedrefs`** might be responsible for much of the 31-bit storage usage. From Java 7.1, **`-Xcompressedrefs`** is the default, so **`-Xnocompressedrefs`** might need to be specified to reduce the 31-bit storage usage to avoid 878-10 abends.

The amount of 64-bit z/OS storage that can be used by a JVM server in addition to the Java Heap might range between 100 MB - 1 GB or higher. In addition to this, up to 20 MB per thread is allocated. The amount is calculated as the DFHAXRO STACK64 maximum stack size + 4 MB. For example, if STACK64(1M,1M,16M) is used, the value is 16 MB + 4 MB, hence 20 MB is allocated. All of this counts towards the CICS view of **`MEMLIMIT`** when GDSA expansion is required. But for the z/OS MEMLIMIT check, a value between 3 MB and the allocated size is counted depending on the peak requirement for the Usable area of the stack.

The biggest challenge to running a JVM server is the amount of 31-bit, and possibly 24-bit, storage that is required as that is always limited to some extent. 64-bit usage is limited by **`MEMLIMIT`**, and can be set to an artificially high value initially and reduced if required.

It might be possible to significantly reduce the amount of storage that is used by tuning Language Environment and Java options.

It is possible to see Language Environment owned 31-bit z/OS storage subpool 1 usage growing over time, and while it might look like a Storage Leak, that is not necessarily the case. In most instances, both the growth and the total amount of subpool 1 storage can be corrected by tuning the Language

Environment **HEAP64** runtime option 31-bit parameters, or by setting **-Xnocompressedrefs**, or both. Similarly, growth over time in 64-bit storage might have the same underlying cause and can be corrected by tuning the Language Environment **HEAP64** 64-bit parameters.

The following procedure aims to provide a simple estimate for the initial sizing, and room for expansion of both 31-bit and 64-bit storage must be possible to mitigate against an estimate that is too small. DFH0STAT reports show the actual storage usage when a JVM server is active.

Note: In terms of the number of threads, assume a background value of 100 when a CICS Liberty JVM server is used, or 50 otherwise, and to that add the THREADLIMIT of the JVM server.

Procedure

1. Run the sample statistics program DFH0STAT before starting a JVM server.
   a) The 24-bit requirement is 256 K plus the number of threads * 4 K.
   b) Start with the 31-bit requirement for **SHRLIBRGNSIZE** shown by the output of `D OMVS,L`.
   c) The remainder of the requirements cannot be accurately estimated. You can add a best guess of 50 - 300 MB per JVM server to the size of the 31-bit requirement depending on whether it is expected to run a simple or complex workload.

      The sums can be compared to the `Private Area storage available below 16 Mb` and `Private Area storage available above 16 Mb` from DFH0STAT to ensure that the allocations fit.

   - If you want to optimize the 31-bit storage settings to match the allocated storage more closely to the actual usage, you can adjust the **SHRLIBRGNSIZE** parameter and the Language Environment options. See "Tuning the z/OS shared library region" on page 242 and "Modifying the enclave of a JVM server with DFHAXRO" on page 241.
2. Calculate how much 64-bit storage is needed for each additional JVM server by adding up the following storage requirements:
   - The **-Xmx** value.
   - The value from **-Xcmsx**, if used.
   - The estimated total number of threads * 20 MB. (This is to account for the way that CICS GDSA expansion views **MEMLIMIT**.)
   - The remainder of the 64-bit requirements cannot be estimated. You can add a best guess of 256 MB - 1 GB.
   - Round up to the next GB multiple.
3. Check the **MEMLIMIT** value to determine whether you have enough 64-bit storage available to run the JVM server. The z/OS **MEMLIMIT** parameter limits the amount of 64-bit storage for the CICS region.

   If you want to optimize the 64-bit storage usage, see "Modifying the enclave of a JVM server with DFHAXRO" on page 241.

Tuning JVM server heap and garbage collection

Garbage collection in a JVM server is handled by the JVM automatically. You can tune the garbage collection process and heap size to ensure that application response times and processor usage are optimal.

About this task

The garbage collection process affects application response times and processor usage. Garbage collection temporarily stops all work in the JVM and can therefore affect application response times. If you set a small heap size, you can save on memory, but it can lead to more frequent garbage collections and more processor time spent in garbage collection. If you set a heap size that is too large, the JVM makes inefficient use of MVS storage and this can potentially lead to data cache misses and even paging. CICS provides statistics that you can use to analyze the JVM server. You can also use IBM Health Center, which provides the advantage of analyzing the data for you and recommending tuning options.

Procedure

1. Collect JVM server and dispatcher statistics over an appropriate interval. The JVM server statistics can tell you how many major and minor garbage collections take place and the amount of time that elapsed performing garbage collection. The dispatcher statistics can tell you about processor usage for T8 TCBs across the CICS region.
2. Use the dispatcher TCB mode statistics for T8 TCBs to find out how much processor time is spent on JVM server threads.

   The "Accum CPU Time / TCB" field shows the accumulated processor time taken for all the TCBs that are, or have been, attached in this TCB mode. The "TCB attaches" field shows the number of T8 TCBs that have been used in the statistics interval. Use these numbers to work out approximately how much processor time each T8 TCB has used.
3. Use the JVM server statistics to find the percentage of time that is spent in garbage collection.

   Divide the time of the statistics interval by how much elapsed time is spent in garbage collection. Aim for less than 2% of processor usage in garbage collection. If the percentage is higher, you can increase the size of the heap so that garbage collection occurs less frequently.
4. Divide the heap freed value by the number of transactions that have run in the interval to find out how much garbage per transaction is being collected.

   You can find out how many transactions have run by looking at the dispatcher statistics for T8 TCBs. Each thread in a JVM server uses a T8 TCB.
5. Optional: Write the verbosegc log data to a file, which can be done with the parameter **`-Xverbosegclog:`***path_to_file*. This data can be analyzed by another ISA tool - Garbage Collection and Memory Visualizer.

   The JVM writes garbage collection messages in XML to the file that is specified in the STDERR option in the JVM profile. For examples and explanations of the messages, see IBM SDK for z/OS, Java Technology Edition, Version 7, Troubleshooting and support section.

   Tip: You can use the file in the Memory Analyzer tool to perform more detailed analysis.

Results

The outcome of your tuning can vary depending on your Java workload, the maintenance level of CICS and of the IBM SDK for z/OS, and other factors. For more detailed information about the storage and garbage collection settings and the tuning possibilities for JVMs, see IBM SDK for z/OS, Java Technology Edition, Version 7, Troubleshooting and support section.

IBM Health Center and Memory Analyzer are two IBM monitoring and diagnostic tools for Java that are supplied by the IBM Support Assistant workbench. You can download these tools free of charge from the site.

Tuning the JVM server startup environment

If you are running multiple JVM servers, you can improve performance by tuning the JVM startup environment.

About this task

When a JVM server starts, the server has to load a set of libraries in the `/usr/lpp/cicsts/cicsts54/lib` directory. If you start a large number of JVM servers at the same time, the time taken to load the required libraries might cause some JVM servers to time out, or some JVM servers might take an excessively long time to start. To reduce JVM server startup time, you should tune the JVM startup environment.

Procedure

1. Create a shared class cache for the JVM servers to load the libraries a single time.

To use a shared class cache, add the **-Xshareclasses** option to the JVM profile of each JVM server. For more information see Class data sharing between JVMs in IBM SDK, Java Technology Edition 7.0.0.

2. Increase the timeout value for the OSGi framework.

   The DFHOSGI.jvmprofile contains the `OSGI_FRAMEWORK_TIMEOUT` option that specifies how long CICS waits for the JVM server to start and shut down. If the value is exceeded, the JVM server fails to initialize or shut down correctly. The default value is 60 seconds, so you should increase this value for your own environment.

Language Environment enclave storage for JVMs

A JVM server has both static and dynamic storage requirements, primarily in 64-bit storage. It may use a significant amount of 31-bit storage.

Note: The amount of 31-bit storage used will depend on several factors:

- The configuration parameters
- The design and use of other products
- The design of the JVM
- The Java workload.

For example, the use of **-Xcompressedrefs** might improve performance, but requires 31-bit storage and should always be used with **-XXnosuballoc32bitmem** to ensure that the JVM dynamically allocates 31-bit storage for compressed references based on demand. For more information about of these options, see Default settings for the JVM in IBM SDK, Java Technology Edition 7.0.0. *Just-in-time compilation* (JIT) also requires 31-bit storage for the compiled class code.

A JVM runs as a z/OS UNIX System Services process in a Language Environment enclave that is created using the Language Environment preinitialization module, **CELQPIPI**.

JVM storage requests are handled by Language Environment, which in turn allocates z/OS storage based on the defined runtime options.

The Language Environment runtime options are set by DFHAXRO. The default values provided by these programs for a JVM enclave are shown in Table 26 on page 237:

Table 26. Language Environment runtime options used by CICS for the JVM enclave

Language Environment runtime options	**Example JVM server values**
Heap storage	`HEAP64(256M,4M,KEEP,4M,1M,FREE,1K,1K,KEEP)`
Library heap storage	`LIBHEAP64(5M,3M)`
Library routine stack frames that can reside anywhere in storage	`STACK64(1M,1M,16M)`
Optional heap storage management for multithreaded applications (64 bit)	`HEAPPOOLS64(ALIGN)`
Optional heap storage management for multithreaded applications (31 bit)	`HEAPPOOLS(ALIGN)`
Amount of storage reserved for the out-of-storage condition and the initial content of storage when allocated and freed	`STORAGE(NONE,NONE,NONE)`

Note: For current JVM server values, refer to the DFHAXRO member in Library SDFHSAMP.

Language Environment runtime options, such as HEAP64, work on the principle of an initial value for that type of storage: for example, 256 MB 64-bit. When HEAP64 cannot contain a new request, an increment is allocated of the specified size (4 MB above) or of the request size plus control information, whichever is larger. Extra increments are allocated as required to meet demand. When an increment is empty, Language Environment will either KEEP or FREE the z/OS storage based on the runtime value.

For full information about Language Environment runtime options, see z/OS Language Environment Customization.

Where possible, the 31-bit and 64-bit initial size should cover the total 31-bit and 64-bit storage requirements, although a few increments is acceptable. This reduces both overall z/OS storage requirements and CPU time, compared to when there are many increments.

The HEAP64 31-bit increment size should not be set to less than 1M and the FREE option should be used. In the previous example, the 31-bit parameters were set to 4M, 1M, and FREE.

Language Environment 31-bit and 64-bit HEAP usage can be seen by activating the RPTO(ON) and RPTS(ON) options in DFHAXRO. An Language Environment storage report is produced when the JVM server is stopped.

You can override the Language Environment runtime options by modifying and recompiling the sample program DFHAXRO, which is described in "Modifying the enclave of a JVM server with DFHAXRO" on page 241. This program is set on the JVMSERVER resource, so you can use different names, which is why there are different options for individual JVM servers, if required.

The amounts of storage required for a JVM in a Language Environment enclave might require changes to installation exits, IEALIMIT or IEFUSI, which you use to limit the **REGION** and **MEMLIMIT** sizes. A possible approach is to have a Java owning region (JOR), to which all Java program requests are routed. Such a region runs only Java workloads, minimizing the amount of CICS DSA storage required and allowing the maximum amount of MVS storage to be allocated to JVMs.

Identifying Language Environment storage needs for JVM servers

After identifying the actual storage needs, it is possible to determine whether the supplied **DFHAXRO** options need to be modified or not. This allows values to be chosen that either avoid the need for incremental storage allocations, or reduce the number to an acceptable level.

About this task

The HEAP64 runtime option in DFHAXRO controls the heap size of the Language Environment enclave for a JVM server. This option includes settings for 64-bit, 31-bit, and 24-bit storage. You can use your own program instead of DFHAXRO if preferred. The program must be specified on the JVMSERVER resource.

Procedure

1. Set the `RPTO(ON)` and `RPTS(ON)` options in DFHAXRO.

   These options are in comments in the supplied source of DFHAXRO. Specifying these options causes Language Environment to report on the storage options and to write a storage report showing the actual storage used.
2. Disable the JVMSERVER resource.

   The JVM server shuts down and the Language Environment enclave is removed.
3. Enable the JVMSERVER resource.

   CICS uses the Language Environment runtime options in DFHAXRO to create the enclave for the JVM server. The JVM also starts up.
4. Run your Java workloads in the JVM server to collect data about the storage that is used by the Language Environment enclave.
5. Remove the `RPTO(ON)` and `RPTS(ON)` options from DFHAXRO.
6. Disable the JVMSERVER resource to generate the storage reports.

   The storage reports include a suggestion for the initial Language Environment enclave heap storage. The entry "Suggested initial size" in the 64-bit user heap statistics contains the suggested value and is equal to the total amount of Language Environment enclave heap storage that was used by the JVM server.

Results

The storage reports are saved in an `stderr` file in z/OS UNIX, or can also go to your CICS JES output if you are using the **`JOBLOG`** or **`DD://`** routing syntax. The directory depends on whether you have redirected output for the JVM in the JVM profile. If no redirection exists, the file is saved in the working directory for the JVM. If no value is set for WORK_DIR in the profile, the file is saved in the `/tmp` directory.

Use the information in the storage reports to select a suitable value for the Language Environment enclave heap storage in the **`DFHAXRO HEAP64`** option. Storage requirements might change from one CICS execution to the next, and are typically not the same for different CICS systems that share the one DFHAXRO, thus requiring a compromise.

The normal aim is to set the HEAP64 initial allocations to the suggested sizes to avoid or reduce the number of increments. The more increments that are used, the more likely that the ratio of z/OS storage compared to actively used Language Environment storage increases. Many increments can also cause an increase in the amount of CPU time that is used by Language Environment to manage the HEAP64 storage requests. Java 7.0 allocates the Java Heap in Language Environment 64-bit HEAP storage and this might result in a large initial allocation being suggested. In this case, you should subtract the **`-Xmx`** value from the suggested size and use the remainder, which should be less than **`-Xmx`**, as the initial allocation. This forces Language Environment to allocate the Java Heap in its own Memory Object as a Language Environment HEAP increment. Later Java releases allocate the JVM Heap as a Memory Object via IARV64 and not through a Language Environment storage request. If a Java migration is performed with an initial allocation that includes **`-Xmx`**, it normally doubles the storage that is used for the Java Heap, and might result in MEMLIMIT being too small.

Allocating many increments might produce the effect of a Storage Leak, which manifests as a continual increase in z/OS storage over time. In practice, this is more likely to be Storage Creep, which is characterized by an increase in both z/OS allocated storage and Language Environment free storage. A Storage Leak shows a continual increase in both z/OS and Language Environment used storage. 31-bit HEAP64 storage is allocated in z/OS subpool 1. In Java 7.0, JIT storage is allocated in z/OS subpool 1 and will naturally grow in size over time in increments of slightly over 2MB in size. In later Java releases, JIT storage is allocated in z/OS subpool 2 in 2MB increments.

The effect of the revised options should be evaluated at least one time and adjusted as required. Tuning should also be repeated at suitable intervals to assess the effect of any changes to storage usage due to application changes and other changes. Tuning should also be repeated whenever the CICS or Java release changes as storage usage patterns might change.

Note: If you increase the 31 bit **HEAP64** initial size, you must also change **HEAPP** to avoid over-allocating **`HEAPPOOLS`** 31-bit storage. In the example below, the HEAPPOOLS percentage values should be reduced from 10% to 1%.

HEAPPOOLS and HEAPPOOLS64 are active in the default DFHAXRO and can be effective when configured, but the correct values are dependent on the workload and hence precise tuning might be difficult.

STACK64 should be checked to ensure that the maximum storage used is not close to the defined limit, which is typically 16 MB. Exceeding the limit will results in runtime errors.

What is not obvious from LE RPTSTG output is that, while using STACK64(1M,1M,16M) provides a safe value for JVM thread stack expansion, it can result in a large MEMLIMIT being required to avoid CICS SOS Above the Bar during GDSA expansion. With the 16M maximum, 20 MB is allocated per JVM thread in three Memory Objects - one of 16+1 MB, one of 2 MB and one of 1 MB. Only 3 MB is normally usable (leaving 17 MB as hidden storage) and this is counted towards the z/OS IARV64 MEMLIMIT check. However, CICS counts all 20 MB to decide whether it can expand the GDSA by a multiple of GB without exceeding MEMLIMIT. A single JVM server can legitimately use more than 200 threads, and 200 threads equates to approximately 4 GB towards the CICS MEMLIMIT check. This is not a bug. Therefore, reducing the STACK64 maximum to a lower value that still permits some expansion can help towards reducing the MEMLIMIT size and the possibility of SOS Above the Bar. It is recommended that a smaller STACK64 maximum be validated in a suitable load test environment before being used in production.

Example

The following example is **RPTOPTS** output based on these DFHAXRO options:

```
HEAPPOOLS(ALIGN,8,10,32,10,128,10,256,10,1024,10,2048,10,0,10,0,10,0,10,0,10,0,10,0,10)
HEAPPOOLS64(ALIGN,8,4000,32,2000,128,700,256,350,1024,100,2048,50,3072,50,4096,50,8192,
            25,16384,10,32768,5,65536,5)
HEAP64(256M,4M,KEEP,4194304,1048576,KEEP,1024,1024,KEEP)
LIBHEAP64(3M,3M,FREE,16384,8192,FREE,8192,4096,FREE)
STACK64(1M,1M,16M)
THREADSTACK64(OFF,1M,1M,128M)
```

The following example is partial **RPTSTG** output:

```
STACK64 statistics:
Initial size:                                     1M
Increment size:                                   1M
Maximum used by all concurrent threads:           1M
Largest used by any thread:                       1M - no change required
Number of increments allocated:                   0
THREADSTACK64 statistics:
Initial size:                                     1M
Increment size:                                   1M
Maximum used by all concurrent threads:           0M
Largest used by any thread:                       0M - not used
Number of increments allocated:                   0
64bit User HEAP statistics:
Initial size:                                     256M
Increment size:                                   4M
Total heap storage used:                          730857472
Suggested initial size:                           697M - use this
Successful Get Heap requests:                     783546
Successful Free Heap requests:                    780785
Number of segments allocated:                     135 - too many increments
Number of segments freed:                         0
31bit User HEAP statistics:
Initial size:                                     4194304
Increment size:                                   1048576
Totalheap storage used (suggested initial size):  137165672 - use this
Successful Get Heap requests:                     1345332
Successful Free Heap requests:                    1345260
Number of segments allocated:                     125 - too many increments
Number of segments freed:                         0
64bit Library HEAP statistics:
Initial size:                                     3M
Increment size:                                   3M
Total heap storage used:                          4640032
Suggested initial size:                           5M
Successful Get Heap requests:                     113381
Successful Free Heap requests:                    112860
Number of segments allocated:                     1 - low, so no change required
Number of segments freed:                         0
31bit Library HEAP statistics:
Initial size:                                     16384
Increment size:                                   8192
Total heap storage used (suggested initial size): 520
Successful Get Heap requests:                     33725
Successful Free Heap requests:                    33725
Number of segments allocated:                     1 - low, so no change required
Number of segments freed:                         0

Suggested Percentages for current CellSizes:
HEAPP(ALIGN,8,1,32,1,128,1,256,1,1024,1,2048,1,0)
```

When reviewing **RPTSTG** output, remember that the **HEAP64** increment sizes are for the minimum amount of storage that Language Environment allocates, and any increment could be substantially bigger than that value. Hence it is not possible to accurately determine how much z/OS storage was used when 1 or more increments have been allocated. The actual number of increments is reported for 64bit HEAP (that is, 135), for 31bit **HEAP** the actual number of increments is one less than is shown (that is, 124 not 125).

Because of the way that Language Environment's storage management works when increments are used, the amount of 31-bit and 64-bit z/OS storage allocated may be significantly higher than shown in **RPTSTG** "maximum used".

The suggested **DFHAXRO** changes are:

```
* Heap storage
DC     C'HEAP64(700M,'         Initial 64bit heap - change (Note 1)
DC     C'4M,'                   64bit Heap increment
DC     C'KEEP,'                 64bit Increments kept
DC     C'128M,'                 Initial 31bit heap - change (Note 2)
DC     C'2M,'                   31bit Heap increment - change (Note 3)
DC     C'FREE,'                 31bit Increments freed - change (Note 4)
DC     C'1K,'                   Initial 24bit heap
DC     C'1K,'                   24bit Heap increment
DC     C'KEEP) '                24bit Increments kept

* Heap  pools
DC     C'HP64(ALIGN) '
DC     C'HEAPP(ALIGN,8,1,32,1,128,1,256,1,1024,1,2048,1,0) ' - change (Note 5)

* Library Heap storage
DC     C'LIBHEAP64(3M,3M) '     Initial 64bit heap - do not change (Note 6)

* 64bit stack storage
DC     C'STACK64(1M,1M,16M) ' - consider a change (Note 7)
```

Note:

1. As shown by **RPTSTG** output 64bit "Suggested initial size" plus a small increase.
2. As shown by **RPTSTG** output 31bit "Suggested initial size" but with a small reduction as we are using FREE.
3. The 31-bit **HEAP** increment may be better as a value of 2M instead of 1M.
4. Optionally, using 31-bit **HEAP FREE** may result in less z/OS storage being allocated to map the "Total heap storage used" than with KEEP.
5. As recommended by **RPTSTG** output after the **HEAPPOOLS** statistics, but may benefit from further optimization. The default of 10% of the 31-bit Heap initial size of 128MB is likely to result in an excessive amount of storage being allocated. A minimum of 6 pools each of 10% of the initial heap size of 128MB causes 77MB to be allocated. This will be included in the "Total heap storage used" value (because the **HEAPPOOLS** storage extents are allocated there), irrespective of what percentage of the pool s is productively used. Using **HEAPPOOLS** cell sizes greater than 256 bytes might result in inefficient use of Language Environment HEAP storage.
6. Only one increment was required, which is not a problem.
7. The largest used was 1MB. Reducing the maximum of 16M to a value such as 8 MB or even lower would significantly reduce the amount of STACK64 storage that CICS counts towards **MEMLIMIT** when checking to see whether it can allocate a new GDSA extent. STACK64 changes should be tested thoroughly before migrating them into a production environment.

This is an example of using 31-bit **HEAP FREE** on another run of the same JVM server. The "Number of segments" shows the number of **GETMAIN**s and **FREEMAIN**s performed, which was low for the time that the JVM server was active. The difference of 2 shows that the enclave terminated with only the initial allocation plus one increment, which is likely to be less than the "Total heap storage" and shows the effectiveness of FREE. "Total heap storage used" was higher, but any total often changes from one run of a JVM server to another, hence basing changes on only one set of **RPTSTG** may not provide the best possible settings.

```
31bit User HEAP statistics:
Initial size: 134217728
Increment size: 2097152
Total heap storage used (suggested initial size): 154056664
Successful Get Heap requests: 3253239
Successful Free Heap requests: 3253176
Number of segments allocated: 149
Number of segments freed: 147
```

It is important to read the Language Environment Debugging Guide in order to correctly interpret **RPTSTG** output.

Modifying the enclave of a JVM server with DFHAXRO

DFHAXRO is a sample program that provides a default set of runtime options for the Language Environment enclave in which a JVM server runs. For example, it defines storage allocation parameters for the JVM heap and stack. It is not possible to provide default runtime options that are optimized for all workloads. Consider identifying actual storage usage, and override the defaults as required, to optimize the ratio of used storage to allocated storage.

About this task

You can update the sample program to tune the Language Environment enclave or you can base your own program on the sample. The program is defined on the JVMSERVER resource and is called during the CELQPIPI preinitialization phase of the Language Environment enclave that is created for a JVM server.

You must write the program in assembly language and it must not be translated with the CICS translator. The options are specified as character strings, comprising a 2-byte string length followed by the runtime option. The maximum length for all Language Environment runtime options is 255 bytes, so use the abbreviated version of each option and restrict your changes to a total of under 200 bytes.

Procedure

1. Copy the DFHAXRO program to a new location to edit the runtime options.

   If maintenance is applied to your CICS region, you might want to reflect the changes in your program. The source for DFHAXRO is in the CICSTS54.CICS.SDFHSAMP library.
2. Edit the runtime options, using the abbreviation for each option.

   The z/OS Language Environment Programming Guide has complete information about Language Environment runtime options.

   - Keep the size of the list of options to a minimum for quick processing and because CICS adds some options to this list.
   - Use the HEAP64 option to specify the initial heap allocation.
   - The ALL31 option, the POSIX option, and the XPLINK option are forced on by CICS. The ABTERMENC option is set to (ABEND) and the TRAP option is set to (ON,NOSPIE) by CICS.
   - The output that is produced by the RPTO and RPTS options is written to the CESE transient data queue.
   - Any options that produce output do so at each JVM termination. Consider the volume of output that might be produced and directed to CESE.
3. Use the DFHASMVS procedure to compile the program.

Results

When you enable the JVMSERVER resource, CICS creates the Language Environment enclave by using the runtime options that you specified in the DFHAXRO program. CICS checks the length of the runtime options before it passes them to Language Environment. If the length is greater than 255 bytes, CICS does not attempt to start the JVM server and writes error messages to CSMT. The values that you specify are not checked by CICS before they are passed to Language Environment.

Tuning the z/OS shared library region

The shared library region is a z/OS feature that enables address spaces to share dynamic link library (DLL) files. This feature enables your CICS regions to share the DLLs that are needed for JVMs, rather than each region having to load them individually. This can greatly reduce the amount of real storage used by MVS, and the time it takes for the regions to load the files.

The storage that is reserved for the shared library region is allocated in each CICS region when the first JVM is started in the region. The amount of storage that is allocated is controlled by the **`SHRLIBRGNSIZE`** parameter in z/OS, which is in the BPXPRMxx member of SYS1.PARMLIB. The minimum is 16 MB, and the z/OS default is 64 MB. You can tune the amount of storage that is allocated for the shared library region by investigating how much space you need, bearing in mind that other applications besides CICS might be using the shared library region, and adjusting the **`SHRLIBRGNSIZE`** parameter accordingly.

If you want to reduce the amount of storage that is allocated for the shared library region, first check that you do not have wasted space in your shared library region. Bring up your normal workload on the z/OS system, then issue the command **D OMVS,L** to display the library statistics. If there is unused space in the shared library region, you can reduce the setting for **SHRLIBRGNSIZE** to remove this space. If CICS is the only user of the shared library region, you can reduce the **SHRLIBRGNSIZE** to the minimum of 16 MB, because the DLLs needed for the JVM only use around 10 MB of the region.

If you find that all the space in the shared library region is being used, but you still want to reduce this storage allocation in your CICS regions, there are three possible courses of action that you can consider:

1. It is possible to set the shared library region size smaller than the amount of storage that you need for the files. When the shared library region is full, files are loaded into private storage instead, and do not benefit from the sharing facility. If you choose this course of action, you should make sure that you restart your more important applications first, to ensure that they are able to make use of the shared library region. This course of action is most appropriate if most of the space in the shared library region is being used by non-critical applications.
2. The DLLs that are placed in the shared library region are those marked with the extended attribute +l. You can remove this attribute from some of your files to prevent them going into the shared library region, and so reduce the amount of storage that you need for the shared library region. If you choose this course of action, select files that are less frequently shared, and also try not to select files that have the extension .so. Files with the extension .so, if they are not placed in the shared library region, are shared by means of user shared libraries, and this sharing facility is less efficient than using the shared library region. This course of action is most appropriate if large files that do not have the extension .so are using most of the space in the shared library region.
3. If you remove the extended attribute +l from all the files relating to the CICS JVM, then your CICS regions do not use the shared library region at all, and no storage is allocated for it within the CICS regions. If you choose this course of action, you do not benefit from the shared library region's sharing facility. This course of action is most appropriate if other applications on the z/OS system require a large shared library region, and you do not want to allocate this amount of storage in your CICS regions.

If you choose to remove the extended attribute +l from any of your files, when you replace those files with new versions (for example, during a software upgrade), remember to check that the new versions of the files do not have this attribute.

Improving ONC RPC performance

Important: This information contains Diagnosis, Modification, or Tuning Information.

The performance of a single client request is affected by various aspects of the client, the network, CICS ONC RPC, the user-replaceable programs, and CICS.

The client

The client timeout interval must take account of the possible delays in dealing with a client request in CICS ONC RPC and in CICS.

If a client request cannot be processed, an error reply is sent to the client.

The network

This manual does not deal with performance problems of TCP/IP networks.

z/Os Communications Server resources

If, while registering 4-tuples, you cause the connection manager to register too many 3-tuples with z/OS Communications Server, you might reduce the service that CICS ONC RPC can give to incoming client requests.

CICS ONC RPC

The allocation of different alias transaction names to different 4-tuples must be coordinated with the priorities given to the transactions in CICS.

The converter URM

Getlengths is called only by the connection manager, and has no effect on the performance of a single client request, or on throughput.

Decode is called by the server controller. Delays here can reduce the throughput of CICS ONC RPC as well as reducing the performance of a single client request. The following recommendations are made:

- Do not use CICS trace here except to solve specific problems.
- Use NOSUSPEND on EXEC CICS GETMAIN. If GETMAIN errors occur because there is not enough storage, look for solutions that do not involve using the SUSPEND option.

Encode is called by the alias. Delays here reduce the performance of single client requests, but not the throughput of CICS ONC RPC.

The resource checker URM

The resource checker is called by the alias, so delays here affect the performance of a single client request, but have no effect on throughput.

CICS HTTP support: Performance and tuning

You can tune several aspects of your system to improve the performance of CICS HTTP support.

What affects the limit on the number of HTTP connections

Several aspects of your CICS region configuration can affect the maximum number of HTTP connections the region can support:

Storage available in the CICS region

In practical terms, the number of connections that can be active between a single CICS region and the web is primarily limited by the storage available in the CICS region. “Storage requirements for CICS web support” on page 246 explains the most significant storage requirements for CICS web support activities.

Effect of the MXT setting

In each CICS region, the maximum number of concurrent connections between CICS as an HTTP server and web clients, or between CICS as an HTTP client and a server on the web, can in theory be up to 64,000. The MXT setting does not directly limit the number of concurrent connections:

- For CICS as an HTTP server, the CWXN transaction (web attach task) does not remain in the system for the duration of a persistent connection, but terminates after each request for which it was required. This means that between requests, a persistent connection can exist, being monitored by the Sockets listener task (CSOL), without being associated with an active task.
- For CICS as an HTTP client, an application program can open and maintain more than one persistent connection to a server on the web, and these are covered by the single active task for the application program.

However, the MXT setting, and any limitations that you set in the transaction class definitions for CICS web support transactions, can be used to limit the amount of CICS web support activity in the CICS region. For information about how to set MXT, see Setting the maximum task specification (MXT) .

Using performance tuning for HTTP connections to protect CICS from unconstrained resource demand

The SOTUNING SIT parameter specifies whether performance tuning for HTTP connections occur to protect CICS from unconstrained resource demand. If the SOTUNING SIT parameter is set to the default value of YES, if the region becomes overloaded CICS temporarily stops listening for new HTTP connection requests. If overloading continues, CICS closes existing HTTP persistent connections and marks all new HTTP connections as non-persistent. These actions prevent oversupply of new HTTP work from being received and queued within CICS, allowing feedback to TCP/IP port sharing and Sysplex Distributor, promoting a balanced sharing of workload with other regions that are sharing the same IP endpoint and allowing the CICS region to recover more quickly. If SOTUNING is set to YES, CICS periodically closes persistent connections to allow more efficient sharing of workload across regions that share IP endpoints, using technologies such as TCP/IP shared ports and Sysplex distributor.

CICS TCP/IP statistics can be used to illustrate whether performance tuning has taken place. For more information, see “Performance tuning statistics” on page 250.

Performance considerations for CICS as an HTTP server

What affects performance

Priority of transactions is significant

For CICS as an HTTP server, the priority of the various transactions involved for CICS web support is significant, and incorrect settings might lead to a short-on-storage situation. “Priorities for CICS web support transactions (CWXN, CWXU, CWBA, CW2A)” on page 247 tells you how to set priorities to avoid issues in this area.

Types of HTTP responses influence performance

For CICS as an HTTP server, HTTP responses can be generated by an application, or provided from a static document, which can be either an zFS file or a CICS document template. A CICS document template can be one of a variety of different CICS resources, including a program, a partitioned data set, or a file. “Relative performance of CICS web support response methods” on page 249 explains the differences in performance between these response types.

How to improve performance

Connection throttling can prevent a CICS region from being overloaded

By default, CICS attempts to keep HTTP connections open as persistent connections. Persistent connections between a web client and a server can be reused for more than one exchange of a request and a response, so a new connection does not have to be established for each request. Persistent connections improve network performance because compared with making a request use an existing connection, establishing a new connection consumes significant additional network resources. However, if multiple web clients set up long-lived persistent connections to CICS as an HTTP server and use the connections heavily, it is possible for a CICS region handling the connections to become overloaded and experience performance problems.

If you experience such problems, you can set up connection throttling to make excess web clients connect to other CICS regions that share the port and provide the same service.

- Use the MAXPERSIST attribute on the TCPIPSERVICE resource definition to set a limit on the number of persistent HTTP connections that a CICS region accepts for a particular port. Subsequent web clients are subject to connection throttling, and are required to close their connection after each response. When the new clients reconnect, if they connect to another CICS region that shares the port and has not reached its limit they can maintain a persistent connection there.
- An HTTP/1.1 server should normally allow persistent connections, so only set up connection throttling in a CICS region that has experienced performance problems due to long-lived persistent connections. If you do limit the number of persistent connections, CICS as an HTTP server is still compliant with the HTTP/1.1 specification (provided that you do not set a zero limit), but refusing persistent connections is not recommended as a normal practice.
- Be aware that the performance of web clients can be affected when they fail to obtain a persistent connection that they expected, especially if they reconnect to the CICS region where connection throttling is in place. The performance returns to normal when the web clients connect to another CICS region that shares the port and has not reached its limit.

For more information about persistent connections and connection throttling, see How CICS web support handles persistent connections.

CICS TCP/IP statistics can be used to illustrate how well connections are reused. For more information, see “Connection persistence statistics” on page 252.

Performance considerations for CICS as an HTTP client

How to improve performance

Connection pooling can provide performance benefits in some situations

For CICS as an HTTP client, by default CICS closes client HTTP connections after an application has finished using the connection. If you set up connection pooling, CICS can leave the connection open and place it in a pool in a dormant state, and it can be reused by the same application or by another application that connects to the same host and port. "Connection pooling for HTTP client performance" on page 253 explains the situations in which connection pooling provides performance benefits for CICS web support applications and web services applications.

Performance considerations for SSL connections

- If you are using the Secure Sockets Layer (SSL), the security measures (such as encryption and decryption and the SSL handshake) cause a slight increase in CPU per transaction. For CICS as an HTTP client, if you implement connection pooling CICS applications can reuse client HTTP connections that have already been made using SSL. The security measures do not need to be repeated when the connection is reused, so applications that are given a connection from the pool do not experience this CPU increase.
- You can monitor the cipher suites that are being selected for SSL connections between each CICS region and its clients. The performance data field SOCIPHER (320) in the DFHSOCK group shows the code for the cipher suite that was used for each SSL inbound connection. Use this information to identify any cipher suites that are offered by the CICS region but are not being selected for SSL connections. You can also identify any less efficient or less secure cipher suites that are being selected for SSL inbound connections but that you would prefer to eliminate. Customizing encryption negotiations explains how to customize the list of cipher suites that can be used in encryption negotiations.
- SSL uses S8 TCBs. For HTTP requests over SSL, all CICS processing occurs on the S8 TCB (there is no switching between the S8 and SO TCBs). This means that the web attach task (CWXN by default) will stay on the S8 TCB until all the data has been sent or received. This could result in the task that is sending or receiving the messages hitting the runaway limit and being terminated. If sending or receiving large messages, you should determine if you need to increase the transaction **RUNAWAY** value for the web server HTTP attach transaction (CWXN by default), the web server alias transactions, and any transactions that issue the Web client API commands SEND, RECEIVE and CONVERSE.

  You can monitor S8 TCBs using the dispatcher TCB statistics.

  See CICS dispatcher: performance and tuning for guidance on setting the system initialization parameters that control the number of TCBs that CICS uses.

Storage requirements for CICS web support

The number of connections that can be sustained between a single CICS region and the web is primarily limited by the storage available in the CICS region.

The major influences on storage usage for CICS web support are:

- **Basic storage requirement for each connection.** About 4K of storage is required for each connection between CICS and the Web.
- **For CICS as an HTTP server: Size of requests received from web clients.** The total amount of data that CICS accepts for a single request can be limited by the MAXDATALEN attribute of the TCPIPSERVICE definition.
  - The request line and HTTP headers are stored in a container.
  - The request body is stored in CICS main storage. Unwanted data from a request body can be ignored by an application program, but the web attach task always reads the complete message in order to clear the data from the socket, and places all the data in storage.
  - Storage used for a request received from a web client is freed when a response has been sent to the request.

- **For CICS as an HTTP client: Size of responses received from web servers.** There is no specific way to limit the amount of data that CICS accepts for a response. (The HTTP client facility of CICS web support is not designed for use as a browser, so your requests should return only known resources that you have selected.)
  - The status line and HTTP headers are stored in a container.
  - The response body is stored in CICS main storage. Unwanted data from a response body can be ignored by an application program, but the complete message is read and placed in CICS storage.
  - Storage used for responses received from web servers is required for the duration of the connection with the web server. The storage obtained for the first response is overwritten by each subsequent response that is received for the connection, or released and re-obtained if a subsequent response requires a larger amount of storage. The storage is freed when the connection is closed by the application program using a WEB CLOSE command (or at end of task if the connection has not previously been closed). The maximum amount of storage that is obtained for each connection is therefore equal to the size of the largest message received on the connection.
- **For CICS as an HTTP server and CICS as an HTTP client: Size of messages produced by CICS (requests or responses).** While a request or response is being assembled for sending out from CICS, storage is required for the HTTP headers, and also for the message body.
  - The HTTP headers are stored in a container.
  - The message body is stored in CICS main storage.
  - Storage can be freed in two ways for WEB SEND (Client and Server) and WEB CONVERSE commands. These options should only be used if you do not intend (later in this transaction) to retrieve the contents of the document that has been sent:
    - Specify the WEB SEND or CONVERSE command with the ACTION(IMMEDIATE) option.
    - Specify the WEB SEND command with the ACTION(EVENTUAL) and DOCSTATUS(DOCDELETE) options.
- **Code page conversion.** Code page conversion can be carried out for any message body received or sent by CICS, if it is specified by an application program, analyzer program, or URIMAP definition in the processing path for the request. The message body is in CICS main storage.
  - For conversion between the EBCDIC code page 037 and the ASCII code page ISO-8859-1, CICS writes the converted message body to the same area of storage as the original message body, so no additional storage is used.
  - For other types of code page conversion, CICS requires additional storage to contain the converted message body. Depending on the character sets used, the size of this additional storage area can range from the same size as the original message body, to a theoretical maximum of four times the size of the original message body (which is unlikely). For example, 2MB of message body data would require at least 4MB of storage in total. Double-byte character sets (DBCS) or multi-byte character sets are likely to require larger storage areas within this range.

Priorities for CICS web support transactions (CWXN, CWXU, CWBA, CW2A)

When CICS receives a request which is not eligible for being processed by directly attached alias transaction, a web attach task is used to receive requests from web clients and perform initial checks. The default transaction ID for a web attach task is CWXN for the HTTP protocol, or CWXU for the USER protocol. Alias transactions are used to cover subsequent processing for application-generated responses. The default transaction ID for a CICS web support alias transaction is CWBA, and for an Atom feed alias transaction, it is CW2A.

Task structure for CICS web support has more information about the transactions that are used for CICS web support processing, and how they interact with each other.

If you set the priority of the CWXN or CWXU transaction (or its alias) higher than the priority of the alias transactions such as CWBA or CW2A, this can result in a build-up of requests that have been received but not yet processed, which might lead to a short-on-storage situation.

The default priorities for the CWXN and CWXU transactions are set to 1, and the default priorities for the CICS-supplied CWBA and CW2A transactions are also set to 1. You can adjust these priorities depending on your workload. Make sure the priorities of the alias transactions like CWBA or CW2A are higher than the priority of the transactions associated with web attach tasks like CWXN or CWXU.

Consider using TCLASS definitions to limit the number of web attach tasks that can run to less than 50% of the maximum user task limit, in order to give priority to the processing of alias transactions. At peak workload, this means that storage is freed in preference to being acquired and that inflight tasks are processed in preference to new requests.

Processing HTTP requests by using directly attached user transactions

The socket listener task CSOL is optimized to attach user transactions directly for fast arriving HTTP requests. This bypasses the web attach task, so reduces the CPU time that is required to process each request.

To qualify for optimization, the HTTP request and the TCPIPService and URIMAP resource attributes must meet the criteria shown in Table 27 on page 248. Ensure that your requests are fully configured by selecting all the resource attribute values from the "Required attribute values" column in this table.

Note: For requests that require a static response, the web attach task is always attached to process them. No other transaction is involved.

Table 27. Criteria to process HTTP requests using directly attached user transactions

Resource or request	**Attribute or request condition**	**Required attribute values and conditions for the direct route: CSOL > user transaction** **ALL criteria must apply**	**Attribute values and conditions for the indirect route: CSOL > web attach task > user transaction** **Used if any criteria apply**
TCPIPService	PROTOCOL	HTTP	USER
	SSL	NO | ATTLSAWARE	YES | CLIENTAUTH
URIMAP	ANALYZER	NO	YES
	USAGE	SERVER | PIPELINE | ATOM	CLIENT
HTTP REQUEST	HEADER	Arrived	Not arrived
	URIMAP	Matched	Not matched

The socket listener task CSOL detects inbound TCP/IP connection requests on all ports that are defined to CICS, and starts the CICS service that is associated with the port. When the port is intended for CICS web support that uses the PROTOCOL of HTTP and SSL support NO or ATTLSAWARE, CSOL uses the available request data to assess whether a directly attached user transaction can handle the request.

For example, in the following situations, the request cannot be optimized and CSOL attaches the web attach task to process the request:

- CSOL cannot determine the URI and the USER ID from the request header data (for example, an HTTP 1.0 request that did not contain the host header was received).
- No matching URIMAP is found or enabled for the request.
- The matching URIMAP indicates that the request needs to go through the analyzer program (that is, the URM attribute on TCPIPService).

For web service requests and atom feeds (that is, URIMAP with usage of PIPELINE or ATOM), no analyzer is involved, so these requests can be optimized and processed by the transactions CPIH or CW2A, provided that no SSL session is used. This mechanism is ready to adopt the direct route and gain performance enhancement, without any changes.

For an HTTP server request, if the matching URIMAP indicates that the client expects an application-generated response and other criteria are also met, CSOL directly attaches the transaction CWBA that is defined in the URIMAP to process the request. The transaction's task receives the request data from the web client and processes it as normal.

Note: CWXN must always be configured so that the web attach task can process requests that do not meet the optimization criteria.

Relative performance of CICS web support response methods

Application-generated responses use more resources than static responses. Application-generated responses require a user transaction to be attached. An analyzer program, converter program, or more than one user-written application program might be involved in processing the request and producing the response. Typically, greater elapsed time and processor time is required to produce the response.

Static responses involve only the web attach task, a URIMAP definition, and the source document for the response body. Performance for a static response is better than for an application-generated response, so if you are using an architecture with an application program and analyzer program to deliver a simple response document, consider converting this to a static response. For those application-generated responses, the requests that are targeting a URIMAP with ANALYZER(NO) specified might qualify for being processed by directly attached user transactions, and bypassing the web attach task. For more information, see "Processing HTTP requests by using directly attached user transactions" on page 248.

For UNIX files only, the HTTP If-Unmodified-Since header is respected. If the UNIX file is unmodified since the time selected in this header, the file is not returned, and an HTTP 304 (Not Modified) response with no message body is returned instead. This can considerably reduce the amount of data transmitted.

Within this category, performance is further influenced by the choice of source document used for the response body, which can be:

- A z/OS UNIX System Services file, called directly from the URIMAP definition using the HFSFILE option.
- A z/OS UNIX System Services file, defined as a CICS document template, and called from the URIMAP definition using the DOCTEMPLATE option.
- A document template stored in a z/OS partitioned data set or PDSE.
- A document template stored in a transient data queue.
- A document template stored in a temporary storage queue.
- A document template stored in a CICS file (ESDS, RRDS or another type of data set).
- A document template contained in a CICS program.
- A document template generated by an exit program. The content of the document template could be loaded from a location such as DB2 or another database manager. Document templates generated by an exit program are classed with static responses when they operate through a URIMAP definition and not through a web-aware application program. However, they do involve an application program, so in terms of resource and performance, they can be similar to an application-generated response.

Introduction to documents and document templates has more information about the different types of CICS document template, and how to set them up. If you are using a CICS document template to provide a static response, ensure that the definition is installed before you use it.

To improve performance, the CICS document handler caches a copy of most document templates when they are installed. Subsequent references to the template use the cached copy. This means that the relative speed of access for the document template type is not important after the first retrieval from the source. Caching does not take place for document templates retrieved from CICS programs, because programs are already managed by the CICS loader, and have fast retrieval times. For document templates generated by exit programs, you can specify whether or not a copy is to be cached.

When storage is constrained, the performance of document templates can be impacted. Programs containing document templates are managed like other CICS loaded programs, and may be flushed out by program compression. Document templates cached by the document handler may be released, and on the next reference of these document templates, they will need to be retrieved from the source.

Performance tuning statistics

Performance tuning for HTTP connections protects CICS from unconstrained resource demand. You can use TCP/IP statistics to see if performance tuning has taken place.

Using TCP/IP statistics to see whether CICS has paused listening for new HTTP connection requests because it is under stress

If a region becomes overloaded, CICS pauses listening for new HTTP connection requests, and any new requests queue outside of CICS in the TCP/IP backlog queue. The size of this queue is set per TCPIPSERVICE by the resource BACKLOG attribute. If enough requests are received to fill the queue, new connection requests will be dropped. The following fields in TCP/IP Global statistics indicate whether CICS has paused listening for new HTTP connection requests as a result of performance tuning:

Performance tuning for HTTP connections (SOG_SOTUNING)
Indicates whether performance tuning for HTTP connections is enabled.

Socket listener has paused listening for HTTP connections (SOG_PAUSING_HTTP_LISTENING)
Indicates whether the listener has paused listening for HTTP connection requests because the number of tasks in the region has reached the limit for accepting new HTTP connection requests.

Number of times socket listener notified at task accept limit (SOG_TIMES_AT_ACCEPT_LIMIT)
Indicates the number of times the listener has been notified that the number of tasks in the region has reached the limit for accepting new HTTP connection requests.

Last time socket listener paused listening for HTTP connections (SOG_TIME_LAST_PAUSED_HTTP_LISTENING)
Indicates the last time the socket listener paused listening for HTTP connection requests because the number of tasks in the region had reached the limit for accepting new HTTP connection requests.

For a TCPIPSERVICE, the following fields in TCP/IP services: Resource statistics indicate whether requests are queuing in the TCP/IP backlog queue and whether the queue is full and new connection requests are being dropped:

Current backlog (SOR_CURR_BACKLOG)
Indicates the current number of connection requests waiting in the backlog queue, summed over all appropriate stacks if the TCP/IP service is listening on multiple stacks.

Connections dropped (SOR_CONNS_DROPPED)
Indicates the total number of connections that were dropped because the backlog queue of the TCP/IP service was full, summed over all appropriate stacks if the TCP/IP service is listening on multiple stacks.

Time connection last dropped (SOR_CONN_LAST_DROPPED)
Indicates the time that a connection was last rejected because the backlog queue of the TCP/IP service was full.

Current maximum backlog (SOR_CURR_MAX_BACKLOG)
Indicates the maximum number of connection requests that the TCP/IP service currently allows in the backlog queue for the service, summed over all appropriate stacks if the TCP/IP service is listening on multiple stacks. This value can be greater than the value that is specified in the BACKLOG attribute (SOR_BACKLOG) of the TCP/IP service because the TCP/IP service might temporarily increase this value if, for example, it determines that there is a SYN flood.

Note: If a CICS region has reached the maximum number of user tasks, which is specified in the **MXT** SIT parameter, you might not be able to log into the CICS system and issue a **STATS** command. If this happens, you can use the **NETSTAT ALL** command to obtain the information about the backlog queue and connections dropped.

Using TCP/IP statistics to see whether existing persistent HTTP connections are being closed because a region is under stress

If the region continues to become overloaded, it closes existing persistent HTTP connections after their next request completes, and does not allow new HTTP connections to be persistent until the region is no longer under stress. The following statistics in TCP/IP Global statistics indicate whether CICS is temporarily disallowing HTTP connection persistence as a result of performance tuning:

Performance tuning for HTTP connections (SOG_SOTUNING)
Indicates whether performance tuning for HTTP connections is enabled.

Region stopping HTTP connection persistence (SOG_STOPPING_PERSISTENCE)
Indicates whether the region is closing existing persistent connections when their next request completes and is making new connections non-persistent, because the number of tasks in the region has exceeded the limit.

Number of times region stopped HTTP connection persistence (SOG_TIMES_STOPPED_PERSISTENT)
Indicates the number of times the region took action to close existing persistent connections when their next request completes and make new connections non-persistent, because the number of tasks in the region had exceeded the limit.

Last time stopped HTTP connection persistence (SOG_TIME_LAST_STOPPED_PERSISTENT)
Indicates the last time the region took action to close existing persistent connections when their next request completes and make new connections non-persistent, because the number of tasks in the region had exceeded the limit.

Number of persistent connections made non-persistent (SOG_TIMES_MADE_NON_PERSISTENT)
Indicates the number of times a persistent HTTP connection was made non-persistent because the number of tasks in the region had exceeded the limit.

Number of times disconnected an HTTP connection at max uses (SOG_TIMES_CONN_DISC_AT_MAX)
Indicates the number of times a persistent HTTP connection was disconnected because the number of uses had exceeded the limit.

For a TCPIPSERVICE, the following statistics in TCP/IP services: Resource statistics indicate whether existing persistent connections are being disconnected and whether new persistent connections will be made non-persistent:

Made non-persistent at task limit (SOR_NONP_AT_TASK_LIMIT)
Indicates the number of times a new persistent HTTP connection was made non-persistent because the number of tasks in the region had exceeded the limit.

Disconnected at task limit (SOR_DISC_AT_TASK_LIMIT)
Indicates the number of times an existing persistent HTTP connection was closed because the number of tasks in the region had exceeded the limit.

Disconnected after maximum uses (SOR_DISC_AT_MAX_USES)
Indicates the number of times a persistent HTTP connection was disconnected because its number of uses had exceeded the limit.

If there are suddenly a lot more non-persistent connections than normal, check whether performance tuning temporarily disallowed connection persistence to allow the region to recover from overloading.

Multiple stacks considerations

If the TCPIPSERVICE is listening on multiple stacks, each stack has its own independent backlog queue for the TCPIPSERVICE. The following statistics fields are the sum of values over all the stacks:

- Current backlog (SOR_CURR_BACKLOG)
- Connections dropped (SOR_CONNS_DROPPED)

The following statistics fields report the greatest value among all stacks:

- TCPIPSERVICE Backlog Setting (SOR_BACKLOG)
- Time connection last dropped (SOR_CONN_LAST_DROPPED)

- Current maximum backlog (SOR_CURR_MAX_BACKLOG)

This means that the reported number of requests currently queuing in the backlog queue (SOR_CURR_BACKLOG) can be greater than the reported backlog setting (SOR_CURR_MAX_BACKLOG) because the latter is the size of the queue per stack.

Connection persistence statistics

By default, CICS attempts to keep HTTP connections open as persistent connections. From TCP/IP statistics, you can obtain a view of connection persistence for a CICS region and for connections into a specific TCPIPSERVICE.

Obtaining a view of connection persistence for a CICS region

The following statistics in TCP/IP Global statistics provide a view of connection persistence for a CICS region:

Inbound

Total number of inbound sockets created (SOG_INB_SOCKETS_CREATED)
The total number of inbound sockets created

Total number of non-persistent inbound sockets created (SOG_NPERS_INB_SOCKETS_CREATED)
The total number of non-persistent inbound sockets created

Current number of inbound sockets (SOG_CURR_INBOUND_SOCKETS)
The current number of inbound sockets

Current number of non-persistent inbound sockets (SOG_CURR_NPERS_INB_SOCKETS}
The current number of non-persistent inbound sockets

Current number of persistent inbound sockets (calculated for the reports)
The current number of persistent inbound sockets

Peak number of inbound sockets (SOG_PEAK_INBOUND_SOCKETS)
The peak number of inbound sockets

Peak number of non-persistent inbound sockets (SOG_PEAK_NPERS_INB_SOCKETS)
The peak number of non-persistent inbound sockets

Peak number of persistent inbound sockets (SOG_PEAK_PERS_INB_SOCKETS)
The peak number of persistent inbound sockets

Outbound

Total number of outbound sockets created (SOG_OUTB_SOCKETS_CREATED)
The total number of outbound sockets created

Total number of persistent outbound sockets created (SOG_PERS_OUTBOUND_CREATED)
The total number of persistent outbound sockets created

Current number of non-persistent outbound sockets (SOG_CURR_OUTB_SOCKETS)
The current number of non-persistent outbound sockets

Current number of persistent outbound sockets (SOG_CURR_PERS_OUTB_SOCKETS)
The current number of persistent outbound sockets

Peak number of outbound sockets (SOG_PEAK_BOTH_OUTB_SOCKETS)
The peak number of outbound sockets

Peak number of non-persistent outbound sockets (SOG_PEAK_OUTB_SOCKETS)
The peak number of non-persistent outbound sockets

Peak number of persistent outbound sockets (SOG_PEAK_PERS_OUTB_SOCKETS)
The peak number of persistent outbound sockets

Obtaining a view of connection persistence for connections into a TCPIPSERVICE

The following statistics in TCP/IP services: Resource statistics can give you a view of connection persistence for connections into a specific TCPIPSERVICE:

Total Connections (SOR_TOTAL_CONNS)
The total number of connections made for the TCP/IP service

Non-Persistent Connections (SOR_TCPIPS_NON_PERSIST)
The number of connections where CICS did not allow the Web client to have a persistent connection

Maximum Persistent Connections (SOR_TCPIPS_MAX_PERSIST)
The maximum number of persistent connections from Web clients that the CICS region accepts at any one time

Made non-persistent at MAXPERSIST (SOR_NONP_AT_MAXPERSIST)
The number of times a new persistent connection was made non-persistent because MAXPERSIST was reached

Requests processed (SOR_REQUESTS)
The number of requests processed by the TCP/IP service

Many factors control connection persistence. For details, see How CICS web support handles persistent connections. Note that IPIC connections are always persistent. If there are suddenly a lot more non-persistent connections than normal, check whether performance tuning temporarily disallowed connection persistence to allow the region to recover from overloading. For more information, see "Performance tuning statistics" on page 250.

As non-persistent connections are by definition single use only, you can use the TCPIPSERVICE statistics fields to calculate how many times persistent connections were reused:

Number of requests processed by existing persistent connections = Requests processed (SOR_REQUESTS) - *Total Connections (SOR_TOTAL_CONNS)*

Note: If performance tuning for HTTP connections is active, CICS periodically closes persistent HTTP connections to allow more efficient sharing of workload across regions that share IP endpoints. This limits the maximum number of requests that can be processed by a connection.

Connection pooling for HTTP client performance

For CICS as an HTTP client, connection pooling can provide performance benefits where multiple invocations of CICS web support applications, web services applications, or the HTTP EP adapter make connection requests for the same host and port, or where a web services application makes multiple requests and responses.

By default, CICS closes a client HTTP connection after the connection has been used:

- For CICS web support applications that make HTTP client requests, CICS closes the connection when the application program issues a **`WEB CLOSE`** command to notify CICS that it has finished using the connection. If the application does not issue the **`WEB CLOSE`** command, CICS closes the connection when end of task is reached.
- For CICS web services applications that are service requesters, CICS closes the connection after the application program has issued its service request using the INVOKE SERVICE command and received the response if appropriate.
- For CICS event processing, CICS closes the connection after the HTTP EP adapter has emitted the business event.

When you set up connection pooling, instead of closing the client HTTP connection after use, CICS keeps the connection open and stores it in a pool in a dormant state. The dormant connection can be reused by the same application or by another application that connects to the same host and port. When an application issues a command to open a client HTTP connection, CICS checks whether a dormant connection is available in the pool for that host and port, and if so supplies it to the application rather than opening a new connection.

When a pooled connection is reused, CPU use is reduced compared with opening a new connection. An additional saving occurs where connections use the Secure Sockets Layer (SSL), because the security measures do not need to be repeated when the connection is reused. Applications reuse a pooled connection in exactly the same way as they use a new connection, and the connection can be pooled again after use.

Connection pooling is specified by the SOCKETCLOSE attribute in a URIMAP resource, which defines if, and for how long, CICS keeps a socket open after the CICS application has finished using its client HTTP connection. Applications must specify the URIMAP resource when they open the connection.

The HTTP EP adapter requires a URIMAP resource to open a connection. URIMAP resources are not a requirement for HTTP client requests from CICS web support and web services applications, but they have a number of advantages. In addition to enabling connection pooling, when you use URIMAP resources for client connections, system administrators can manage any changes to the endpoint of the connection, and you do not need to recompile applications if the URI of a service provider changes. For CICS web support applications, URIMAP resources can also be used to store and administer client certificates and cipher suite codes for SSL.

Connection pooling can provide performance benefits for CICS applications that use client HTTP connections in the following scenarios:

- Multiple CICS web support applications, either invocations of the same application or different applications, make HTTP client requests to the same host and port.
- Multiple CICS web services applications that are service requesters make requests to the same web service provider.
- A CICS web services application that is a service requester makes multiple requests to a web service provider in a single task.
- The HTTP EP adapter sends events frequently to the HTTP server specified in the event binding.

Connection pooling does not enhance the performance of a single invocation of a user-written CICS web support application, or a single web services request and response, or a single event emission.

How CICS manages connection pooling

Connection pooling is transparent to applications, and does not require any involvement from administrators after the SOCKETCLOSE attribute is specified in the URIMAP resource definition. CICS holds a pool of connections for each URIMAP resource that has a nonzero SOCKETCLOSE attribute.

Before placing an HTTP client connection in a pool of connections, CICS checks the state of the connection. Connections are not placed in the pool if they are found or suspected to be in a poor state. For example, the following situations would cause the connection to be in a poor state:

- The last HTTP response from the server was not OK.
- The number of requests and the number of responses on the connection are not equal.
- The connection still has data present on it that the CICS application did not receive.

When an application issues a command to open an HTTP client connection and suitable connections are available in the pool, CICS selects the connection that was placed in the pool most recently. This practice enables older connections to expire in the situation where connection usage has reduced from a previous peak level. When a pooled connection reaches the time limit that you specified in the SOCKETCLOSE attribute without being reused, CICS closes the socket and removes the connection.

CICS also closes dormant connections in a pool in the following situations:

- The web server closes a connection.
- You discard the URIMAP resource associated with the pool of connections.
- MAXSOCKETS is reached for the CICS region, and sockets are required for use by different connections.

The resource statistics for URIMAP definitions, in DFHWBRDS, include statistics for connection pooling as follows:

- The SOCKETCLOSE setting for the URIMAP resource
- The current and peak number of pooled connections in the pool for this URIMAP resource
- The number of dormant connections that CICS reclaimed from the pool because the CICS region had reached the MAXSOCKETS limit
- The number of dormant connections that were not reused and expired while they were in the pool

These statistics are reported in the DFHSTUP and DFH0STAT reports.

The number of times that a connection is reused is reported region-wide in TCP/IP: Global statistics.

How to determine a SOCKETCLOSE value for Outbound Connection Pooling in CICS

Start by specifying a SOCKETCLOSE value of one minute 000100 (hhmmss) or a value of your choosing. Then, make use of your DFHSTUP statistics output or use the STAT transaction online to capture URIMAP Resource statistics. Specifically observing the following four statistics:

- URIMAP Socket pool size
- URIMAP Peak socket pool size
- URIMAP Sockets reclaimed
- URIMAP Sockets timed-out

You should not be too concerned about the "Socket pool size" or "Peak socket pool size" as they are purely influenced by the number of concurrent requests using the specific URIMAP. A high value is only a concern if the specific URIMAP is not expected to be being used very much.

The number of "Sockets reclaimed" is where CICS had no available sockets to service a new request so one of the idle sockets from the pool was closed and reused for the new request. If the number of reclaimed sockets is high then that could indicate the system is processing too much "sockets based work" compared to its MAXSOCKETS setting within the CICS system initialization parameters (DFHSIT). If this is your case you should consider increasing your MAXSOCKETS setting.

The most important value for tuning is the number of "Sockets timed out". If there is a huge number being timed out then this would indicate that one of the following is happening:

- The sockets are not being used very much and are instead remaining idle in the pool until the SOCKETCLOSE value is reached
- The SOCKETCLOSE value is too small and the sockets are getting closed before they have had a chance to get reused

In this case, the value for SOCKETCLOSE should be increased to allow the sockets to get reused. If the number of timeouts is 0 or a relatively small number then that indicates the pool is working well and the sockets are being actively reused.

Setting up connection pooling

If any of the client HTTP connections in your CICS regions are good candidates for connection pooling, you can use CICS resources to set it up.

About this task

For information about how connection pooling can improve performance, see “Connection pooling for HTTP client performance” on page 253.

Procedure

1. Specify the SOCKETCLOSE attribute in the URIMAP resource definitions for the selected client HTTP connections for CICS web support, CICS web services, or CICS event processing. You can choose the length of time for which CICS keeps each pooled connection open before discarding it.

   For information about the SOCKETCLOSE attribute, see URIMAP resources.

2. For CICS web support and web services applications, ensure that CICS applications are specifying the URIMAP resource definitions on the commands to open the client HTTP connections.

   CICS web support applications open a client HTTP connection by using the **WEB OPEN** command. CICS web services applications open a client HTTP connection by using the **INVOKE SERVICE** command, or its synonym **INVOKE WEBSERVICE**. The applications must not specify the URI directly in the command, or for service requesters, use a URI from the web service description.
3. For CICS web support applications only, verify that the application does not use the option `CLOSESTATUS(CLOSE)` on any of the **WEB SEND** or **WEB CONVERSE** commands that it issues. `CLOSESTATUS(CLOSE)` requests the server to close the connection, and closed connections cannot be pooled.
4. For CICS web support applications only, verify that the application is issuing a **WEB CLOSE** command when it finishes using the client HTTP connection.

   CICS does not place connections in the pool if CICS web support applications do not issue the **WEB CLOSE** command, in case the connections are not in a good state. In CICS web services applications, the **INVOKE SERVICE** command completes the use of the connection by the application, so CICS web services applications do not have to issue any additional commands to complete their use of the connection.

Results

CICS keeps the HTTP connection open and stores it in a pool for reuse.

Performance report

You can learn about the performance characteristics of releases of CICS TS from the Performance Report. The report gives a broad understanding of important concepts of CICS performance, and describes many of the performance improvements that can be realized by upgrading CICS. This information was previously published as an IBM Redbook: *IBM CICS Performance Series: CICS TS for z/OS V5 Performance Report*, (SG24-8298).

The information is useful for the following people:

- System architects who want to understand the performance characteristics and capabilities of a release of CICS TS.
- Capacity planners and performance analysts who want to understand how an upgrade to the latest release of CICS TS can affect their environment.
- Application developers who want to design and code highly optimized applications for deployment into a CICS TS environment.

The Performance Report includes the following information:

- Description of factors that are involved in the interaction between IBM Z hardware and a z/OS software environment.
- Definition of key terminology that is used in the descriptions of the results of CICS TS performance benchmarks.
- Presentation of how to collect the required data, and the methodology used, when you apply Large Scale Performance Reference (LSPR) capacity information to a CICS workload in your environment.
- Outline of the techniques that are applied by the IBM CICS Performance Team to achieve consistent and accurate performance benchmark results.
- High-level descriptions of several key workloads that are used to determine the performance characteristics of a CICS TS release.
- Introduction to the open transaction environment and task control block (TCB) management logic in CICS TS, including a reference that describes how several configuration attributes combine to affect the behavior of the CICS TS dispatcher.
- Detailed information about the changes in performance characteristics between successive releases of CICS TS.

- Results of small performance studies from the IBM CICS Performance Team to determine the cost of using a specific CICS TS functional area.

View the Performance Report

Notices

Programming interface information

IBM CICS supplies some documentation that can be considered to be Programming Interfaces, and some documentation that cannot be considered to be a Programming Interface.

Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 5 Release 4 (CICS TS 5.4) are included in the following sections of the online product documentation:

- Developing applications
- Developing system programs
- Securing overview
- Developing for external interfaces
- Application development reference
- Reference: system programming
- Reference: connectivity

Information that is NOT intended to be used as a Programming Interface of CICS TS 5.4, but that might be misconstrued as Programming Interfaces, is included in the following sections of the online product documentation:

- Troubleshooting and support
- Reference: diagnostics

If you access the CICS documentation in manuals in PDF format, Programming Interfaces that allow the customer to write programs to obtain the services of CICS TS 5.4 are included in the following manuals:

- Application Programming Guide and Application Programming Reference
- Business Transaction Services

- Customization Guide
- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- Installation Guide
- Security Guide
- CICS Transactions
- CICSPlex System Manager (CICSPlex SM) Managing Workloads
- CICSPlex SM Managing Resource Usage
- CICSPlex SM Application Programming Guide and Application Programming Reference
- Java Applications in CICS

If you access the CICS documentation in manuals in PDF format, information that is NOT intended to be used as a Programming Interface of CICS TS 5.4, but that might be misconstrued as Programming Interfaces, is included in the following manuals:

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSPlex SM Problem Determination Guide

Trademarks

UNIX is a registered trademark of The Open Group in the United States and other countries.

Zowe™, the Zowe logo and the Open Mainframe Project™ are trademarks of The Linux Foundation.

The Stack Exchange name and logos are trademarks of Stack Exchange Inc.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability
These terms and conditions are in addition to any terms of use for the IBM website.

Personal use
You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use
You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights
Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM online privacy statement

IBM Software products, including software as a service solutions, (*Software Offerings*) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information (PII) is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect PII. If this Software Offering uses cookies to collect PII, specific information about this offering's use of cookies is set forth below:

For the CICSPlex SM Web User Interface (main interface):
Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name and other PII for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface (data interface):
Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other PII for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface ("hello world" page):
Depending upon the configurations deployed, this Software Offering may use session cookies that do not collect PII. These cookies cannot be disabled.

For CICS Explorer:

Depending upon the configurations deployed, this Software Offering may use session and persistent preferences that collect each user’s user name and password, for purposes of session management, authentication, and single sign-on configuration. These preferences cannot be disabled, although storing a user's password on disk in encrypted form can only be enabled by the user's explicit action to check a check box during sign-on.

If the configurations deployed for this Software Offering provide you, as customer, the ability to collect PII from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM Privacy Policy and IBM Online Privacy Statement, the section entitled *Cookies, Web Beacons and Other Technologies* and the IBM Software Products and Software-as-a-Service Privacy Statement.

Index