z/OS
Version 2 Release 3

DFSORT Tuning Guide

SC23-6882-30

Note

Before using this information and the product it supports, read the information in “Notices” on page 85.

This edition applies to Version 2 Release 3 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2019-02-16

Contents

List of Figures

List of Tables

About this document

Sorting is one of the most frequently used functions at most data processing sites.

This document provides information about tuning IBM DFSORT, offering suggestions for reducing its use of system resources and achieving better turnaround time for the many applications that use DFSORT without adversely affecting system performance.

This document is intended for systems engineers, performance analysts, system programmers, and application programmers, who have some experience with DFSORT. For those unfamiliar with DFSORT, *z/OS DFSORT Installation and Customization* and *z/OS DFSORT Application Programming Guide* provide information that will help you use this document effectively.

In this document, one gigabyte (GB) is equal to 1024 megabytes (MB), which is equal to 1048576 kilobytes (KB), which is equal to 1073741824 bytes.

How to use this document

This document contains the following sections:

- Chapter 1, "Introduction ," on page 1, describes the importance of tuning DFSORT, an example of successful tuning, system resources and tuning of DFSORT, and the performance indicators for DFSORT.
- Chapter 2, "DFSORT performance features ," on page 5, describes the performance features of DFSORT.
- Chapter 3, "Environment considerations ," on page 13, describes the relationship between DFSORT and the environment in which it runs including the storage hierarchy, virtual storage, and system-managed storage.
- Chapter 4, "Installation considerations," on page 19, describes how installation options, run-time options, DFSORT capabilities, installation defaults, site-wide options, and installation exits affect the performance of DFSORT applications.
- Chapter 5, "Run-time considerations," on page 35, describes DFSORT features and how to use them to improve the run-time performance of DFSORT.
- Chapter 6, "Application considerations," on page 51, describes how new and existing applications can make efficient and effective use of the DFSORT facilities.
- Chapter 7, "DFSORT performance data ," on page 61, describes the actions you can take to tune DFSORT, the type and location of information you need to tune DFSORT, and the methods you can use to collect the information.
- Appendix A, "Sample ICETEXIT," on page 75, includes sample source code for an ICETEXIT routine which creates a summary performance record each time DFSORT is used, and a SVC to write the resulting user records to SMF.

Required product knowledge

To use this document effectively, you should be familiar with the following information:

- Job control language (JCL)
- DFSORT and ICETOOL control statement syntax
- Data management
- Tape and disk hardware
- System Management Facilities (SMF)
- Resource Measurement Facility (RMF™)

You should also be familiar with the information presented in the following related documents:

Table 1: Related documents.
Documentation Title
z/OS DFSORT Application Programming Guide
z/OS DFSORT Installation and Customization
z/OS DFSORT Messages, Codes and Diagnosis Guide
z/OS MVS JCL Reference
z/OS MVS JCL User's Guide
z/OS DFSMS Using Data Sets
z/OS DFSMS Using Magnetic Tapes
z/OS MVS System Management Facilities (SMF)
z/OS Language Environment Programming Guide

Referenced documents

This document refers to the following documents:

Table 2: Referenced documentation.
Documentation
z/OS RMF Report Analysis
z/OS MVS Initialization and Tuning Reference
z/OS MVS Initialization and Tuning Guide
z/OS DFSMS Installation Exits

z/OS DFSORT Tuning Guide is a part of a more extensive DFSORT library. These documents can help you work with DFSORT more effectively.

Task	Documentation
Planning for and customizing DFSORT	*z/OS DFSORT Installation and Customization*
Learning about DFSORT	*z/OS DFSORT: Getting Started*
Application programming	*z/OS DFSORT Application Programming Guide*
Interpreting messages and codes, and diagnosing failures	*z/OS DFSORT Messages, Codes and Diagnosis Guide*

z/OS information

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

To find the complete z/OS® library, go to IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

Performance comparisons

The performance comparisons shown in this document are derived from averaging numbers from three runs of each component of the comparison in a production environment. The applications used are not meant to be representative of any particular user's environment. The performance comparisons shown are examples of the effects of various tuning techniques in particular situations.

Unless otherwise stated, applications were run:

- On an IBM 2084 CPU with 24 GB of central storage in a production z/OS environment. All of the input, output, and work data sets resided on IBM RAID 3390-3 or 3390-9 emulated disk connected to IBM 2105 or 3990-6 control units.
- With z/OS DFSORT V1R5 using the IBM-supplied installation defaults.
- Using 1500 MB input data sets with RECFM=FB, LRECL=1500, BLKSIZE=27000 and 20-byte randomly generated keys.

  Exceptions: For the comparisons shown in Figure 8 on page 47, variations of this data set with different blocksizes were used. For the COBOL comparisons in Chapter 6, a 150 MB input data set with RECFM=FB, LRECL=160, and BLKSIZE=27840 was used.

The actual performance of a particular sort application is dependent on many factors including record length, data set size, region size, total storage available, type and number of auxiliary storage devices, and specific functions and exits used.

CPU time represents the sum of the following five fields: TCB, SRB, RCT, HPT, and IIP. See “Processor utilization ” on page 3 for a description of these fields.

Elapsed time results for sorting in a multi-tasking environment are application profile and workload dependent. Therefore, the results might differ from user to user.

IBM does not represent nor warrant that your applications will achieve the same performance data as the examples in this document.

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xv.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Knowledge Center function
If your comment or question is about the IBM Knowledge Center functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Knowledge Center Support at ibmkc@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS DFSORT Tuning Guide, SC23-6882-30
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Chapter 1. Introduction

This document offers information and recommendations on tuning DFSORT. It provides advice to the system programmer on setting DFSORT's installation default parameters as well as to the application programmer on improving the performance of individual DFSORT applications. It describes the main indicators used to measure performance and emphasizes the advantages of using DFSORT's Blockset technique for improved performance. Since sort applications tend to be more complex and time-consuming than merge or copy applications, this document concentrates on techniques for improving the performance of sort applications. However, many of these techniques are also appropriate for copy and merge applications.

This document also assumes that DFSORT's most efficient technique is used. See "Blockset technique" on page 5 for more information on ensuring that Blockset is used.

Note: Although a DFSORT "application" can comprise one or more parts of a job, for simplicity's sake the terms "application" and "job" are used interchangeably in this document.

This chapter describes the following:

- The importance of tuning DFSORT
- Examples of successful tuning of DFSORT
- System resources and tuning DFSORT
- The performance indicators for DFSORT

DFSORT on the web

For articles, online documents, news, tips, techniques, examples, and more, visit the DFSORT Home Page (www.ibm.com/storage/dfsort).

DFSORT FTP site

You can obtain DFSORT articles and examples using anonymous FTP to:

```
ftp://ftp.software.ibm.com/storage/dfsort/mvs/ (ftp://ftp.software.ibm.com/storage/dfsort/mvs/)
```

The importance of tuning

If you are unsure whether your site can benefit from the advice in this document, consider the following questions:

- Do you know how frequently DFSORT is invoked at your site?
- Is there growing pressure to reduce your batch window?
- Are you confident that you and your users are using DFSORT efficiently?
- If better tuning could result in significant savings in elapsed time, CPU time, device connect time, or EXCP counts, would it be worthwhile?

The tuning of DFSORT and the way applications use it is important because sorting and copying are two of the most frequently used functions at z/OS sites. DFSORT applications typically consume from 10 to 25 percent of the total processor resources and 15 to 30 percent of the total I/O channel resources.

More efficient use of DFSORT can lead to:

- Reduced use of system resources

- Reduced job turnaround time
- Improved productivity

All of these benefits can result in cost savings.

Because of the internal optimizations that DFSORT performs (for instance, selecting the best work data set block size for the run), users often do not take the time to tune DFSORT or their applications. Experience shows that many sites only begin to focus on DFSORT performance when there is a crisis: perhaps as data volumes increase, or the batch window becomes smaller. While this document offers help for these situations, it is also intended to help avoid them.

Successful tuning

Some of the practical advice in this document is based on the experience of IBM developers working with DFSORT customers to help them in specific situations.

This document offers many recommendations, ranging from advice on setting the most appropriate option values to guidelines for optimizing virtual storage and work space. You should decide which recommendations are best for your site, based on site requirements and the amount of resources available to implement them.

System resources

The purpose of tuning DFSORT is to use system resources more efficiently. This is important at most sites, since there are usually too many demands made for limited resources. Although DFSORT automatically optimizes many of its tuning decisions, there are additional actions which a site and its DFSORT users can take to further improve DFSORT performance. These actions depend on the priority given to various performance objectives.

Different sites and programmers define *efficient performance* in different ways. A site with a primarily batch environment or an application programmer with a single task to complete probably measures performance based on elapsed time. Alternatively, a site experiencing high CPU usage or a programmer who is charged based on CPU time is more likely to evaluate efficiency in terms of reduced CPU time.

While improved performance is the objective of tuning, often it is necessary to compromise. That is, improving the use of one system resource can have a negative impact on other resources, in much the same way that giving more resources to one application can make it perform much better at the expense of degrading the performance of other applications that are competing for the same resources.

The main trade-offs that you should consider are among:

Processor Load
Accounting charges are often based on the number of CPU service units used by a job or address space. The more CPU time used, the higher the charges. Reducing a job's CPU time not only reduces these charges, but also enables other jobs competing for the same CPU resource to complete sooner.

Paging Activity
System paging activity reflects how the system is managing central storage to meet the virtual storage requirements of applications and user programs. *Paging* is the process of moving virtual storage pages from central storage to auxiliary storage, and takes a large amount of elapsed time to perform.

System paging activity can increase elapsed time for user programs. If the activity is too high, the system reduces its workload by reducing the number of jobs running, and might suspend processing of some jobs (known as *swapping*) until the paging activity returns to an acceptable level. The result is that the system spends more time managing virtual storage, while many user programs take longer to complete.

I/O Activity
Some accounting charges are based on the I/O performed by a job. This is frequently measured by execute channel program (EXCP) counts. While EXCP counts might not represent a completely

accurate usage of I/O resources, they are important to many users and sites, and steps can be taken to reduce them.

Elapsed Time
Elapsed time is likely to be most important for sites with a limited batch window, where processing of particular applications has to be completed within a certain period. It is also important to users whose productivity depends on having their applications complete as soon as possible.

Disk Utilization
In environments where disk space is constrained, the amount of auxiliary storage required by an application is a primary concern. For DFSORT applications, this usually involves the efficient use of output and work data sets.

Performance indicators

This section describes how you can measure the performance factors listed previously for DFSORT. Often, performance is evaluated as a combination of some or all of them. The priorities given to each depend upon site objectives.

Processor utilization

CPU fields are used to measure the amount of work performed by the processor, as opposed to work performed by the I/O and storage subsystems. CPU time consists of the sum of the following five components.

Task Control Block (TCB)
The CPU time used to perform user program activity on behalf of user tasks for a job step.

Service Request Block (SRB)
The CPU time used to perform system service requests on behalf of user tasks for a job step.

Hiperspace™ Processing Time (HPT)
If the user task reads from or writes to a Hiperspace using the HSPSERV macro, this is the amount of CPU time used to service these reads and writes.

I/O Interrupt Processing (IIP)
The CPU time used to handle input/output (I/O) interrupts on behalf of user tasks for a job step.

Region Control Task (RCT)
If the user task is swapped out while running, this is the amount of CPU time used to swap the task out and back in again.

Central storage and system paging

Central storage is a critical resource exploited by DFSORT to reduce I/O and improve performance. DFSORT exploits available storage to create memory objects, Hiperspaces and data spaces. Overcommitment of these resources can lead to excessive paging that negatively impacts the performance of DFSORT and other workloads running on the same system.

To be meaningful, paging and swapping activity measurements are usually associated with particular workloads, or groups of applications that are run simultaneously on a given processor. As such, system paging activity is more a measure of the performance and throughput of the entire system than of the performance of individual applications being run on the system.

Paging activity is often measured by the page-in, page-out, and page-reclaim rates of different address space types, such as Hiperspace, VIO, or non-VIO address spaces. For swapping activity, there is also the additional distinction between logical and physical swaps.

Central storage resource statistics and system paging and swapping activity data are usually gathered by the Resource Measurement Facility (RMF) or a similar system tool. See *z/OS RMF Report Analysis* for more information on RMF.

I/O activity

This represents the movement of data between the processor and disk or tape devices. The effective use of I/O resources is important to a site and I/O activity is often an important component of site accounting methodologies.

Because performing input to and output from disk and tape devices typically takes much longer than manipulating the data in real storage, the amount of I/O performed is a key component of an application's Elapsed time. Therefore, reducing I/O generally improves an application's Elapsed time.

I/O activity is primarily measured in the following ways:

EXCPs
The number of execute channel program (EXCP) commands issued (logical I/Os)

SSCHs
The number of start subchannel (SSCH) commands issued (physical I/Os)

Device Connect Time
The amount of time a particular device is dedicated for the I/O transfer

Channel Usage
The percentage of time a channel is busy initiating, transferring, or completing the movement of data between a device and the CPU

While EXCPs are often used as a measure of I/O activity, their counts can be extremely misleading. For example, one EXCP can be used to transfer a few bytes or dozens of megabytes! SSCHs measure the number of physical I/Os to a data set and thus can be more useful than EXCPs. A complete analysis of total I/O performance should consider device connect time, channel usage and SSCHs.

Elapsed time

Elapsed time refers to the amount of “wall clock” time from initiation to termination of the application. For typical sorting applications, elapsed time is composed primarily of I/O time, with CPU time and I/O queueing time also contributing significantly.

The Elapsed time for an application can differ from run to run, depending upon the amount of competition from other applications for system resources. Accurate Elapsed time comparisons can be done only if the system has no other applications running.

Disk utilization

In certain environments, disk space usage is a more important characteristic of an application's performance than CPU time or elapsed time. Inefficient disk usage is usually measured in terms of the amount of disk space that is allocated, compared to the amount of disk space actually needed. Frequently, large amounts of disk space are wasted as a result of inefficient blocking

Chapter 2. DFSORT performance features

The performance of a DFSORT application is largely determined by the use of a special set of product features. This chapter provides an introduction to the performance features of DFSORT including:

- Blockset technique
- OUTFIL
- Memory object sorting, Hipersorting and dataspace sorting
- Dynamic Storage Adjustment
- Cache fast write
- ICEGENER
- Compression
- Striping
- Dynamic allocation of work data sets
- System-determined block size
- Larger tape blocksizes (greater than 32K)
- Managed tape data sets
- IDCAMS BLDINDEX
- DFSORT's Performance Booster for The SAS System

How to use these features to gain the most effective performance from DFSORT is described in Chapter 4, “Installation considerations,” on page 19 and in Chapter 5, “Run-time considerations,” on page 35.

Blockset technique

The Blockset technique is DFSORT's most efficient method for sorting, merging, and copying. Blockset uses optimized algorithms and makes the most efficient use of IBM hardware. DFSORT uses Blockset whenever possible. DFSORT's other techniques, Peerage/Vale and Conventional, are not as efficient as Blockset and are only used when Blockset cannot be used.

The Blockset technique can reduce CPU time, I/O activity, and Elapsed time. It is strongly recommended that you remove any obstacles to using Blockset whenever possible. For the purposes of this document, any recommendations to improve performance assume that Blockset is the DFSORT technique used.

It is worth checking to see if Blockset was used for a given application; message ICE143I in the SYSOUT data set shows which technique was used. If ICE143I is not shown or shows that a technique other than Blockset was selected, resubmit the application with a SORTDIAG DD statement (unnecessary if the application already had a SORTDIAG DD statement or if your site has specified installation option DIAGSIM=YES). Additional DFSORT messages and diagnostic information are then shown, including:

- ICE802I, which shows the technique used
- ICE800I or error messages

ICE800I gives a code indicating why Blockset was not used. If the code is 1, one or more error messages are also shown.

Note: Blockset messages are suppressed if another technique is selected, unless a SORTDIAG DD statement is present, or installation option DIAGSIM=YES has been specified for your site.

Blockset will **not** be selected if you specify work data sets on tape devices. Blockset only supports tape devices for input and output data sets. It is very strongly recommended that you use disk rather than tape for work data sets.

OUTFIL

OUTFIL is a control statement that allows you to create one or more output data sets for a sort, copy, or merge application with only one pass over an input data set. You can use multiple OUTFIL statements, with each statement specifying the OUTFIL processing to be performed for one or more output data sets.

With a single pass over an input data set, OUTFIL can create one or more:

- Output data sets containing unedited or edited records
- Output data sets containing different ranges, samples, or subsets of records. Those records that are not selected for any subset can be saved in another output data set.
- Reports containing different header records, detail records and trailer records.
- FB output data sets from a VB input data set.
- VB output data sets from an FB input data set.

OUTFIL has many additional features as well; see *z/OS DFSORT Application Programming Guide* for complete details on all of the tasks you can perform with OUTFIL.

Benefits

Because OUTFIL allows you to create output data sets with only one pass over an input data set, OUTFIL can improve performance. This is especially true for elapsed time and EXCPs when you compare an OUTFIL job to create many output data sets with many non-OUTFIL jobs each creating one output data set.

Memory object sorting, hipersorting and data space sorting

DFSORT can use available central storage to improve elapsed time and reduce I/O to disk work data sets for sort applications. DFSORT evaluates the characteristics of each sort along with available resources to determine the best technique for providing optimal performance.

A memory object is a data area in virtual storage that is allocated above the bar and backed by central storage. With memory object sorting, memory objects can be used either as intermediate work space or as additional main storage. Memory objects can be used exclusively, or along with disk, for temporary storage of records during a sort.

Hipersorting is DFSORT's ability to use Hiperspaces as intermediate work space for sorting. Hipersorting was originally designed to exploit expanded storage in 31-bit architecture. With 64-bit architecture, Hipersorting now effectively exploits central storage instead. DFSORT's use of Hiperspaces for intermediate work space, alone or in combination with work data sets, is called Hipersorting.

A data space is a data area in virtual storage that is allocated below the bar and backed by central storage. Programs that access data in a data space run in access register address space control (AR ASC) mode. With data space sorting, data spaces can be used as additional main storage. Data spaces can be used exclusively or along with disk, for temporary storage of records during a sort.

Benefits

Memory objects and Hiperspaces used as intermediate work space

When virtual storage is smaller than the input data set to be sorted, DFSORT uses temporary intermediate storage to perform the sort. This intermediate storage can be of three types; it can be memory objects, Hiperspaces, or disk work data sets. Understanding how all programs use central storage can help you see the benefits of using memory objects or Hiperspaces as work space.

- Memory Object or Hiperspace Storage used as work space

  A program's request for data in a memory object or Hiperspace work file results in a synchronous or other high-speed transfer in central storage. Upon completion, the program continues.

- Disk Storage used as work space

  If the request is for data to or from disk, an asynchronous I/O operation is started, the program interrupted, and control returned to the dispatcher to dispatch another program. When the I/O completes, the program waits to be dispatched again, since it is now able to process its data.

A memory object or Hiperspace read or write operation involves the transfer of data within central storage, whereas a disk I/O operation involves the transfer of data between disk and central storage. Since data movement in central storage is faster than from disk, elapsed time is likely to improve. Note that disk volumes connected to cached controllers can provide significant, but smaller, elapsed time improvements compared to central storage. The use of memory object or Hiperspace work space eliminates the need for work data set I/O, therefore eliminating EXCPs and channel usage for the work devices.

Memory objects and data spaces used as additional main storage

A memory object or data space can be used as a large contiguous area of virtual storage for sorting records. The benefit of using central storage in this way is twofold:

- Due to the potentially large size of a memory object or data space, this type of sorting enables a greater percentage of sort jobs to be processed completely within main storage, without the need to write intermediate data to disk work space. This can reduce CPU and elapsed times, as well as EXCP counts and channel usage.
- If an in-main-storage sort is not possible, memory object or dataspace sorting usually picks the optimal amount of virtual storage for memory object or data space. This is frequently larger than the default amount of main storage and enables DFSORT to sort larger amounts of data at a time before writing them to the work data sets. This often provides a savings in elapsed time, I/O activity, and CPU time.

Operation

In addition to DFSORT, central storage can be used by many other applications and system components, such as DB2® buffer pools, VSAM Hiperspace, Hiperbatch, virtual lookaside facility (VLF), VIO, and the paging subsystem. To avoid overusing storage, which can lead to paging data set space shortages and increased system paging activity, DFSORT uses several safeguards.

To begin with, DFSORT is always aware of the future storage needs of other concurrent DFSORT applications. A DFSORT application never attempts to back its memory object, Hiperspace or data space data with storage that is needed by another DFSORT application.

Next, DFSORT dynamically determines the amount of available storage prior to creating each memory object, Hiperspace or data space. "Available" storage is the storage used to back new Memory Object, Hiperspace or data space data. It consists of two types:

- Free storage is storage that is not being used by any application.
- Old storage is storage that is being used by other applications, but whose data has been unreferenced for a long period of time. This time period is long enough that the system deems it eligible for migration to auxiliary storage to make room for new data.

Knowing both the amount of storage needed by other DFSORT applications and the amount of available storage, along with the values of DFSORT installation options EXPMAX, EXPOLD, EXPRES and TUNE, DFSORT can assess the impact of creating a memory object, Hiperspace or data space. This greatly reduces the likelihood of overusing central storage, especially as a result of multiple large concurrent DFSORT applications. It also allows sites to customize their total use of storage by DFSORT applications through the EXPMAX, EXPOLD, EXPRES and TUNE installation options. These can only be specified as installation wide defaults using an ICEPRMxx member of PARMLIB or the ICEMAC macro.

MOSIZE, HIPRMAX and DSPSIZE control the amount of memory object, Hiperspace and data space each individual DFSORT application can use while EXPMAX, EXPOLD, and EXPRES control the total amount of storage used by all DFSORT applications running in the system. These can be specified as installation wide defaults using an ICEPRMxx member of PARMLIB or the ICEMAC macro, overridden at run-time with an EXEC PARM option or OPTION control statement, or overridden with an installation wide ICEIEXIT exit routine.

The MOWRK installation default controls whether memory objects can be used as intermediate work space. If MOWRK=NO is in effect, then memory objects can only be used as additional main storage. MOWRK can be specified as an installation wide default using an ICEPRMxx member of PARMLIB or the ICEMAC macro, or overridden at run-time with an EXEC PARM option or OPTION control statement. MOWRK cannot be overridden with an ICEIEXIT exit routine.

When memory objects or data spaces are used as additional main storage, each sort creates one large area at the beginning and no adjustments are made until the memory object or data space is deleted when the sort terminates.

When memory objects or Hiperspaces are used as intermediate work space, DFSORT has the ability to create up to 16 memory objects or Hiperspaces which can be allocated incrementally or all at once. The advantage of allocating storage in increments is that multiple concurrent sorts can more evenly share available central storage resources since available central storage is evaluated prior to creating each new memory object or Hiperspace. When storage is allocated in this way, DFSORT's dynamic allocation of work data sets must allocate additional space to safeguard against any unexpected increase in disk work space requirements if central storage resources become constrained during the sort. When each sort allocates all of it's memory object or Hiperspace immediately, DFSORT's dynamic allocation can be more aggressive in reducing space allocations since it is less likely that the disk work space allocations will be larger than expected. However, when multiple sorts are running concurrently, a small number of them may monopolize the available central storage while others are forced to use only disk work space.

DFSORT provides the TUNE installation default to control how DFSORT evaluates available resources and allocates memory objects, Hiperspaces and data spaces. If balancing storage usage by multiple concurrent sorts is a priority, the shipped default of TUNE=STOR is recommended. If conserving disk work space is a priority, TUNE=DISK is recommended. TUNE can be specified as an installation wide default using an ICEPRMxx member of PARMLIB or the ICEMAC macro. TUNE cannot be overridden with an ICEIEXIT exit routine.

The following are actions you can take which might increase DFSORT's use of memory objects, Hiperspaces and data spaces:

- Ensure that MOSIZE=MAX, MOWRK=YES, HIPRMAX=OPTIMAL and DSPSIZE=MAX are used. These parameters can be specified as installation-wide defaults using an ICEPRMxx member of PARMLIB or the ICEMAC macro, or overridden at run-time with an EXEC PARM option or OPTION control statement. MOSIZE, HIPRMAX and DSPSIZE can also be overridden with an installation-wide ICEIEXIT exit routine.
- Verify that a sufficient size for memory objects is defined by the MEMLIMIT parameter on the JOB or EXEC JCL statement. Note that if REGION=0M is specified, MEMLIMIT=NOLIMIT is assumed. If REGION is set equal to a value other than 0M and MEMLIMIT is not specified, the default MEMLIMIT value defined in the SMFPRMxx member of PARMLIB is used.
- Verify that IEFUSI does not place any restrictions on the size of the memory objects, Hiperspaces or data spaces a single address space can create.
- Ensure that DFSORT has accurate information about the input file size. DFSORT can automatically estimate the file size for disk input data sets, and tape data sets managed by DFSMSrmm or a tape management system that uses ICETPEX. See “File Size and Dynamic Allocation” in *z/OS DFSORT Application Programming Guide* for information on situations where DFSORT cannot determine the file size accurately, and what to do about it.
- Verify that there is sufficient available central storage present to back DFSORT's memory objects, Hiperspaces and data spaces.

See Chapter 4, “Installation considerations,” on page 19 and Chapter 5, “Run-time considerations,” on page 35 for further details on how best to tune DFSORT's use of memory objects, Hiperspaces and data spaces.

Dynamic storage adjustment

Dynamic Storage Adjustment (DSA) is DFSORT's ability to automatically increase the amount of virtual storage committed to a sort when doing so should significantly improve performance. Like dataspace sorting, DSA lets DFSORT tune the right amount of virtual storage for a sort application.

Benefits

Increasing the amount of virtual storage available to DFSORT can significantly improve the performance of many sort applications, especially large sorts. The optimal amount of virtual storage varies with the amount of data being sorted. Consequently, the best performance and most efficient use of virtual storage can only be obtained by tuning the virtual storage of each individual sort. DSA does most of this tuning automatically, relieving system and application programmers of the task. By using the optimal amount of virtual storage, DFSORT maximizes its performance while minimizing the impact of DFSORT applications on the system.

Operation

DFSORT's use of virtual storage from the primary address space is limited by several factors, most notably system (for example, REGION, IEFUSI) and DFSORT (for example, TMAXLIM, SIZE) parameters, defaults, and exits. While DFSORT can never exceed the system limits, those limits usually exceed the virtual storage DFSORT needs for most applications. As a result, the DFSORT limits usually control the amount of virtual storage used by DFSORT.

Dynamic Storage Adjustment permits a range to be specified for the default amount of virtual storage for sorts, provided SIZE/MAINSIZE=MAX is in effect. This range starts with the value specified for TMAXLIM and goes up to the value specified for DSA. For most sorts, DFSORT limits its use of virtual storage to TMAXLIM. However, for larger sorts, DFSORT automatically optimizes performance by increasing the amount of virtual storage it uses, up to but not exceeding the DSA value.

See Chapter 4, “Installation considerations,” on page 19 and Chapter 5, “Run-time considerations,” on page 35 for further details on the best way to tune DFSORT's use of DSA.

Cache Fast Write (CFW)

The term cache fast write (CFW) is normally used to refer to the capability of the cached 3990 storage control units to use cache memory to improve average I/O times. In this document, cache fast write is also used to refer to DFSORT's ability to take advantage of the storage control unit's cache fast write function, by writing data sets to and reading data sets from cache only.

Benefits

You can gain elapsed time savings by using cache fast write. The benefits of CFW are twofold:

- When writing data to the work data sets, the write operations can complete at cache speed rather than at disk speed, as long as overall cache activity permits it. If the cache usage is very heavy, then there may be very little or no benefit to using CFW, since the majority of write operations will be immediately directed to disk.
- When reading data back from the work data sets, the read operations can complete at cache speed rather than at disk speed, as long as the data to be read still resides in cache. If the required data does not reside in cache, a disk access is required and no performance benefit results. The higher the cache activity, the less the performance improvement with CFW, because it is more likely that written data will be destaged to disk before it can be read back.

Operation

DFSORT's use of cache fast write for the work data sets can be controlled with installation option CFW, and run-time option CFW or NOCFW on the DEBUG statement. Note that cache fast write can only be used

if the DFSORT SVC has been installed. Also note that some control units override the CFW setting and decide when and when not to cache the data.

ICEGENER

DFSORT's ICEGENER facility allows you to more efficiently process IEBGENER jobs. Qualifying IEBGENER jobs are processed by the equivalent, but more efficient, DFSORT copy function. If the DFSORT copy function cannot be used (for example, IEBGENER control statements are specified), control is automatically transferred to the IEBGENER program.

If your site has installed ICEGENER as an automatic replacement for IEBGENER, you need not make any changes to your jobs to use ICEGENER. If your site has not chosen automatic use of ICEGENER, you can use ICEGENER by substituting the name ICEGENER for IEBGENER on any EXEC statement or LINK macro call.

The use of ICEGENER rather than IEBGENER for copy applications can result in significant savings in elapsed time, CPU time, and EXCP counts. Figure 1 on page 10 shows an example of the tremendous performance advantages of ICEGENER over IEBGENER.

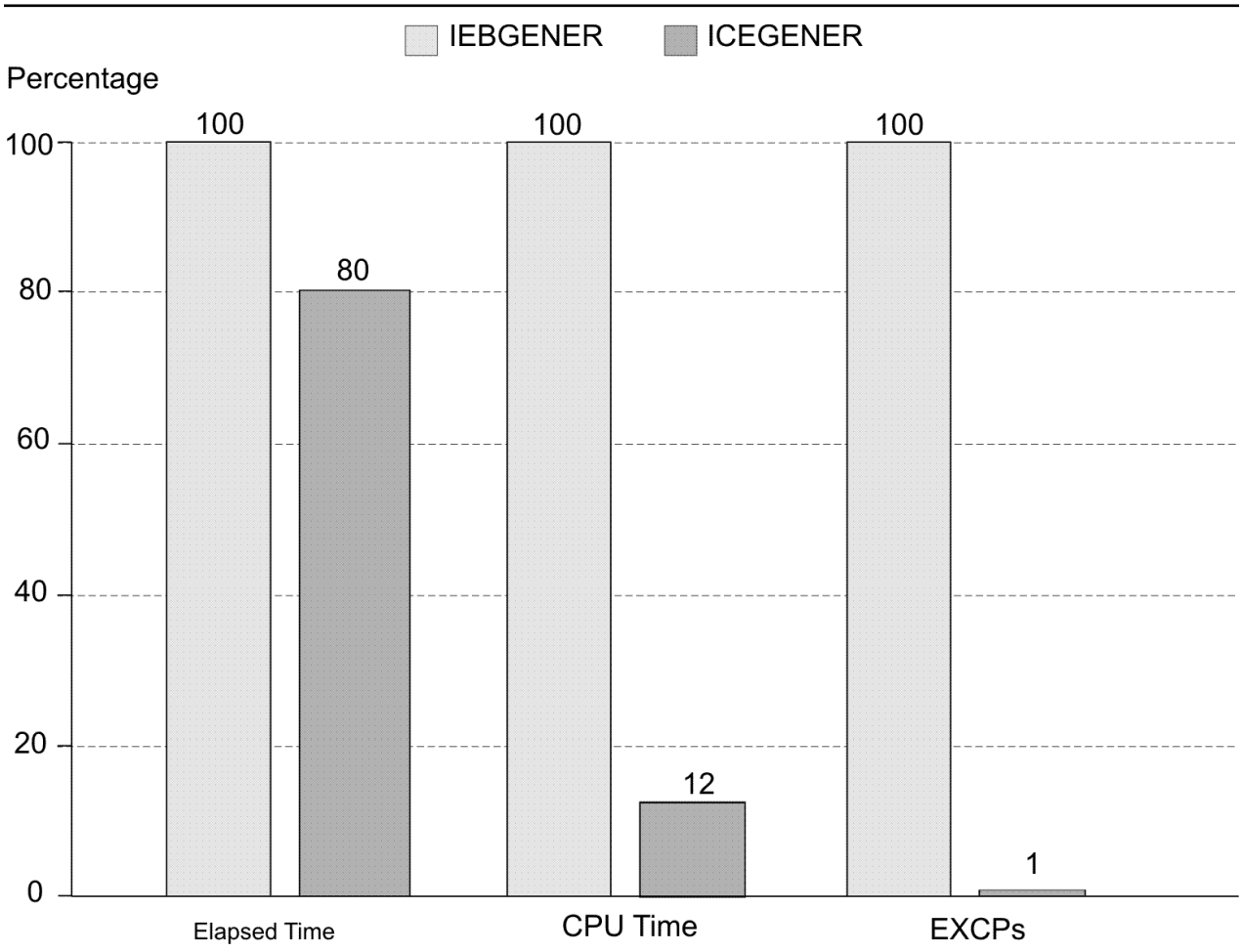

Figure 1: ICEGENER versus IEBGENER Copy Comparison

Extended format datasets

Extended format data sets provide features that can significantly reduce the elapsed time DFSORT spends reading and writing data.

The Sequential Data Striping feature can reduce the time required to read and write your DFSORT input and output data sets. We recommend using sequential data striping for your very large DFSORT input and output data sets as a way to improve elapsed time and performance.

Extended format data sets also provide functions to assist in managing the space requirements of very large data sets. These features include compression, extended addressability (for VSAM), increased extent and volume support, and space constraint relief options. Consult with your storage administrator to learn how these features can be exploited in your environment.

Dynamic allocation of work data sets

When a sort application cannot be performed entirely in virtual storage, DFSORT must use work space. Dynamic allocation of work data sets makes more efficient use of this work space.

You set the DYNAUTO installation option to control whether dynamic allocation is used automatically, or only when requested by the DYNALLOC run-time option. DYNAUTO also controls whether dynamic allocation or JCL allocation takes precedence when JCL work data sets are specified.

DYNAUTO can be set as follows:

- If DYNAUTO=IGNWKDD, dynamic allocation takes precedence over JCL allocation. If you want the opposite result for selected applications, use the USEWKDD run-time option.
- If DYNAUTO=YES, JCL allocation takes precedence over dynamic allocation. If you want the opposite result, remove all JCL allocation statements.
- If DYNAUTO=NO, dynamic allocation of work data sets is not used unless you specify the DYNALLOC run-time option. JCL allocation takes precedence over dynamic allocation.

Dynamic allocation of work data sets has the following advantages:

- As the characteristics (for example, file size and virtual storage size) of an application change over time, DFSORT can automatically optimize the amount of dynamically allocated work space for the application. This eliminates unneeded allocation of disk space.
- As the amount of Hiperspace available to the application varies from run to run, DFSORT can automatically adjust the amount of space it dynamically allocates to complement the amount of Hiperspace. This eliminates unneeded allocation of disk space.
- The amount of work space actually used is often less than the amount allocated. DFSORT tries to minimize dynamic over-allocation while making certain that the application does not fail due to lack of space. With JCL allocation, you could minimize the amount of allocated space manually, but this might require changes to JCL allocation as the characteristics of the application change.
- Dynamically allocated work data sets are automatically allocated as large format so they can use more than 65535 tracks on a single volume.
- When disk work space requirements are larger than expected, DFSORT can automatically recover by increasing allocation sizes or using additional work data sets.

System-Determined Block Size (SDB)

Use the system-determined block size (SDB) facility whenever possible to allow the system to select optimal block sizes for your output data sets. SDB provides better use of output disk space and improved elapsed time. DFSORT's use of SDB can be controlled using installation options SDB and SDBMSG.

Larger tape block sizes (greater than 32K)

With z/OS, block sizes greater than 32760 bytes can be used for tape data sets. A larger tape block size can improve elapsed time and tape utilization. The larger block sizes allow the tape device to perform better because of the decreased need to start and stop reading blocks. DFSORT's SDB=LARGE or SDB=INPUT option allows DFSORT to select a block size greater than 32760 for a tape output data set. However, you must insure that subsequent applications that use such data sets can handle larger block sizes.

Managed tape data sets

The use of tapes managed by DFSMSrmm, or a tape management system that uses ICETPEX, allows DFSORT to obtain accurate information about input file size and data set characteristics. This results in improved performance and more efficient use of both main storage and intermediate work storage.

IDCAMS BLDINDEX

DFSORT provides support that enables IDCAMS BLDINDEX to automatically use DFSORT to improve the performance of most BLDINDEX jobs that require BLDINDEX external sorting.

DFSORT's performance booster for the SAS system

DFSORT provides significant CPU time improvements for The SAS System applications. To take advantage of this feature, contact SAS Institute Inc. for details of the support they provide to enable this enhancement.

Chapter 3. Environment considerations

While DFSORT automatically optimizes many of its tuning decisions to improve performance, the environment in which it runs affects its overall performance and, therefore, the programs and applications that use DFSORT. DFSORT is designed to take advantage of the major components of the z/OS environment.

The purpose of this chapter is to describe how DFSORT modifies its operation to take advantage of these components and the benefits it derives from them.

The majority of information in this chapter applies only to sorting applications. While copying and merging are also commonly used features of DFSORT, it is sorting that uses the most resources and is the most important to tune.

This chapter includes the following information:

- Storage hierarchy
- Virtual storage
- System-managed storage

Storage hierarchy

Within a z/OS environment, data resides across a storage hierarchy. Each level of this hierarchy is represented in hardware by a different component, and the amount of each component in a system is usually variable between some minimum and maximum levels. A typical representation of these components (from top to bottom) would be:

1. Processor cache
2. Central storage
3. Storage control cache
4. Disk
5. Tape

The components at the top of the hierarchy (processor cache, central storage) are somewhat expensive on a per-byte basis (and thus relatively small in capacity), but have very fast access times. In contrast, the components at the bottom of the hierarchy (disk, tape) are relatively inexpensive and have very large capacities, but their access rates are considerably slower. As you go from top to bottom in the hierarchy, the components typically get less expensive per byte, have higher capacities, and have slower access times.

DFSORT attempts to take advantage of all the levels of the storage hierarchy. The following sections briefly describe how DFSORT accomplishes this.

Processor cache

Processor cache is the special high-speed memory from which processors access their instructions and data. This memory has a much faster access rate than central storage, and is an integral part of IBM processors. It contains a copy of those portions of central storage that have been recently referenced. When an instruction or piece of data is needed by the processor and is not in the processor cache, a "cache miss" takes place and the processor waits while a copy of the data is brought into the cache. This results in higher CPU times.

DFSORT is designed to make efficient use of the processor cache by reducing cache misses as much as possible.

Central storage

Of all the components that affect DFSORT's performance, central (or main), storage is the most crucial. Sorting is a memory-intensive operation. Without enough central storage to back the virtual storage needs of DFSORT, its performance (as well as the performance of other applications on the system) degrades significantly due to excessive system paging activity.

To sort efficiently, DFSORT needs large amounts of virtual storage. Its needs grow with the amount of data being sorted; a data set four times as large as another requires roughly twice as much virtual storage to sort with the same degree of efficiency. If this virtual storage is not backed by central storage when DFSORT is running, there is a noticeable performance degradation on the system.

Central storage also plays an important role in the use of Hipersorting, dataspace sorting, and memory object sorting. DFSORT bases its use of these very efficient sorting methods on the amount of central storage it can use without causing excessive paging on the system. If too much central storage paging would result, DFSORT limits its use of these methods and relies on work data set I/O to auxiliary storage.

See "Hipersorting" on page 37, "Memory object sorting" on page 35, and "Sorting with data space" on page 40 for more information about exploiting central storage to reduce I/O and improve elapsed time.

Storage control cache

Certain storage control models, such as IBM's Enterprise Storage Server (ESS) and TotalStorage DS8000 Series, contain a special high-speed (relative to disk) memory known as storage control cache. This cache serves two purposes:

- When reading data from disk, if the data is already in the storage control cache, the program can access it directly from the cache without having to wait for the (relatively slow) disk to retrieve it.
- When writing data to disk, by writing directly to the cache (through use of cache fast write), applications can complete their write operations significantly faster than if they had written to the disk directly.

DFSORT takes advantage of the storage control cache by writing to its work data sets with cache fast write enabled. This speeds up the time needed to write to these work data sets as well as to read back from them.

To benefit from DFSORT's ability to use cache fast write (CFW), ensure that the CFW feature is activated on your storage control units (if appropriate).

DFSORT also takes advantage of the storage control cache by selecting the appropriate caching mode for input and output data sets. This reduces elapsed time for DFSORT applications and also helps other non-DFSORT applications make better use of the cache.

Disk

In most cases, DFSORT's Blockset technique adjusts automatically to take advantage of the geometry of the particular IBM disk being used. This is especially true for work data sets, whose block sizes and distribution of data play crucial roles in the performance of DFSORT.

The location of input, output, and work data sets, as well as the speed of the disks on which they reside has a significant effect on the performance of a sort application. For best results, work data sets should be placed on different storage subsystems than the input and output data sets. This helps avoid channel, control unit path, and device contentions. To attain the maximum performance benefit, these data sets (or at least the input and output data sets) should be placed on the fastest disks so that DFSORT can take advantage of their speed.

In general, while DFSORT has no control over where its data sets are allocated, it does automatically optimize its access patterns based on data set location to achieve the best possible performance.

When allocating DFSORT work data sets on devices attached to non-synchronous storage control units or connected to ESCON channels, elapsed time may be degraded for certain applications. To avoid this degradation, it is especially important to follow the virtual storage guidelines described in "Virtual storage guidelines" on page 43 and to ensure that DFSORT has an accurate knowledge of the size of the data set

being sorted. See *z/OS DFSORT Application Programming Guide* for more details about non-synchronous storage subsystem considerations.

In addition to data set location, certain data set characteristics, such as block size, are also very important when considering performance. As mentioned before, DFSORT automatically chooses an optimal block size for its work data sets. Using DFSORT's installation option SDB=LARGE, SDB=INPUT or SDB=YES enables DFSORT to choose optimal block sizes for its output data sets on disk as well. Of less importance, using installation option SDBMSG=YES enables DFSORT to choose optimal block sizes for its message data sets on disk.

Tape

Tape is the least expensive media on a per-byte basis. It has the highest capacity for data storage of any component in the storage hierarchy. But it also has a relatively slow access time and must be accessed sequentially. However, automatic tape libraries and IDRC compacted tape make tape a good choice for SORTIN and SORTOUT. Further, if these tape data sets are managed by DFSMSrmm or a tape management system that uses ICETPEX, DFSORT can obtain important information such as input file size, and input and output attributes, that help DFSORT optimize performance.

Tape devices are not recommended for use as work data sets. Since a fast access rate is critical for work data sets, and work data tends to be accessed in a nonsequential manner, performance is significantly better when disk devices are used as work data sets rather than tape devices. In fact, tape work data sets should be avoided if at all possible, because they prevent DFSORT from using its efficient Blockset technique.

Certain data set characteristics, such as block size, are important when considering performance on tape devices. Using DFSORT's installation option SDB=LARGE or SDB=INPUT enables DFSORT to choose optimal block sizes greater than 32K for its output data sets on tape. Using installation option SDB=YES enables DFSORT to choose optimal block sizes up to 32K for its output data sets on tape. Of less importance, using installation option SDBMSG=YES enables DFSORT to choose optimal block sizes up to 32K for its message data sets on tape as well

Virtual storage

Logically, DFSORT (like any other application) works within a virtual address space. Installation defaults such as TMAXLIM and run-time options such as REGION and MAINSIZE determine the size of this address space. With the exception of dataspace sorting (see “Memory object sorting, hipersorting and data space sorting” on page 6 for a discussion of dataspace sorting) and memory object sorting (see “Memory object sorting, hipersorting and data space sorting” on page 6 for more information about memory object sorting), this size remains constant throughout most of the sorting process.

DFSORT attempts to make the best use of the virtual storage it has available. If you provide DFSORT with enough virtual storage, it might be able to sort the input data set entirely in virtual storage (an "in-main-storage" sort) without need of work data sets or Hiperspace. This is the preferred method of sorting small data sets up to a few megabytes in size. To sort larger input data sets, DFSORT may use data space or memory objects to perform an "in-main-storage" sort.

When an in-main-storage sort is not possible or practical, DFSORT processes a portion of the input data set at a time, then combines all of these processed portions together into the final sorted data set. It is here that DFSORT excels at allocating virtual storage effectively in order to minimize both the number of portions as well as the time spent combining the portions into the output data set.

With dataspace sorting, DFSORT creates a data space to help carry out its processing. This is a new area of virtual storage, and is in addition to the original (specified or defaulted) virtual storage requested. The size of the data space is sufficient to guarantee an efficient sort (or dataspace sorting is not used).

In addition, DFSORT adjusts the size of the data space during processing, as necessary, in response to system paging levels. When system paging levels rise, DFSORT reduces its use of virtual storage (as long as this reduction does not significantly degrade DFSORT performance).

DFSORT also tries to move as many of its data areas above 16 MB virtual as it can to help provide virtual storage constraint relief for the system.

With z/OS, the MVS™ address space expands up to 16 exabytes in size. The architecture that creates this address space provides 64-bit addresses. The 64-bit address space marks the 2-gigabyte virtual line called "the bar". The bar separates storage below the 2-gigabyte address, called "below the bar", from storage above the 2-gigabyte address, called "above the bar". DFSORT obtains storage in virtual storage above the bar as a "memory object" for sorting.

Main storage

Main storage is the portion of virtual storage in the primary address space that DFSORT limits itself to using. In general, the more main storage you make available to DFSORT, the better the performance for larger jobs. To prevent excessive paging, insure that sufficient real storage is available to back up the amount of main storage used. This is especially important with main storage sizes greater than 32 MB. The default amount of main storage that will be made available to DFSORT is defined when it is installed.

Although it is possible to run a "very minimal" sort application in 88 KB of storage, the actual minimum amount of storage required is generally 128 KB to 440 KB depending on the application. The recommended minimum to avoid severe performance degradation is about 6MB but a greater amount may be required for optimal performance when sorting large files. Guidelines for setting these values are given in Table 4 on page 43 and Table 5 on page 44. It is recommended that the user not override the mainsize storage amount, but rather allow DFSORT to adjust the storage up to the DSA limit (DFSORT only makes this adjustment for a sort application when doing so can improve performance).

System-managed storage

The Storage Management Subsystem (SMS) makes data set allocation very easy and efficient. Having SMS manage the temporary data sets can be a good first step in migrating to system-managed storage. However, the SMS automatic class selection (ACS) routines can unintentionally affect DFSORT data set allocations and job performance, so you might need to coordinate changes to the ACS routines with the site's storage administrator, as described here.

When any data set is allocated on an SMS system, the allocation request passes through the system's data class and storage class ACS routines. If the data set will be system-managed, the request also passes through the system's management class and storage group ACS routines. There is only one set of ACS routines per site, and they are very powerful. They can override DFSORT installation options, such as VIO=NO, and can even override requests for a certain data, storage, or management class, or a certain unit or volume.

It is important to use &maxsize rather than &size in the ACS routines for volume assignment decisions related to the size of DFSORT work data sets. &size only takes into account the primary allocation, whereas &maxsize takes into account both the primary and possible total secondary allocation. Thus, the use of &maxsize will allow for a more accurate decision on the assignment of unit, storage class, storage group, and so on.

As an example of how ACS routines can affect the performance of DFSORT applications, consider a storage class ACS routine that assigns all temporary data sets to the STANDARD storage class, as shown in Figure 2 on page 17. The storage class is then used to assign the temporary data sets to a storage group. Because the routine does not differentiate between DFSORT temporary data sets and other temporary data sets, the storage group ACS routine cannot selectively prevent DFSORT temporary data sets from being assigned to VIO. Allocating temporary data sets to VIO works well in most cases, but might not be desirable for DFSORT temporary data sets, as explained in “VIO for DFSORT data sets ” on page 48.

```
PROC 1 STORCLAS
     ......
     ......
     SELECT
        WHEN(&DSTYPE = 'TEMP')
          SET &STORCLAS = 'STANDARD'
     END
     ......
     ......
END /* END OF PROC */
```

Figure 2: ACS Storage Class Routine

One way to avoid allocating DFSORT temporary data sets to VIO is to write the ACS storage class routine so it assigns all DFSORT temporary output and work data sets (for example, those with ddnames SORTOUT, SORTOFdd, SORTWKdd, and SORTDKdd) to a special NONVIO storage class. Using the &DD variable is the most efficient way to determine whether or not a data set is a DFSORT data set. Because you cannot use the &DD variable to check the ddname in the storage group ACS routine, you must check for DFSORT temporary data sets in the storage class ACS routine as shown in Figure 3 on page 17.

```
PROC 1 STORCLAS

/* DEFINE DFSORT TEMPORARY DATA SETS  */

FILTLIST DFSORTDD INCLUDE(SORTWK*,SORTDK*,SORTOF*,'SORTOUT')
     ......
     ......
/* ASSIGN 'NONVIO' STORAGE CLASS TO DFSORT TEMPORARY DATA SETS */
/* AND 'STANDARD' TO ALL OTHER TEMPORARY DATA SETS               */

     SELECT
        WHEN(&DSTYPE = 'TEMP')
          IF (&DD = &DFSORTDD)
           THEN SET &STORCLAS = 'NONVIO'
           ELSE SET &STORCLAS = 'STANDARD'
        OTHERWISE SET &STORCLAS = '
     END
     ......
     ......
END /* END OF PROC */
```

Figure 3: Storage Class ACS Routine

The storage group ACS routine can then look for the SORT storage class, and assign the data sets to a non-VIO storage group, as shown in Figure 4 on page 18. The site's storage administrator will need to create the special NONVIO storage class and alter the storage class and storage group ACS routines.

```
PROC 1 STORGRP
     ......
     ......
/* ASSIGN ALL TEMPORARY DATA SETS THAT ARE NOT DFSORT    */
/* DATA SETS TO 'SGVIO'                                  */

     SELECT
        WHEN(&DSTYPE = 'TEMP' && &STORCLAS; ¬= 'SORT')
          SET &STORGRP = 'SGVIO','PRIME'

/* ASSIGN DFSORT TEMPORARY DATA SETS TO 'PRIME'          */

        OTHERWISE SET &STORGRP = 'PRIME'
     END
     ......
     ......
END /* END OF PROC */
```

Figure 4: Storage Group ACS Routine

Be aware that the ddnames SORTOUT, SORTWKdd, and SORTDKdd can be changed to other names when a program calls DFSORT. If other DFSORT output and work data set ddnames are in common use at your site, you should also include them in the ACS routines. This scheme does not work for OUTFIL data sets specified with the FNAMES operand unless common ddnames are specified.

Note: Filtering on SORTIN in the ACS routines cannot prevent DFSORT temporary input data sets from being allocated to VIO. These data sets are allocated in preceding steps and passed to DFSORT. Thus, SORTIN is not the ddname used when these data sets are allocated.

If the only DFSORT temporary data sets are dynamically allocated work data sets, another way to avoid VIO allocation is to use a non-VIO generic device type for the installation option DYNALOC and the run-time option DYNALLOC. The storage group ACS routine could then be written to assign a pool storage group to data sets with that generic device type. Again, the site's storage administrator will need to establish a non-VIO generic device type and alter the storage group ACS routine.

Note: DFSORT's dynamic allocation uses a primary allocation of 0 and non-zero secondary extents. In this way, space is only used when the data set is extended. If the attempts to extend the data sets fail or the required space can still not be obtained, DFSORT may do one of two things:

- delete and then reallocate a dynamically allocated work data set using a non-zero primary allocation to force selection of a volume with enough free space to satisfy the request or
- retry the allocation using a device type matching the unit type of the original allocation (for example, 3390) in an effort to allocate the required space in a volume from another pool storage group. This behavior, though rare, should be taken into account when designing ACS routines that make assignments based solely on device type.

There might be other cases as well where the site's ACS routines unintentionally alter DFSORT performance or data set allocation. Be aware of this, and if you encounter problems, consult with your storage administrator to work out a joint solution.

Software that changes disk space allocations or adds additional volumes should be excluded from changing DFSORT work data set space. DFSORT's dynamic allocation function calculates the optimal work space for each sort application, adjusts the allocated work space as needed, and terminates if the required space is not available. Software that reduces DFSORT's disk work data set allocations may cause a sort application to fail. To avoid this situation, you should take the appropriate steps to have such software exclude the following ddnames from being changed: SORTWK*, STATWK*, DATAWK*, DAnnWK*, STnnWK*, SWnnWK* and ccccWK* (where cccc is defined with the SORTDD=cccc option). Additionally if you are using DFSMS data classes, you should not assign those ddnames to a DFSMS data class that uses the Space Constraint Relief feature.

Chapter 4. Installation considerations

Improving the performance of DFSORT consists of a number of activities, including:

- Tuning on a site-wide or system level
- Tuning of individual applications
- Designing efficient applications

To some extent, the success of each depends upon the success of the others. For instance, suppose we have a DFSORT application that has been carefully designed and tuned, and would be a good candidate for dataspace sorting. It might be adversely affected by the site's decision to set the default DSPSIZE parameter to 0. Such a setting would turn off dataspace sorting (unless the programmer has specifically overridden it in the application).

In a similar fashion, suppose that a significant site-wide tuning effort had been undertaken to find the optimal value for the installation parameter TMAXLIM. This parameter controls DFSORT's default maximum virtual storage. If a programmer overrides this default by specifying an unusually small SIZE or MAINSIZE value in an application, the application might make larger demands on the system's disk and processor resources. This, in turn, could cause performance problems for that application as well as any other active applications on the system.

Even the best system and application tuning may be wasted on applications that use sorts unnecessarily. For example, an application that sorts two already sorted data sets into one could be replaced with a more efficient merge application. Likewise, an application that uses a sort to extract a subset of the records, but does not rearrange the records in any way, could be replaced with a more efficient copy application.

This chapter offers advice for system programmers and application developers who are responsible for installing and using DFSORT. It includes the following topics:

- Running DFSORT resident
- Making the DFSORT SVC available
- Using ICEGENER as a replacement for IEBGENER
- Understanding storage options
- Using DFSORT capabilities
- Changing installation defaults
- Understanding installation performance options
- Using installation exits

Running DFSORT resident

By running DFSORT resident, that is, with DFSORT's SORTLPA library in LPALST and DFSORT's SICELINK library in LINKLST, you can gain three performance benefits:

- Two or more applications can use the same copy of DFSORT in main storage at the same time. This enables central storage to be used more efficiently and cuts down on system paging.
- The DFSORT load modules do not have to be loaded each time DFSORT is run. This also saves unnecessary paging and time. This is especially noticeable for the smaller DFSORT applications, which tend to make up the bulk of DFSORT jobs at most sites.
- The space for the DFSORT load modules is not charged against the virtual storage limits of individual applications. This saves storage that can be used by DFSORT to do a more efficient sort.

Since DFSORT is invoked so frequently, it is a prime candidate for running resident. When the DFSORT ICEGENER facility is used to replace IEBGENER, as described in “Using ICEGENER as a replacement for

IEBGENER” on page 20, DFSORT's use greatly increases, making it even more important to run DFSORT resident.

Making the DFSORT SVC available

The DFSORT-supplied SVC enables DFSORT to run authorized functions without itself being authorized. In particular, the following performance-related functions are impaired if DFSORT's SVC is not available:

SMF type-16
DFSORT's type-16 SMF record contains useful information for analyzing the performance of DFSORT (see “Using SMF data ” on page 68). Without the SVC, DFSORT cannot write the SMF record to an SMF system data set, although the record can still be obtained through an ICETEXIT routine. If DFSORT's SMF feature is activated (installation or run-time option SMF=SHORT or SMF=FULL) and a properly installed SVC is not available, then all DFSORT applications will abend.

Cache fast write
Cache fast write (CFW) enables DFSORT to save elapsed time because DFSORT is able to write its intermediate data into storage control cache, and read it from the cache (see “Cache Fast Write (CFW)” on page 9). Without the SVC, DFSORT cannot use CFW, and issues message ICE191I. Processing continues with possibly degraded elapsed time performance.

Caching mode
For storage control units that support cache, DFSORT selects the caching mode that appears to be the best for the circumstances. Without the SVC, DFSORT cannot set these caching modes, and issues message ICE191I. This results in the default modes being selected, with possibly degraded system and DFSORT elapsed time performance.

In addition to the functions described previously, there are other performance enhancements that are available to DFSORT through use of the SVC.

Recommendation: Make the DFSORT SVC available for best performance. Make sure that the installation of the SVC has been completed correctly so that the SVC can be used.

Using ICEGENER as a replacement for IEBGENER

At many sites, the copy utility IEBGENER is a frequently used program. Actions that improve its performance greatly benefit user productivity and resource utilization.

DFSORT's ICEGENER facility allows qualifying IEBGENER jobs to be routed to the more efficient DFSORT copy function. In most cases, using the DFSORT copy function instead of IEBGENER requires less CPU time, less elapsed time, and results in fewer EXCPs (see Figure 1 on page 10). If the DFSORT copy function cannot be used, DFSORT automatically transfers control to the IEBGENER utility. You can install ICEGENER so that your existing IEBGENER jobs do not require changes.

These are some of the circumstances that prevent the use of ICEGENER:

- A SYSIN DD statement other than SYSIN DD DUMMY is present.
- Detection of an error before DFSORT has started the copy operation
- A condition listed in DFSORT message ICE160A as follows:

1
The SYSUT1 or SYSUT2 data set was BDAM.

2
FREE=CLOSE was specified.

3
An attempt to open a data set caused a system error.

4
The SYSUT1 or SYSUT2 data set resided on an unsupported device.

5
ASCII tapes had the following parameters:

(LABEL=AL or OPTCD=Q) and RECFM=D and BUFOFF¬=L

or

(LABEL=AL or OPTCD=Q) and RECFM¬=D and BUFOFF¬=0

6
An attempt to read the DSCB for the SYSUT1 data set caused an error.

7
An attempt to read the DSCB for the SYSUT2 data set caused an error.

8
The SYSUT1 data set had keyed records.

9
User labels were present.

Under such circumstances, DFSORT transfers control to IEBGENER. However, IEBGENER may not be able to process the copy application either.

See *z/OS DFSORT Installation and Customization* for complete details on installing ICEGENER as an automatic replacement for IEBGENER.

Storage options

By using appropriate values for DFSORT installation storage options, you can ensure that the majority of applications have sufficient virtual storage. If the IBM supplied installation default value for an installation storage option is inappropriate for the majority of DFSORT applications at your site, you should change it to a more appropriate value. For specific applications, the installation values can be overridden using run-time options. See *z/OS DFSORT Installation and Customization* and *z/OS DFSORT Application Programming Guide* for details about DFSORT storage options and the relationships between the options.

Recommendations

You must provide sufficient virtual storage to DFSORT using the guidelines given in “Virtual storage guidelines” on page 43 and “Virtual storage and sorting with data space or memory objects ” on page 44.

The following installation storage option values are recommended:

SIZE
The default, SIZE=MAX, is recommended. This enables DFSORT to use as much virtual storage as possible, both above and below 16 MB virtual, subject to the limits set by MAXLIM and TMAXLIM.

When installation option SIZE=MAX, EXEC PARM option SIZE=MAX, or run-time option MAINSIZE=MAX is in effect, the TMAXLIM, RESALL, and RESINV options are used. These options are not used when SIZE=n or MAINSIZE=n is in effect.

You should also be aware of how the JCL REGION parameter can affect DFSORT virtual storage allocation. While subject to the constraints of your site's IEFUSI and IEALIMIT exits, the JCL REGION value limits the amount of below 16 MB virtual storage.

DFSORT attempts to place as much of its storage as possible above 16 MB virtual. DFSORT, however, still needs sufficient storage below 16 MB virtual to run effectively. In general, a REGION value of at least 512 KB is best. If DFSORT is called by a program, the REGION value should be large enough to allow sufficient storage for DFSORT and the program. If E15 or E35 user exits are used, a larger REGION value might improve performance because these functions use more storage below 16 MB virtual.

In general, the minimum of:

- REGION plus OVERRGN
- SIZE or MAINSIZE
- MAXLIM

determines the maximum storage available below 16 MB virtual. Thus, with REGION=100K, OVERRGN=65536 (64 KB), SIZE=4194304 (4 MB), and MAXLIM=1048576 (1 MB), a total of 4 MB is available to DFSORT, but only 164 KB can (and probably will) be available below 16 MB virtual. On the other hand, if REGION=2M and SIZE=819200 (800 KB), then a total of 800 KB is available to DFSORT and all of it can (but probably will not) be allocated below 16 MB virtual.

If the available storage below 16 MB virtual is insufficient, DFSORT issues message ICE039A, which indicates the minimum additional storage required below 16 MB virtual. Although the application might run if you add the minimum storage indicated, it is recommended that you increase either the REGION value, or the SIZE or MAINSIZE value (or both) to provide sufficient virtual storage for the application to run efficiently.

PARM parameter SIZE=n or OPTION parameter MAINSIZE=n can be used to allow more (or less) storage for specific applications for which the installation default is not appropriate. For example, you might want to specify MAINSIZE=8M to improve performance for a critical large application when your installation default is 6 MB. Note that DSA is not used when SIZE=n or MAINSIZE=n is specified.

When SIZE=n or MAINSIZE=n is in effect, RESALL and RESINV are not used. Generally, this does not cause a problem, but if it does you should ensure that the user exit size values in your MODS statement, if any, are correct. Because the user exit size in the MODS statement is only an estimate, you can raise it if necessary to allow more reserved storage. Alternatively, you could raise the REGION value or go back to using SIZE=MAX or MAINSIZE=MAX to resolve the problem. See *z/OS DFSORT Installation and Customization* for details of the relationships between SIZE, MAINSIZE, RESALL, RESINV, REGION, and other storage parameters.

TMAXLIM
Although different sites have different requirements, experience indicates that the TMAXLIM default of 6 MB is a good general purpose value. Raising TMAXLIM from the default value might provide additional performance benefits, particularly if a high percentage of your site's DFSORT usage is spent on large sorting applications. However, the DSA parameter (described later in this section) only raises the storage for selected applications. Since a larger TMAXLIM affects every job and could result in higher CPU times for smaller jobs, setting DSA appropriately is a better choice than increasing TMAXLIM.

Alternatively, you could specify SIZE=n or MAINSIZE=n at run-time to provide more (or less) storage for specific DFSORT applications.

DSA
DSA (Dynamic Storage Adjustment) allows DFSORT to change the value of TMAXLIM dynamically if doing so should improve the performance of a sort job. In general, the default of 64 (MB) is sufficient to handle most sorts, but if your site runs very large sorts (multiple GB of input data), you might consider increasing the DSA to 128.

MAXLIM
The default of 1 MB is generally sufficient. DFSORT performance might be improved by specifying larger MAXLIM and REGION values for applications with E15 or E35 user exits that use storage below 16 MB virtual.

MINLIM
A MINLIM value of at least the default value of 440 KB is recommended. The major reason for using a lower MINLIM value is if your site has applications that fail with a higher MINLIM value (and for which there is no easy fix for the failures).

The MINLIM value is important only for jobs which specify a SIZE or MAINSIZE value that is less than the MINLIM value. By not using a value of MAX, these jobs become locked into a specific virtual storage limit. They must also reserve storage without using RESALL and RESINV, usually by making the SIZE or MAINSIZE value less than the REGION value or by coding user exit sizes on a MODS control statement. Such applications are prime candidates for your tuning efforts.

One way to improve the performance of these applications is to raise the MINLIM value (for example, to the MAXLIM value). This enables such applications to run with a larger amount of virtual storage. This strategy, however, might cause some of these applications to fail due to insufficient reserved storage. Using run-time options, you should change the failing applications to use SIZE=MAX or MAINSIZE=MAX and set appropriate values for user exit sizes (on the MODS control statement), RESALL, and RESINV.

OVERRGN
The default and recommended values for this option are 64 KB for directly-invoked and 16 KB for program-invoked applications.

RESALL
The default is 4 KB and should normally not be modified. If a storage related failure occurs when sufficient virtual storage (REGION and SIZE) has been specified, try setting RESALL to a larger value to correct the problem.

RESINV
Normally, a RESINV value of 16 KB is sufficient and is recommended.

ARESALL
ARESALL is seldom needed and can be kept at its default of 0.

ARESINV
ARESINV is seldom needed and can be kept at its default value of 0.

See *z/OS DFSORT Installation and Customization* for more information about these parameters.

Hipersorting, memory object sorting, and data space sorting

Hipersorting, memory object sorting and data space sorting are all methods of exploiting central storage to reduce work data set I/O and improve DFSORT efficiency. DFSORT selects the method that best fits the characteristics of the sort being executed and the available resources at the time of execution. This section recommends ways to use Hipersorting, memory object sorting, and data space sorting to improve DFSORT performance.

Recommendations

The recommended installation settings for Hipersorting, memory object sorting, and data space sorting are the IBM-supplied defaults:

- EXPMAX=MAX
- EXPOLD=50%
- EXPRES=10%
- TUNE=STOR
- HIPRMAX=OPTIMAL
- MOSIZE=MAX
- MOWRK=YES
- DSPSIZE=MAX

These settings are described in detail in the sections that follow.

EXPMAX=MAX

This setting allows maximum Hipersorting, memory object sorting, and data space sorting activity on a system, especially during periods when applications (both DFSORT and non-DFSORT) are making little use of central storage. Setting EXPMAX=n, where n is a number of megabytes, or EXPMAX=p%, where p% is a percentage of the configured central storage, should only be considered when it is important for a site to limit the total amount of storage used by all DFSORT applications on the system.

Setting EXPMAX to n or p% can help prevent situations where a long running sort application holds storage resources that may be needed by new work entering the system later. Note that this only limits DFSORT's storage usage. The total storage used by DFSORT and other address spaces can exceed EXPMAX depending on the values in effect for EXPRES and EXPOLD.

EXPOLD=50%

This setting is recommended only if migration of storage data to auxiliary storage does not cause a problem on your system. During batch processing windows, migrating old data to auxiliary storage can improve overall system performance, since it allows more active data, like DFSORT work data, to use storage in place of disk work data sets. But, the one-time migration can have a negative impact on system performance, especially if the paging subsystem is not large enough to hold the number of frames that are migrated. Using EXPOLD=50%, along with TUNE=STOR, causes DFSORT to continually examine the amount of old storage and never allow a sort to cause more than half of it to be used by DFSORT.

If the migration of large amounts of old storage data is not a concern and you want to maximize DFSORT's use of central storage, consider setting EXPOLD=MAX. This setting should only be used if there is a robust paging subsystem large enough to support a large spike in the number of frames being migrated to auxiliary storage.

If the migration of large amounts of old storage data is a concern for your site, set EXPOLD to a small number or zero. Setting EXPOLD=0 limits DFSORT to only consider available frames in its evaluation of resources for Hipersorting, memory object sorting or data space sorting.

EXPRES=10%

This setting permits DFSORT almost full use of available storage when non-DFSORT activity is light, but greatly reduces the possibility of an overuse of storage when there is a sudden increase in the use of storage by non-DFSORT applications. If you need to keep central storage available for non-DFSORT applications, increase this value. If you want DFSORT to have full use of available storage, set EXPRES=0.

TUNE=STOR

This setting causes DFSORT to allocate central storage in increments sized to balance usage when multiple sorts are executing concurrently on the same system. DFSORT continually examines available resources and dynamically makes adjustments to the increment size. If EXPMAX, EXPOLD or EXPRES are specified as percentages, DFSORT will also recalculate those values based on available resources. DFSORT's dynamic allocation will increase the space allocations to reduce the risk of failure if central storage resources become constrained during execution of a sort.

If conserving disk work space is more important in your environment, you can use TUNE=DISK which will cause each sort to allocate all of the available work space it plans to use immediately. This can cause a small number of sorts to monopolize available central storage but DFSORT's dynamic allocation will be able to more aggressively reduce the disk work space allocations based on the expected central storage usage.

Note that even sorts which use JCL or pre-allocated work data sets will allocate storage based on the TUNE parameter. This is necessary so that all sorts compete equally for the same storage. Similarly, this is why TUNE cannot be overridden as a run-time option. TUNE=DDYN can be used to cause sorts that use DFSORT's dynamic allocation to allocate all of the available storage they plan to use immediately, and cause sorts using JCL or pre-allocated work data sets to allocate in increments. This is useful for special situations where dynamic allocation is frequently used by the majority of sorts, and optimizing disk space is a priority, but there might be a known time period when the majority of sorts do not use dynamic allocation and would benefit from balancing central storage.

TUNE=OLD is available for customers who prefer that DFSORT allocate central storage in fixed increment sizes. If EXPMAX, EXPOLD or EXPRES is specified as a percentage, DFSORT calculates that value once based on configured central storage.

HIPRMAX=OPTIMAL, MOSIZE=MAX, DSPSIZE=MAX, MOWRK=YES

These settings allow the DFSORT installation options EXPMAX, EXPOLD, and EXPRES to control the total Hipersorting, memory object sorting, and data space sorting activity on a system.

HIPRMAX=n or HIPRMAX=p% should be reserved for use as a run-time override for applications that have a special reason to limit the amount of Hiperspace available for Hipersorting.

MOSIZE=n or MOSIZE=p% should be reserved for use as run-time overrides for applications that have a special reason to limit the amount of virtual storage available for memory object sorting.

MOWRK=NO should only be used for installations that have a special need to only allow memory objects to be used as an extension of main storage instead of intermediate work space.

DSPSIZE=n or DSPSIZE=p% should be reserved for use as run-time overrides for applications that have a special reason to limit the amount of virtual storage available for data space sorting.

When Hiperspace, memory object and data space usage is limited, JCL work data sets (SORTWKdd) may no longer be large enough to complete a sort, or the installation DYNSPC default may not be large enough for unknown file size conditions. As a result, error messages (for example, ICE046A and ICE083A) may be issued due to less central storage being used. The use of dynamic allocation with an appropriately large installation DYNSPC value is recommended. If you have JCL work data sets and you want DFSORT to ignore them, use DYNAUTO=IGNWKDD.

DFSORT installation defaults

DFSORT is shipped with a set of installation defaults that are used by all DFSORT applications at a site. These defaults set values for various DFSORT parameters. They can be changed on a site-wide level for various invocation environments and time-of-day environments using ICEPRMxx members of PARMLIB or the ICEMAC macro. See *z/OS DFSORT Installation and Customization* for complete details of installation defaults.

Most of the installation defaults can be overridden for specific applications by setting the appropriate run-time options. In addition, some of the defaults (as well as run-time options) can be overridden for all or a subset of applications at a site through use of an ICEIEXIT routine. See *z/OS DFSORT Application Programming Guide* for complete details of DFSORT's option override scheme.

Modifying installation defaults

ICEPRMxx members in PARMLIB or the ICEMAC macro can be used to modify the IBM-supplied installation default values for DFSORT. Many of these installation options have an impact on performance, including:

- Storage limits
- Hiperspace, data space, and memory object limits
- Use of system-determined block size
- Reallocation of VIO work data sets
- Use of dynamic allocation for work data sets
- Number of work data sets for dynamic allocation
- Device type for dynamically allocated data sets
- Use of control interval access for VSAM data sets
- Number of I/O buffers to use
- IDRC tape compaction ratio
- Use of VERIFY, EQUALS, ALTSEQ, LOCALE and COLLKEY

Modifications to the installation defaults should be carefully chosen to reflect how you want DFSORT to run at your site. You should only override installation options for which the supplied default values are not acceptable. You can easily change installation options using a START ICEOPT command from the console or in a COMMNDxx member in PARMLIB. For example, to use the installation options specified in an ICEPRM01 member of PARMLIB, type the following at the console:

```
start iceopt,iceprm=01
```

ICEPRMxx members in PARMLIB are the recommended way to change your installation defaults. You can activate different ICEPRMxx members for different LPARs.

Alternatively, you can use the ICEMAC macro and a USERMOD to update the source distribution library to reflect the changed options.

Invocation installation environments

DFSORT allows separate sets of installation defaults for four invocation installation environments, as follows:

JCL-invoked (ICEAM1)
DFSORT invoked directly (that is, not through programs) by batch jobs

Program-invoked (ICEAM2)
DFSORT invoked through batch programs

TSO-invoked (ICEAM3)
DFSORT invoked directly by foreground TSO users

TSO program-invoked (ICEAM4)
DFSORT invoked through programs by foreground TSO users

Time-of-Day installation environments

DFSORT allows each of the invocation installation environments (ICEAM1-4) to specify time-of-day installation environments (ICETD1–4) with separate sets of installation defaults to be used for runs on specific days and times (for example, from 8:00am to 5:00pm on Saturday and Sunday). You might want to take advantage of this feature to allow, for example, DFSORT to use larger storage values (DSA, TMAXLIM, and so on) for runs at certain days and times.

Listing the installation defaults with ICETOOL

You can use an ICETOOL job similar to the one in Figure 5 on page 26 to list the merged PARMLIB/ICEMAC installation defaults actually in use at your site for the eight installation environments.

```
//DFRUN   JOB A402,PROGRAMMER
//LISTDEF EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=A
//DFSMSG  DD SYSOUT=A
//SHOWDEF DD SYSOUT=A
//TOOLIN  DD *
  DEFAULTS LIST(SHOWDEF)
/*
```

Figure 5: Using ICETOOL to List Installation Defaults

See *z/OS DFSORT Installation and Customization* for complete details on installation options and defaults.

Note: The supplied DFSORT installation defaults were chosen to balance DFSORT versus system performance and resource usage, as well as to have DFSORT operate in a manner appropriate for most customers. Therefore, the supplied defaults should only be changed on an exception basis.

Installation defaults that are inappropriate for an application should be overridden for that application with the corresponding run-time options. See *z/OS DFSORT Application Programming Guide* for complete details of run-time options.

Installation options and performance

Table 3 on page 27 shows site-wide installation options that influence the performance of DFSORT, a description of each option, and additional comments about the option.

Table 3: Installation Options That Influence DFSORT Performance.

Type	Installation Option	Description	Comments
Options that tailor main storage	SIZE	Upper limit for total storage above and below 16 MB virtual	Limited by TMAXLIM when SIZE=MAX is in effect. The recommended value is MAX.
	OVERRGN	Upper limit for storage over and above that specified by REGION.	Limited by the IEFUSI or IEALIMIT installation-wide exits using the "region limit" values.
	MAXLIM	Upper limit for storage below 16 MB virtual.	Always used. The recommended value is 1 MB.
	TMAXLIM	Upper limit for total storage above and below 16 MB virtual.	Only used when SIZE=MAX or MAINSIZE=MAX is in effect. The recommended value is 6 MB.
	DSA	Upper limit for dynamic storage adjustment.	Only used when SIZE=MAX or MAINSIZE=MAX is in effect and the use of additional storage should improve performance. The recommended value is 64 MB.
	MINLIM	Lower limit for SIZE or MAINSIZE.	Only used when SIZE=n or MAINSIZE=n is less than MINLIM. The recommended value is 440 KB.
	RESALL	Storage below 16 MB virtual that is reserved for system use.	Only used when SIZE=MAX or MAINSIZE=MAX is in effect. Can reduce the amount of virtual storage available for use by DFSORT.
	RESINV	Storage below 16 MB virtual that is reserved for use by an invoking program.	Only used when SIZE=MAX or MAINSIZE=MAX is in effect and DFSORT is program-invoked.
	ARESALL	Storage above 16 MB virtual that is reserved for system use.	Can reduce the amount of virtual storage available for use by DFSORT.
	ARESINV	Storage above 16 MB virtual that is reserved for use by an invoking program.	Only used when DFSORT is program-invoked.

Table 3: Installation Options That Influence DFSORT Performance. (continued)

Type	Installation Option	Description	Comments
Options that affect use of Hiperspace, memory objects and data space	HIPRMAX	Upper limit for Hiperspace for a single application.	Hiperspaces limited by IEFUSI installation exit. Recommended setting is OPTIMAL. Use EXPMAX, EXPOLD, and EXPRES to control Hipersorting on a system-wide basis.
	MOSIZE	Upper limit for a memory object for a single application.	Recommended setting is MAX. Memory objects are limited by IEFUSI and JCL MEMLIMIT. If central storage is overcommitted, set MOSIZE to a low value and test it. If resource contention is a problem only at specific times, set MOSIZE to MAX in your environment installation modules and use a time-of-day installation module to restrict the MOSIZE value for all jobs during those specific times.
	MOWRK	Whether memory objects can be used as intermediate work storage or only as an extension of main storage.	Recommended setting is YES. DFSORT's use of memory objects as work space provides more efficient operation and improved balancing of central storage resources across concurrent sort applications.
	DSPSIZE	Upper limit for data space size.	Recommended setting is MAX. If central storage is overcommitted, set DSPSIZE to a low value and test it. If resource contention is a problem only at specific times, set DSPSIZE to MAX in your environment installation modules and use a time-of-day installation module to restrict the DSPSIZE value for all jobs during those specific times. Data spaces are limited by IEFUSI.
	EXPMAX	Upper limit for total available central storage used for all Hipersorting, memory object and data space sorting activity.	Available central storage is subject to non-Hipersorting, non-memory object and non-data space sorting activity. Should be set to MAX unless you want to limit Hipersorting, memory object and data space sorting activity to a fixed portion of storage.
	EXPOLD	Upper limit for total old expanded and central storage used for all Hipersorting, memory object and data space sorting activity.	Old expanded and central storage are subject to non-Hipersorting, non-memory object and non-data space sorting activity. Should be set to 50% unless a large migration of old storage data causes a problem at your site.

Table 3: Installation Options That Influence DFSORT Performance. (continued)

Type	Installation Option	Description	Comments
	EXPRES	Lower limit for available expanded and central storage reserved for non-Hipersorting and non-memory object sorting use.	Available expanded and central storage are subject to non-Hipersorting and non-memory object sorting activity. Should be set to 10% unless you want DFSORT to have lower access priority to storage than other applications.
Options that influence allocation of work data sets	DYNAUTO	Whether work data sets should be dynamically allocated automatically.	Set to YES, or to IGNWKDD if you want to use dynamic allocation even when JCL work data sets are specified. Can cause an increase in CPU time for small sort applications but helps minimize the amount of disk space required for sorting.
	DYNALOC	Default device type and number of work data sets when dynamic allocation is requested.	Does not request dynamic allocation; only supplies defaults.
	DYNAPCT	Default additional work data sets (as percentage) to be dynamically allocated and used only when needed.	If set to OLD, additional data sets are only allocated when DFSORT cannot determine the input file size. Exceptions are documented in *z/OS DFSORT Installation and Customization*
	DYNSPC	Default amount of space to allocate for dynamically allocated work data sets.	Only used when DFSORT cannot determine the input file size.
	IDRCPCT	Compaction ratio to use for IDRC-compacted tape data sets when computing data set size.	Tells DFSORT how much work space to dynamically allocate when input is on an IDRC-compacted tape and DFSORT cannot determine the input file size.
	VIO	Whether VIO work data sets should be automatically reallocated to real disk locations.	ACS routines can override this option.
Options that affect VSAM performance	CINV	Whether control interval access is used for VSAM input data sets.	Improves performance for VSAM input data sets.
	VSAMBSP	Amount of VSAM buffer space to use.	To improve VSAM performance, set OPTIMAL or MAX.

Table 3: Installation Options That Influence DFSORT Performance. (continued)

Type	Installation Option	Description	Comments
Options that affect input and output data set performance	IOMAXBF	The maximum buffer space to be used for disk SORTIN and SORTOUT data sets.	Recommended setting is 34 MB. Should only be lowered when device contention or long channel connect times are a problem. Lowering the default could result in larger EXCP counts for these data sets, but could also decrease the channel connect time per EXCP.
	ODMAXBF	The maximum buffer space to be used for each OUTFIL data set.	Recommended setting is 2 MB. Lowering the value can cause performance degradation for the application. When you use more than 2 MB, the performance improvements are small except for EXCPs, and, there is an increased need for storage.

Table 3: Installation Options That Influence DFSORT Performance. (continued)

Type	Installation Option	Description	Comments
Other options that affect performance	CFW	Whether cache fast write is used for DFSORT work data sets.	Only applicable for work data sets located on disks attached to cached 3990 storage control units. Can be used only if DFSORT SVC is installed. Cache fast write can reduce the elapsed time of sorting applications. CFW is more beneficial to DFSORT performance for small and intermediate sized sorts, where the work data sets are relatively small.
	EQUALS	Whether input order is preserved for records with equal keys.	Can cause an increase in CPU time. The default setting of VLBLKSET should be changed to NO if possible.
	VERIFY	Whether output records are checked for correct order.	Can cause an increase in CPU time.
	ALTSEQ	Whether a collating sequence other than EBCDIC is used.	Can cause an increase in CPU time.
	SDB and SDBMSG	Whether system-determined block size is used for output data sets whose block size is zero.	Use of optimum block sizes for output data sets provides more efficient performance than using other block sizes.
	IGNCKPT	Specifies whether Checkpoint/Restart requests at run-time should be ignored.	When CKPT is specified, the Blockset technique cannot be selected. Therefore, the recommended setting is YES so that the Blockset technique can be used.
	CHALT	Specifies whether ALTSEQ is to be applied to character format fields (CH).	As with ALTSEQ, can cause an increase in CPU time. Recommended setting is NO.
	COBEXIT	Specifies the library for COBOL E15 and E35 routines.	Use COB2. COB1 is obsolete.
	COLLKEY	Specifies whether Unicode data processing is to be used and, if so, designates the active collation rules version.	Should only be used when required since it can show degraded performance relative to collation using character encoding values.
	EFS	Specifies the name of a user-written Extended Function Support program to be called by DFSORT.	Can cause an increase in CPU time and elapsed time. Use only when necessary for your application.

Table 3: Installation Options That Influence DFSORT Performance. (continued)

Type	Installation Option	Description	Comments
LOCALE	Specifies whether locale processing is to be used and, if so, designates the active locale.	Should only be used when required since it can show degraded performance relative to collation using character encoding values.	

See *z/OS DFSORT Installation and Customization* for a complete list of site-wide installation options. See *z/OS DFSORT Application Programming Guide* for corresponding run-time overrides.

Installation exits

DFSORT allows you to use user-written, installation-wide initialization and termination exit routines to perform a variety of functions, such as overriding the options currently in effect and collecting statistical data. For tuning purposes it is often advantageous to install these exits for the following reasons:

- These exits can be used for performance data gathering to help you understand the use of DFSORT at your site and make the appropriate tuning decisions based on this information.
- An initialization routine allows you to override the run-time values set for certain options, which enforces your decisions for those option values for all DFSORT applications at your site.

ICEIEXIT

A site-supplied ICEIEXIT routine can exercise control over certain DFSORT run-time options.

If present and activated, the ICEIEXIT routine is called and passed installation and run-time information by DFSORT. The ICEIEXIT routine can then use current DFSORT and system information to determine whether to change certain options in effect. This also permits site-wide control of certain options whose installation defaults have been overridden at the application-level.

An ICEIEXIT routine can examine installation and run-time information related to:

- Storage limits
- Hiperspace limits
- Data space limits
- Memory object limits
- Use of VERIFY
- OUTFIL buffer space limits

and additional run-time information related to:

- DFSORT technique used
- Type of DFSORT application
- Method of DFSORT invocation
- Storage above 16 MB virtual
- Configured expanded storage

An ICEIEXIT routine can also change certain run-time options including:

- Storage limits
- Hiperspace limits
- Data space limits
- Memory object limits

- Use of VERIFY
- OUTFIL buffer space limits

A site could use an ICEIEXIT routine to control applications and enforce site standards. For example:

- Many options (for example, MAXLIM, SIZE, TMAXLIM, DSA) can affect the virtual storage used by DFSORT. An ICEIEXIT routine could specify the amount of virtual storage to be used depending on such factors as performance requirements and jobname.

  Note: The time-of-day installation modules allow you to specify the virtual storage to be used depending on the day and time when an application runs. See "Time-of-Day installation environments" on page 26 for more information.
- Before creating a data space, DFSORT checks to see how much central storage either is not being used or has gone unreferenced for a sufficient period of time. This is to make sure enough real storage is available to back the data space without causing excessive system paging activity. An ICEIEXIT routine can further reduce the risk of overcommitting central storage by limiting the amount of data space that a single DFSORT application can use. This would also override any run-time specifications that try to get around the installation default.

See *z/OS DFSORT Installation and Customization* for information about coding an ICEIEXIT routine and a sample ICEIEXIT routine, which shows how the storage available to DFSORT can be dynamically modified based on the jobname/stepname and type of application.

ICETEXIT

If present and activated, the ICETEXIT routine is called at the end of DFSORT application processing. It is available for those who wish to make a thorough analysis of DFSORT performance data using a single source of information. See "Using ICETEXIT data " on page 70 for more information about using this routine; see *z/OS DFSORT Installation and Customization* for complete information on how to write and install an ICETEXIT routine.

Chapter 5. Run-time considerations

This chapter offers advice about improving the performance of individual DFSORT applications by using run-time options related to the following areas:

- Memory object sorting
- Hipersorting
- Sorting with data space
- Cache fast write
- File size
- Storage
- Input and output data sets

The last section includes a table with information on run-time options available with DFSORT that can affect performance.

Memory object sorting

Memory object sorting is a DFSORT capability that uses memory objects in 64-bit virtual storage. For a more detailed description of this capability, see "Memory object sorting, hipersorting and data space sorting" on page 6. The sections that follow include information on how to use memory object sorting in the most efficient way at your site.

DFSORT selects the most appropriate mode (memory objects as intermediate work space or memory objects as additional main storage) for each particular run. Not every sort application can use memory object sorting, and even for those sorts that can use memory object sorting, it may be more advantageous not to use it under certain circumstances.

A memory object, used in memory object sorting, is allocated in virtual storage above the bar and backed by central storage. The use of central storage for memory object sorting must always be weighed against the possibility of degrading performance for a particular job, or for the entire system, by overusing central storage. If DFSORT were to use memory object sorting indiscriminately, there could be a significant increase in paging activity and a resulting reduction in total system performance, affecting all jobs, including sorts.

The recommended setting for MOSIZE is MAX and the recommended setting for MOWRK is YES. MOSIZE=MAX enables DFSORT to dynamically determine the total amount of memory object storage to be used for memory object sorting, taking into account the size of the file being sorted and the central storage usage activity of the system. MOWRK=YES enables DFSORT to determine whether to use memory object storage as intermediate work space or as an extension to main storage. DFSORT has full control over total memory object sorting activity, and sites can customize their definition of MOSIZE=MAX with the installation options EXPMAX, EXPOLD, and EXPRES. MOSIZE=n and MOSIZE=p% should be reserved for use as a run-time override for applications that have a special reason to limit the size of a memory object for memory object sorting.

The number of megabytes used for a memory object during the sort is displayed in message ICE199I or ICE299I depending on whether DFSORT uses it as intermediate work space or as an extension to main storage. If memory object sorting is not used, both messages show a value of 0M.

Limitations

The maximum amount of memory object storage used by DFSORT is limited to the minimum of the following values:

- The MEMLIMIT parameter limit on the total size of memory objects that can be allocated in a single job step. See *z/OS MVS JCL Reference* for a description of the MEMLIMIT parameter.
- The IEFUSI exit limit on the total size of memory objects that can be allocated in a single job step. See *z/OS DFSMS Installation Exits* for a description of IEFUSI.
- The MOSIZE value in effect, when set to a value other than MAX. The MOSIZE value in effect is either the installation default, an overriding value specified at run-time, or an overriding value specified in the installation ICEIEXIT routine. Note that the value specified in the ICEIEXIT routine overrides any other value.
- Available central storage. Throughout the run, DFSORT checks the amount of available central storage. If, as a result of any such check, either a central storage shortage is predicted, or one of the site limits for total central storage usage by all memory object sorting applications is reached, DFSORT switches from using a memory object to using disk work data sets.

  In addition, all future DFSORT applications are prevented from using memory object sorting until the central storage situation is relieved. This prevents memory object sorting applications by themselves from causing a shortage of central storage.

The following are those cases for which you should not attempt to adjust your application; in these cases the best performance for the individual job and for the system is achieved by not using memory object sorting:

- **Other performance features are in use:** Memory object sorting is not used when DFSORT decides to use Hipersorting or data space sorting. DFSORT dynamically chooses between using memory object sorting, data space sorting, and Hipersorting and selects the one that provides the best performance for the particular sort. Messages ICE180I and ICE188I indicate whether Hipersorting or data space sorting was used for a particular run.
- **The size of the input data set is very small:** If the amount of data to be sorted is known to be small enough that the sort can be accomplished in main memory, memory object sorting is not used.

Application adjustments

The following are those cases for which you may want to adjust your application in order to take advantage of memory object sorting.

- **The Blockset technique was not selected:** Memory object sorting is supported only for the Blockset technique. If Blockset is not selected, message ICE800I indicates why it was not selected. Note that the ICE800I message is printed only when a SORTDIAG DD statement was coded in the sort's JCL, or installation option DIAGSIM=YES has been specified for your site. Use the ICE800I reason code to determine the exact condition that is preventing the use of Blockset. If you are interested in using memory object sorting for the job, change your application appropriately to eliminate the particular condition, so that Blockset can be used.
- **Insufficient available virtual storage:** In some cases, the amount of virtual storage available to DFSORT can influence the potential effectiveness of memory object sorting. The third value in message ICE092I or ICE093I indicates the amount of storage available for a particular sort job. To help reduce the likelihood of not using memory object sorting because of insufficient virtual storage, ensure that this value is at least the minimum recommendation given in Table 5 on page 44.
- **Insufficient available above the bar virtual storage:** The MEMLIMIT parameter on the JOB or EXEC JCL statement limits the total amount of memory object storage that can be allocated in a single job step. For more effective use of memory object sorting, specify a MEMLIMIT value that is at least equal to the size of the sorted data set, or specify an unlimited size for memory objects using the MEMLIMIT=NOLIMIT parameter on the JOB or EXEC JCL statement. Note that if MEMLIMIT is not specified, but REGION=0K or REGION=0M is specified on the JOB or EXEC JCL statement, then MEMLIMIT=NOLIMIT is implied.
- **Insufficient available central storage:** The total amount of memory object storage DFSORT needs is directly related to the size of the input data set. Memory objects are backed by central storage. Therefore, insufficient available central storage for backing DFSORT's memory objects has an effect on the performance of memory object sorting. If the total amount of memory object storage DFSORT needs would exceed the available central storage, DFSORT chooses not to use memory object sorting.

If you would like memory object sorting to be used, there are several possible approaches you can take:

- Make sure that the MOSIZE value, the MEMLIMIT value, or the installation IEFUSI exit are not limiting the application to a small amount of memory object storage. Setting MOSIZE=MAX (or setting MOSIZE to a very large value), and having MEMLIMIT and IEFUSI allow the use of a memory object at least the size of the input data set, will remove this limitation.
- Rerun the application when system activity, especially other concurrent memory object sorting and Hipersorting activity, is lower so that more central storage is available for the sort.
- Reduce the size of the input data set, so that less central storage is required for the sort. For some applications it is not necessary to sort all of the data, since only a subset is needed for processing. For example, INCLUDE, OMIT, SKIPREC, or STOPAFT can significantly reduce the amount of intermediate storage required by DFSORT. See *z/OS DFSORT Application Programming Guide* for more details about these features of DFSORT.
- Ensure that DFSORT has accurate information about the input file size. DFSORT can automatically estimate the file size for disk input data sets and tape data sets managed by DFSMSrmm or a tape management system that uses ICETPEX. However, there are certain situations, which DFSORT reports with message ICE118I, in which DFSORT cannot determine the file size. See "File Size and Dynamic Allocation" in *z/OS DFSORT Application Programming Guide* for more information on these situations, and what to do about them.

- **Insufficient work data set usage:** When it is possible to use a combination of memory object and disk storage as intermediate work space, DFSORT must decide how best to use disk work data sets. Specify generous extent sizes for disk work data sets, especially for secondary extents.

In general, DFSORT takes into account the effects of using memory objects on both the application's performance and the system's performance when determining whether or not to use memory object sorting. If either effect is not desirable, DFSORT chooses not to use memory object sorting.

See "Memory object sorting, hipersorting and data space sorting" on page 6 for information on the benefits and operation of memory object sorting and "Hipersorting, memory object sorting, and data space sorting" on page 23 for additional information on using memory object sorting effectively.

Hipersorting

Hipersorting is a DFSORT capability that uses Hiperspaces for sorting. For a more detailed description of this capability, see "Memory object sorting, hipersorting and data space sorting" on page 6. The sections that follow include information on how to use Hipersorting in the most efficient way at your site.

DFSORT selects the most appropriate mode (Hiperspace-only, Hiperspace-mixed, or disk-only mode) for each particular run. Not every sort application can use Hipersorting, and even for those sorts that can use Hipersorting, it may be more advantageous not to use it under certain circumstances. This section examines the most common reasons for not using Hiperspace and explains the possible actions that can be undertaken to allow more jobs to take advantage of Hipersorting.

DFSORT adjusts its use of hiperspaces appropriately taking into consideration that real storage is a resource available to all users of the system.

Some customers have expressed concerns that they would like to see Hipersorting used more often. However, the use of hiperspace storage must always be weighed against the possibility of degrading performance for a job or for the entire system, by overusing storage. If DFSORT were to use Hipersorting indiscriminately, there could be a significant increase in paging activity and a resulting reduction in total system performance, affecting all jobs, including sorts.

The recommended setting for HIPRMAX is OPTIMAL. With dynamic Hipersorting, DFSORT has full control over total Hipersorting activity, and sites can customize their definition of HIPRMAX=OPTIMAL with the installation options EXPMAX, EXPOLD, and EXPRES. HIPRMAX=n and HIPRMAX=p% should be reserved for use as a run-time override for applications that have a special reason to limit the amount of Hiperspace available for Hipersorting.

The number of kilobytes of Hiperspace storage used during the sort is displayed in message ICE180I. If Hipersorting is not used, the message shows a value of 0K.

Limitations

The maximum amount of Hiperspace used by DFSORT is limited to the minimum of the following values:

- The IEFUSI exit limit on the total amount of Hiperspace and data space that can be allocated in a single job step. See *z/OS MVS Installation Exits* for a description of IEFUSI.
- The HIPRMAX value in effect, when set to a value other than OPTIMAL. The HIPRMAX value in effect is either the installation default, an overriding value specified at run-time, or an overriding value specified in the installation ICEIEXIT routine. Note that the value specified in the ICEIEXIT routine overrides any other value.
- Available storage. Throughout the run, DFSORT determines the pages available on the system, subtracts from this the amount of storage needed by other concurrent Hipersorting applications, and factors in the values specified for installation options EXPMAX, EXPOLD, and EXPRES. If as a result of any such check, either a storage shortage is predicted or one of the site limits for total storage usage by all Hipersorting applications is reached, DFSORT switches from using Hiperspace to using disk work data sets for all currently running Hipersorting applications.

  In addition, all future DFSORT applications are prevented from using Hipersorting until the storage situation is relieved. This prevents Hipersorting applications by themselves from causing excessive paging.

  Since this last criteria depends very heavily on system activity, especially other concurrent Hipersorting and memory object sorting activity, DFSORT applications can use varying amounts of Hiperspace when run at different times and under different conditions. In fact, it is possible for such applications to not use any Hiperspace.

The following are those cases for which you should not attempt to adjust your application; in these cases the best performance for the individual job and for the system is achieved by not using Hipersorting:

- **Other performance features are in use.** Hipersorting is not used when DFSORT decides to use dataspace or memory object sorting. DFSORT dynamically chooses between using dataspace sorting, memory object sorting and Hipersorting; DFSORT selects the one that provides the best performance for the particular sort. Messages ICE188I and ICE199I indicate whether dataspace sorting or memory object sorting was used for a particular run.
- **The size of the input data set is very small.** If the amount of data to be sorted is known to be small enough that the sort can be accomplished in main memory, Hipersorting is not used. Since no intermediate data is generated, neither Hiperspace nor disk work data sets are needed. The presence of message ICE080I indicates that a sort was processed in main memory.

Application adjustments

The following are those cases for which you may want to adjust your application in order to take advantage of Hipersorting.

- **The Blockset technique was not selected**. Hipersorting is supported only for the Blockset technique. If Blockset is not selected, message ICE800I indicates why it was not selected.

  Note that the ICE800I message is printed only when a SORTDIAG DD statement was coded in the sort's JCL, or installation option DIAGSIM=YES has been specified for your site. Use the ICE800I reason code to determine the exact condition that is preventing the use of Blockset. If you are interested in using Hipersorting for the job, change your application appropriately to eliminate the particular condition, so that Blockset can be used.
- **Insufficient available virtual storage.** In some cases, the amount of virtual storage available to DFSORT can influence the potential effectiveness of a Hiperspace. A Hiperspace of a certain size could be too small to improve performance when an insufficient amount of virtual storage is available, whereas the same size Hiperspace might be large enough to improve performance when a sufficient

amount of storage is available. Since DFSORT does not use Hiperspace when doing so would not result in a performance benefit, insufficient virtual storage can indirectly prevent the use of Hipersorting.

Supply DFSORT with sufficient virtual storage if you would like Hipersorting to be used. The third value in message ICE092I or ICE093I indicates the amount of storage available for a particular sort job. To help reduce the likelihood of not using Hipersorting because of insufficient virtual storage, ensure that this value is at least the maximum recommendation given in Table 4 on page 43. If necessary, increase the amount of virtual storage available to the job by specifying a larger MAINSIZE value on the OPTION control statement and/or raising the REGION value on the sort step EXEC statement.

- **Insufficient available central storage.** The size of the input data set in relation to the total available central storage has an important effect on the performance of Hipersorting. If the size of the Hiperspace that could be created is too small to hold a significant percentage of the intermediate data, then the performance of the run would be degraded compared to using disk-only mode. Therefore, DFSORT chooses not to use Hipersorting in this situation.

  If you would like Hipersorting to be used, there are several possible approaches you can take:

  – Make sure that the HIPRMAX value or the installation IEFUSI exit is not limiting the application to a small amount of Hiperspace. Setting HIPRMAX=OPTIMAL (or to a very large value) and having IEFUSI allow at least 2 GB of Hiperspace per application will remove this limitation.

  – Make sure that the EXPMAX, EXPOLD, and EXPRES values allow significant amounts of Hipersorting. This is accomplished by setting EXPMAX and EXPOLD to large values (or MAX) and EXPRES to a small value.

  – Rerun the application when system activity, especially other concurrent Hipersorting and memory object sorting activity, is lower so that more storage is available for the sort. The more storage available, the larger the Hiperspace that can be created by DFSORT, and the larger the data set size for which Hipersorting can be allowed.

  – In situations where business critical applications are executing concurrent with non-critical applications, it may be desirable to limit Hiperspace usage by a non-critical sort application to leave resources available for the business critical sort applications.

    Remember that some data sets are so large that Hipersorting can **never** be used to sort them, even if all of the storage installed on the system were available for the sort. To allow Hipersorting in such cases, you can either break up the large sort into multiple smaller sorts, or install more storage on the system.

  – Reduce the size of the input data set, so that less storage is required for the sort. For some applications it is not necessary to sort all of the data, since only a subset is needed for processing. For example, INCLUDE, OMIT, SKIPREC, or STOPAFT can significantly reduce the amount of intermediate storage required by DFSORT. See *z/OS DFSORT Application Programming Guide* for more details about these features of DFSORT.

  – Ensure that DFSORT has accurate information about the input file size. DFSORT can automatically estimate the file size for disk input data sets and tape data sets managed by DFSMSrmm or a tape management system that uses ICETPEX. However, there are certain situations, which DFSORT reports with message ICE118I, in which DFSORT cannot determine the file size. See "File Size and Dynamic Allocation" in *z/OS DFSORT Application Programming Guide* for more information on these situations, and what to do about them.

  – The parameters that control the system resources manager (SRM) can indirectly affect the amount of storage that is available for all the jobs on your system, including sort jobs. For example, PWSS=(0,100) may cause DFSORT to not use Hipersorting. See *z/OS MVS Initialization and Tuning Reference* for information about SRM and its parameters.

- **Inefficient work data set usage.** When a Hiperspace-mixed mode run is possible, DFSORT must decide how best to use both Hiperspace and disk work data sets. In most cases, trade-offs can be made such that both types of intermediate storage can be used efficiently. In some cases, however, it is impossible to use both Hiperspace and disk work data sets efficiently, in which case DFSORT chooses not to use Hipersorting.

In order to avoid such cases, use only 3390 or later model disks, and supply sufficient virtual storage to DFSORT, as described in Table 4 on page 43. Sometimes, it is necessary to rerun the jobs when there is less system activity (to allow selection of Hiperspace-only mode) in order to take advantage of Hipersorting.

In general, DFSORT takes into account the potential effects of using Hiperspace on both the application's performance and the system's performance when determining whether or not to use Hipersorting. If either effect is not desirable, DFSORT chooses not to use Hipersorting.

See "Memory object sorting, hipersorting and data space sorting" on page 6 for information on the benefits and operation of Hipersorting and "Hipersorting, memory object sorting, and data space sorting" on page 23 for additional information on using Hipersorting effectively.

Sorting with data space

Dataspace sorting is a DFSORT capability that uses data space. See "Memory object sorting, hipersorting and data space sorting" on page 6 for a detailed description of dataspace sorting.

The DSPSIZE parameter

The DSPSIZE parameter specifies the maximum amount of data space to be used with dataspace sorting. The maximum size of the data space allocated by DFSORT for a job is determined by the minimum of the following values:

- The limit placed on the size of the data space by the system's IEFUSI installation exit. For a description of IEFUSI, refer to *z/OS MVS Installation Exits*.
- The DSPSIZE value (either the installation value, or an override value specified at run-time).
- The amount of system paging activity at the start of the run. The size of the data space can be adjusted throughout the run. DFSORT determines the optimal data space size, based on system activity, to help avoid a negative impact on system paging when using dataspace sorting. If the system paging levels are high, DFSORT's data space size limit will be low.
- The limit of 2 gigabytes placed on the size of a single data space.

How DFSORT uses data space

The recommended setting for DSPSIZE is MAX. This enables DFSORT to dynamically determine the size of a data space to be used for data space sorting, taking into account the size of the file being sorted and the central storage usage activity of the system. DFSORT has full control over total data space sorting activity, and sites can customize their definition of DSPSIZE=MAX with the installation options EXPMAX, EXPOLD, and EXPRES. DSPSIZE=n and DSPSIZE=p% should be reserved for use as a run-time override for applications that have a special reason to limit the size of a data space for data space sorting.

The amount of data space storage used for each sort job is displayed in message ICE188I. DFSORT dynamically determines how best to use data spaces for each particular run:

- If the input data set size is known to be small enough so that the sort can be accomplished in main storage, no data space is created.
- If the size of the input data set in relation to the maximum available data space amount is too large, no data space is created. Dataspace sorting is only used if the size of the data space that could be created is large enough to improve the performance of the sort application.
- If the size of the input data set in relation to the total available main storage is too large, no data space is created. To ensure that you have enough main storage to use dataspace sorting, follow the recommended virtual storage guidelines for DFSORT (Table 5 on page 44).

In general, DFSORT takes into account the effect on the application's performance and the effect on the system's performance before using data space. If either effect is not desirable, DFSORT chooses not to use dataspace sorting.

See “Memory object sorting, hipersorting and data space sorting” on page 6 for information about the benefits and operation of dataspace sorting, and “Hipersorting, memory object sorting, and data space sorting” on page 23 for additional information on using data space sorting effectively.

Cache Fast Write

With DFSORT, cache fast write (CFW) refers to the capability of DFSORT to take advantage of the storage control unit's cache fast write function when writing to the work data sets. The recommended setting for cache fast write is CFW=YES. If you want to change the CFW setting for a specific application, you can use the CFW or NOCFW options of the DEBUG statement at run-time for that application.

See “Cache Fast Write (CFW)” on page 9 for information on the benefits and operation of cache fast write and “Hipersorting, memory object sorting, and data space sorting” on page 23 for additional information on using cache fast write effectively. Note that some control units will override this setting and determine for themselves whether or not to cache the data.

File size

The input file size is important for sort applications, since it is used for several internal optimizations as well as for dynamic work data set allocation. DFSORT can automatically estimate the file size for disk input data sets and tape data sets managed by DFSMSrmm or a tape management system that uses ICETPEX. However, there are certain situations, which DFSORT reports with message ICE118I, in which DFSORT cannot determine the file size. See "File Size and Dynamic Allocation" in *z/OS DFSORT Application Programming Guide* for more information on these situations, and what to do about them.

Storage

DFSORT sorts most efficiently when sufficient virtual storage is available to enable an optimal balance between placing data in virtual and auxiliary storage. When virtual storage is limited, DFSORT must expend more resources to transfer data between virtual and auxiliary storage, which causes increased CPU time, elapsed time, and I/O usage.

Sufficient real storage must be available to support DFSORT's virtual storage requirements. Supplying DFSORT with more virtual storage might not improve performance if the available system resources cannot accommodate the corresponding increase in virtual storage activity. If real storage resources become overcommitted, excessive paging can result. This can cause the performance of both DFSORT and the system to degrade. It is important, therefore, to balance virtual storage resources supplied to DFSORT with the overall system resource requirements.

DFSORT's Dynamic Storage Adjustment (DSA) feature can let DFSORT tune the right amount of virtual storage for sort applications relieving, system and application programmers of the task. See “Dynamic storage adjustment ” on page 9 for more information on the benefits and operation of DSA.

See “Virtual storage ” on page 15 for a description of virtual storage.

Data set size and virtual storage

The relationship between data set size and amount of virtual storage available is critical to the performance of DFSORT. Basically, there are four separate cases to consider.

- When virtual storage is larger than the data set, DFSORT may be able to perform the sort entirely within virtual storage, without need to store intermediate data. This is called an *in-main-storage sort*. Indeed, this is the preferred method for sorting small data sets, since it minimizes I/O usage as well as CPU and elapsed time.
- When virtual storage is smaller than the data set, Hiperspace or work data sets are needed to store the intermediate data. Provided virtual storage is sufficient (see Table 4 on page 43 for guidelines), DFSORT is still able to perform an efficient sort, with elapsed and CPU times close to those of an in-

main storage sort. I/O or Hiperspace usage is increased, however, reflecting the need to write intermediate data to work data sets or Hiperspace.

- When virtual storage is reduced further or the data set size is increased, DFSORT is forced to make less efficient use of Hiperspace or work data sets. DFSORT does what it can to maintain performance but is forced to use Hiperspace or work data sets less efficiently as the ratio of data set size to available storage increases.The loss of efficiency adversely affects elapsed time and EXCP counts.

  This performance degradation can be especially dramatic when using work data sets allocated on devices attached to non-synchronous storage control units or connected to ESCON channels. In such cases, it is especially important to follow the virtual storage guidelines explained in "Virtual storage guidelines" on page 43.
- When virtual storage is very small or the data set size is very large, DFSORT may require several additional passes over the data to perform the sort. This phenomenon is known as *intermediate merging*. DFSORT issues message ICE247I to indicate intermediate merging was required; processing continues with degraded performance. Figure 6 on page 42 shows the benefit of increasing virtual storage to eliminate intermediate merging.

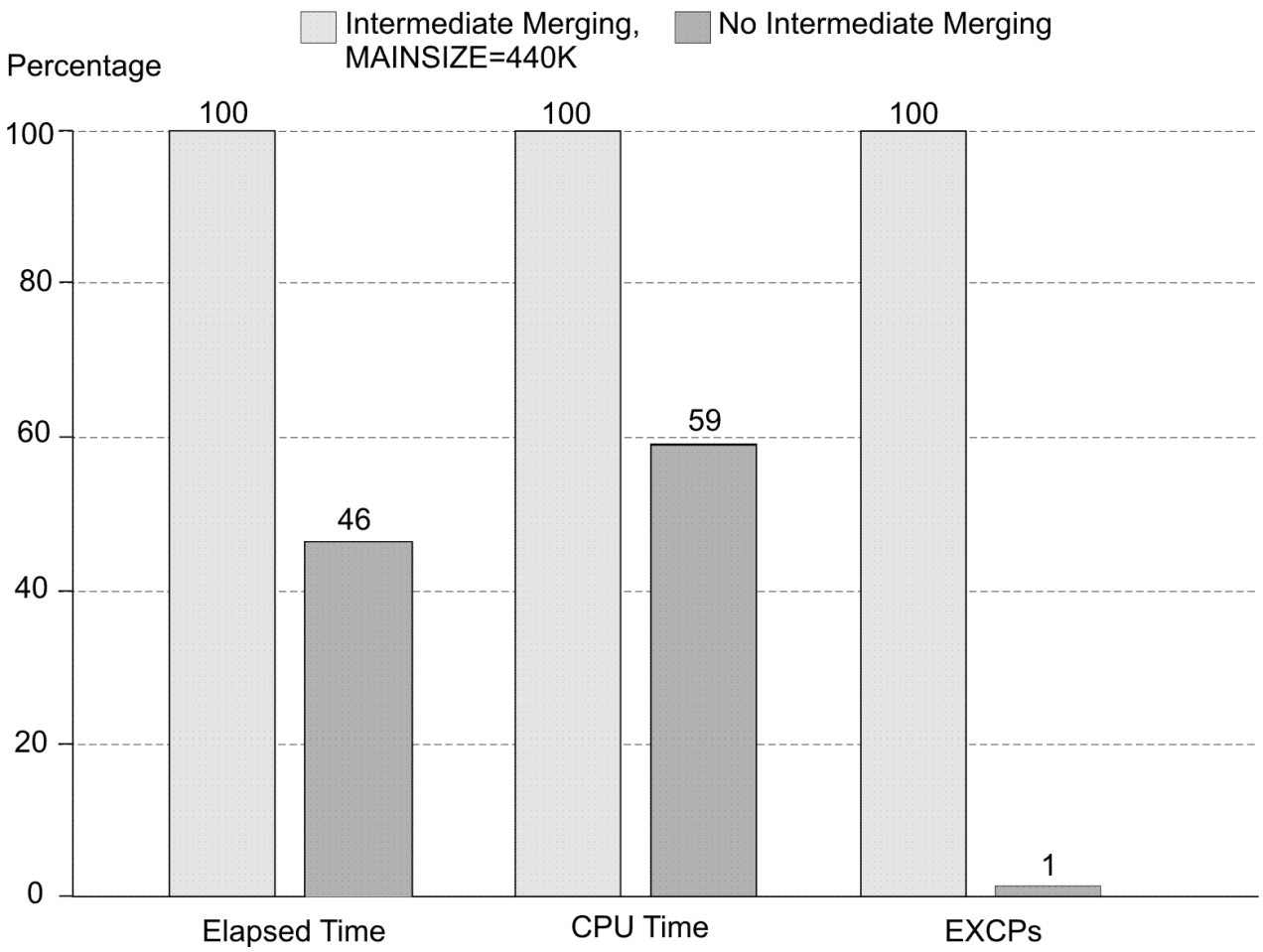

Figure 6: Benefits of Eliminating Intermediate Merging

All other factors being equal, the range of data set sizes that DFSORT can sort efficiently (or sort without requiring intermediate merging) grows roughly as the square of the virtual storage size. That is, doubling the virtual storage in an application enables the application to handle data sets four times as large with the same degree of efficiency. Likewise, halving the virtual storage causes the application to handle data sets only one-fourth as large with the same efficiency.

Virtual storage limitations

With the possible exception of in-main storage sorts, providing more storage than needed to do an efficient sort (see Table 4 on page 43 for storage guidelines) will probably not result in any significant performance improvement. In fact, elapsed time (and possibly CPU time) may even increase slightly. While this degradation might not be very noticeable, increasing virtual storage increases the overall effect DFSORT has on the system by tying up more central storage than necessary. This can result in fewer jobs being able to run at the same time as well as increased paging activity on the system.

If user exit routines are used, they will affect DFSORT virtual storage requirements. The exit routines will occupy virtual storage, and any storage requests they issue will reduce the amount of storage available to DFSORT. The MODS control statement should be used to reserve storage for exit routines.

If the storage available to DFSORT below 16 MB virtual is severely limited (for example, to less than 256 KB), the use of any of the following can result in storage failures or terminations:

- Spanned records
- COBOL exit routines
- CHALT, LOCALE, or SMF options
- ALTSEQ, INCLUDE, OMIT, SUM, OUTFIL, OUTREC, or INREC control statements
- Very large blocks or logical records
- VSAM data sets
- An Extended Function Support (EFS) program
- An ICETEXIT routine
- A large ICEIEXIT routine
- A large number of JCL or dynamically allocated work data sets
- Processing Unicode data formats

You can avoid storage problems and achieve better DFSORT performance by making sure MINLIM is always set to a reasonable value (for example, the supplied default of 440 KB) and by using SIZE/MAINSIZE=MAX with DSA at 32 or more (the supplied default is 64), or SIZE/MAINSIZE=nM with n set to at least the minimum value recommended in Table 4 on page 43.

The DSA run-time option can be used to override the DSA installation option for a particular sort application.

Virtual storage guidelines

DFSORT's Dynamic Storage Adjustment feature (DSA) provides the best use of virtual storage for large sorts. However, if DSA cannot be used because the file size is unknown (as indicated by message ICE118I) or because SIZE=MAX is not in effect, the following guidelines can be used to determine the best amount of virtual storage for the sort.

Table 4 on page 43 gives guidelines for the recommended minimum virtual storage to use for a sort application based on its data set size. If you do not know the data set size, you can run the application and look at message ICE134I. The table gives a range of virtual storage sizes for each possible data set size. The low end of each range should produce an efficient sort for the given data set size. The high end, in some cases, will enable an even more efficient sort. Using less than the low end will likely produce noticeable degradation while using more than the high end will probably not have a significant impact on performance.

Table 4: Recommended Minimum Storage Guidelines for Sorting Without Data Space.

Input Data Set Size	Recommended Minimum Storage
Less than 50 MB	4 MB

Table 4: Recommended Minimum Storage Guidelines for Sorting Without Data Space. (continued)

Input Data Set Size	Recommended Minimum Storage
50 MB to 100 MB	4-6 MB
100 MB to 200 MB	4-8 MB
200 MB to 500 MB	6-12 MB
500 MB to 1 GB	8-16 MB
1 GB to 2 GB	12-24 MB
More than 2 GB	16-32 MB

In order to guarantee the most efficient sorting, use the higher end of the range shown. In order to guarantee efficient, but not necessarily optimum sorting, use the lower end. These values are intended for sorting without data space. See Table 5 on page 44 for storage recommendations for sorting with data space.

Although sort applications can usually run with less virtual storage than the recommended minimum, the recommended amount enables DFSORT to sort most efficiently. Using less than the recommended amount can result in the effects described in “Data set size and virtual storage ” on page 41.

Virtual storage and sorting with data space or memory objects

Dataspace sorting and memory object sorting have a different set of guidelines regarding virtual storage. For one thing, dataspace sorting creates and uses a data space to hold the records currently being processed. The size of this data space is chosen to be large enough to guarantee an efficient sort. Otherwise, dataspace sorting is not used. The same applies to memory object sorting with regards to a memory object.

As a result of the ability of dataspace sorting or memory object sorting to adjust its virtual storage requirements dynamically to the data set being sorted, and the fact that the virtual storage made available through the data space or memory object is in addition to the virtual storage available to DFSORT normally (through the SIZE or MAINSIZE parameter), the guidelines in Table 4 on page 43 are not applicable to dataspace sorting or memory object sorting. Instead, use the values found in Table 5 on page 44 for dataspace sorting or memory object sorting applications.

Table 5: Recommended Minimum Storage Guidelines for Sorting with Data Space or Memory Objects.

Input Data Set Size	Recommended Minimum Storage for Dataspace Sorting or Memory Object Sorting
Less than 200 MB	4 MB
200 MB to 500 MB	4-6 MB
500 MB to 1 GB	4-8 MB
1 GB to 2 GB	4-10 MB
More than 2 GB	4-12 MB

In order to guarantee the most efficient sorting, use the higher end of the range shown. In order to guarantee efficient, but not necessarily optimum, sorting, use the lower end. These values are intended for sorting with data space or memory objects. See Table 4 on page 43 for storage recommendations for sorting without data space or memory objects.

Input and output data sets

The performance of DFSORT can be affected by your choice of block sizes, the types of devices for input and output data sets, user exits, VIO, and some enhancements for input and output data sets. The sections that follow describe these items in more detail.

Block sizes

Choosing an efficient block size can improve space utilization and I/O performance. DFSORT's SDB=LARGE, SDB=INPUT or SDB=YES installation option allows DFSORT to automatically select the system determined optimum block size for output data sets as long as you do not specify the BLKSIZE explicitly. Use one of these installation options, as appropriate, for optimum space utilization and performance. SDB=INPUT is the supplied default. See *z/OS DFSORT Installation and Customization* for details about the SDB installation option values. You can use the SDB run-time option to override the SDB installation option when appropriate for particular jobs.

Space utilization

The amount of space on a track or cylinder occupied by user data depends on the block size specified for the data set. Grouping logical records into blocks reduces the amount of space needed to store data. Because fewer physical records are needed to store the same number of logical records, the amount of space for count and key areas, and for gaps between records is reduced.

Larger block sizes offer better opportunities for increased disk space utilization. An appropriately selected block size can result in higher space utilization than a smaller block size. An inappropriately selected block size (large or small) can result in poor space utilization.

Figure 7 on page 46 shows the 3390 space utilization with various block sizes for a fixed-blocked data set with a logical record length of 160.

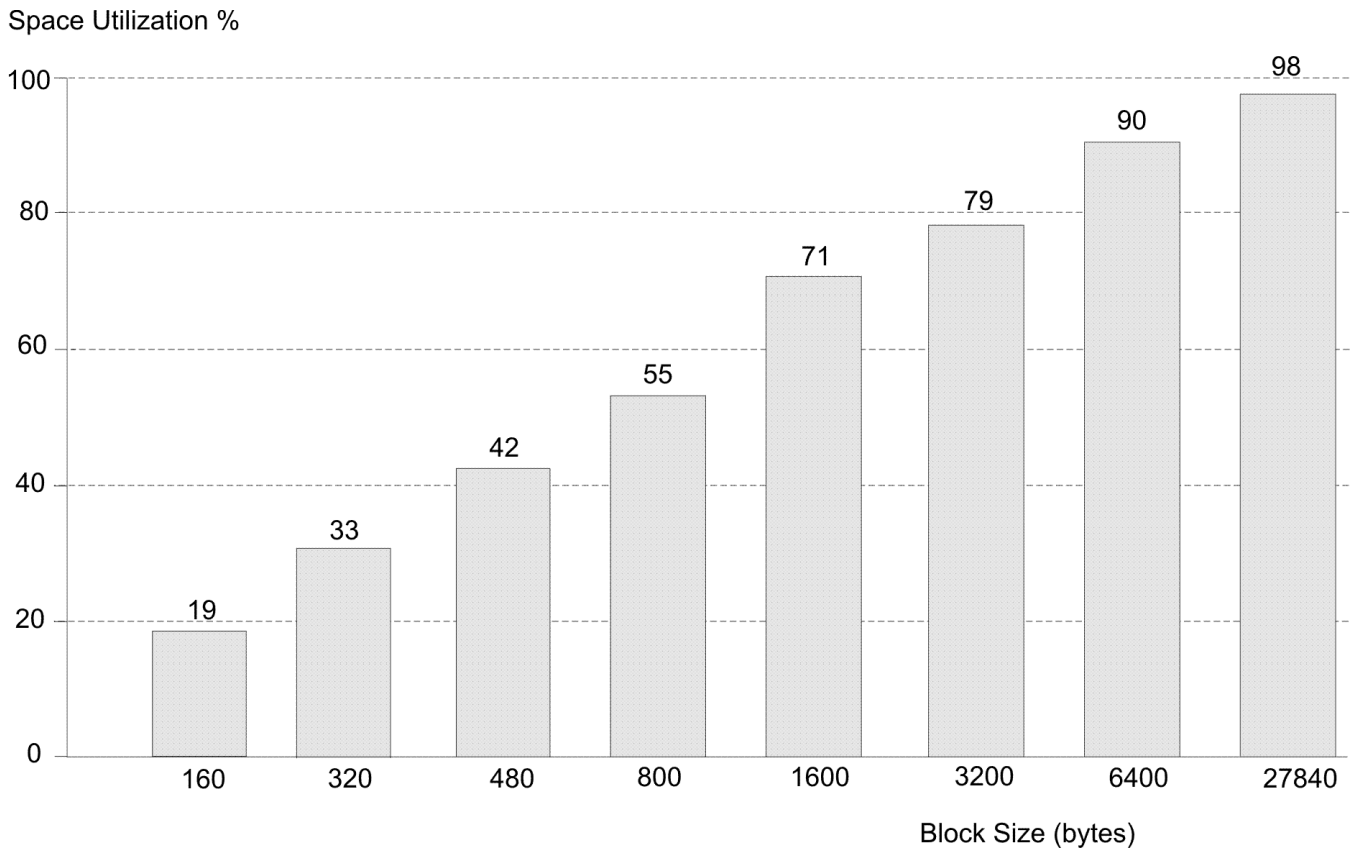

Figure 7: 3390 Utilization for Various Block Sizes

I/O performance

Although small block sizes permit more concurrent channel operations, they reduce the net data transfer rate (the actual amount of data transferred per second). This can impact the elapsed time of a DFSORT application performing a significant amount of I/O. Small block size transfer also requires more CPU involvement and can, therefore, increase CPU time.

Large block sizes enable a higher net data transfer rate for sequential data sets, such as for input and output, and reduce the amount of processor time needed to service a channel program. For tapes, larger block sizes (up to 256 megabytes), provide the best performance and are recommended. Always use system determined blocksizes (SDB) for the best utilization and performance. An example of the benefits of appropriately large input and output data set block sizes is shown in Figure 8 on page 47.

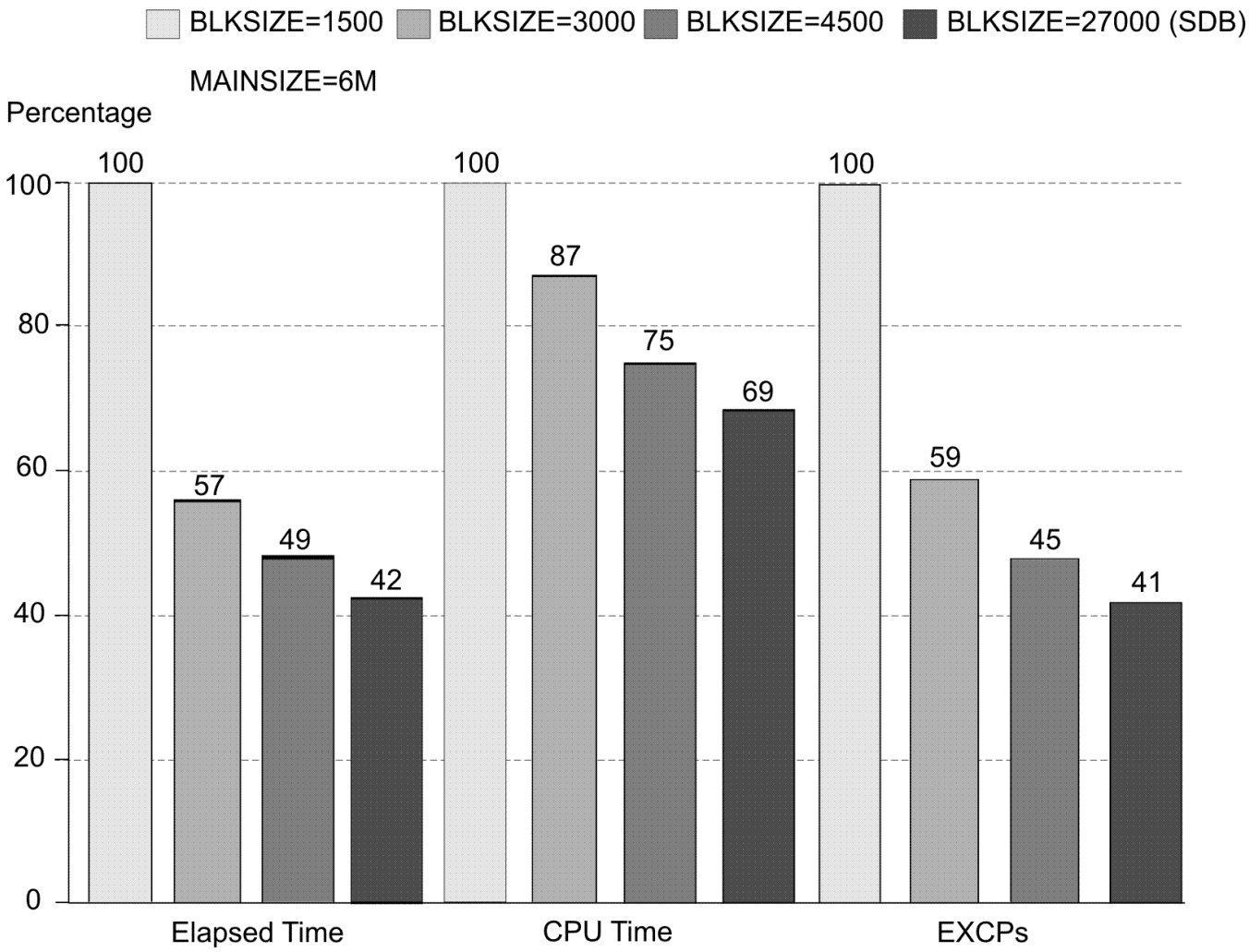

Figure 8: Benefits of Large Input/Output Data Set Block Sizes

Recommendations

When selecting a block size for input or output, consider these factors:

- Smaller data set sizes generally result in less efficient use of disk space.
- DFSORT applications that process data sets with small block sizes will generate higher EXCP counts and probably increase CPU time.

Type of device

For optimal performance, use high performance devices such as IBM's Enterprise Storage Server (ESS) or TotalStorage DS8000 for input, output, and work data sets to gain the advantages of higher data transfer rates and multiple path access. Other ways of improving DFSORT processing of the input and output data sets are as follows:

- Use multiple channel paths
- Allocate enough primary space for the output data sets to avoid the need for additional extents.
- Use separate devices for the input and work data sets, and for the output and work data sets. (DFSORT application data sets that are accessed concurrently should reside on separate devices.)

VIO for DFSORT data sets

DFSORT temporary data sets allocated to virtual devices (VIO) can provide significant elapsed time improvements. However, the trade-off for improving elapsed time using VIO is a serious CPU time degradation.

VIO is recommended for DFSORT input and output data sets only. While DFSORT can use VIO for work data sets, Hiperspace is the preferred method of using storage to eliminate I/O to work data sets on auxiliary storage.

In a DFSMS environment, data sets used by DFSORT might be allocated to virtual devices by the automatic class selection (ACS) routines, overriding the VIO installation option in some cases. "System-managed storage" on page 16 explains how the ACS routines can be changed to avoid VIO allocation for DFSORT temporary data sets.

Input and output data set enhancements

You can also use certain enhancements for input and output data sets to improve performance. These enhancements include compression and striping. See "Extended format datasets" on page 11 for a more detailed description of each of these items.

Using compression and striping affects performance as follows:

- The time needed to transfer data is decreased, sometimes dramatically.
- The time needed to perform the DFSORT application is decreased, sometimes dramatically.
- Work data set I/O is much more likely to be a bottleneck in sort applications that use these enhancements. To eliminate the need for work data set I/O when using compression or striping, do one of the following:
  - Use Hipersorting for all sorting, or
  - Sort entirely in main storage or data space for small to medium size sorts.

Run-time options and performance

Table 6 on page 48 shows run-time options that influence the performance of DFSORT, a description of each option, any restrictions, the IBM-supplied installation default value and a possible reason for modifying that value at run-time. Any IBM-supplied installation default value can be changed to a different site installation default value using ICEPRMxx members in PARMLIB or the ICEMAC macro.

Table 6: Run-time Options That Influence DFSORT Performance.

Run-time Option	Description	Restriction	IBM-supplied Default Value and Reason for Modifying
SIZE and MAINSIZE	Upper limit for total storage above and below 16 MB virtual.	Limited by TMAXLIM or MAXLIM when SIZE=MAX or MAINSIZE=MAX is in effect.	The default is MAX. Modify when sorting unusually large data sets.
DSA	Upper limit for dynamically adjusted storage.	Only used when SIZE=MAX or MAINSIZE=MAX is in effect and DFSORT can determine that performance will benefit.	The default is 64 MB. Set to a lower value if you do not want DFSORT to adjust storage up to 64 MB.
RESALL	Storage below 16 MB virtual that is reserved for system use.	Only used when SIZE=MAX or MAINSIZE=MAX is in effect. Can reduce the amount of virtual storage available for use by DFSORT.	The default is 4 KB. Modify when sufficient REGION is specified but application terminates for lack of below 16 MB virtual storage.

Table 6: Run-time Options That Influence DFSORT Performance. (continued)

Run-time Option	Description	Restriction	IBM-supplied Default Value and Reason for Modifying
RESINV	Storage below 16 MB virtual that is reserved for use by an invoking program.	Only used when SIZE=MAX or MAINSIZE=MAX is in effect and DFSORT is program-invoked.	The default is 0.
ARESALL	Storage above 16 MB virtual that is reserved for system use.	Can reduce the amount of virtual storage available for use by DFSORT.	The default is 0.
ARESINV	Storage above 16 MB virtual that is reserved for use by an invoking program.	Only used when DFSORT is program-invoked.	The default is 0.
HIPRMAX	Upper limit for Hiperspace for a single application.	Hiperspaces limited by IEFUSI installation exit.	The default is OPTIMAL. Set to 0 to disable Hipersorting.
DSPSIZE	Upper limit for data space size.	Data spaces limited by IEFUSI.	The default is MAX. Set to 0 to disable dataspace sorting.
MOSIZE	Maximum size of a memory object for a single application.	Memory objects limited by MEMLIMIT JCL and IEFUSI installation exit.	The default is MAX. Set to 0 to disable memory object sorting.
MOWRK and NOMOWRK	Whether memory objects can be used as intermediate work storage.	NOMOWRK can cause an increase in CPU time.	The default is MOWRK. Use NOMOWRK only when necessary
DYNALLOC	Requests dynamic allocation and specifies device type and number of work data sets.		The default is (SYSDA, 4). The unit name can be changed to a name used exclusively by DFSORT to isolate work data set volumes from the general pool of devices.
DYNAPCT	Default additional work data sets (as percentage) to be dynamically allocated.	Allocated with zero space and only used when needed.	The default is 10. Increase if failures due to insufficient work space are a concern. If set to OLD, additional work data sets are not allocated.
DYNSPC	Default amount of space to allocate for dynamically allocated work data sets.	Only used when DFSORT cannot determine the input file size.	The default is 256 MB. Set to a higher value to allow larger sorts when DFSORT cannot determine the file size.
CINV and NOCINV	Whether control interval access is used for VSAM input data sets.	Improves performance for VSAM input data sets.	The default is CINV.
CFW and NOCFW	Whether cache fast write is used for DFSORT work data sets.	Only applicable for work data sets located on disks attached to cached 3990 storage control units. Can be used only if DFSORT SVC is installed. Cache fast write can reduce the elapsed time of sorting applications. CFW is more beneficial to DFSORT performance for small and intermediate sized sorts.	The default is CFW. Use NOCFW for large sorts.
EQUALS and NOEQUALS	Whether input order is preserved for records with equal keys.	Can cause an increase in CPU time.	The default is VLBLKSET. Use NOEQUALS whenever possible.
VERIFY and NOVERIFY	Whether output records are checked for correct order.	Can cause an increase in CPU time.	The default is NOVERIFY. Use VERIFY only when necessary.

Table 6: Run-time Options That Influence DFSORT Performance. (continued)

Run-time Option	Description	Restriction	IBM-supplied Default Value and Reason for Modifying
ALTSEQ	Whether a collating sequence other than EBCDIC is used.	Can cause an increase in CPU time.	There is no default value. Use ALTSEQ only when necessary.
AVGRLEN	Specifies the average input record length in bytes for variable-length sort applications.	This value is used when necessary to determine the input file size. The resulting value is important for sort applications, since it is used for several internal optimizations as well as for dynamic work data set allocation (see OPTION DYNALLOC).	There is no default value. Using a value close to the actual average record length may improve variable-length record sort performance.
FILSZ	Specifies either the exact number of records to be sorted or an estimate of the number of records to be sorted.	The type of value specified can have a significant effect on performance and work data set allocation. See "File size" on page 41 for more information.	There is no default value. Using a value close to the actual file size may improve sort performance when message ICE118I is received.
ODMAXBF	The maximum buffer space to be used for each OUTFIL data set.	Lowering the value can cause performance degradation for the application. When you use more than 2 MB, the performance improvements are small except for EXCPs, and, there is an increased need for storage.	The default is 2 MB. When you are running OUTFIL applications with a large number of output data sets and constrained storage, use a smaller value to reduce total virtual storage usage.
NOBLKSET	Controls the use of Blockset techniques.	If necessary, DFSORT can still use non-Blockset techniques for other reasons. Using non-Blockset techniques significantly degrades performance.	There is no default value. Specify NOBLKSET only temporarily to bypass a Blockset problem.
CHALT and NOCHALT	Specifies whether ALTSEQ is to be applied to character format fields (CH).	As with ALTSEQ, can cause an increase in CPU time.	The default is NOCHALT. Use CHALT only when necessary.
COBEXIT	Specifies the library for COBOL E15 and E35 routines.	COBEXIT=COB1 is obsolete.	The default is COB2. Use COB1 only if necessary for compatibility.
COLLKEY	Specifies whether Unicode data processing is to be used and, if so, designates the active collation rules version.	Should only be used when required since it can show degraded performance relative to collation using character encoding values.	The default is UCA600. Use COLLKEY processing only when necessary for your application.
EFS	Specifies the name of a user-written Extended Function Support program to be called by DFSORT.	Can cause an increase in CPU time and elapsed time.	The default is NONE. Use an EFS program only when necessary for your application.
LOCALE	Specifies whether locale processing is to be used and, if so, designates the active locale.	Should only be used when required since it can show degraded performance relative to collation using character encoding values.	The default is NONE. Use LOCALE processing only when necessary for your application.

See *z/OS DFSORT Application Programming Guide* for a complete list of run-time override options.

Chapter 6. Application considerations

Most sites have many applications involving sorting. DFSORT is often used by these applications either directly by invoking DFSORT with JCL, or indirectly by invoking DFSORT from a program. In particular, the COBOL SORT statement, and the PL/I sort routines result in calls to DFSORT (see the appropriate language documents for complete details).

The way in which COBOL interfaces with DFSORT depends on the use of COBOL features such as FASTSRT, NOFASTSRT, USING, GIVING and INPUT and OUTPUT PROCEDUREs, and DFSORT features such as COBOL exits and DFSORT control statements. In general, the features you use are dictated by the needs of your application. But in many cases, an application can achieve its results using one or another of these features, that is, you have a choice. This chapter describes some of the COBOL and DFSORT features you can choose and the performance and productivity implications of doing so.

See the appropriate COBOL guide if you are not familiar with any of the COBOL features described in this chapter. See *z/OS DFSORT Application Programming Guide* if you are not familiar with any of the DFSORT features described in this chapter. See *z/OS DFSORT: Getting Started* for tutorials on DFSORT control statements and features.

Note: Many of the techniques described in this chapter can also be used with other languages that can call DFSORT such as PL/I and Assembler.

[Programming Interface Information] COBOL interfaces to DFSORT

Understanding the interfaces between DFSORT and languages such as COBOL can help you design more efficient applications. Knowing which interfaces are used enables you to:

- Obtain more information about these interfaces in the DFSORT documents.
- Determine the best way to take advantage of these interfaces.

[End Programming Interface Information]

Invoking DFSORT from COBOL

In order to invoke DFSORT from COBOL, you must code a COBOL SORT statement. This statement has the following variations which affect the way in which DFSORT and COBOL pass information back and forth:

- USING
- GIVING
- INPUT PROCEDURE
- OUTPUT PROCEDURE

In addition, the COBOL compile-time options FASTSRT and NOFASTSRT also affect the interfaces between COBOL and DFSORT.

The interfaces that result from the COBOL SORT statement and the FASTSRT/NOFASTSRT compile-time options are described in “Processing with FASTSRT ” on page 52 and “Processing with NOFASTSRT ” on page 52.

When a COBOL calling program is used, DFSORT control statements can be specified using the COBOL data set defined by IGZSRTCD or the DFSORT data set defined by SORTCNTL or DFSPARM. There are some differences in how these data sets can be used. For example:

- SORTCNTL allows you to specify comment statements, blank statements, remarks, and labels. DFSPARM allows comment statements, blank statements, and remarks, but not labels. IGZSRTCD does not allow comment statements, remarks, or labels, and eliminates blank statements.

- DFSPARM and IGZSRTCD enable you to pass certain DFSORT run-time options (such as MSGDDN) that are ignored in SORTCNTL.
- Using the COBOL special register SORT-CONTROL enables you to pass different IGZSRTCD data sets to DFSORT when you have multiple SORT statements. The statements in IGZSRTCD are actually placed in the parameter list in storage that COBOL passes when it calls DFSORT. A separate parameter list with the appropriate control statements is passed for each call.
- SORTCNTL and DFSPARM enable you to pass DFSORT control statements without increasing the storage used for the parameter list.

For our examples, we use SORTCNTL, although IGZSRTCD or DFSPARM would do just as well.

Processing with FASTSRT

COBOL's FASTSRT compile-time option is a special feature of the language that can improve performance for qualifying applications which use the COBOL SORT statement. You should always specify the FASTSRT option. COBOL decides automatically at compile-time whether FASTSRT can actually be used. For example, FASTSRT cannot be used for input processing when an INPUT PROCEDURE is specified. See the appropriate COBOL guide for the complete list of FASTSRT requirements and restrictions.

When FASTSRT is in effect for input processing, COBOL passes the ddname of the input data set to DFSORT. DFSORT uses this ddname to read the input data set directly.

When FASTSRT is in effect for output processing, COBOL passes the ddname of the output data set to DFSORT. DFSORT uses this ddname to write the output data set directly.

Input or output processing by DFSORT rather than COBOL can result in reductions in CPU time, EXCPs, and elapsed time.

Note: PL/I's PLISRTA subroutine is equivalent to using FASTSRT for both input and output processing. PLISRTB is equivalent to using FASTSRT for output processing, and PLISRTC is equivalent to using FASTSRT for input processing.

Processing with NOFASTSRT

When NOFASTSRT is in effect for input processing, the USING or INPUT PROCEDURE causes COBOL to generate a DFSORT E15 user exit routine and pass its address to DFSORT. When an INPUT PROCEDURE is used, COBOL incorporates the INPUT PROCEDURE code into the E15 routine it generates.

DFSORT does not read the input data set directly, but instead obtains all the input records from the E15 routine. The E15 routine generated by COBOL reads the input data set and passes one record to DFSORT each time it is called.

When NOFASTSRT is in effect for output processing, the GIVING or OUTPUT PROCEDURE causes COBOL to generate a DFSORT E35 user exit routine and pass its address to DFSORT. When an OUTPUT PROCEDURE is used, COBOL incorporates the OUTPUT PROCEDURE code into the E35 routine it generates.

DFSORT does not write the output data set directly, but instead passes all the output records to the E35 routine. DFSORT calls the E35 routine generated by COBOL once for each record so the E35 routine can write the record to the output data set.

Input or output processing by COBOL rather than DFSORT can result in degraded performance.

Note: PL/I's PLISRTD subroutine is equivalent to using NOFASTSRT for both input and output processing. PLISRTC is equivalent to using NOFASTSRT for output processing, and PLISRTB is equivalent to using NOFASTSRT for input processing

Performance

When FASTSRT is in effect for a COBOL sort, DFSORT reads the input data set and writes the output data set rather than COBOL. This results in reductions in elapsed time, CPU time and EXCPs. An example of the benefits of FASTSRT is shown in Figure 9 on page 53. For this comparison, the same COBOL sort with USING and GIVING was run with FASTSRT and NOFASTSRT.

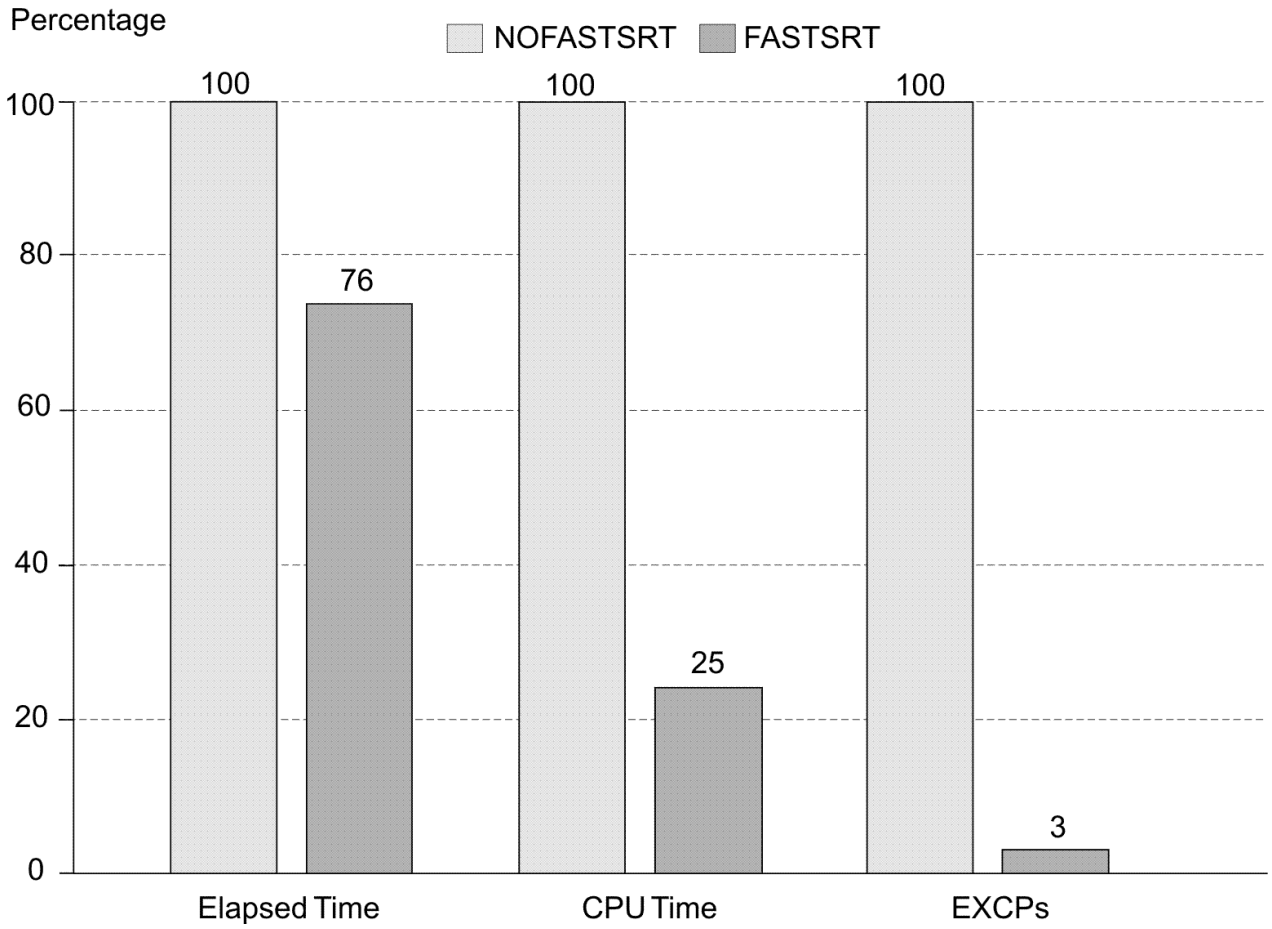

Figure 9: Benefits of FASTSRT

Sample sorting application

The remainder of this chapter describes and compares three different methods for doing the following application:

1. Pre-processing: Delete all input records which have zero in a particular field (call it the OMIT field). The OMIT field is columns 30-34 of the records.
2. Sorting: Sort the remaining records in descending order by a particular character format field (call it the KEY field). The KEY field is in columns 5-24 of the records.
3. Post-processing: Write one output record with each key.

The three methods are as follows:

- Method 1: COBOL program with INPUT/OUTPUT PROCEDUREs
- Method 2: COBOL program with DFSORT control statements
- Method 3: DFSORT with control statements

[Programming Interface Information] Method 1: COBOL program with INPUT/OUTPUT PROCEDUREs

Method 1 uses a SORT statement INPUT PROCEDURE for pre-processing and a SORT statement OUTPUT PROCEDURE for post-processing.

NOFASTSRT is in effect for input and output processing due to the use of INPUT and OUTPUT PROCEDUREs.

An INPUT PROCEDURE or OUTPUT PROCEDURE can add, delete, alter, edit, or otherwise modify the records.

The INPUT PROCEDURE and OUTPUT PROCEDURE are actually special forms of E15 and E35 exits which are called by DFSORT during its processing, but controlled by the COBOL calling program.

The INPUT PROCEDURE is responsible for supplying all of the input records to be sorted to DFSORT, while the OUTPUT PROCEDURE is responsible for disposing of all the records after they are sorted.

[End Programming Interface Information]

COBOL Calling Program for Method 1

```
*-------------------------------------------------------------
*  METHOD 1: COBOL INPUT AND OUTPUT PROCEDURES.
*-------------------------------------------------------------
*  1. PRE-PROCESS:
*     THE PROGRAM USES A SORT INPUT PROCEDURE TO DELETE RECORDS
*     WITH A ZZZZZ OMIT FIELD BEFORE SORTING. THE OMIT FIELD IS
*     IN COLUMNS 30-34.
*  2. SORT
*     THE PROGRAM CALLS DFSORT TO SORT THE RECORDS IN DESCENDING
*     ORDER. THE KEY IS IN COLUMNS 5-24.
*  3. POST-PROCESS:
*     THE PROGRAM USES A SORT OUTPUT PROCEDURE TO WRITE ONE
*     RECORD WITH EACH KEY AFTER SORTING.
*
*  INPUT/OUTPUT: READS INDS AND WRITES OUTDS.
*                DFSORT PASSES RECORDS TO THE PROCEDURES.
*-------------------------------------------------------------
 ID DIVISION.
   PROGRAM-ID. CASE1.
 ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
   FILE-CONTROL.
        SELECT INDS    ASSIGN TO INDS.
        SELECT OUTDS   ASSIGN TO OUTDS.
        SELECT SORT-FILE  ASSIGN TO SORTFILE.
 DATA DIVISION.
   FILE SECTION.
   FD INDS RECORD CONTAINS 160 CHARACTERS
      LABEL RECORD STANDARD BLOCK 27840
      DATA RECORDS ARE INDS-RECORD.
   01 INDS-RECORD.
    05 FILLER        PIC X(4).
    05 INDS-KEY      PIC X(20).
    05 FILLER        PIC X(5).
    05 INDS-OMIT     PIC X(5).
    05 FILLER        PIC X(126).
   FD OUTDS RECORD CONTAINS 160 CHARACTERS
     LABEL RECORD STANDARD BLOCK 27840
     DATA RECORDS ARE OUTDS-RECORD.
  01 OUTDS-RECORD.
   05 FILLER          PIC X(160).
  SD SORT-FILE RECORD CONTAINS 160 CHARACTERS
     DATA RECORD SORT-RECORD.
  01 SORT-RECORD.
   05 FILLER        PIC X(4).
   05 SORT-KEY      PIC X(20).
   05 FILLER        PIC X(136).
```

```
  WORKING-STORAGE SECTION.
  01 FLAGS.
   05 INDS-EOF     PIC X VALUE SPACE.
      88 SFLAG     VALUE "Y".
   05 TEMP-EOF     PIC X VALUE SPACE.
      88 TFLAG     VALUE "Y".
  01 PSTART     PIC 9(1) VALUE 0.
  01 SAVE-KEY   PIC X(20).
   01 TEMP-RECORD.
    05 FILLER        PIC X(4).
    05 TEMP-KEY      PIC X(20).
    05 FILLER        PIC X(136).
   PROCEDURE DIVISION.
   MASTER SECTION.
*-----------------------------------------------------------------
*     CALL DFSORT TO SORT THE RECORDS IN DESCENDING ORDER.
*-----------------------------------------------------------------
       SORT SORT-FILE
        ON DESCENDING KEY SORT-KEY
        INPUT PROCEDURE INPUT-PROC
        OUTPUT PROCEDURE OUTPUT-PROC.
       IF SORT-RETURN > 0
          DISPLAY "SORT FAILED".
       STOP RUN.
*-----------------------------------------------------------------
*     SORT INPUT PROCEDURE:
*       READ INDS.
*       DELETE ALL RECORDS WITH A 'ZZZZZ' OMIT FIELD.
*       SEND ALL OTHER RECORDS TO DFSORT FOR SORTING.
*-----------------------------------------------------------------
   INPUT-PROC SECTION.
      OPEN INPUT INDS
      READ INDS AT END SET SFLAG TO TRUE
      END-READ
      PERFORM UNTIL SFLAG
        IF INDS-OMIT NOT = "ZZZZZ"
          RELEASE SORT-RECORD FROM INDS-RECORD
        END-IF
        READ INDS AT END SET SFLAG TO TRUE
        END-READ
      END-PERFORM.
      CLOSE INDS.
*-----------------------------------------------------------------
*     SORT OUTPUT PROCEDURE:
*       RECEIVE RECORDS FROM DFSORT INTO TEMP.
*       WRITE ONE RECORD WITH EACH KEY TO OUTDS.
*-----------------------------------------------------------------
    OUTPUT-PROC SECTION.
       OPEN OUTPUT OUTDS
       RETURN SORT-FILE INTO TEMP-RECORD AT END SET TFLAG TO TRUE
       END-RETURN
       PERFORM UNTIL TFLAG
         IF PSTART = 0
 *-----------------------------------------------------------------
 *     FIRST RECORD - SAVE KEY AND WRITE RECORD TO OUTDS.
 *-----------------------------------------------------------------
           MOVE TEMP-KEY TO SAVE-KEY
           WRITE OUTDS-RECORD FROM TEMP-RECORD
           MOVE 1 TO PSTART
         ELSE
           IF TEMP-KEY NOT = SAVE-KEY
*-----------------------------------------------------------------
*     RECORD HAS NEW KEY - SAVE KEY AND WRITE RECORD TO OUTDS.
*-----------------------------------------------------------------
            MOVE TEMP-KEY TO SAVE-KEY
            WRITE OUTDS-RECORD FROM TEMP-RECORD
           END-IF
        END-IF
        RETURN SORT-FILE INTO TEMP-RECORD
          AT END SET TFLAG TO TRUE
        END-RETURN
      END-PERFORM.
      CLOSE OUTDS.
```

Operation (NOFASTSRT in effect)

The SORT statement results in a call to DFSORT with a parameter list that contains a SORT control statement and other information.

DFSORT treats the INPUT PROCEDURE as an E15 user exit which must supply all the input records. DFSORT calls the INPUT PROCEDURE once for each input record. The INPUT PROCEDURE reads the input data set and uses RELEASE to pass each record with a non-zero OMIT field to DFSORT.

DFSORT sorts the records passed to it by the INPUT PROCEDURE as requested by the SORT statement passed to it by the calling program.

DFSORT treats the OUTPUT PROCEDURE as an E35 user exit which must dispose of all the output records. DFSORT calls the OUTPUT PROCEDURE once for each sorted record. The OUTPUT PROCEDURE uses RETURN to obtain the records passed from DFSORT and writes one record with each key to the output data set.

Performance

Since the COBOL program's INPUT and OUTPUT PROCEDUREs must, by definition, read and write the data sets, NOFASTSRT is in effect for Method 1. FASTSRT is in effect for the other methods we will describe, providing significant performance improvements.

[Programming Interface Information] Method 2: COBOL program with DFSORT control statements

Method 2 eliminates the COBOL pre-processing and post-processing code by using DFSORT control statements to provide the equivalent functions. An OMIT statement is used instead of an E15, and a SUM statement is used instead of an E35.

FASTSRT is used for input and output processing.

DFSORT provides a powerful set of collating and editing functions available though the use of control statements and options. Collating and editing functions can be used to replace program code. They are designed to adapt to the run-time characteristics of an application in order to provide significant performance benefits.

DFSORT's most significant collating and editing functions are:

- SORT or MERGE: enable you to override the SORT or MERGE control statement generated by the compiler. The override statement can contain the DFSORT year 2000 field format (Y2x).
- INCLUDE or OMIT: enable you to include or delete records whose fields meet certain criteria.
- INREC and OUTREC: enable you to delete and rearrange fields in your records, to insert blanks, zeros, timestamps, sequence numbers and constants in records, to transform characters, to perform lookup and change operations, to edit and convert numeric fields in many ways, and to do arithmetic on numeric fields and decimal constants.
- SUM: enables you to sum fields in records with equal keys and to keep only one record with each key.
- OUTFIL: enables you to perform a wide variety of tasks for multiple output data sets including all of those listed previously for INCLUDE or OMIT and OUTREC, as well as FB to VB conversion, VB to FB conversion, selection of subsets and samples of records, and simple or complex reports.
- SKIPREC and STOPAFT: enable you to delete records at the beginning or end of your data set.

When a COBOL calling program is used, INCLUDE, OMIT, INREC, OUTREC, SUM, and OUTFIL can be specified in the data set defined by IGZSRTCD, SORTCNTL, or DFSPARM. SKIPREC and STOPAFT can be specified on an OPTION statement in the data set defined by IGZSRTCD, SORTCNTL, or DFSPARM.

The figures that follow show the COBOL calling program and DFSORT control statements for Method 2.

[End Programming Interface Information]

Operation (FASTSRT in effect)

The SORT statement results in a call to DFSORT with a parameter list that contains a SORT control statement and other information.

DFSORT reads the input data set and deletes each record with a zero OMIT field as requested by the OMIT statement.

DFSORT sorts the remaining records as requested by the SORT statement passed from the calling program.

DFSORT writes one record with each key to the output data set as requested by the SUM statement.

Productivity

The use of DFSORT editing functions rather than COBOL code reduces considerably the effort required to perform the application. Source code for pre-processing and post-processing is eliminated along with the time to compile and debug the code.

In addition, future changes to the editing functions performed by the application can be made by simply changing the control statements. Access to source code or recompiles are not necessary.

Control statements

SORTCNTL contains the OMIT and SUM statements used with the COBOL calling program.

```
//SORTCNTL DD *
   OMIT COND=(30,5,CH,EQ,C'ZZZZZ')
   SUM FIELDS=NONE
/*
```

Figure 10: DFSORT Control Statements for Method 2

COBOL calling program

```
*-----------------------------------------------------------------
*  METHOD 2: COBOL MAIN PROGRAM.
*-----------------------------------------------------------------
*  1. PRE-PROCESS:
*     A DFSORT OMIT CONTROL STATEMENT DELETES RECORDS WITH A
*     'ZZZZZ' OMIT FIELD BEFORE SORTING. THE OMIT FIELD IS IN
*     IN COLUMNS 30-34.
*  2. SORT
*     THE PROGRAM CALLS DFSORT TO SORT THE RECORDS IN DESCENDING
*     ORDER. THE KEY IS IN COLUMNS 5-24.
*  3. POST-PROCESS:
*     A DFSORT SUM CONTROL STATEMENT WRITES ONE RECORD WITH
*     EACH KEY AFTER SORTING.
*
*  INPUT/OUTPUT: DFSORT READS SORTIN AND WRITES SORTOUT.
*-----------------------------------------------------------------
 ID DIVISION.
  PROGRAM-ID. CASE2.
 ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
    FILE-CONTROL.
       SELECT SORTIN  ASSIGN TO SORTIN.
       SELECT SORTOUT ASSIGN TO SORTOUT.
       SELECT SORT-FILE  ASSIGN TO SORTFILE.
 DATA DIVISION.
   FILE SECTION.
    FD SORTIN RECORD CONTAINS 160 CHARACTERS
       LABEL RECORD STANDARD BLOCK 27840
       DATA RECORDS ARE SORTIN-RECORD.
    01 SORTIN-RECORD.
      05 FILLER          PIC X(160).
    FD SORTOUT RECORD CONTAINS 160 CHARACTERS
       LABEL RECORD STANDARD BLOCK 27840
       DATA RECORDS ARE SORTOUT-RECORD.
    01 SORTOUT-RECORD.
```

```
      05 FILLER          PIC X(160).
    SD SORT-FILE RECORD CONTAINS 160 CHARACTERS
       DATA RECORD SORT-RECORD.
     01 SORT-RECORD.
      05 FILLER        PIC X(4).
      05 SORT-KEY      PIC X(20).
      05 FILLER        PIC X(136).
    WORKING-STORAGE SECTION.
  PROCEDURE DIVISION.
    MASTER SECTION.
*-----------------------------------------------------------------
*     CALL DFSORT TO SORT THE RECORDS IN DESCENDING ORDER.
*-----------------------------------------------------------------
      SORT SORT-FILE
        ON DESCENDING KEY SORT-KEY
        USING SORTIN
        GIVING SORTOUT.
      IF SORT-RETURN > 0
        DISPLAY "SORT FAILED".
      STOP RUN.
```

Performance

Since Method 2 uses USING and GIVING rather than INPUT and OUTPUT PROCEDUREs, FASTSRT is in effect. In addition, by eliminating the overhead related to passing each record between DFSORT and the user exits, and enabling DFSORT to use its highly optimized editing code, Method 2 achieves significant performance gains in CPU time, elapsed time, and EXCPs compared to Method 1. Figure 11 on page 58 shows a performance comparison between Method 1 and Method 2.

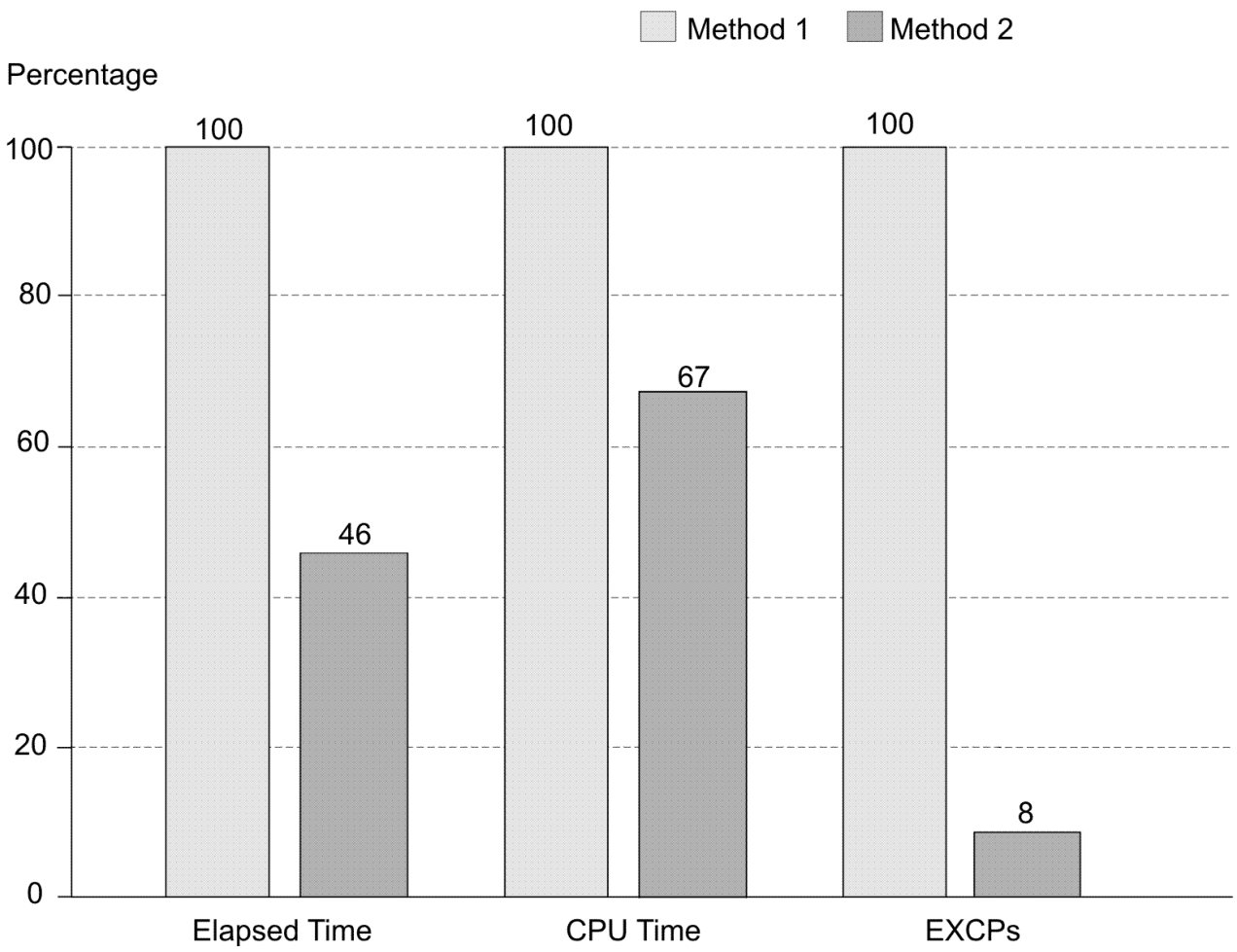

Figure 11: Method 1 versus Method 2 Performance Comparison

Method 3: DFSORT with control statements

Method 3 takes the final step of eliminating the COBOL calling program by invoking DFSORT directly (using PGM=ICEMAN or PGM=SORT on the JCL EXEC statement). Calling DFSORT directly is only feasible if all of the functions of the COBOL program can be duplicated using DFSORT control statements, options, or user exits.

When DFSORT is invoked directly, control statements can be specified using the data set defined by SYSIN or DFSPARM. DFSPARM can be used to specify certain options (for example, MSGDDN) that are ignored in SYSIN. For this example, we use SYSIN, although DFSPARM would do just as well.

The COBOL SORT statement causes COBOL to generate a DFSORT SORT control statement. The equivalent DFSORT SORT control statement must be supplied when DFSORT is invoked directly. When the following JCL statements are added to the COBOL job, the needed equivalent DFSORT SORT control statement is displayed in DFSHOW.

```
//SORTDIAG DD DUMMY
//DFSHOW DD SYSOUT=*
//DFSPARM DD *
  MSGDDN=MYSHOW
/*
```

Control statements

SYSIN contains the OMIT, SORT and SUM statements used with the direct invocation of DFSORT.

```
//SYSIN DD *
   OMIT COND=(30,5,CH,EQ,C'ZZZZZ')
   SORT FIELDS=(5,20,CH,D)
   SUM FIELDS=NONE
/*
```

Figure 12: DFSORT Control Statements for Method 4

Operation

The system calls DFSORT directly.

DFSORT reads the input data set and deletes each record with a zero OMIT field as requested by the OMIT statement.

DFSORT sorts the remaining records as requested by the SORT control statement.

DFSORT writes one record with each key to the output data set as requested by the SUM statement.

Productivity

Elimination of all COBOL code reduces significantly the effort required to perform an application. You can use control statements to duplicate complex code logic quickly and effectively. Source code for pre-processing, sorting, and post-processing is eliminated along with the time to compile and debug the code.

In addition, future changes to the editing and sorting functions performed by the application can be made by simply changing the control statements. Access to source code, and recompiles, are not necessary.

Performance

Performance for Method 3 is comparable to that for Method 2.

Chapter 7. DFSORT performance data

Tuning activity is often started when a particular job is taking too long to complete or is using system resources excessively. But if such a “problem job” does not exist at your site, where do you start to look for ways to improve DFSORT's performance or balance its use of resources with other requirements?

The purpose of this chapter is to outline the actions available for those who want to tune the use of DFSORT at their site. It also shows what information you need about DFSORT, where to find it, and what methods are available for collecting the data, including use of the DFSORT installation exits ICEIEXIT and ICETEXIT. The specific topics include:

- Where to find performance indicators
- An overview of DFSORT performance information
- Sources of DFSORT performance information
- Techniques for analyzing DFSORT performance data
- Balancing DFSORT requirements with system resources
- Understanding trade-offs in improving performance

DFSORT performance indicators

Performance indicators can be found in a number of different places. Where you choose to find them depends on your own knowledge of z/OS, available tools, and how much effort you want to spend.

Sources you can use include:

JES Log

For batch jobs, JES writes a job log including messages issued and system accounting information. Figure 13 on page 62 shows a sample JES2 log for a DFSORT job. The JES2 log produced at your site will, of course, be different.

```
                  J E S 2  J O B  L O G  --  S Y S T E M  E D S 3  --  N O D E  S N J M A S 3

15.07.39 JOB11137 ---- TUESDAY,   04 OCT 2011 ----
15.07.39 JOB11137  ICH70001I Y897797  LAST ACCESS AT 15:00:41 ON TUESDAY, OCTOBER 4, 2011
15.07.39 JOB11137  $HASP373 TGSORT   STARTED - INIT 8    - CLASS A - SYS EDS3
15.07.55 JOB11137  -                                          --TIMINGS (MINUTES)--
15.07.55 JOB11137  -JOBNAME  STEPNAME PROGRAM     RC   EXCP    CPU    SRB  CLOCK   SERV  PG
15.07.55 JOB11137  -TGSORT   SORT     SORT      0000    573    .03    .00     .2   855K   0
15.07.55 JOB11137  -TGSORT   ENDED. TOTAL CPU TIME=   .03  TOTAL ELAPSED TIME=    .2 MINUTE
15.07.55 JOB11137  $HASP395 TGSORT   ENDED
------ JES2 JOB STATISTICS ------
  04 OCT 2011 JOB EXECUTION DATE
           13 CARDS READ
          147 SYSOUT PRINT RECORDS
            0 SYSOUT PUNCH RECORDS
            9 SYSOUT SPOOL KBYTES
         0.26 MINUTES EXECUTION TIME
        1 //TGSORT   JOB (DA26,005,098,J69),'FL YAEGER',CLASS=A,                  JOB11137
          // MSGCLASS=H,USER=Y897797,MSGLEVEL=(1,1),TIME=(2,0),NOTIFY=Y897797
        2 //JOBLIB DD DSN=SORTDEV.V2R1.DEV.SICELINK,DISP=SHR
        3 //       DD DSN=SORTDEV.V2R1.DEV.SORTLPA,DISP=SHR
        4 //SORT EXEC PGM=SORT
        5 //SYSOUT DD SYSOUT=*
        6 //SORTDIAG DD DUMMY
        7 //SYSABEND DD SYSOUT=*
        8 //SORTIN DD DSN=Y897797.TGSORT.IN,DISP=SHR
        9 //SORTOUT DD DSN=&&SRTOUT,DISP=(NEW,PASS),
          // SPACE=(CYL,(250,250),RLSE),UNIT=SYSDA
       10 //SYSIN DD *
ICH70001I Y897797  LAST ACCESS AT 15:00:41 ON TUESDAY, OCTOBER 4, 2011
IEF236I ALLOC. FOR TGSORT SORT
IEF237I 4701 ALLOCATED TO JOBLIB
IEF237I 4701 ALLOCATED TO
IEF237I JES2 ALLOCATED TO SYSOUT
IEF237I DMY  ALLOCATED TO SORTDIAG
IEF237I JES2 ALLOCATED TO SYSABEND
IGD103I SMS ALLOCATED TO DDNAME SORTIN
IGD100I 5417 ALLOCATED TO DDNAME SORTOUT  DATACLAS (        )
IEF237I JES2 ALLOCATED TO SYSIN
IGD100I 5D06 ALLOCATED TO DDNAME SORTWK01 DATACLAS (        )
IGD100I 5D5B ALLOCATED TO DDNAME SORTWK02 DATACLAS (        )
IGD100I 5C0B ALLOCATED TO DDNAME SORTWK03 DATACLAS (        )
IGD100I 5D06 ALLOCATED TO DDNAME SORTWK04 DATACLAS (        )
IGD100I 5D06 ALLOCATED TO DDNAME SORTWK05 DATACLAS (        )
IEF285I   SYS11277.T150739.RA000.TGSORT.R0394529       DELETED
IEF285I   VOL SER NOS= SCR308.
IEF285I   SYS11277.T150739.RA000.TGSORT.R0394530       DELETED
IEF285I   VOL SER NOS= SCR304.
IEF285I   SYS11277.T150739.RA000.TGSORT.R0394531       DELETED
IEF285I   VOL SER NOS= SCR304.
IEF285I   SYS11277.T150739.RA000.TGSORT.R0394527       DELETED
IEF285I   VOL SER NOS= SCR304.
IEF285I   SYS11277.T150739.RA000.TGSORT.R0394528       DELETED
IEF285I   VOL SER NOS= SCR302.
IEF142I TGSORT SORT - STEP WAS EXECUTED - COND CODE 0000
IEF285I   Y897797.TGSORT.JOB11137.D0000102.?           SYSOUT
IEF285I   Y897797.TGSORT.JOB11137.D0000103.?           SYSOUT
IGD104I Y897797.TGSORT.IN                           RETAINED,  DDNAME=SORTIN
IEF285I   Y897797.TGSORT.JOB11137.D0000101.?           SYSIN
IEF373I STEP/SORT    /START 2011277.1507
IEF032I STEP/SORT    /STOP  2011277.1507
        CPU:     0 HR  00 MIN  02.19 SEC    SRB:     0 HR  00 MIN  00.08 SEC
        VIRT:  1132K  SYS:   376K  EXT:    20784K  SYS:    11268K
IEF285I   SORTDEV.V2R1.DEV.SICELINK                    KEPT
IEF285I   VOL SER NOS= SRTLIB.
IEF285I   SORTDEV.V2R1.DEV.SORTLPA                     KEPT
IEF285I   VOL SER NOS= SRTLIB.
IEF237I 5417 ALLOCATED TO SYS00001
IEF285I   SYS11277.T150755.RA000.TGSORT.R0394532       KEPT
IEF285I   VOL SER NOS= SCR307.
IEF285I   SYS11277.T150739.RA000.TGSORT.SRTOUT.H03     DELETED
IEF285I   VOL SER NOS= SCR307.
IEF375I  JOB/TGSORT  /START 2011277.1507
IEF033I  JOB/TGSORT  /STOP  2011277.1507
        CPU:     0 HR  00 MIN  02.19 SEC    SRB:     0 HR  00 MIN  00.08 SEC
```

Figure 13: A Sample JES2 Log

Messages

Some programs (like DFSORT) issue messages quantifying their use of certain system resources. This is the simplest way of accessing such information. In the case of DFSORT, when a SORTDIAG DD statement is present or installation option DIAGSIM=YES has been specified for your site, additional messages useful for tuning are issued. An example of the DFSORT messages is shown in Figure 14 on

page 63. See *z/OS DFSORT Messages, Codes and Diagnosis Guide* for explanations of these messages.

```
ICE805I 1 JOBNAME: TGSORT   , STEPNAME: SORT
ICE802I 0 BLOCKSET     TECHNIQUE IN CONTROL
ICE201I 0 RECORD TYPE IS F - DATA STARTS IN POSITION 1
ICE992I 0 RA 0 WR 0 TR 1
ICE751I 0 C5-BASE   C6-BASE   C7-BASE   C8-BASE   E4-BASE   C9-BASE   E5-BASE   E6-BASE   C4-BASE   E7-BASE
ICE143I 0 BLOCKSET     SORT  TECHNIQUE SELECTED
ICE250I 0 VISIT http://www.ibm.com/storage/dfsort FOR DFSORT PAPERS, EXAMPLES AND MORE
ICE000I 1 - CONTROL STATEMENTS FOR 5650-ZOS, Z/OS DFSORT V2R1  - 15:07 ON TUE OCT 04, 2011 -
            SORT FIELDS=(5,20,CH,D)
ICE201I 0 RECORD TYPE IS F - DATA STARTS IN POSITION 1
ICE992I 0 RA 0 WR 0 TR 1
ICE751I 0 C5-BASE   C6-BASE   C7-BASE   C8-BASE   E4-BASE   C9-BASE   E5-BASE   E6-BASE   C4-BASE   E7-BASE
ICE187I 0 DFSORT SVC LEVEL INCOMPATIBLE WITH DFSORT PROGRAM LEVEL
ICE193I 0 ICEAM1 INVOCATION ENVIRONMENT IN EFFECT - ICEAM1 ENVIRONMENT SELECTED
ICE088I 2 TGSORT  .SORT    .        , INPUT LRECL = 1500, BLKSIZE = 27000, TYPE = FB
ICE093I 0 MAIN STORAGE = (MAX,20861041,20861041)
ICE156I 0 MAIN STORAGE ABOVE 16MB = (20760161,20760161)
ICE127I 0 OPTIONS: OVFLO=RC0 ,PAD=RC0 ,TRUNC=RC0 ,SPANINC=RC16,VLSCMP=N,SZERO=Y,RESET=Y,VSAMEMT=Y,DYNSPC=256
ICE128I 0 OPTIONS: SIZE=20861041,MAXLIM=1048576,MINLIM=450560,EQUALS=N,LIST=Y,ERET=RC16 ,MSGDDN=SYSOUT
ICE129I 0 OPTIONS: VIO=N,RESDNT=NONE,SMF=NO   ,WRKSEC=Y,OUTSEC=Y,VERIFY=N,CHALT=N,DYNALOC=(SYSDA   ,004),ABCODE=MSG
ICE130I 0 OPTIONS: RESALL=4096,RESINV=0,SVC=109 ,CHECK=Y,WRKREL=Y,OUTREL=Y,CKPT=N,COBEXIT=COB2
ICE131I 0 OPTIONS: TMAXLIM=6291456,ARESALL=0,ARESINV=0,OVERRGN=65536,CINV=Y,CFW=Y,DSA=64
ICE132I 0 OPTIONS: VLSHRT=N,ZDPRINT=Y,IEXIT=N,TEXIT=N,LISTX=N,EFS=NONE    ,EXITCK=S,PARMDDN=DFSPARM ,FSZEST=N
ICE133I 0 OPTIONS: HIPRMAX=OPTIMAL,DSPSIZE=MAX ,ODMAXBF=0,SOLRF=Y,VLLONG=N,VSAMIO=N,MOSIZE=MAX
ICE235I 0 OPTIONS: NULLOUT=RC0
ICE236I 0 OPTIONS: DYNAPCT=10 ,MOWRK=Y,TUNE=STOR,EXPMAX=MAX    ,EXPOLD=50%    ,EXPRES=10%
ICE084I 0 EXCP ACCESS METHOD USED FOR SORTOUT
ICE084I 0 EXCP ACCESS METHOD USED FOR SORTIN
ICE750I 0 DC 1500012000 TC 0 CS DSVVV KSZ 20 VSZ 20
ICE887I 0 CSES 17182181,17437734,17437734 ES 0,0,0
ICE886I 0 SYS 524288 TSTG 15693961 FS 368169 INIT 3682 MAX 18409 LEN 0
ICE752I 0 FSZ=1000008 RC  IGN=0 E  AVG=1500 0  WSP=1948258 C  DYN=4500 49120
ICE915I 0 MOFSZ=1439,MOSZ=0,MOSYS=9496(2),MOSTG=61304,MEML=9496(1)
ICE916I 0 MOFR=0402,MOVR=VV
ICE996I 0 ESM=17437734,ESO=127776,ESR=1743773,ESP=4096,ESS=16384,CES=82143488,HSZ=16777216
ICE997I 0 HWSP=915535,HMAX=0,MOMAX=2430976,ASV=15693961,EQ=I2,HN=1
ICE898I 0 OMAX=17182181,NMAX=17437734,ENQT=7675904,CMAX=770048,HU=97,BUN=49120,MD=NK,M3,DU=83,DR=0,HN=1
ICE880I 0 QP=188 QA=2146 HI=3585 LI=1687 MI=3832 TZ=770048 N1=228883 N2=770048 SZ=189 HN=1
ICE889I 0 CT=MAX     , SB=241, L=0, D=0000, CCW=1MAM
ICE901I 0 W 03PP33 04PP33 05PP33 01PP33 02NP33
ICE902I 0 O PP10  I PP10
ICE751I 1 CF-BASE   CE-BASE   D3-BASE   ED-BASE   E8-BASE
ICE900I 0 CON=1,MUV=0,VOL=1,ENU=0,SBK=1,SRC=10,VEM=0
ICE999I 0 PWK=4 PSP=4500 SWK=0 SSP=0 TWK=0 TSP=0 RWK=0 RSP=0 AWK=4 AWP=4500
ICE090I 0 OUTPUT LRECL = 1500, BLKSIZE = 27000, TYPE = FB   (SDB)
ICE055I 0 INSERT 0, DELETE 0
ICE054I 0 RECORDS - IN: 1000000, OUT: 1000000
ICE134I 0 NUMBER OF BYTES SORTED: 1500000000
ICE253I 0 RECORDS SORTED - PROCESSED: 1000000, EXPECTED: 1000008
ICE165I 0 TOTAL WORK DATA SET TRACKS ALLOCATED: 4500 , TRACKS USED: 0
ICE199I 0 MEMORY OBJECT USED AS MAIN STORAGE = 0M BYTES
ICE299I 0 MEMORY OBJECT USED AS WORK STORAGE = 1478M BYTES
ICE180I 0 HIPERSPACE STORAGE USED = 0K BYTES
ICE188I 0 DATA SPACE STORAGE USED = 0K BYTES
ICE891I 1 20797856 WMAIN, 19664 CMAIN, MAX CALLOC, N SCN, B BA, 0 AZ, 0 BZ, NN QC, 0 CZ, 0 DZ, 1 PLE
ICE892I 1 1500 RIN  27000 BLI  27000 BLO  1500 RUN  49120 BUN  2817 CPU  00 CVC
ICE893I 1 376 XIN 215 WIN 0 GIN NDEY PFP00 B00 CM000 CIX UPTH LMD VS   RUX
ICE894I 0 148 STR  148 MOR  212 IPB  377 OPB  0 CYL  O   MN
ICE881I 0 EQ=I2 DX=0 D2=95 D3=92 D4=1 AS=0 SA=0 SB=0 SC=0 HN=1
ICE885I 0 DAT 04 DSR 0402 BINS 0 BSZ 0 RCP 0 AJC 0 RLC 0 DUNIT 0
ICE895I 0 204 MUNIT  1 SUNIT  34 OUNIT
ICE896I 0 148 SET  147 DEXTOT  0 BLK   1000008 CSZ  0 WE
ICE804I 1 SORTWK05 EXCP COUNT: 0
ICE804I 1 SORTWK04 EXCP COUNT: 0
ICE804I 1 SORTWK03 EXCP COUNT: 0
ICE804I 1 SORTWK02 EXCP COUNT: 0
ICE804I 1 SORTWK01 EXCP COUNT: 0
ICE804I 1 SORTOUT  EXCP COUNT: 156
ICE804I 1 SORTIN   EXCP COUNT: 150
ICE899I 0 HSR=2080,HSW=150,HRE=31474,HWE=150,HRP=377688,HWP=377688,HWM=378240,HNM=1
ICE052I 0 END OF DFSORT
```

Figure 14: DFSORT Messages

ICEIEXIT

The installation initialization exit (ICEIEXIT) allows you to examine certain installation and run-time information for each DFSORT application. The ICEIEXIT routine could be used to collect the information relevant to performance, and to write this information to a data set which can be analyzed. Refer to “Using ICEIEXIT data ” on page 69 for information on using ICEIEXIT data to do a moderate analysis of your DFSORT applications. Refer to “Installation exits ” on page 32 for a brief overview of ICEIEXIT.

SMF

System management facilities (SMF) collects a variety of system and application-related information into a number of different SMF records written by the system and various programs.

SMF type-30 (subtype 4) records, for instance, are written for each job step, and summarize the step's consumption of major system resources, such as CPU time (broken down into these fields: TCB, SRB, RCT, HPT, and IIP), elapsed time, EXCP counts, and device connect times. In addition, DFSORT can write an SMF type-16 record, which summarizes the key statistics from a particular DFSORT run. By running reports against SMF data, sites can use this information for workload analysis, planning, and accounting purposes.

To extract performance information from SMF data, you can run an SMF report, use a data reduction program, use the analysis and reporting features of DFSORT or its ICETOOL utility, or write a program of your own. Since SMF data often contains accounting and other sensitive data, access might be limited.

ICETEXIT

The installation termination exit (ICETEXIT) allows you to collect extensive performance-related information about all DFSORT applications at a site. ICETEXIT provides comprehensive data for each DFSORT application including the information contained in DFSORT's type-16 SMF record. Refer to “Using ICETEXIT data ” on page 70 for information on using ICETEXIT data to do a moderate analysis of your DFSORT applications. Refer to “Installation exits ” on page 32 for a brief overview of ICETEXIT.

RMF

Resource Measurement Facility (RMF) measures system, address space, and workload activity. You can use RMF to generate reports online or off-line, or as a real-time monitor. Most of these reports concern performance-related statistics for processor, storage, I/O devices, and system data sets.

RMF's primary use is for system-level tuning, but it can also detail the performance characteristics of subsystems and workloads. Many system programmers rely heavily on RMF reports for these analysis activities. RMF writes a series of records into the SMF system data sets which can later be processed by RMF, or other SMF analysis programs.

In terms of DFSORT performance analysis, RMF can be used to understand DFSORT's use of resources, to examine the effects of running DFSORT applications, and to highlight contention for resources. For example, RMF can help you identify situations where multiple work data sets are being placed on the same disk device, by showing a high device utilization for that particular device. As with SMF, access to RMF and its data might be restricted.

For a detailed cross reference of sources of performance indicators, refer to Table 7 on page 65.

Overview of DFSORT performance information

Table 7 on page 65 lists the main DFSORT performance areas, sources of information about each area, and suggested analysis activities for each area.

Each level of analysis discussed in “Analysis techniques for DFSORT performance data ” on page 67 produces a set of data to interpret. This figure is intended to help you associate that data with certain performance areas in DFSORT. For example, if you have decided to do a simple analysis and are concerned with your use of Blockset, you would check DFSORT messages ICE143I and ICE800I to determine whether or not the Blockset technique is being used, and if it is not, why it is not.

You can also use this figure to help you determine which level of analysis you want to do, given the performance areas you are interested in. For example, if you were concerned about intermediate merging, you would be able to tell from this figure that you would need to do moderate analysis to obtain data about this performance area.

Table 7 on page 65 uses the following abbreviations:

ICExxxx
DFSORT messages (The ICE8xxI messages only appear if you have coded the SORTDIAG DD statement or if the installation option DIAGSIM=YES has been specified for your site.). See *z/OS DFSORT Messages, Codes and Diagnosis Guide* for explanations of the DFSORT messages.

JLOG
Job/JES log.

SMF16
SMF type-16 record (issued by DFSORT).

SMF30
SMF type-30 (subtype 4) record.

IEXIT
Information passed to the DFSORT ICEIEXIT routine.

TEXIT
Information passed to the DFSORT ICETEXIT routine (includes contents of the SMF type-16 record).

An individual source might not contain all of the information listed under Analysis.

Table 7: Sources of Performance Indicators.

Area	Source	Analysis
Blockset	ICE143I, ICE189A, ICE800I, SMF16, IEXIT, TEXIT	Ensure the Blockset technique is being used or determine why it is not being used.
Intermediate merge	ICE247I, SMF16, TEXIT	Ensure there are no intermediate merges for a Blockset sort (if appropriate). Message ICE247I indicates an intermediate merge has taken place. An intermediate merge is a condition caused by a very low virtual storage to data set size ratio, and usually results in significant performance degradation. Providing sufficient virtual storage to DFSORT eliminates intermediate merges.
Unknown file size	ICE118I, SMF16	Indicates DFSORT cannot determine the input file size. If DFSORT terminates with ICE046A or ICE083A, ensure that an appropriate value is used for FILSZ=En or DYNSPC=n.
Virtual storage	ICE039A, ICE080I, ICE092I, ICE093I, ICE115A, ICE156I, ICE231I, JLOG, SMF16, SMF30, TEXIT	Ensure adequate storage is available both above and below 16 MB virtual or determine how much more storage is needed. **Note:** JES log messages and SMF type-30 fields can be misleading as to how much storage DFSORT actually uses. The DFSORT messages and ICETEXIT values contain the actual amount of storage used by DFSORT.
DFSORT storage options	ICE128I, ICE130I, ICE131I, IEXIT, TEXIT	Ensure SIZE=MAX or MAINSIZE=MAX is used and RESALL, TMAXLIM, MAXLIM, OVERRGN, DSA and MINLIM values are appropriate.
Work data sets	ICE129I, ICE165I, JLOG, SMF16, SMF30, TEXIT,	Determine whether dynamic allocation or JCL allocation of work data sets is used, how much work space is allocated and used, and to what types of devices the data sets are allocated. In addition, ensure that the work space is minimized.
Hipersorting	ICE133I, ICE180I, IEXIT, SMF16, TEXIT	Ensure Hipersorting is used, if appropriate, and determine how much Hiperspace is used.

Table 7: Sources of Performance Indicators. (continued)

Area	Source	Analysis
Dataspace sorting	ICE133I, ICE188I, IEXIT, SMF16, TEXIT	Ensure dataspace sorting is used, if appropriate, and determine how much data space is used.
Memory object sorting	ICE133I, ICE236I, ICE199I, ICE299I, IEXIT, SMF16, TEXIT	Ensure memory object sorting is used, if appropriate, and determine how much memory object storage is used, and how it was used (as intermediate work storage or as an extension to main storage)
CPU time	JLOG, SMF16, SMF30, TEXIT	Determine the effects of tuning on CPU time.
Elapsed time	JLOG, SMF16, SMF30, TEXIT	Determine the effects of tuning on Elapsed time.
EXCPs	ICE804I, JLOG, SMF16, SMF30, TEXIT	Determine the effects of tuning on EXCPs.
Device connect time	SMF30	Determine the effects of tuning on device connect time.
Storage control cache	TEXIT	Ensure storage control units with cache are used, and that cache fast write is used.
System determined block size	ICE090I, ICE210I	Ensure system-determined block size is used for output data sets, when appropriate.
Block sizes	ICE088I, ICE090I, ICE2101, SMF16, TEXIT	Ensure adequately large block sizes are used for input and output data sets.
Disk extents	SMF16, TEXIT	Ensure the number of extents required for input, output, and work data sets is minimized.
DFSORT SVC	ICE145A, ICE187I, ICE191I, ICE194I, ICE816I	Ensure that the DFSORT SVC is available.
Residency	ICE129I	Ensure that DFSORT modules are resident, if appropriate.
DFSORT level	ICE000I, SMF16, TEXIT	Ensure that the latest DFSORT level is installed.
Control field length	SMF16, TEXIT	Ensure that the control fields are only as long as necessary to distinguish the records.
EQUALS	ICE128I, TEXIT, SMF16	Ensure that the EQUALS option is used only when necessary.
VERIFY	ICE129I, IEXIT, TEXIT	Ensure that the VERIFY option is used only when necessary.
Number of disk work data sets and devices	ICE804I, JLOG, SMF16, SMF30, TEXIT	For disk-only work data set applications, ensure that at least three data sets are used and that the data sets are on separate devices, if possible.
Type of application (sort, merge, or copy)	ICE143I, SMF16, IEXIT, TEXIT	Understand the type of application and tune accordingly.

Table 7: Sources of Performance Indicators. (continued)

Area	Source	Analysis
Record length and format	ICE088I, ICE089I, ICE090I, ICE091I, ICE210I, SMF16, TEXIT	Understand what type of data is being processed, and tune accordingly.
Input, work, and output data set sizes	ICE054I, ICE055I, ICE098I, ICE134I, ICE227I, ICE228I, SMF16, TEXIT	Understand the characteristics of the data being processed, and tune accordingly.
VIO	JLOG, SMF30	Ensure that VIO is only used when appropriate.

Sources of DFSORT performance information

z/OS systems provide a wealth of performance data which is often difficult to analyze in its raw form. You need to convert this raw data into processed data that can be used for trend analysis and management reports. Various reporting products, and DFSORT/ICETOOL, can also provide useful performance information.

Performance analysis and reporting products

Various analysis and reporting products are available that post-process system log data sets (such as SMF records written by the system and by products such as DFSORT), produce a variety of reports that can help you analyze your DFSORT performance information, and identify and resolve bottlenecks.

DFSORT/ICETOOL

If you do not have access to a performance analysis and reporting product, you can use the DFSORT editing functions in combination with the reporting capabilities of ICETOOL or OUTFIL to generate performance data reports. The INCLUDE/OMIT, OUTFIL, and INREC/OUTREC control statements are helpful in selecting the particular fields of the particular SMF, RMF, or other data records to be analyzed. ICETOOL or OUTFIL can then be used to generate printable reports from the resulting raw data. See *z/OS DFSORT Application Programming Guide* for more information about ICETOOL, OUTFIL, and editing functions.

Analysis techniques for DFSORT performance data

After identifying where and how you can obtain the performance data, you need to analyze the performance data. You need to understand your current use of DFSORT in order to assess whether any changes are needed. Tuning might not be necessary.

Investigating DFSORT use does not need to take a lot of effort or a long time. You can investigate DFSORT a number of times, and with each iteration, look at a different aspect or go into a greater level of detail. Where you begin depends on how much time you have and the effort you are prepared to invest.

You can also investigate DFSORT use to ensure that any tuning you have done has had the desired effect. Alternative techniques for investigating current DFSORT use are described in this section and ordered according to how much effort is involved.

Regardless of which technique you choose, you might want to use the ICETOOL DEFAULTS operator to list the installation defaults currently in effect at your site. You can use this operator at any time to produce a list of the values for each installation option for each of the four invocation installation environments and each of the four time-of-day installation environments. Refer to “Modifying installation defaults” on page 25 for information about installation defaults and an example of an ICETOOL defaults job. The output from DEFAULTS can help you analyze how you are using DFSORT and is beneficial to have regardless of

the level of analysis you are performing. See *z/OS DFSORT Application Programming Guide* and *z/OS DFSORT Installation and Customization* for more information about the ICETOOL DEFAULTS operator.

Simple analysis

Generally, a few DFSORT applications at a site can be identified as being the largest and longest running, or on a critical path. An application is said to be on a critical path if any delay in its completion would delay finishing the workload. Tuning activities on the critical path can have a beneficial effect in reducing the overall elapsed time of the batch workload. The performance of these applications can determine whether you finish inside the batch window or overrun it, or whether you are using too many system resources. The "80/20 rule of thumb" applies to DFSORT: often 80% of the system's resources used by DFSORT are used by 20% of DFSORT applications.

Start your investigation by concentrating on these applications. Use a SORTDIAG DD statement (or make sure the ICEMAC option DIAGSIM=YES has been specified for your site) to obtain all available messages.

DFSORT messages are a simple and effective source of information for individual applications. They give you a picture of the processing and help you develop an action plan.

Performance-related information in the DFSORT messages includes:

- Amount of virtual storage available
- DFSORT technique used
- Total bytes and records sorted
- Number and distribution of EXCPs by data set
- Work data set space allocated and used
- Hiperspace, data space, or memory object storage used
- Availability of the DFSORT SVC
- Access methods and block sizes used
- Settings of various options (for example, EQUALS, VERIFY) that can affect performance

Moderate analysis

If you can take the time to do some analysis of all your DFSORT applications, you can pinpoint more precisely where to tune your use of DFSORT to perform more efficiently. The easiest way to start is to use information optionally provided by DFSORT. To do this, you can use data from SMF records or an ICEIEXIT routine.

Using SMF data

Three SMF record types are particularly useful for analyzing the performance of DFSORT: type-30, type-16, and type-42. To collect an SMF record type, make sure that the active SMFPRMxx member is set up to collect that particular type. For type-30 records, make sure subtype 4 (step total) records are collected. If device connect time and EXCP statistics are desired in the type-30 (subtype 4) records, the DETAIL option must be in effect. For type-42 records, make sure subtype 6 (data set statistics) records are collected. See *z/OS MVS Initialization and Tuning Reference* for more information on setting up an SMFPRMxx member.

Type-30 records

SMF type-30 (subtype 4) records contain useful information on the resource consumption of a particular job step. These include:

- CPU time (total, and broken down by field)
- Elapsed time
- EXCPs (total, and broken down by device)
- Device connect time, broken down by device
- Paging statistics

See *z/OS MVS System Management Facilities (SMF)* for more detailed information on the type-30 fields.

Type-16 records

DFSORT produces SMF type-16 records. They contain data on a particular DFSORT step. If you want to produce DFSORT SMF records, DFSORT's supplied ICEMAC default of SMF=NO must be changed to SMF=FULL or SMF=SHORT, or SMF=FULL or SMF=SHORT must be specified on an OPTION statement in DFSPARM or in the extended parameter list at run-time. This must be done in addition to setting up the appropriate SMFPRMxx member. The DFSORT SVC must be available for DFSORT to be able to write the SMF records to an SMF data set. Refer to "Making the DFSORT SVC available" on page 20 for additional information.

Some of the fields in the type-16 record containing performance-related data are:

- Type of application (sort, merge, or copy)
- DFSORT technique used
- Elapsed, TCB, and SRB time used
- Type and length of records being sorted
- Total bytes and records sorted
- Work data set allocation data
- Access methods and block sizes used
- Control field length
- Amount of Hiperspace, data space, and memory object storage used
- Number of input and output data set I/O calls
- Number of work data set EXCPs
- Number of intermediate merges performed

See *z/OS DFSORT Installation and Customization* for a complete list of the type-16 fields.

There are several reasons why it might be convenient to use the DFSORT type-16 records:

- You already have procedures to analyze this information.
- The scope of the information you are interested in is limited to the information available in the SMF records.
- You do not wish to write and maintain an analysis routine.

Type-42 records

DFSMS/MVS produces SMF type-42 (subtype 6) records. They contain information on a particular disk data set OPEN for a particular job, such as:

- Start Subchannels (SCCHs) (physical I/Os)
- Response time components (Connect, Disconnect, IOSQ and Pending times)
- Cache performance information

Both interval and CLOSE-time records are produced (unless interval recording is disabled).

See *z/OS MVS System Management Facilities (SMF)* for more detailed information on the type-42 subtype 6 fields.

Using ICEIEXIT data

DFSORT enables you to provide an installation initialization exit (ICEIEXIT) routine which can be used to examine some performance-related information about all DFSORT applications at a site.

If present and activated, the ICEIEXIT routine is passed installation and run-time information. As this document illustrates in later sections, the values of installation and run-time parameters affect DFSORT performance.

The ICEIEXIT routine can examine installation and run-time information related to:

- Virtual storage limits
- Hiperspace limits
- Memory object limits
- Data space limits
- Use of VERIFY
- OUTFIL buffer space limits

And run-time information about:

- Type of DFSORT application (sort, merge, or copy)
- Method of invocation
- Use of the Blockset technique
- Use of storage above 16 MB virtual

See *z/OS DFSORT Installation and Customization* for complete information about how to write and install an ICEIEXIT routine.

Thorough analysis

To tune DFSORT thoroughly, you need extensive data about all DFSORT applications run by you or your site, including elapsed time, CPU time, number of EXCPs, and types and amounts of virtual storage used. JES logs and DFSORT messages can be used to analyze an isolated number of applications, but this is not practical to apply to a large number of DFSORT jobs. In addition, these sources do not supply all of the data needed to perform such a thorough analysis of DFSORT resource consumption.

Using ICETEXIT data

DFSORT enables you to provide an installation termination exit (ICETEXIT) routine which can be used to collect extensive performance-related information about all DFSORT applications at a site.

If present and activated, the ICETEXIT routine is called at the end of DFSORT application processing. It is available for those who wish to make a thorough analysis of DFSORT performance data using a single source of information. See *z/OS DFSORT Installation and Customization* for complete information on how to write and install an ICETEXIT routine.

ICETEXIT provides comprehensive information on each DFSORT application, including the information contained in DFSORT's type-16 SMF record.

Note: The SMF record is available to the ICETEXIT routine even if the SMF facility is not being used (in this case, the SMF record is constructed and passed to the ICETEXIT routine, but not written to the SMF data set).

Additional information available to the ICETEXIT routine includes:

- Control statements specified
- User exits used
- Options (for example, VERIFY) in effect
- Control field formats used
- Work data set EXCPs broken down by data set and DFSORT phase
- Use of cache fast write
- Virtual storage statistics
- DFSORT phase timing statistics (elapsed, TCB, SRB)
- Input and output data set statistics
- Hipersorting statistics
- Sorting with data space statistics
- Memory object sorting statistics

In some cases, you only need a portion of this information for performance and tuning reasons.

The advantage of using an ICETEXIT routine is that all the information about each DFSORT application is available to the routine. You do not need to use information from a number of other sources. How you process the data depends on your requirements.

Examples of how you can use an ICETEXIT routine include:

- Collating information from SMF type-16 records with run-time and installation option values to identify applications with options which degrade performance.
- Write an SMF record using your own format.

You can write the record to a private data set for subsequent processing.

You can process SMF records or private data set records using an appropriate data reduction program.

The example in Appendix A, “Sample ICETEXIT,” on page 75 provides a sample ICETEXIT assembler routine. This installation-wide exit creates a user SMF record from the data passed by DFSORT.

The assembler source for an SVC which writes the user SMF record to SMF is also included. This enables the ICETEXIT routine to run unauthorized but write SMF records using the SVC.

See *z/OS DFSORT Installation and Customization* for more information on how to write and install an ICETEXIT routine,

Using RMF data

Using ICETEXIT data enables you to understand the details of how DFSORT is being used at your site. But in addition to optimizing the performance of individual or groups of sort applications, it is also very important to pay some attention to the system's overall performance when making tuning decisions for DFSORT.

Using SMF type-30 records and certain RMF reports, such as the I/O device activity, the paging activity, and the address space resource data reports, enables you to measure the amount of impact (if any) on system resources that is caused by DFSORT applications. With this information it is possible to balance DFSORT's resource requirements with those of the other applications on the system.

See *z/OS RMF Report Analysis* for more information about RMF reports.

DFSORT requirements and system resources

The previous sections described the significance of understanding your site's current use of DFSORT, in making the proper DFSORT tuning decisions. But in addition to optimizing the performance of individual or groups of sort applications, it is also very important to pay some attention to the system's overall performance when making tuning decisions for DFSORT (or tuning decisions for any application, for that matter). System resources are always limited, and a heavy overcommitment of resources usually means a severe performance degradation for the entire system. So it is advisable to balance DFSORT's resource requirements with those of the other applications on the system.

Analyzing system performance is usually accomplished with the Resource Measurement Facility (RMF) or an equivalent product. Many different types of reports can be generated with RMF, such as CPU Activity Reports, I/O Activity Reports, and Paging Activity Reports. A lot of very useful information is contained in these reports, but for the purpose of this discussion on DFSORT tuning, only the I/O and Paging Activity Reports will be addressed. For detailed information on system-wide tuning in general, see *z/OS MVS Initialization and Tuning Guide*.

Placement of data sets

The RMF I/O Device Activity Report provides device related information such as average response times, average connect and disconnect times, and average activity rates. This data is furnished for each online I/O device in one or more device classes. The information in the I/O Activity Report (in conjunction with

the Channel Activity and I/O Queueing Activity reports) can help you to identify possible areas of device contention.

For instance, if the report shows a high device activity rate and an unusually large pending time for a particular device (or group of devices) containing DFSORT work data sets, the performance of the sort application was most certainly adversely affected. If it turns out that the volumes selected for the work data sets happened to, for example, also contain the tables for a very active data base, it might be advantageous to make changes to the applications or to the I/O configuration that would prevent future allocation of DFSORT work data sets to those particular devices.

The same idea holds true for DFSORT input and output data sets, and in general for any application on the system; if there is a lot of contention for a few I/O devices, the data on those devices should be distributed over more (less active) volumes to achieve an overall balance in I/O activity rates. Keep in mind that for best DFSORT performance, the input and work data sets, as well as the work and output data sets should be located on different volumes, or even different storage subsystems if possible.

Use of virtual storage

The RMF Paging Activity Report contains two fields that can be especially useful in determining contention for central storage resources. The high unreferenced interval count (HUIC) is shown in the central storage movement rates section, and the minimum total available central storage frames is shown in the frame and slot counts section. If the Paging Activity Report indicates a very low HUIC and an unusually low available frame count for extended periods of time, then the system is suffering from contention for central storage.

You can use SMF type-30 records to pinpoint which jobs or set of jobs are using the most virtual storage. If these are not DFSORT applications, you may be able to reduce the REGION parameters in the JCL for those jobs, or restrict the available region sizes and limits for those jobs in the IEFUSI system exit. If they are DFSORT jobs, there are two possible actions:

1. Restrict the virtual storage available to DFSORT. This can be done by setting lower TMAXLIM/MAXLIM/DSA installation defaults, or by dynamically restricting the storage size in an ICEIEXIT, or by a combination of the two.
2. Restrict DFSORT's use of dataspace and memory object sorting. This can be accomplished by setting DSPSIZE=n and MOSIZE=m values as installation defaults where n is the dataspace size limitation and m is the memory object size limitation for every sort job. The appropriate values to use for n and m is best determined by experimentation on the system in question. A good technique to use would be to start out with a very low value and then keep raising it by 10 MB or 20 MB for as long as system performance is not affected significantly. In cases of severe overcommitment of central storage, DSPSIZE=0 and MOSIZE=0 defaults may be advisable.

However, the possible performance consequences for the DFSORT applications should be taken into account. Refer to “Virtual storage and sorting with data space or memory objects ” on page 44 for more information. Performance critical DFSORT jobs may have to be excluded from these storage restrictions.

Use of VIO data sets

Another useful measurement contained in the RMF Paging Activity Report is the total system VIO page-in and page-out rates, which are displayed in the central storage paging rates section. Unusually high VIO paging rates can indicate an overuse of VIO data sets. With the exception of sorting applications that process only very small files, it is generally undesirable to allow the use of VIO devices for DFSORT output and especially DFSORT work data sets. Refer to “System-managed storage” on page 16 for details on how to prevent VIO allocations for DFSORT data sets when using DFSMS.

Performance trade-offs

Table 8 on page 73 is a summary of the techniques described in this document for improving DFSORT performance. In many cases (use of Blockset is a notable exception), an improvement in performance is

not free. The figure summarizes these potential trade-offs: increased paging, increased swapping, increased CPU time, and changes needed to applications.

For example, to use this figure, assume you want to improve the elapsed time for a DFSORT application. A number of techniques exist to choose from, ranging from ensuring that DFSORT uses Blockset to modifying the way the application uses DFSORT. You can decide which technique, or combination of techniques, is appropriate based on the effort required and the trade-off, if any, you are prepared to accept.

Table 8: Summary of Potential Performance Trade-Offs.

Improvement Area	Technique	Potential Trade-Offs			
		Increased Paging	Increased Swapping	Increased CPU Time	Change Application
CPU Time	Use Blockset				
	Increase Virtual Storage	Y	Y		
	Use appropriate large block sizes for input and output data sets				
	Use DFSORT functions and exits				Y
Elapsed Time	Use Blockset				
	Use memory object sorting				
	Use dataspace sorting				
	Use multiple work data set devices				
	Increase Virtual Storage	Y	Y		
	Use Hipersorting			Y	
	Use VIO data sets	Y		Y	
	Use appropriate large block sizes for input and output data sets				
	Use DFSORT functions and exits				Y
	Use striping				
I/O Activity	Use Blockset				
	Increase Virtual Storage	Y	Y		
	Use Hipersorting			Y	
	Use memory object sorting				
	Use dataspace sorting				
	Use appropriate large block sizes for input and output data sets				
	Use DFSORT functions and exits				Y
	Use striping				

Table 8: Summary of Potential Performance Trade-Offs. (continued)

Improvement Area	Technique	Potential Trade-Offs			
		Increased Paging	**Increased Swapping**	**Increased CPU Time**	**Change Application**
Work Data Set Space	Use Hipersorting			Y	
	Use memory object sorting				
	Use dataspace sorting				
	Increase Virtual Storage	Y	Y		
Specify number of records when appropriate				Y	

Appendix A. Sample ICETEXIT

This appendix contains Programming Interface information.

This appendix contains a sample ICETEXIT routine which creates a composite record containing all of the data areas passed to ICETEXIT and subsequently writes it out to SMF as user record 129. When ICEMAC option TEXIT=YES is specified, DFSORT calls the ICETEXIT routine at the end of each DFSORT application run at the site.

Because this routine does not run as an authorized program, an SVC is required to write the record to SMF. (This user SVC is unrelated to the SVC used by DFSORT itself.) "Making the DFSORT SVC available" on page 20 shows a sample SVC that can be used for this purpose. However, before installing it as a user SVC at your site, you should consider the security implications of using the SVC as is.

Note: The use of this ICETEXIT routine and associated SVC does not replace the DFSORT capability to issue its own type-16 SMF record. If you want DFSORT to issue the type-16 SMF record, use ICEMAC or run-time option SMF=SHORT or SMF=FULL. Depending upon how you install the DFSORT SVC, you may have to change ICEMAC option SVC=n. See *z/OS DFSORT Installation and Customization* for details.

You can obtain the sample ICETEXIT and SVC shown in this appendix from the DFSORT FTP site. The file is TGSAMP.ZIP.

Sample ICETEXIT

```
ICETEXIT TITLE 'SAMPLE ICETEXIT ROUTINE'
*-------------------------------------------------------------------*
*       SAMPLE ICETEXIT ROUTINE.                                    *
*       PURPOSE:                                                    *
*           THIS ROUTINE WILL COMBINE ALL OF THE DATA AREAS PASSED  *
*           TO ICETEXIT IN ONE USER SMF RECORD ID 129.  THESE       *
*           RECORDS CAN THEN BE ANALYZED AS A GROUP.                *
*                                                                   *
* NOTES:                                                            *
*   1) SEE "DFSORT INSTALLATION AND CUSTOMIZATION" FOR FULL DETAILS *
*      ON ICETEXIT                                                  *
*   2) THIS EXAMPLE ASSUMES THE USE OF SVC 249 - CHANGE IT AS       *
*      APPROPRIATE IF YOU USE A DIFFERENT SVC                       *
*                                                                   *
* REQUIRED DFSORT MACROS: ICESMF, ICEDTEX                           *
* REQUIRED SYSTEM MACROS: SAVE, GETMAIN, TIME, FREEMAIN, RETURN,    *
*                         CVT, IEESMCA, YREGS                       *
*-------------------------------------------------------------------*
         SPACE 1
ICETEXIT CSECT
ICETEXIT AMODE 24
ICETEXIT RMODE 24
         SPACE 1

*-------------------------------------------------------------------*
*        INITIALIZATION.                                            *
*-------------------------------------------------------------------*
         SPACE 1
         SAVE  (14,12),,ICETEXIT-&SYSDATE;-&SYSTIME;
         LR    R12,R15             EPA FOR BASE
         USING ICETEXIT,R12
         LR    R11,R1              SAVE ICETPAR ADDR
         USING ICETPAR,R11
         LA    R0,DSALTH           LENGTH OF DSA TO GETMAIN
       GETMAIN RC,LV=(0)
         LTR   R15,R15             DID WE GET STORAGE?
         BNZ   S01L990                NO..THEN GET OUT
         ST    R1,8(,R13)          FORWARD CHAIN
         ST    R13,4(,R1)          BACKWARD CHAIN
         LR    R13,R1              ->NEW SAVE AREA
         USING DSA,R13
         SPACE 1
*-------------------------------------------------------------------*
```

```
*        COMPUTE SIZE OF USER SMF RECORD TO BE BUILT.                *
*--------------------------------------------------------------------*
         SPACE 1
         L     R0,SDASMFL          LENGTH OF CONSTANT PORTIONS
         ICM   R1,B'1111',ICETSMFA  ->SORT SMF RECORD
         BZ    S01L100                BRANCH IF NOT ONE..
         USING ICESMFH,R1
         AH    R0,ICERDW           + LENGTH
         DROP  R1
         SPACE 1
S01L100  DS    0H
         ST    R0,DSASMFL          SAVE FOR LATER FREEMAIN
       GETMAIN RC,LV=(0)
         LTR   R15,R15             DID WE GET THE STORAGE?
         BNZ   S01L950               NO..THEN EXIT
         LR    R8,R1               LOAD FOR BASE
         USING ICE81RCD,R8
         SPACE 1
         LR    R0,R1               TO ADDR
         L     R1,DSASMFL          TO LENGTH
         SR    R14,R14             NO FROM ADDR
         LR    R15,R14             NO FROM LENGTH
         MVCL  R0,R14              ZERO THE AREA
         SPACE 1
*--------------------------------------------------------------------*
*        NOW BUILD THE SMF HEADER.                                   *
*--------------------------------------------------------------------*
         SPACE 1
         MVI   ICE81RTY,129        SET RECORD TYPE
         TIME  BIN
         STM   R0,R1,ICE81TME      SAVE TIME/DATE
         L     R1,16               ->CVT
         USING CVT,R1
         L     R1,CVTSMCA          ->SMCA
         USING SMCABASE,R1
         MVC   ICE81SID,SMCASID    GET SYSID
         DROP  R1
         LA    R9,ICE81LTH         LENGTH OF HEADER
         LA    R10,0(R9,R8)        ->FIRST AVAILABLE POSITION
         SPACE 1
*--------------------------------------------------------------------*
*        APPEND THE DFSORT SMF RECORD.                               *
*--------------------------------------------------------------------*
         SPACE 1
         ICM   R2,B'1111',ICETSMFA    -> DFSORT SMF RECORD
         BZ    S01L110                BRANCH IF NONE
         SPACE 1
         STH   R9,ICE81ROF         SAVE OFFSET
         USING ICESMFH,R2
         LH    R1,ICERDW           LENGTH OF SMF RECORD
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         DROP  R2
         SPACE 1
*--------------------------------------------------------------------*
*        APPEND THE SMF STATISTICS.                                  *
*--------------------------------------------------------------------*
         SPACE 1
S01L110  DS    0H
         ICM   R2,B'1111',ICETSST    -> SMF STATISTICS
         BZ    S01L120               BRANCH IF NONE
         STH   R9,ICE81SOF         SAVE OFFSET
         USING ICESST,R2
         L     R1,ICESMFST         LENGTH OF SMF STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         DROP  R2
         SPACE 1
*--------------------------------------------------------------------*
*        APPEND THE GENERAL STATISTICS.                              *
*--------------------------------------------------------------------*
         SPACE 1
S01L120  DS    0H
         ICM   R2,B'1111',ICETGEN    -> GEN STATISTICS
         BZ    S01L130               BRANCH IF NONE
         STH   R9,ICE81GOF         SAVE OFFSET
```

```
         USING ICEGEN,R2
         L     R1,ICEGSTAT         LENGTH OF GEN STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         DROP  R2
         SPACE 1
*------------------------------------------------------------------*
*        APPEND THE OPTIONS STATISTICS.                            *
*------------------------------------------------------------------*
         SPACE 1
S01L130  DS    0H
         ICM   R2,B'1111',ICETOPTS   -> OPTIONS STATISTICS
         BZ    S01L140               BRANCH IF NONE
         STH   R9,ICE81OOF         SAVE OFFSET
         USING ICEOPTS,R2
         L     R1,ICEOSTAT         LENGTH OF OPTIONS STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         DROP  R2
         SPACE 1
*------------------------------------------------------------------*
*        APPEND THE SORT/MERGE STATISTICS.                         *
*------------------------------------------------------------------*
         SPACE 1
S01L140  DS    0H
         ICM   R2,B'1111',ICETSMS    -> SORT/MERGE STATISTICS
         BZ    S01L150               BRANCH IF NONE
         STH   R9,ICE81MOF         SAVE OFFSET
         USING ICESMS,R2
         L     R1,ICEFSTAT         LENGTH OF SORT/MERGE STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         DROP  R2
         SPACE 1
*------------------------------------------------------------------*
*        APPEND THE VIRTUAL STORAGE STATISTICS.                    *
*------------------------------------------------------------------*
         SPACE 1
S01L150  DS    0H
         ICM   R2,B'1111',ICETVIRT   -> VIRTUAL STORAGE STATISTICS
         BZ    S01L160               BRANCH IF NONE
         STH   R9,ICE81VOF         SAVE OFFSET
         USING ICEVSTOR,R2
         L     R1,ICEVSTAT         LENGTH OF VIRTUAL STORAGE STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         DROP  R2
         SPACE 1
*------------------------------------------------------------------*
*        APPEND THE PHASE TIMING STATISTICS.                       *
*------------------------------------------------------------------*
         SPACE 1
S01L160  DS    0H
         ICM   R2,B'1111',ICETPTIM   -> PHASE TIMING STATISTICS
         BZ    S01L170               BRANCH IF NONE
         STH   R9,ICE81TOF         SAVE OFFSET
         USING ICEPHAST,R2
         L     R1,ICEPTIME         LENGTH OF PHASE TIMING STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         DROP  R2
         SPACE 1
*------------------------------------------------------------------*
*        APPEND THE SORTIN STATISTICS.                             *
*------------------------------------------------------------------*
         SPACE 1
S01L170  DS    0H
```

```
         ICM   R2,B'1111',ICETSIN    -> SORTIN STATISTICS
         BZ    S01L180               BRANCH IF NONE
         STH   R9,ICE81XOF         SAVE OFFSET
         USING ICESRTIN,R2
         L     R1,ICESORTI         LENGTH OF SORTIN STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         DROP  R2
         SPACE 1
*------------------------------------------------------------------*
*        APPEND THE SORTOUT STATISTICS.                            *
*------------------------------------------------------------------*
         SPACE 1
S01L180  DS    0H
         ICM   R2,B'1111',ICETSOUT   -> SORTOUT STATISTICS
         BZ    S01L190               BRANCH IF NONE
         STH   R9,ICE81ZOF         SAVE OFFSET
         USING ICESRTOT,R2
         L     R1,ICESORTO         LENGTH OF SORTOUT STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         DROP  R2
         SPACE 1
*------------------------------------------------------------------*
*        APPEND THE SORTWK STATISTICS.                             *
*------------------------------------------------------------------*
         SPACE 1
S01L190  DS    0H
         ICM   R15,B'1111',ICETSWK    -> SORTWK STATISTICS
         BZ    S01L200               BRANCH IF NONE
         STH   R9,ICE81WOF         SAVE OFFSET
         USING ICEWRKDS,R15
         L     R1,ICESORTW         LENGTH OF SORTWK STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R2,R15              FROM ADDRESS
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         SPACE 1
*------------------------------------------------------------------*
*        APPEND THE SORTWK DATA SET ENTRIES.                       *
*------------------------------------------------------------------*
         SPACE 1
         LH    R7,ICESWKEN         # SORTWK DATA SET ENTRIES
         STH   R7,ICE81EEN         SAVE IN SMF RECORD
         STH   R9,ICE81EOF         SAVE OFFSET
         LA    R15,ICESWK01        ->FIRST IN LIST
         DROP  R15
S01L195  DS    0H
         L     R2,0(,R15)          ->ICEWORK
         USING ICEWORK,R2
         L     R1,ICESWORK         LENGTH OF SORTWK DATA SET ENTRY
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         SPACE 1
         LA    R15,4(,R15)         ->NEXT IN LIST
         BCT   R7,S01L195          DO ALL IN LIST
         SPACE 1
*------------------------------------------------------------------*
*        APPEND THE HIPERSORTING STATISTICS.                       *
*------------------------------------------------------------------*
         SPACE 1
S01L200  DS    0H
         ICM   R2,B'1111',ICETHIPR   -> HIPERSORTING STATISTICS
         BZ    S01L210               BRANCH IF NONE
         STH   R9,ICE81HOF         SAVE OFFSET
         USING ICEHIPER,R2
         L     R1,ICEHSORT         LENGTH OF HIPERSORTING STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
```

```
         MVCL  R0,R2               APPEND DFSORT SMF RECORD
         DROP  R2
         SPACE 1
*------------------------------------------------------------------*
*        APPEND THE SORTING WITH DATA SPACE STATISTICS              *
*------------------------------------------------------------------*
         SPACE 1
S01L210  DS    0H
         ICM   R2,B'1111',ICETDATA   -> SORTING WITH DATA SPACE STATS
         BZ    S01L215               BRANCH IF NONE
         STH   R9,ICE81DOF         SAVE OFFSET
         USING ICEDATAS,R2
         L     R1,ICEDSORT         LENGTH OF DATA SPACE STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              ->AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT DATA SPACE STATS
         DROP  R2
         SPACE 1
*******************************************************************
*        APPEND THE DFSORT MEMORY OBJECT STATISTICS.               *
*******************************************************************
         SPACE 1
S01L215  ICM   R2,B'1111',ICETMO     -> MEMORY OBJECT STATISTICS
         BZ    S01L220               BRANCH IF NONE
         STH   R9,ICE81NOF         SAVE OFFSET
         USING ICEMOSEC,R2
         L     R1,ICEMOLNG         LENGTH OF MEMORY OBJECT STATISTICS
         LR    R0,R10              TO ADDRESS
         AR    R9,R1               INCREMENT LENGTH/OFFSET
         AR    R10,R1              -> AFTER SMF RECORD
         LR    R3,R1               FROM LENGTH
         MVCL  R0,R2               APPEND DFSORT MEMORY OBJECT STATS
         DROP  R2
         SPACE 1
*------------------------------------------------------------------*
*        SMF RECORD IS BUILT.  SEND IT OFF TO SMF.                 *
*------------------------------------------------------------------*
         SPACE 1
S01L220  DS    0H
         STH   R9,ICE81LEN         SAVE IN RDW
         BAL   R15,S01L700         BRANCH AROUND LABEL
         DC    C'USMF'             SVC PASSWORD
S01L700  DS    0H
         L     R0,0(,R15)          LOAD INTO PARM REGISTER
         LR    R1,R8               -> USER SMF RECORD
*******************************************************************
* SVC 249 IS USED IN THIS EXAMPLE - CHANGE THIS CALL AS APPROPRIATE
*******************************************************************
         SVC   249                 CALL SVC TO WRITE RECORD
         SPACE 1
*------------------------------------------------------------------*
*        ALL DONE.  FREE STORAGE AND RETURN.                       *
*------------------------------------------------------------------*
         SPACE 1
         LR    R1,R8               -> SMF RECORD
         L     R0,DSASMFL          LENGTH OF AREA
      FREEMAIN R,LV=(0),A=(1)
         SPACE 1
S01L950  DS    0H
         LA    R0,DSALTH           LENGTH OF AREA
         LR    R1,R13              ->AREA
         L     R13,4(,R1)          ->CALLER'S SAVEAREA
      FREEMAIN R,LV=(0),A=(1)
         SPACE 1
S01L990  DS    0H
        RETURN (14,12),,RC=0
         TITLE 'S D A - STATIC STORAGE AREA (CONSTANTS)'
         SPACE 1
*--------CONSTANT DEFINING THE COMBINED LENGTHS OF ALL DATA AREAS
*        EXCEPT FOR THE DFSORT SMF RECORD (ICESMF)
         SPACE 1
SDASMFL  DC    A(ICE81LTH+L'ICESMEND+L'ICEGNEND+L'ICEOEND+L'ICESSEND+L'X
               ICEVEND+L'ICETMEND+L'ICESNEND+L'ICESOEND+L'ICESWEND+(32*X
               L'ICESEEND)+L'ICEHREND+L'ICEDSEND+L'ICEMOEND)
         TITLE 'D S A - DYNAMIC STORAGE AREA'
DSA      DSECT
DSASAVE  DS    18A                 SAVE AREA
DSASMFL  DS    F                   LENGTH OF USER SMF RECORD AREA
DSALTH   EQU   (*-DSA)             LENGTH OF DSA
         TITLE 'DFSORT USER SMF RECORD GENERATED FROM ICETEXIT ROUTINE'
```

```
         SPACE 1
ICE81RCD DSECT
ICE81LEN DS    H                  RECORD LENGTH
ICE81SEG DS    H                  SEGMENT DESCRIPTOR
ICE81FLG DS    XL1                SYSTEM INDICATOR
ICE81RTY DS    AL1                RECORD TYPE = AL1(129)
ICE81TME DS    XL4                TIME, IN HUNDREDTHS OF A SECOND
ICE81DTE DS    PL4                DATE, IN FORM 00YYDDDF
ICE81SID DS    CL4                SYSTEM IDENTIFICATION
         SPACE 1
ICE81ROF DS    H                  OFFSET TO SORT SMF RECORD (ICESMF  )
ICE81SOF DS    H                  OFFSET TO SMF STATS       (ICESST  )
ICE81GOF DS    H                  OFFSET TO GENERAL STATS   (ICEGEN  )
ICE81OOF DS    H                  OFFSET TO OPTION STATS    (ICEOPTS )
ICE81MOF DS    H                  OFFSET TO SORT/MERGE STAT (ICESMS  )
ICE81VOF DS    H                  OFFSET TO VIRT STOR. STAT (ICEVSTOR)
ICE81TOF DS    H                  OFFSET TO TIMING STATS    (ICEPHAST)
ICE81XOF DS    H                  OFFSET TO SORTIN STATS    (ICESRTIN)
ICE81ZOF DS    H                  OFFSET TO SORTOUT STATS   (ICESRTOT)
ICE81WOF DS    H                  OFFSET TO SORTWK STATS    (ICEWRKDS)
ICE81EOF DS    H                  OFFSET TO SORTWK ENTRY    (ICEWORK )
ICE81EEN DS    H                  # OF ICEWORK ENTRIES
ICE81HOF DS    H                  OFFSET TO HIPER STATS     (ICEHIPER)
ICE81DOF DS    H                  OFFSET TO DATASPACE STATS (ICEDATAS)
ICE81NOF DS    H                  OFFSET TO MEM. OBJ. STATS (ICEMOSEC)
ICE81LTH EQU   *-ICE81RCD         LENGTH OF COMMON PORTION
        ICESMF
        ICEDTEX
         PRINT NOGEN
         CVT   LIST=NO,DSECT=YES
       IEESMCA
         YREGS
         END
```

```
IGC0024I TITLE 'SAMPLE SVC 249 - WRITE A USER SMF RECORD'
*--------------------------------------------------------------------*
*       SAMPLE SVC 249 - WRITE A USER SMF RECORD                     *
*       PURPOSE:                                                     *
*                THIS SVC IS INVOKED BY THE SAMPLE ICETEXIT          *
*                ROUTINE TO WRITE THE USER SMF RECORD WITHOUT        *
*                BEING APF AUTHORIZED.                               *
*                                                                    *
* NOTES:                                                             *
*   1) THIS EXAMPLE ASSUMES THE USE OF SVC 249 - CHANGE IT AS        *
*      APPROPRIATE IF YOU USE A DIFFERENT SVC.                       *
*   2) THIS EXAMPLE USES A SIMPLE PASSWORD CHECK TO VERIFY THAT IT   *
*      IS BEING CALLED BY ICETEXIT.  SINCE THIS PASSWORD SCHEME COULD *
*      EASILY BE COMPROMISED, YOU SHOULD CAREFULLY EVALUATE THE      *
*      RISKS IN USING IT AS IS IF YOU INSTALL IT IN YOUR SYSTEM.     *
*                                                                    *
* REQUIRED SYSTEM MACROS: SMFWTM, YREGS                              *
*--------------------------------------------------------------------*
         SPACE 1
IGC0024I CSECT
         USING IGC0024I,R6
         C     R0,PASSWORD          IS THE PASSWORD CORRECT?
         BNER  R14                     NO..RETURN TO CALLER
         SMFWTM (1)                 WRITE THE SMF RECORD
         BR    R14                  RETURN TO CALLER
         SPACE 1
         DS    0F
PASSWORD DC    CL4'USMF'
         YREGS
         END
```

Figure 15: Sample SVC 249 to Write a SMF User Record

Appendix B. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed email message to mhvrcfs@us.ibm.com.

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or `3.1` or `3.1.1`. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number `3.1`) are mutually exclusive alternatives. If you hear the lines `3.1 USERID` and `3.1 SYSTEMID`, your syntax can include either `USERID` or `SYSTEMID`, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*** indicates an optional syntax element that is repeatable**
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.

3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line `6.1+ data area`, you must include at least one data area. If you hear the lines `2+, 2 HOST,` and `2 STATE`, you know that you must include `HOST, STATE,` or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for the Knowledge Centers. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

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 is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information 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's Privacy Policy at ibm.com/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming interface information

This publication primarily documents information that is NOT intended to be used as Programming Interfaces of DFSORT.

This publication also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of DFSORT. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

[Programming Interface Information] [End Programming Interface Information]

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

UNIX is a registered trademark of The Open Group in the United States and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

SC23-6882-30