System z

Managing a z/OS Environment on a Uni-processor LPAR

This document can be found on the web, www.ibm.com/support/techdocs
Search for document number WP100925 under the category of “White Papers.

Version Date: 6/30/2014

IBM Advanced Technical Support

Kenneth Stine
Kathy Walsh

Trademarks

The following terms are registered trademarks of International Business Machines Corporation in the United States and/or other countries: AIX, AS/400, DB2, IBM, Micro Channel, MQSeries, Netfinity, NUMA-Q, OS/390, OS/400, Parallel Sysplex, PartnerLink, POWERparallel, RS/6000, S/390, Scalable POWERparallel Systems, Sequent, SP2, System/390, ThinkPad, WebSphere.

The following terms are trademarks of International Business Machines Corporation in the United States and/or other countries: DB2 Universal Database, DEEP BLUE, e-business (logo), ~, GigaProcessor, HACMP/6000, Intelligent Miner, iSeries, Network Station, NUMACenter, POWER2 Architecture, PowerPC 604,pSeries, Sequent (logo), SmoothStart, SP, xSeries, zSeries. A full list of U.S. trademarks owned by IBM may be found at http://iplswww.nas.ibm.com/wpts/trademarks/trademar.htm. NetView, Tivoli and TME are registered trademarks and TME Enterprise is a trademark of Tivoli Systems, Inc. in the United States and/or other countries.
Oracle, MetaLink are registered trademarks of Oracle Corporation in the USA and/or other countries.
Microsoft, Windows, Windows NT and the Windows logo are registered trademarks of Microsoft Corporation in the United States and/or other countries.
UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.
LINUX is a registered trademark of Linus Torvalds.
Intel and Pentium are registered trademarks and MMX, Pentium II Xeon and Pentium III Xeon are trademarks of Intel Corporation in the United States and/or other countries.
Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries.
Other company, product and service names may be trademarks or service marks of others.

Table of Contents

Introduction

The focus of this paper is on managing work on a uni-processor system including the impact of looping or CPU-intensive tasks. The topics covered are:

- Changes in the MVS-ready dispatcher to help SRM run more work
- WLM policy practices and MVS settings to help manage this environment
- Application looping work and its impact on the system
- System type work which is CPU-intensive at times
- z/OS functions which override WLM assigned dispatching priorities
- New features of the EC/BC 12 that could impact performance

Uni-processor systems may exist as either an LPAR on a single-engine physical processor (a one way), or as a "small LPAR" on a larger n-way physical processor. For this discussion, a "small LPAR" is one with a PR/SM weight guarantee less than 50% of an engine based on its assigned weight when the processor runs at capacity. Often, a minimum of two logical CP's will be assigned even to these small LPARs because clients are concerned about the availability of uni-processor systems.

In the past, running a small LPAR as a 2-way could result in "short engines" and cause performance problems. "Short engines" occur on busy processors. A "short engine" is when the per CP share of the logical processors is considerably less than 100%. Generally logical processors with a per CP share less than 60% are considered to have a short per CP share.

The per CP share indicates the amount of capacity over a time interval the logical processor should receive when there is contention. The PR/SM dispatcher manages the allocation of this capacity based on a time slicing algorithm. Physical engines are assigned to a logical processor for the duration of a time slice.

Let's set an example. Say an LPAR is allowed 30% of a 7-way processor. By weight the LPAR is allowed 2.1 CPs of capacity. The HiperDispatch mode and the number of logical CPs assigned to the LPAR will determine the CP share.

In a HiperDispatch=No case the LPAR is assigned four logical CPs. The 2.1 CPs of capacity is spread evenly over the four logicals giving a per CP share of 53%, (2.1 CPs of share / 4 medium logical processors = 53% share). This is a short CP; 53% of the time the logical will have a physical assigned and 47% of the time no physical engine will be assigned. This means 47% of the time any transaction assigned to this logical engine will not be able to execute instructions.

In a HiperDispatch=Yes case the LPAR can still be assigned four logical CPs. However HiperDispatch will allocate the per CP share differently. In this case with 2.1 CPs of capacity allocated HiperDispatch will assign 1 CP of this capacity to a logical engine designated as a vertical high with a 100% per CP share. The remaining 1.2 CPs of capacity are assigned to two logical CPs designated as vertical mediums with 60% share,

(1.2 CPs of share / 2 medium logical processors= 60% share). The remaining fourth logical CP will be designated as a vertical low and will receive 0% share. This logical will be parked or unparked depending upon system utilization and PR/SM weight enforcements.

With the introduction of the zEC12, a new feature called Warning Track, is provided to help environments running HiperDispatch=YES with "short engines." With the Warning Track support a z/OS system with a logical processor who is going to lose it's physical processor due to the end of time slice will now have an opportunity to be warned by PR/SM when a physical engine is going to be taken away from the logical engine. A Warning Track interrupt will be fielded by z/OS which will give it the opportunity to save status for the unit of work currently assigned to the logical processor and re-queue the work unit back to the dispatcher queue. By being moved back to the work queues any other logical processor still running, (hence still has a physical assigned) is eligible to select the work unit and allow the transaction to keep running on a different logical processor.

Before we discuss managing work on a uni-processor, let us review two capabilities in MVS which set the groundwork for being able to run CPU-intensive work on uni-processor systems.

Two Changes in MVS Reduced the Impact of Looping Work on a Uni-processor

Two of the capabilities in z/OS which positively impact the ability to run z/OS on a uni-processor are “reduced preemption” and “fair share” dispatching. Prior to these functions becoming available, looping work at a high priority could have easily dominated a uni-processor.

“Reduced preemption” assumes it is better to allow work to execute until it either gives up the CPU voluntarily or reaches the end of a timed period. At this point, the work queue is searched and if it contains higher priority work, then higher priority work will get dispatched. This results in more efficient use of the processor at minimal cost to the higher priority work.

MVS 5.1 introduced fair share dispatching. Before MVS 5.1, address space selection was on a first come, first served basis for each dispatching priority. A TCB dispatched was not interrupted to dispatch an SRB or TCB of an address space with the same priority. This allowed one address space to dominate others of equal priority which often lead to performance and throughput problems. The Workload Manager goal mode implementation required fairer access and had a need to run a system with fewer dispatch priorities. With fair-share dispatching, any work placed on a logical engine is assigned a slice of time and when the time expires the long-running task is rotated behind other work units with the same dispatch priority. Though a high priority task will lock out work at lower dispatch priorities, tasks at the same or higher dispatch priorities will continue to run,

What does this mean to z/OS running on a uni-processor? It means:

1) A looping program will only impact work that is assigned a lower dispatching priority.
2) Lower priority work which is dispatched will often be allowed to use the processor for one time slice even if higher priority work becomes ready immediately after the dispatch.
3) Two or more CPU-intensive units of work at the same dispatching priority will get equal access to the single CP.

The Characteristics of a Good WLM Policy

Dispatching priority is the key factor in managing work in a uni-processor environment. In goal mode, only the SYSTEM service classes are assigned a specific dispatching priority. It is critical, therefore, a clear dispatch priority order be established for which work should run when the processor is 100% busy. The dispatch priority should be reflected in the use of the importance levels assigned to the service classes.

Recommendations for the WLM policy:

- Use importance levels, not velocity values, to tell WLM how to set dispatching priorities when the processor is at capacity. Remember all work in the same service class runs at the same dispatching priority.
- Set WLM goals that are achievable. Goals for important work should be set so their PI's are close to 1. This will help ensure they do not become donors to lower importance work.
- Select the peak intervals for both online and batch and note how workloads are achieving their goals and revise the service class definitions accordingly.
- Use report classes to provide more granular workload reporting and be able to better identify victims and abusers of CPU-intensive work.
- Ensure you have unique default service and report classes defined for each subsystem type, again to easily identify new work that could potentially be CPU-intensive.
- Watch for large CPU consumers in the SYSSTC service class and if possible, move to a managed service class.
- Use concepts such as WLM CPU Critical sparingly and only to protect critical online components such as DB2.

In addition to a well-defined WLM policy, there are other techniques to manage out of control CPU-intensive work.

Managing CPU-Intensive Work

There are several types of work that can be CPU-intensive:

1) Application programs that loop due to programming errors
2) High-priority system type tasks
3) Other high priority goal-managed tasks

Application Programs:
Looping application programs are very CPU-intensive with minimal or no I/O and may require outside intervention to end the task if it is truly looping. These looping applications can block lower importance workloads from running in the system. To combat this problem, new support was added for blocked workloads, which monitors how long work has been starved of CPU resources and promotes it to a higher dispatch priority for a small amount of time.

> **Recommendation**:
> Define a TSO user id assigned to the SYSSTC service class. This will allow you to get to SDSF to cancel any looping work if needed.

Managing CPU-intensive, non-swappable work:
When a system attempts to run high priority CPU intensive work it can run into a situation where work running at lower dispatch priorities will be starved of resources. For example when a System Programmer initiates an *Activate* command to dynamically update the IODF, the processing necessary to bring units online/offline could end up using all the available capacity of the engine and bring all work running with lower dispatch priorities to a halt. To avoid this kind of outage IBM recommends you use a 'Limit" service class that will allow a CPU intensive job such as an *Activate* command to be set temporally to a lower dispatch priority allowing work to continue to run as the command continues to execute.

To define a "limit" service class, create a new service class called "LIMIT" with a discretionary goal and assign it to a resource group defined with a minimum capacity of 0 and maximum capacity of 1. If you have a looping address space, then use the command E jobname,SRVCLASS=LIMIT to assign the service class to the address space.

Managing CPU-intensive, swappable work:

- Define multi-period service classes so CPU-intensive work can drop to a lower period with a low importance or discretionary goal
- Limit the number of initiators for each JES class
- Use the IEFUJV exit to limit CPU by setting the TIME= parameter
- Define strong JES class standards
- Use the reset command E jobname,QUIESCE.

Dynamic Activate:
There are several required operations that run as a result of an *ACTIVATE* command. The new Input/Output Definitions File (IODF) is compiled to replace the production IODF. During this time the new IODF is compared to the production IODF and the differences are processed by IOS. This processing may dramatically affect your performance depending on the dispatching priority of the issuer. If the command is issued from the HCD dialog it will run in the TSO user's address space and therefore have that user's dispatching priority. Since this is often done by a z/OS systems programmer care should be taken to ensure the WLM goal definition for the system programmer userids are not defined with exceptionally high access.

By allowing work to run at the TSO user's goal concurrently running applications with higher priority levels will still be dispatched. If the command is issued from the command line of a z/OS console it will run in the Master address space and have the highest system dispatching priority, (x'FF'.) All other applications are most likely running at a lower dispatch priority and may see significant CPU demand as the applications wait for the CPU intensive Activate command to complete. This may cause performance problems to high priority online workloads in a uni-processor environment.

Recommendation for Dynamic Activate:

- If possible run the Dynamic Activate on a LPAR with more than one CP assigned to it. If not then run the *ACTIVATE* command via the HMC interface where the dispatch priority will be assigned based on the TSO userid. Verify this TSO id typically defined with a lower dispatch priority than online workloads.
- Another alternative is to dynamically activate a smaller range of devices allowing the scope of the command to be smaller. This is especially true if the command has to be entered via the z/OS console.

HiperDispatch:
HiperDispatch was first introduced in February, 2008 and it provided the ability to dynamically park and unpark a logical processor based on an LPARs relative weight and its demand for additional whitespace. For uni-processor LPARs, there is no advantage or disadvantage to enabling HiperDispatch. With a single processor, neither the ability to park and unpark logical CPs nor the ability to allow work to have dispatch affinities to specific logical processors are an issue.

Recommendation:

- Having HiperDispatch running does not provide any disadvantage. With the introduction of z/OS 1.13, the default is to have HiperDispatch enabled. Rather than having to run with different IEAOPTxx parameter libraries, it is recommended to turn HiperDispatch on.

SYSTEM and High Importance Goal-Managed Work:
System tasks in the SYSTEM and SYSSTC service classes run at a fixed dispatching priority of FF and FE respectively. CPU-intensive system work includes, for example, dumps, contention for spin locks, high volume SRBs, job initiation functions, processes such as page stealing, IRLM processing, system and network automation functions, and large CPU consumers incorrectly classified to SYSSTC.

Goal-managed, i.e., in a managed service class, high importance CPU-intensive work includes such functions as DB2 queries, JAVA garbage collection, and CICS/VSAM. If possible, you want to minimize the time these functions need the CP. When these functions use the one logical CP, work such as CICS onlines may wait and response time can increase or spike for short periods. Refer to the applicable subsystem documentation for tuning guidance.

Job Initiation:
The following discussion focuses on job initiation when the job is being managed under the initiator's dispatching priority. Normally, batch work is assigned a lower importance goal than online work; however, some of the work done as part of batch job initiation runs at the same dispatching priority as SYSSTC. Recent changes to z/OS allow the initiator's dispatching priority to be set lower than SYSSTC's which may be desirable when there is only one logical CP.

When a job is first selected by an initiator, there are a set of activities executed as part of job initiation prior to its assignment to a service class in the SYSEVENT JOBSELECT processing. Generally the CPU demand for this path length is very small but if a large number of jobs enter the system and are initiated at once the CPU intensity of the path increases. On a uni-processor it is more disruptive, as all other work would wait. Prior to z/OS 1.5, this work ran at the fixed dispatching priority of FE, the same as SYSSTC. On a single logical CP system this work could have dominated the processor and impacted higher importance online work.

In z/OS 1.5 and later, WLM will control the dispatching priority for JES, APPC, and OMVS initiators based on the INITIMP parameter setting in the IEAOPT member of PARMLIB. There are five possible values: 0, 1, 2, 3, and E. The default setting is 0 which sets the dispatching priority of the initiator to x'FE', the same as SYSSTC.

When lower importance batch job initiation becomes CPU-intensive on a uni-processor, the INITIMP=0 setting can result in delays to higher importance online work. The settings of 1, 2, or 3 set the dispatching priority to be lower than the dispatching priority for CPU critical work within the corresponding importance level. If there is no CPU critical work in the system, the dispatching priority will be set as for option E. A value of E states the dispatching priority will be calculated dynamically. This dynamically chosen dispatching priority should ensure sufficient access to the processor but not high enough to impact more important work.

Recommendation:

- For uni-processors, set INITIMP=E in the IEAOPTxx PARMLIB member.

The amount of time spent in job/step initiation is reported in the SMF30ICU field in the SMF type 30 subtypes 3 and 4 records. If this time is significant compared to the time actually spent executing the program then you can investigate the areas noted above to try and reduce this time. Each subsystem such as RACF and SMS has their own hints and tips for improving performance. In an ACS routine, for example, the number of volumes in a storage group or a storage pool's free space levels can impact the time it takes to find a candidate volume for data set allocation.

Areas where the Dispatching Priority can be Overridden

Even with a well-defined WLM policy in place, there are situations when the dispatching priority of a higher importance service class may be overridden to allow lower importance work to run for some period of time. In a uni-processor environment these can unexpectedly impact, for example, response time for online transactions. These areas are

- ENQ contention
- LCK contention
- Discretionary Goal Management (DGM)
- Resource Group Minimum
- Dynamic Activate

ENQ Contention:

In z/OS 1.3, ENQ management was changed to ensure critical work is not held back by resources because the unit of work holding the resource is swapped out or not receiving services. When a low priority unit of work holds an ENQ on a resource that a higher dispatching priority unit of work is waiting on, WLM will raise the dispatching priority of the ENQ holder in the hopes that it will finish and release the hold. The length of time for which the dispatching priority is raised is determined by the ERV parameter in the IEAOPTxx member of PARMLIB. The ERV value is the number of CPU service units an address space or enclave is allowed to absorb when it is possibly causing enqueue contention. The dispatching priority is dynamically set every 10 seconds to a value which would give access to CPU where 40% of the capacity is used. The default ERV value is 500. This is a very small value when, for example, the SU/SEC on 2827-701 is over 78,000.

A test was run to see what happens when a low priority "HOLDER" job holds an ENQ on a data set (DISP=OLD) needed by a high importance "WAITER" job. The ERV value was set to over 100,000 in order to use SDSF to observe the changes in dispatching priorities. At one point the dispatching priority, (DP), of the importance 5, velocity 5 HOLDER was raised to the same priority, F9, as the importance 2, velocity 40 work. CPU busy for the importance 5 service class was over 13%. Then the HOLDER was put back to a lower DP and CPU busy was zero. Once the job reached the ERV value, it was

not promoted again even though it continued to hold the ENQ. There was no subsequent indication to SRM of the ENQ contention. The CPU time accumulated when an address space is ENQ promoted is recorded in the SMF type 30 SMF30CEPI field.

Recommendation:

- Increase the ERV value in IEAOPTxx PARMLIB member to allow a lower priority unit of work holding an ENQ to receive some CPU service in the hopes that it will complete and release the ENQ.

Blocked Workload Support

Blocked workloads occur when CPU intensive work utilizes all of the potential CPU resources in an LPAR. This effectively starves all lower priority work of the CPU resources and can bring the whole image to a halt. On an LPAR with two or more engines, this situation is less prevalent because work can always be dispatched to a different engine. However, with a uni-processor LPAR, blocked workloads could cause major performance delays and possibly an outage.

As an example, you have an image running with a single processor assigned to it. During the course of normal operation for example, a started task with a high dispatch priority starts to poll a resource's status bit to see if it has changed. Because of its high dispatch priority, this started task takes control of the processor in a tight CPU loop. While this started task is looping, all other lower priority work is shut out of the system. If the task that will update the status bit is among the lower priority work, it will be unable to be dispatched to update the status bit. The z/OS will require manual intervention to stop the looping task. This is a situation where a uni-processor environment has a lower availability than a two-way or greater environment. In a two-way or greater environment the lower priority work would be able to run on the second engine to clear the status bit. A function in z/OS called Blocked Workload Support is one feature that can help reduce the availability impact of a uni-Processor environment like the one described above.

Blocked Workload Support allows CPU starved workloads to receive trickles of CPU time. Blocked workload support is specified in the IEAOPTxx member and is used to define how long the workload has to be waiting and how much starved work can be promoted at once. To demonstrate blocked workload support, a series of tests were run in an LPAR with one logical CP. A set of six soaker jobs were set at importance three and a set of eight soaker jobs were set as importance five. First, with block workload support disabled, six soaker jobs in the importance three service class were submitted bringing the LPAR to 100% busy. SDSF showed each job used an equal share of the CPU resources. This was fair-share dispatching in action.

Next, the eight importance five service class jobs were submitted. While the high importance jobs continued to share the CPU resources, the lower importance looping jobs were completely shut out.

z/OS blocked workload support was then enabled. The BLWLINTHD parameter was set to five seconds before the workload could be promoted and the BLWLTRPCT value was

set to five to allow a 0.5% cap on promotable work. The test was rerun using the same setup. The RMF reports confirmed the importance five workload now receives CPU resources. Through additional RMF metrics we verified that CPU resources were obtained through Blocked Workload Support.

In the IEAOPTxx member define parameters:

- BLWLINTHD=value range (3-65535) | IBM recommendation = 5
- BLWLTRPCT=value range (0-200)

Discretionary Goal Management:

The objective of Discretionary Goal Management (DGM) is to allow discretionary work to get CPU service when other non-discretionary work is over-achieving its goal. It accomplishes this by capping the over-achieving work. The potential donor is work in a service class with any importance and a velocity goal of 30 or less or a response time goal of more than 1 minute. The over-achieving work must have a PI of less than 0.7. The potential receiver is work in a service class with a discretionary goal. While this may be acceptable on an LPAR with multiple logical CP's, the potential capping of high importance work in a single CP environment may not produce the same desirable results.

Recommendations if you want to prevent DGM from occurring:

- Assign those potential donor service classes to a resource group defined with a null minimum and maximum values.
- Adjust the goals for the potential donor service classes so the resulting PI's are closer to 1 in peak periods

Resource Groups:

It is possible for work in service classes assigned to resource groups to impact higher importance work when the resource group has been defined with a minimum value. This can occur even if it causes the more important work to miss its goal. If a resource group is not meeting its minimum capacity **AND** work in the resource group is not meeting its goal, then WLM will try to give CPU resource to the work in the resource group by increasing the dispatching priority. Note the minimum resource group setting will have no effect if the work is meeting its goal, even if the minimum capacity has not been met. If the work in the resource group is discretionary and the minimum has not been met then it will only get cycles if it does not cause other work to miss its goal.

Recommendation for resource groups:

- Review resource group definitions with minimum capacity settings to better understand the potential impact of work in the service classes assigned to them on higher importance work in the system. Adjust if needed.

Other Items for your Consideration:

z/OS Health Checker Facility

The objective of the z/OS Health Checker Facility is to simplify and automate the identification of potential configuration problems before they impact system availability. This program is integrated into z/OS 1.7 and higher and includes checks on the following subsystems/ functions: GRS, RACF, real storage, virtual storage, system logger, dumping, and XCF. Since these functions run at high dispatching priorities, using the z/OS Health Checker can help to ensure they run efficiently.

Recommendation:

- Install and use the z/OS Health Checker on a regular basis.

GRS Contention Notification System

An enhancement was provided in z/OS 1.8 to allow you to change the GRS Contention Notification System through introduction of a GRS command. If the system which is performing the GRS Contention processing leaves the sysplex for whatever reason it is possible for a small LPAR in the sysplex to be identified as the system now charged with performing the contention notification function.

Recommendation:

- Use the SETGRS CN command to move the contention notification process to a more functional system.

Summary:

Running a uni-processors environment requires detailed understanding of workload behavior and a well-running WLM policy which accurately reflects the business priorities for the work and also contains achievable goals. The ability to actively monitor work, identify abusers and quickly take action is critical to keeping the remaining work running well.

References

Rogers, Bob. "Warning Track." *IBM Destination Z*. N.p., 10 Apr. 2013. Web. 12 Nov. 2013. <http://www.destinationz.org/Mainframe-Solution/Trends/Improving-CPU-utilization-on-zEC12.aspx>.

"MVS Workload Manager Velocity Goals: What you Don't Know Can Hurt You", http://www.ibm.com/servers/eserver/zseries/zos/wlm/documents/velocity/velocity.html, John Arwe author

System Programmer's Guide to: Workload Manager, SG24-6472, IBM Redbook

"z/OS Availability: Blocked Workload Support", Techdoc FLASH10609, Kathy Walsh

z/OS MVS Initialization and Tuning Reference, SA22-7592

"z/OS Workload Manager Unknown Knowns", Techdoc TD101715, Jim McCoy author