

zSeries

**Title: Managing CPU-Intensive Work on Uniprocessor
LPARs**

December 18, 2006

Linda August

IBM Washington Systems Center

Advanced Technical Support - Americas

Trademarks

The following terms are registered trademarks of International Business Machines Corporation in the United States and/or other countries: z/OS, CICS, RACF.

The following terms are trademarks of International Business Machines Corporation in the United States and/or other countries: DB2, MVS, PR/SM, zSeries. A full list of U.S. trademarks owned by IBM may be found at <http://iplswww.nas.ibm.com/wpts/trademarks/trademar.htm>.

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

Introduction

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

- Changes in the ready dispatcher to help SRM run work more efficiently in general
- WLM policy and MVS settings to help manage this environment
- Managing CPU-intensive work - application and system type
- z/OS functions which override WLM assigned dispatching priorities

Uniprocessor systems may exist as either an LPAR on a single-engine physical processor or as a “small LPAR” on a larger n-way physical processor. For this discussion, a “small LPAR” is one that is guaranteed less than 50% of an engine based on its assigned weight when the processor runs at capacity. Often, a minimum of two logical CPs will be assigned to these small LPARs because clients are concerned about uniprocessor systems. However, running a small LPAR as a 2-way can result in “short engines” and cause performance problems. “Short engines” occur on busy processors when the number of logical CPs exceeds the number guaranteed to an LPAR based on its weight. The PR/SM dispatcher manages work based on time slicing and will take the physical CP away at the end of the slice. An LPAR’s guaranteed share is spread across all of the logical CPs assigned. PR/SM is not aware of the importance of the task running on the logical CP and WLM is not aware that the physical CP was taken away. Short engines may potentially impact single-tasking workloads that need the bigger logical CP as well as work on the larger LPARs that need resources being held by work running on the small LPARs.

Two capabilities in MVS set the groundwork for being able to run CPU-intensive work on uniprocessor systems.

Two Changes in MVS Reduced the Impact of Looping Work on a Uniprocessor

Two of the capabilities in z/OS which positively impact the ability to run z/OS on a uniprocessor are “reduced preemption” and “fair share” dispatching. Prior to these functions becoming available, looping work at a high priority could have easily dominated a uniprocessor. “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. In compatibility mode, the FIFO implementation often resulted in complex installation performance specifications (IPS) that could include the use of time slicing. The Workload Manager (WLM) goal mode implementation required fair access so that WLM could manage a few priorities rather than the many unique dispatching priorities that one found in a typical IPS. With fair-share dispatching, the long-running TCB is rotated behind other work units with the same address space priority.

What does this mean to z/OS running as a uniprocessor? 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 be allowed to use the processor 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 uniprocessor environment. In goal mode, only the SYSTEM service classes are assigned a specific dispatching priority. It is critical, therefore, a clear priority order be established for which non-system work should run when the processor is 100% busy. That 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 that 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.

In addition to a well-defined WLM policy, there are other techniques to manage 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 IO and require outside intervention to end the task if it is truly looping. To demonstrate how looping work impacts a uniprocessor, a series of tests were run in an LPAR with a single logical CP using a set of soaker jobs to represent high importance online work as well as low importance looping work. First, multiple looping jobs in the same high importance service class were submitted. SDSF showed each job used an equal share of the CPU. This was fair-share dispatching in action. Next, a tight looping job in an importance 5 service class and multiple CPU-intensive jobs in an importance 2 service class were run. The high importance jobs shared the CPU resource and completely shut out the lower importance looping job. The lower importance job did not get CPU cycles until the higher importance work ended. This demonstrated that looping applications will only impact lower importance work and equal importance work will share the CPU resource on a uniprocessor. This again points out how critical it is to have clear goals allowing WLM to set the priority of how work should run when the processor resource is constrained.

The following recommendations help control looping or CPU-intensive applications:

Recommendation to manage CPU-intensive, non-swappable work: Define a “limit” service class 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.

Recommendations to manage 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`.

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

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. 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.

Job initiation: When a job is first selected by an initiator, there are a set of activities that are executed as part of job initiation prior to its assignment to a service class in the SYSEVENT JOBSELECT processing. These tasks include routines from SAF(RACF), user exit IEFUJI for account code verification, and SMS ACS. It also includes the starting and stopping of WLM-managed initiators. This work can become CPU-intensive depending on the complexity of the exit, the number of data sets referenced, the number of lines in the ACS routines, the extent of RACF profile checking, and even the number of jobs submitted at one time. 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 IEAOPTxx member of PARMLIB. There are five possible values: 0, 1, 2, 3, and E. 0, the default, sets the dispatching priority of the initiator to 254, the same as SYSSTC. When lower importance batch job initiation becomes CPU-intensive on a uniprocessor, the INITIMP=0 setting can result in delays to higher importance online work. A value of 1, 2, or 3 sets 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 uniprocessors, 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 uniprocessor environment these can unexpectedly impact response-oriented work. These areas are

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

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 2084-301 is over 21,000.

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.

Discretionary Goal Management: The objective of Discretionary Goal Management (DGM) is to allow discretionary work to get CPU service when other nondiscretionary work is overachieving its goal. It accomplishes this by capping the overachieving work. The potential donor is work that is 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 overachieving work must have a PI of less than .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 CPs, 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 that the resulting PIs are closer to 1 in peak periods

Resource Group Minimum: 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 group is not meeting its goal, then WLM will try to give CPU resource to that group by increasing the dispatching priority. Note that the minimum 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: 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 as well as virtual storage, 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.

There is an enhancement in z/OS 1.8 that is also rolled back to 1.7 to allow you to define the GRS Contention Notification System. This will ensure that a uniprocessor member of a sysplex is not selected as the system to make notifications by GRS.

Summary

Running a single CP system 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 well-running. There will always be some events you cannot control. For these, try to minimize the likelihood of them occurring as well as their CPU consumption.

Acknowledgments:

Thank you to Kathy Walsh, IBM Distinguished Engineer, eServer-zSeries, Performance, and Ulirch Hild, IBM Development WLM Team, for reviewing this document.

References:

“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 MVS Initialization and Tuning Reference, SA22-7592

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