Cheryl Watson's z/OS 101 Primer

2012

The z/OS 101 Primer

At the 2008 SHARE conference held in Orlando, I gave a lunchtime presentation called *The Tips Your Mentor Forgot to Mention*. It was for the zNextGen project, which was created to help those people who are new to mainframes or new to performance. The response to the project was incredible. One teacher even brought several students from his class in Georgia to the conference JUST to attend the zNextGen sessions.

Because of the project's success, and because several of our readers responded so favorably to my advice for new techies, I decided to include a new section in our Tuning Letters called "z/OS 101". Each z/OS 101 article addresses a topic that should be useful to those new to mainframes, especially in the performance, capacity planning, data center reporting, and charge back areas. They're also good as a review of the basics. If you'd like some special topic addressed, please let me know.

This particular document is a compilation of the 2012 z/OS 101 articles written so far and is offered to the public via our website. Please use it and copy it as you see fit, provided that you use the entire document if you distribute it, always credit us as the source, and make no attempt to resell it. The most recent compilation, as well as all previous complications can always be found at www.watsonwalker.com/articles.html.

So, hats off to our next generation of z/OS techies, and welcome to the exciting world of performance!

z/OS 101

This series of articles is designed for those people who are relatively new to z/OS systems programming or performance. In the first article, I'll describe how and why the Systems Resource Manager (SRM) uses the 'mean time to wait' (MTTW) method in order to improve throughput in a system. The second article is a reminder about an important series of Redbooks.

Multi-Period Batch

The following article was first published in our Tuning Letter 2004 No. 6, pages 29-32. We think it's especially important for people new to MVS to understand the consequences of using multiple periods for batch, and to try to avoid them.

What are the advantages and disadvantages of running batch in single-period service class versus a multi-period service class?

We must have heard this question at least six times at the latest SHARE. Although we did provide an answer in our September 1994 TUNING Letter, we think it's time for an update. We'll address the considerations for both test and production jobs, because they tend to have different requirements.

Test Batch

If your intention is to provide the best turnaround to the most people by allowing large resource consumers to suffer slightly, then you'll want to use the typical method of managing test batch jobs. That method simply consists of getting as many of the small jobs through the system, at a high dispatch priority, as you can. You would then let the larger jobs run at a lower priority, and possibly miss their service goals.

This technique is used in almost every data center today. The only difference is in how it's implemented. Let us describe the two typical methods and the pros and cons of each.

Priority by Job Classes

The most common technique is to define a set of test batch job classes that allow a certain set of resources. For example, you might define the following test batch job classes:

A - Less than 1 second CPU time, no tapes - 10 minute turnaround
B - Less than 5 seconds CPU time, 0 to 1 tape - 30 minute turnaround
C - Unlimited CPU time, 0 or 1 tape - 2 hour turnaround
D - Unlimited CPU time, unlimited tapes - overnight

Then you would define some JES initiators to process these jobs. There are dozens of ways to set up initiators, but a typical scenario, might be:

Init 1 - Classes: A
Init 2 - Classes: A
Init 3 - Classes: B
Init 4 - Classes: BA
Init 5 - Classes: CA
Init 6 - Classes: DCBA

You would then set up a single period service class for each job class. As one example:

TSTBATA - 90% within 10 minutes
TSTBATB - 90% within 30 minutes
TSTBATC - period 1 = velocity of 20%; period 2 = discretionary
TSTBATD - discretionary

We're making an assumption that there aren't enough ended class C jobs to allow a response time goal.

The advantage of this technique is that the initiators will determine the highest priority jobs to allow into MVS. If the operators feel that the system is too busy at the moment, they can close down the initiators in order of 6, 5, 4, 3, 2 and 1. When jobs in classes A and B get onto an initiator, they'll go into a single-period service class and stay at the same dispatch priority while they're executing. For those job classes, the first jobs on an initiator are normally the first jobs completed.

Job classes C and D, on the other hand, have unlimited CPU time. They might need 20 seconds of CPU time or three hours of CPU time - you don't really know. Therefore, the multi-period batch allows you to push the smaller of these large jobs through the system by setting the dispatch priority of period one to provide higher performance.

Priority by Period

Prioritizing test batch jobs by their actual use rather than their anticipated use is another common technique. In this method, there would be just one test batch job class. The initiators would be used to manage the number of test jobs in the system, but wouldn't differentiate between the short jobs or the long jobs.

A service class for this method might have four periods and look like:

Period 1 - 90% within 10 minutes, duration = 1000 Service Units (SUs)
Period 2 - 90% within 30 minutes, duration = 3000 SUs
Period 3 - velocity of 20%, duration = 10000 SUs
Period 4 - discretionary

All test jobs would enter the system in a first-come, first-served order. As soon as MVS sees them, they will be run at a high dispatch priority until they've consumed 1000 service units. Those jobs taking less than 4,000 service units (1000 in period one and 3000 in period two) have the next highest priority and will be completed next. The

longer jobs will compete at the same low priority, with the smaller jobs typically completing first.

Comparison

The first method using job classes takes more effort on the part of the sysprogs and the programmers that submit jobs. The sysprogs will need to analyze the current data to determine appropriate job class groupings, the job class information will need to be distributed to the users, and the users will need to estimate their job's usage ***before*** they submit the job. If they guess too low, the jobs will ABEND with a time-out for the job class. If they guess too high, they'll get poorer service than they deserve. If job class designations change, there may be additional ABENDs because programmers have a tendency to use "old JCL," change a line or two (seldom the job class) and submit the job. If the job runs in the wrong job class, you may have an ABEND. This technique is not too productive for the programmers, but will result in the best turnaround times for the majority of users. (As you'll see next.)

Multi-period batch service classes are one of the primary causes for missing your batch window goals

The second method is very simple to use, but can lead to severe problems. The programmers don't need to be concerned with which job class to use, and the sysprogs don't need to do any analysis (other than determine the appropriate durations for the periods). MVS and SRM will get the shortest jobs out at the highest priority. This method has a large problem however, and that's the possibility that a programmer will overload the initiators with a lot of long-running jobs. If the short jobs can't even get on an initiator, WLM can't get them completed in time. This technique is very useful when you have plenty of resources, and can greatly over-initiate the system. If the system is very constrained, and you have to limit the number of initiators, it may be difficult or impossible to get the small jobs in and out of the system. The biggest problem with this technique is that you can't guarantee turnaround times for your users. It becomes extremely difficult to manage to a set of service level objectives.
The easiest compromise seems to be to define a very, very small number of batch job classes, such as three. Make them significantly different enough in terms of resource usage that it will be easy for the users to choose the correct class. For service levels, you might simply talk about short, medium, and long batch (e.g. class A, B, and C), with response goals for the first two. When setting up your initiators, make sure that classes A and B have at least one dedicated initiator each (otherwise a bunch of class C jobs could grab the initiators during slack times and class A jobs couldn't get started). Then create a multi-period service class for class C. This would give you the capability of managing your test jobs to provide consistent turnaround times for classes A and B. Class C users that use the least amount of resources would tend to get better response than those using more resources.

WLM managed initiators can have similar problems with multi-period batch service classes, because the work is run in the same type of service classes. An additional problem with these initiators is that if all of the current initiators are blocked with long

running jobs and small jobs are missing their goals, then WLM will open up more initiators. It's quite possible that the system can become over-initiated. WLM will eventually stop the unnecessary initiators, but the problem may exist for a period of time.

Production Batch

Production batch jobs present a different problem. The most typical scenario is that all production batch jobs are placed in a single job class with TYPRUN=HOLD in the JCL. Then they are released one job at a time by the operations staff or an automated scheduler. There are no turnaround times associated with production batch jobs. Although the intention of most production batch jobs is to complete before the online systems come up in the morning, there is no way to indicate this to MVS and SRM. Most installations have solved the problem by identifying critical jobs in the batch cycle and assigning them to a higher priority service class (one with a higher dispatch priority).

If you put all production batch in a multi-period service class, you will generally have problems when resources become constrained. One of the major jobs in the critical path, for example, might fall into second or third period. If resources are constrained, other smaller jobs will come in and out of the system at a higher priority than the critical batch job. In a very constrained system, the critical job could take three to four times the normal elapsed time just because it's running at a low priority. Often the solution to this is to move that particular job to a higher priority, single-period, service class. But this solution is applied one job at a time as problems are diagnosed.

In general, multi-period production batch is very frustrating to the operators and schedulers. If they've released a job, it's because they want it to run (now!). They don't want it to lie around the bottom of the resource pool using CPU only when nobody else wants it. For production jobs, usually first-in, first-out is the desired mode of operation. With single-period production batch service classes, that's what you get. With multi-period service classes, the job using the most resources will take the longest, sometimes to the detriment of the critical batch window. ***This has been one of the primary causes for sites not meeting their critical batch window.*** If you're having trouble getting your batch jobs complete before your online systems come up each day, check to see if this is the cause.

One alternative for this is to identify the critical jobs in your batch cycle and place them in a unique job class assigned to a unique single-period service class. This service class would run at a higher velocity and importance than other batch, and would exhibit first-in, first-out characteristics. ■

z/OS Dispatcher

An older version of this article was first published in our Tuning Letter 1999 No. 2, pages 40-43. It's been significantly updated due to the changes in the dispatcher made for specialty processors (zIIPs and zAAPs) and HiperDispatch.

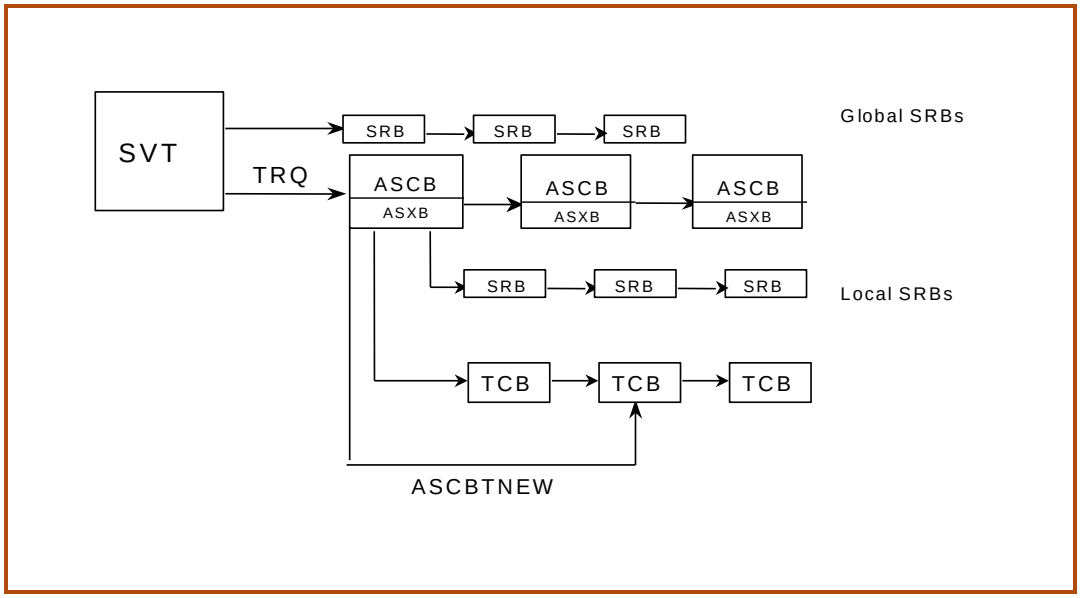
The first section below, *SP 2.1.7 to SP 4.3*, isn't necessary to understanding the z/OS dispatcher today, but we've included it because it might help a few of our older readers. (I mean more experienced, of course!)

SP 2.1.7 to SP 4.3

From MVS/SP 2.1.7 through MVS/SP 4.3, the Supervisor Vector Table (SVT) was the primary anchor for the dispatcher. See Figure 1. When the dispatcher was ready to dispatch some work, it started at the SVT to see if any global SRBs (Service Request Blocks) were ready to be dispatched. If no global SRBs were ready, the address spaces were then looked at. The SVT also had a pointer to the True Ready Queue (TRQ), which was simply a pointer to the first ready ASCB (Address Space Control Block) in a chain of ASCBs that had at least one ready work element and were chained together in descending dispatch priority (DP) sequence. If an ASCB became ready, it was inserted into the queue in the correct DP sequence (after the last ASCB at that priority).

Within an ASCB, the dispatcher searched for any local SRBs for that address space. If none were ready to execute, the dispatcher picked up the chain of TCBs (Task Control Blocks) from the Address Space Extension Block (ASXB) to find the next ready TCB. A pointer, ASCBTNEW, pointed to the next ready TCB. On a multi-processor (MP), the TCB pointed to by ASCBTNEW may have already been dispatched on another CP, so the dispatcher searched the TCBs in priority sequence. The TCBs were in oldest to newest attach sequence unless the address space had "chapped" the TCB priority. "Chapping" is a service to change a TCB's priority within an address space.

Figure 1 – Dispatcher Prior to MVS/SP V5



The problems caused by this dispatcher sequence were:

1. The dispatcher spent a lot of time searching the four queues: global SRB, ASCBs, local SRBs, and TCBs.

2. On an MP, the dispatcher was unnecessarily searching the entire TCB queue instead of being able to use the ready TCB.

3. The oldest TCBs in a multi-tasking address space were always favored in this scenario and could keep newer TCBs from obtaining any CPU.

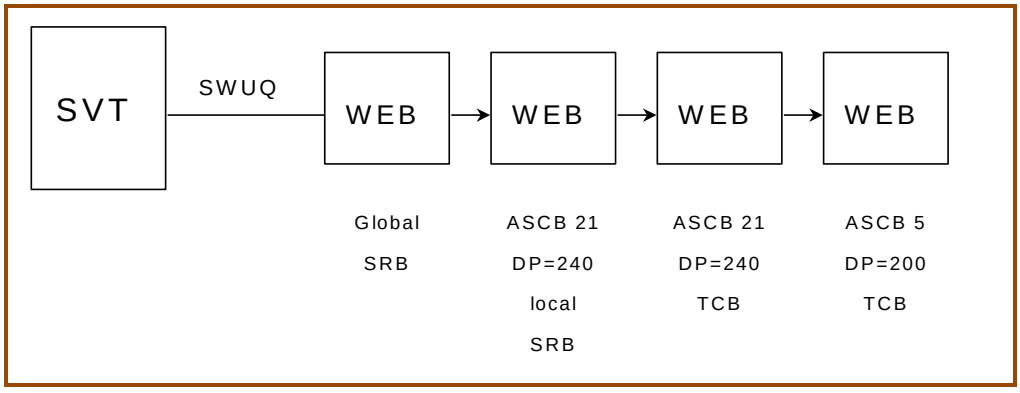
4. Because SRBs are non-preemptable and are always dispatched first, it is imperative that they run only short-running code.

MVS/SP 5.1

In order to resolve some of these performance problems and to provide a solution for DB2 and other new workloads, the dispatcher was rewritten in MVS/SP 5.1. It wasn't until SP 5.2 that the new algorithms were fully exploited by DB2 and DDF using enclaves. OS/390 R3 extended enclave support to include TCBs.

SP 5.1 introduced a Work Element Block (WEB) as shown in Figure 2. The WEB can point to a global SRB, a local SRB, or a TCB. The WEBs are in sequence by descending dispatch priority order and contain pointers only to work that is ready to execute. The SVT (system vector table) contains a pointer, called the System Work Unit Queue (SWUQ) that points to the first and highest priority WEB. When a work unit is dispatched, this highest priority WEB is simply removed from the chain. This allows the dispatcher to eliminate scanning chains when dispatching work. After an interruption, the WEB is placed back on the chain at the end of the queue of equal priorities. This prevents the oldest TCBs from monopolizing the newest TCBs within an address space.

Figure 2 – Dispatcher in MVS/SP V5 and OS/390



As an example, the sequence might be:

WEB - ASCB90, global SRB, DP=210
WEB - ASCB25, global SRB, DP=200
WEB - ASCB101, local SRB, DP=230

WEB - ASCB101, local SRB, DP=230
WEB - ASCB101, TCB (priority 10), DP=230
WEB - ASCB101, TCB (priority 5), DP=230
WEB - ASCB25, TCB (priority 12), DP=200

The global (non-preemptable) SRBs were placed on the queue in descending dispatch priority based on their address space priority, but still run before high priority address spaces.

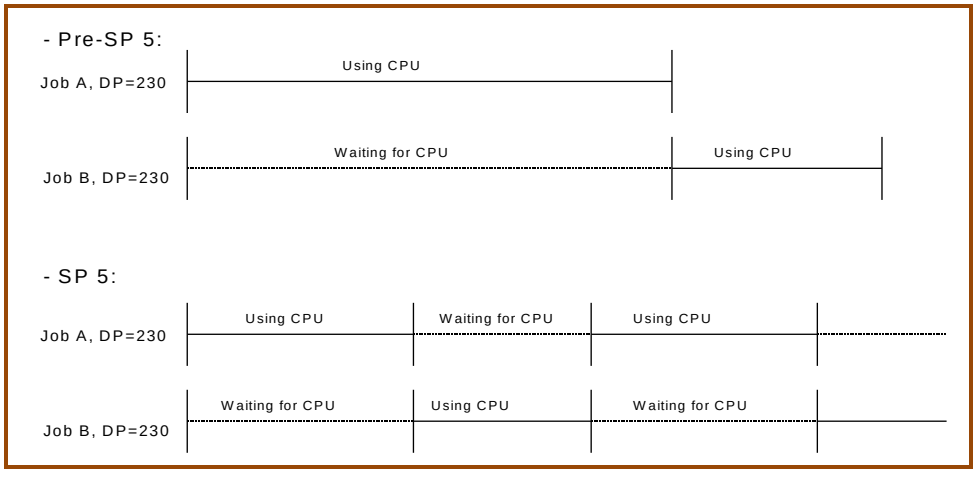
In addition to these changes, SP 5.2 introduced the concept of preemptable SRBs. These SRBs may be preempted by other SRBs or even TCBs with higher (or equal) priority. An SRB can also be scheduled at the same priority as TCBs if invoked with PRIORITY(CURRENT). In that case, the local SRBs would be dispatched at the same priority as the TCB. So you might see:

WEB - ASCB101, preemptable SRB (priority 10)
WEB - ASCB101, TCB (priority 10)
WEB - ASCB101, TCB (priority 5)
WEB - ASCB101, preemptable SRB (priority 5)

Fair Access Priority

Prior to SP 5, when a TCB was dispatched on a machine with multiple processors, the TCB could theoretically use the CPU for many seconds (minutes) while another TCB ***at the same priority*** waited, as shown in the top half of Figure 3. Even if the TCB was interrupted, it still remained in the front of the queue of TCBs for other address spaces with the same DP. That is, it was not-preempted by other TCBs at the same DP unless rotate or MTTW DPs were used. (Rotate DPs went away in MVS/SP 2.1.7.) This often led to inconsistent response times for work with the same DP (e.g. two CICS regions with same DP).

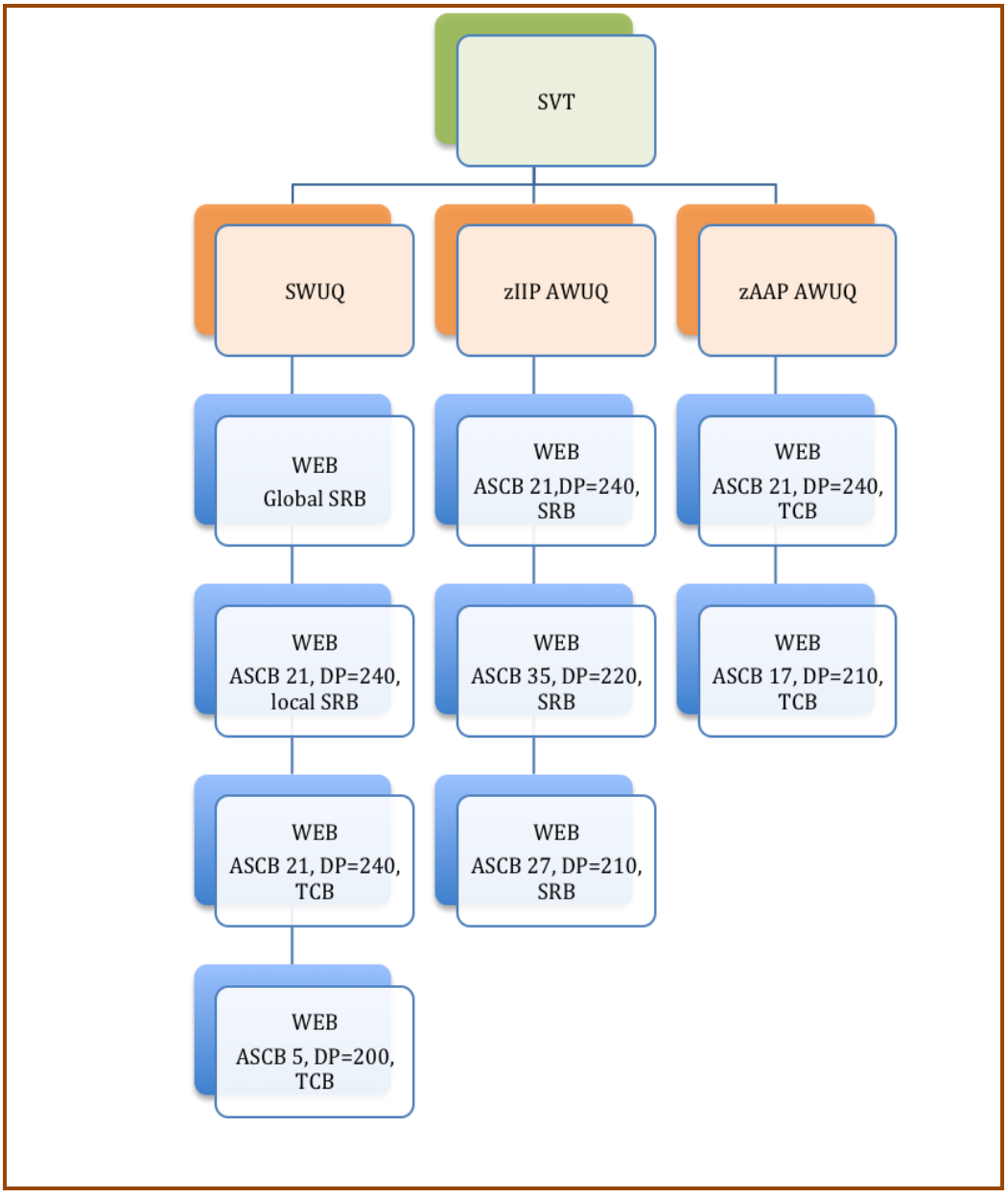
Figure 3 – Fair Access Priority



With SP 5.1, the dispatcher ensures that a TCB cannot dominate one CPU. When a dispatcher timeslice completes, a check will be done for other work being ready to run at the same DP. If so, the dispatcher places the TCB at the end of the WEB queue of elements that have the same DP and selects a new unit of work. See the bottom half of Figure 3.

This same logic holds true now for preemptable SRBs and enclaves. Because of this new logic, you no longer need to keep as many separate dispatch priorities as you used to. As an example, you might currently have one DP for the CICS monitor and a lower one for the CICS TORs that it monitors. The theory was that if a TOR were dispatched for a long time, the monitor could not get dispatched to do its monitoring. With fair access priority, you could put the address spaces in the same DP and both would have fair access to the CPU.

Figure 4 – Work Unit Queues with Specialty Processors (Simplified)

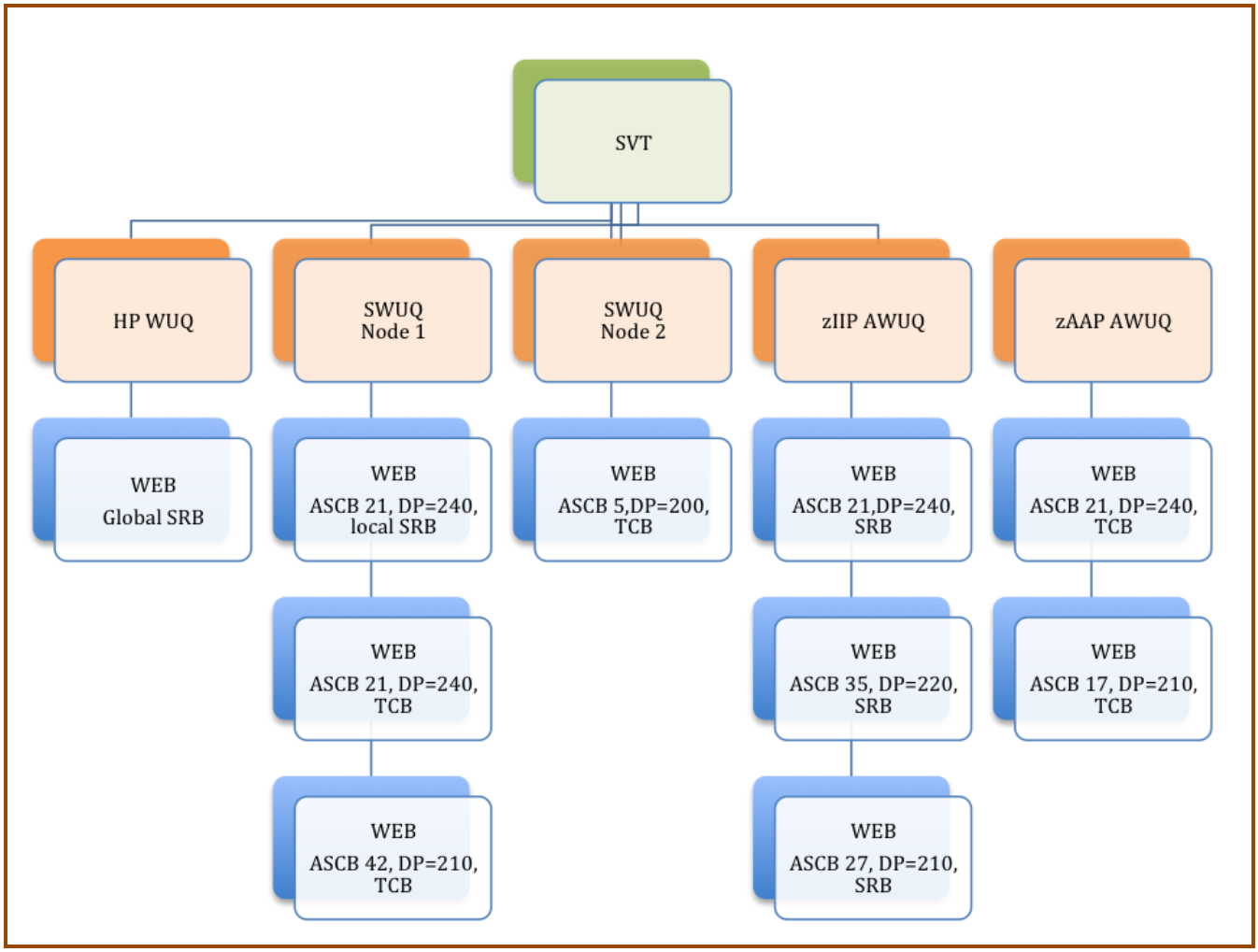


zIIPs and zAAPs

The addition of zIIP and zAAP processors required a change to the single System Work Unit Queue (SWUQ). Additional Alternate Work Unit Queues (AWUQ) were added for the specialty processors, so Figure 2 is changed to look like Figure 4.

zIIP processors may execute work on only the zIIP AWUQ, while zAAP processors may execute work only on the zAAP AWUQ. If a zIIP or zAAP processor notifies the dispatcher that it needs help and HONORPRIORITY=YES, a general processor might dispatch a WEB from one of the AWUQs, depending on dispatch priorities.

Figure 5– Work Unit Queues with HiperDispatch (Simplified)



HiperDispatch

When HiperDispatch was introduced, an enhancement to the dispatcher was created to take into account book affinity. *[For a description of book affinity, please see our* Tuning Letter 2011 No. *4, page 17.] As* we've mentioned in the past, if work can be redispatched on the same book or the same chip, there is increased efficiency due to the opportunity for cache hits at one or more levels of cache related to the processor, the chip, or the book. To accomplish this, PR/SM and z/OS attempt to create separate nodes for groups of processors that share the same chip or book. The method is not fully documented, but Figure 5 will give you a simplified version of running with HiperDispatch turned on. So when a general CP wakes up, the dispatcher will first look at the High Performance Work Unit Queue (that has WEBs from high priority system tasks), and if that queue is empty, will then select work from the Node associated with that CP. Any processor has the ability to search two WUQs concurrently, so that it can select the highest priority. For example, a CP that is helping a zIIP could look at both

the CP WUQ node and the zIIP AWUQ in order to pick the highest priority WEB. You can find more information about this ability to search two queues in a US patent written by **Bernie Pierce** and **Bob Rogers** at www.google.com/patents/US7657889.

Summary

This article has tried to describe the basic changes to the z/OS dispatcher as the technology has advanced. It's been greatly simplified from the actual internals, but we hope that it's provided a greater understanding of the dispatching algorithms. I'd like to thank IBM's **Bernie Pierce** and **Robert Vaupel** for their help in understanding this.

Bibliography

'Dispatching Management in MVS: TCBs to Enclaves', **Bernard R. Pierce**, CMG Transactions, August 1996

'Scalability and High Availability', **Robert Vaupel**, IBM http://os.ibds.kit.edu/downloads_own/teach_wt1011_systemz_lecture/KIT%20WS10-11%20Part%203%20Dispatching-Scalability%20and%20Availability.pdf

'The Evolution of the SRM to The Workload Manager in MVS V5', **Bernard R. Pierce**, CMG Transactions, Winter 1995

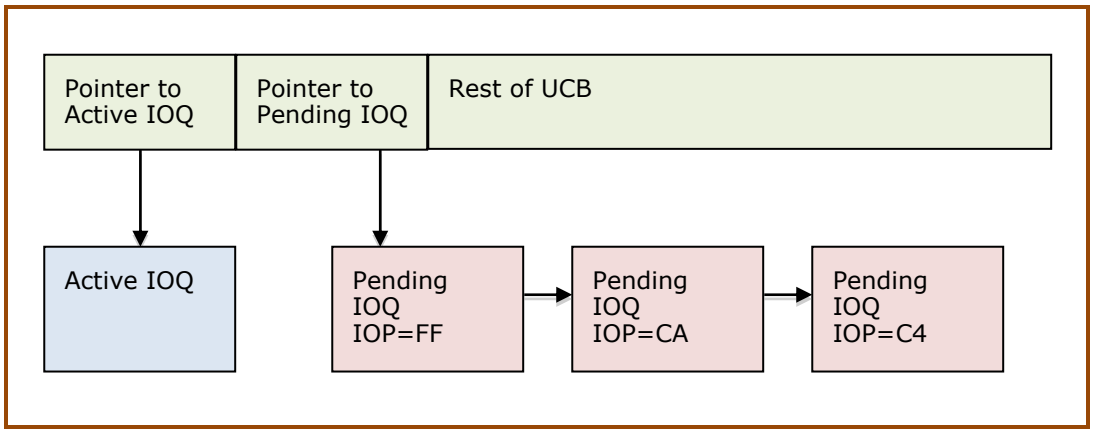
'The value of HiperDispatch on the z10 and zEnterprise 196 processors', **Bernie Pierce**, IBM System z Technical University October 4-8, 2010 (no link available)

GG24-4137-00 - MVS/ESA 5.1.0 Technical Presentation Guide ■

What is I/O Priority?

Figure 6 shows a diagram of control blocks associated with an I/O. For every device, there is one Unit Control Block (UCB). This contains information related to the location, size, and type of device. In addition, it contains the pointers to any I/O requests made to the device. MVS allows a single active I/O to be sent to a device from a single MVS image. Note that multiple requests to the device could be made from other MVS images, but the controller is left to sort that out. Let's just deal with what one MVS system can see.

Figure 6 – I/O Priority



When an I/O request is made to a device, a control block called an IOQ (I/O queue) is built containing a pointer to the actual channel program. The channel program contains the instructions to the device, such as seek, set sector, and read. Figure 6 shows that there are two pointers located in the UCB for IOQs. The first pointer can point to a single IOQ that is currently active on the device (i.e. has been sent to the control unit). The second pointer, if not empty, points to any pending IOQs. When IOS (the I/O Supervisor) tries to schedule an I/O to a device, and there is already an active IOQ, IOS will place the IOQ on the pending IOQ chain. The length of time spent on the pending chain is referred to as IOSQ time and is reported on RMF or CMF device reports. This is used to calculate device response time (sum of pend time, disconnect time, connect time, and IOSQ time). When the active I/O completes, the I/O interrupt processing will schedule the notification to the program waiting for the I/O, move the next IOQ from the pending IOQ to the active IOQ, and the I/O request will be immediately sent to the device. Before returning to normal processing, the interrupt handler will issue a test pending interrupt (TPI) to the channel to see if other interrupts are ready to be handled.

The original default for MVS was to place the IOQ in first come, first served order. This led to some problems if a batch program running with a high I/O load was accessing an online (e.g. CICS) device. The I/Os for CICS would be delayed waiting for all of the queued batch I/Os. So IBM implemented I/O priority queuing.

Way back when (before goal mode). . .

In SRM, the parameter in the IEAIPSxx member called IOP could be set to FIFO (first in, first out) or PRTY (dispatch priority). If IOQ=PRTY was set (and this became the recommendation), then the IOQs were placed on the pending queue in descending dispatch priority sequence. It was possible to modify this even more by placing an IOP=dp parameter on a performance group period if you wanted the I/O to have a different dispatch priority than the address space. This might happen, for instance, if you wanted TSO I/O to be equal to CICS's I/O requests, even though TSO had a lower dispatch priority than CICS.

When goal mode originally appeared, the dispatch priority of the address space was used for all I/O requests. That is, it was as if the user had specified IOQ=PRTY. In OS/390 R3, however, a new facility, called I/O Priority Management, became available. If you turned this facility on (as specified in the same WLM screen where service definition coefficients are set), then the calculation of an address space's velocity changed from:

(CPU Using) / (CPU Using + CPU Delay + Storage Delay)

to:

(CPU Using + I/O Using) /
(CPU Using + I/O Using + CPU Delay + Storage Delay + I/O Delay)

where:

I/O Using = connect time + disconnect time
I/O Delay = pend time + IOSQ time

IOSQ time is the average length of time that a pending IOQ waits for a UCB. As I mentioned in our TUNING Letter, 2001, No. 1, on page 32, I felt that this calculation was flawed, and soon IBM removed the disconnect time from the velocity calculation. Performance analysts have always viewed disconnect time as a delay, not a using time.

But back to the I/O priority. If you turn on I/O priority management, WLM will know who is being delayed by IOSQ time. If work is missing its goal, and is being delayed by IOSQ, then WLM can set a dispatch priority for that work's I/O requests that differs from the address space's own dispatch priority.

If you want to use dynamic PAVs (Parallel Access Volumes) or IRD, I/O priority management MUST be turned on.

HyperPAV Enhancement

There are two changes that occur when HyperPAV is used. To avoid starvation, the priority is incremented for all queued requests every MIH (Missing Interrupt Handler) interval. And when an I/O request finishes for a device, the next highest priority request for the control unit (set of devices) is driven.

Additionally, HyperPAV provides more controls for I/O priority such as giving more link bandwidth to the higher priority requests and deciding which disconnect I/O reconnects first. ■

zIIP Experiences

We've been hearing great success stories from the first users of the zIIP processors. We first described the zIIP processors in early 2006, and have great expectations that these can significantly reduce software and hardware costs for large DB2 installations. But to explain the successes and how to determine whether zIIPs will benefit you, we'd like to first explain a few things.

Additional information can be found in our previous TUNING Letter issues: 2006, No. 3, pages 23-25; 2006, No. 5, pages 26-27; and 2006, No. 1, pages 38-40. A good resource for all things zIIP can be found using the link www.ibm.com/systems/z/ziip. An especially useful set of FAQs is found on their Resources page at www.ibm.com/systems/z/ziip/resources.html.

Preparation

Before you consider a zIIP processor, there are a few things to consider:

1. Do you have the correct levels of hardware and software? zIIPs require a System z9 processor, DB2 Version 8 or 9, z/OS 1.6+ (with PTFs), and the zIIP FMID.

2. Do you have enough DB2 work to move to a zIIP machine? Once you have DB2 V8 and the supporting zIIP maintenance applied to z/OS, there is an easy way to determine this. In the IEAOPTxx parmlib member, specify PROJECTCPU=YES (the default is NO). If you have the correct level of RMF (or the proper maintenance) applied, you can find the projected zIIP work even if you don't have a zIIP processor. With this parameter turned on, you will start to see new fields in your RMF Workload Activity Report (WLMGL). Look at Figure 7 to see the information that can be provided for every service class period. The service times are the number of seconds of CPU time consumed. The AAP in the Service Times column indicates the zAAP service time, and the IIP in the same column indicates the zIIP service time. The next column (APPL %) shows the percent of a CPU consumed by each type of workload. The CP APPL % excludes zAAP and zIIP work executed on standard CPs. The AAPCP APPL % is the percent of a standard CP running zAAP work, and the IIPCP % is the percent of a standard CP running zIIP work. AAP APPL % is the percent of a zAAP CP; and IIP APPL % is the percent of a zIIP CP being used. In this example, there is no work being run on either a zAAP or zIIP CP, but there is eligible work executing on the general CPs. Summing the IIP service times or IIPCP percents for all workloads on the system can show you how much eligible zIIP work is currently running on your machine. This same parameter allows you to provide similar estimates for zAAP processors prior to installation.

Figure 7 - RMF WLMGL Report

```
SERVICE TIMES  ---APPL %---
CPU     342.2  CP       21.8
SRB      12.3  AAPCP     0.6
RCT       5.6  IIPCP     2.3
IIT       3.1
HST       6.7  AAP       0.0
AAP       9.1  IIP       0.0
IIP      34.0
```

3. It's important to understand how you expect to save money with zIIPs. You might be trying to delay an upgrade of your current processors (i.e. you simply want to move off some DB2 work). You might be planning an upgrade, but want to see how much DB2 work can move to a zIIP in order to justify a smaller machine than required to run all of the work, including DB2. You might be considering implementing Variable Workload License Charges, where you want to move some work out of an LPAR in order to reduce the software costs. In any of these situations, it's important to know which time frames will benefit from reducing the CPU time. One installation, for example, investigated a zIIP, but found that they couldn't reduce their peak CPU consumption because there were only stored procedures (not eligible for zIIPs) running during the highest CPU utilization of the LPAR. In other words, their IIPCP percent was quite low when the machine was at its busiest.

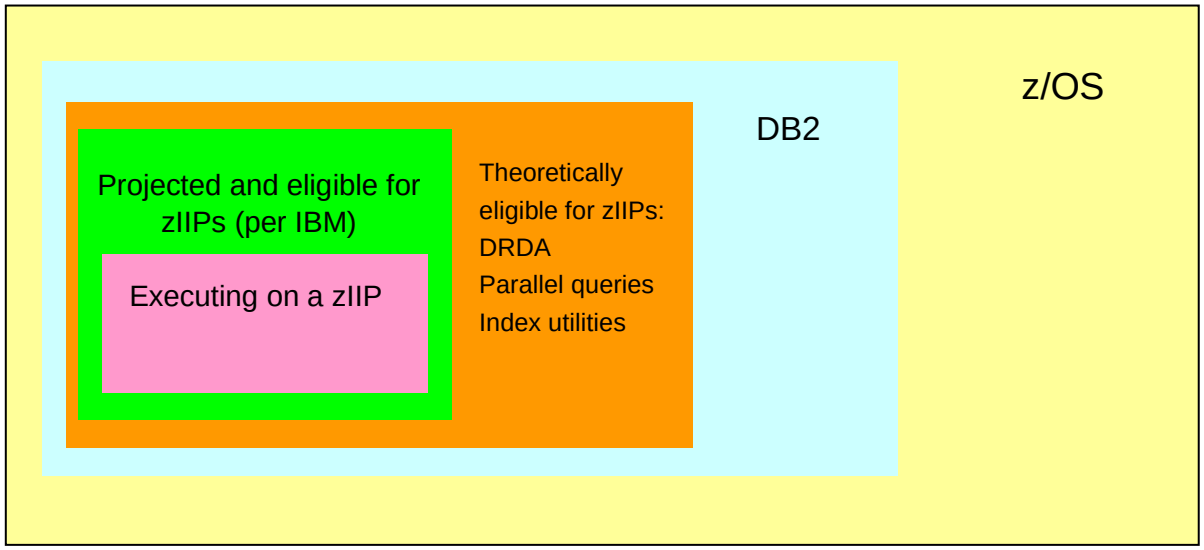
4. Do keep in mind that on sub-capacity machines, the zIIPs will always run at their native speed. So that if you find that you have 50% of a standard CP used by zIIP work, then it may only take 25% of a zIIP CP running at the faster native speed.

zIIP Measurements

Assume that Figure 8 shows the amount of CPU resources consumed by the LPARs on your system. Let's look at each component of resource usage:

- The outer (yellow) box, labeled 'z/OS', shows all of the CPU consumed by all applications on the machine. The amount of CPU here is easily obtained from your RMF or CMF reports - either the CPU Activity Report or the Workload Activity Report.

Figure 8 – CPU Resources



- The next box (turquoise) is labeled 'DB2' and is the subset of the total CPU consumed. This can also be identified from the SMF type 30 records or the Workload Activity Report.

- The next box (orange), labeled 'Theoretically eligible for zIIPs', needs further description and we'll cover that a bit later.

- The fourth box (in green) is labeled 'Projected and eligible for zIIPS'. This can easily be obtained by setting PROJECTCPU=YES as described earlier and collecting the data from the IIPCP fields. This is the amount of zIIP work that DB2 is willing to offload to a zIIP.

- The smallest box (in pink) is labeled 'Executing on a zIIP' and can be measured from the IIP APPL % field as shown in Figure 7.

What's Eligible?

So each of these subsets of CPU time can be measured and reported with the exception of the 'Theoretically eligible for zIIPs'. What is this all about? We can look at the marketing considerations for an answer. IBM needs to compete with other platforms for database applications, and zIIPs are a way of reducing the cost of running DB2 on a mainframe. But if IBM moved all of DB2 off to a smaller cost machine (i.e. zIIP), then they might lose too much revenue. So IBM has set up an internal check to ensure that they don't move too much off the mainframe by implementing two controls:

1. First, they have determined a subset of types of work that can move. Initially, that consisted of certain DB2 functions, such as DRDA, parallel queries, and index utilities that could move to a zIIP. This is really the box called 'Theoretically eligible for zIIPs'. But IBM doesn't tell you how much this represents. By the way, this list will likely change in the future. At SHARE, there were comments relating to using zIIPs for TCP/IP Sec (security) and other database applications. Also, the initial zIIP implementation supported only Star schema parallel queries, but maintenance has now changed that to enable all parallel queries.

2. The software internally checks to ensure that no more than a certain amount of work will move. That might be only 50% of the eligible work. This is the amount of work that IBM provides in their estimation tool. As we mentioned in an earlier TUNING Letter, prior to the PROJECTCPU parameter, IBM said that you could look at all of your DDF work and estimate that 40% of that work could be moved to a zIIP processor. Later releases of DB2 have increased the type of work that can be moved and the percent of work that is eligible.

The Success Stories

Now that zIIPs are in the field, we are hearing some great success stories. In almost all cases, users reported that the actual amount of work moved to the zIIPs was higher than that estimated from the IBM tool. That is, they weren't disappointed with their results and had already determined that if they could move that much work then the zIIP could be justified. So it's very important to perform the estimation process before deciding to order or install a zIIP. And watch out for the time frames! It seems that it's the only place you can go wrong if you at least turn on PROJECTCPU. ■