Goal-Based Initiator Management

John Arwe
IBM Corporation
522 South Road
Poughkeepsie, NY 12601-5400

Abstract

It was only a matter of time: MVS Workload Manager (WLM) and the Job Entry Subsystem (JES2) in OS/390 Release 4 intend to remove another bastion of unfettered tinkering, your initiation structure. How many initiators do you need? In which job classes? When? Is it better to over-initiate or under-initiate? On which systems?

It's not as bad as it sounds of course. Separate the rumors from the facts and the fiction as we explore the changes being made to WLM and JES2. Find out how WLM can now use goals to manage the number and placement of initiators across a sysplex, what long-overdue measurement data will now be reported, why your critical path batch work is safe, the migration issues, and the operational implications of using this new capability. Abstract resource affinity support will be mentioned in passing but will not be considered in detail.

Table of Contents

Introduction

MVS/ESA SP 5 introduced the notion of goal-oriented work management within the sysplex. Initially CPU and storage were the only resources allocated directly based on the goals of the work; OS/390 Release 3 added management of I/O delays. Queueing delay was also managed in OS/390 Release 3, but only for enclave-based work. Of the traditional workloads, only batch work commonly incurs significant queueing delay within the sysplex. Transactional workloads are subject to network delays, but these are not within the control scope of MVS. Several implementations for management of queueing delay have evolved within MVS: job scheduling packages, the JESes, and APPC scheduler (ASCH) are some examples.

JES and ASCH manage pools of server address spaces that provide services to other work. The size of each pool is determined by initialization parameters, which along with the arrival and departure rates of requests from the served queue determine the actual queue time. Pool size is the primary control over queue delay, and there is no required connection between the run-time prioritization of the work and its relative access to the next idle initiator. This has led to the practice of over-initiation, initiating as much work as possible in order to get work prioritized by MVS and in particular by the System Resource Manager (SRM) component of MVS.

Job scheduling packages typically intend to solve a somewhat different problem, and use more detailed information about jobs and inter-job dependencies to initiate at "the right time". Since queue delay is replaced by job scheduling delay, jobs managed by job scheduling packages are not expected to wait for initiation once they are released. The historical information used by job scheduling packages implicitly accounts for the run-time prioritization of the work.

Background

OS/390 Release 4 adds initiation delay to the list of resources managed by WLM and SRM. SRM management of JES2 queue delay based on the goals assigned to work by WLM relaxes the need to over-initiate by establishing a feedback loop between SRM and JES2 that can be used to manage the number of initiators. The sysplex scope of WLM management adds the capability to dynamically balance the number of JES2 initiators across the sysplex. This provides the crucial link between initiation policy and performance management that was missing before. Additional benefits include reduced operator involvement in initiator management and better reporting of JES2 queue delay components. WLM's requirement for a basic sysplex consisting of one or more systems is unchanged.

In order to enable WLM to manage JES2 queue delays, several fundamental changes are required.

1. WLM/SRM must be aware of queue delay as it occurs, rather than finding out about it when a job is finally selected for execution. This implies that batch jobs resident on the input queue must generate state samples.

2. Jobs on the input queue but ineligible for selection must be filtered out so that they do not generate spurious queue delay samples.

3. WLM must know the service class accruing the queue delay in order to attribute the queue delay samples correctly. Thus jobs on the queue must be classified prior to execution.

4. WLM must have control over the resource which controls queue delay, initiators.

5. WLM must recognize that the queue is multisystem in scope and that jobs may be eligible to execute only on a subset of the systems due to affinities.

Along with the existing mechanisms for allocating resources to work based on goals, the feedback loop is complete from queue delay samples and goals to the number of initiators.

The (New) Secret Life of a Job

As in prior releases, a job proceeds from a reader through conversion into initiation and finally execution. The same MVS initiator code runs in the new world, SYS1.PROCLIB(INIT) running IEFIIC , as ran in prior releases. Only occasional dual-pathing in the code distinguishes initiators managed by the installation from those managed by WLM.

At the end of the conversion process, after exits that might modify the JCL have run, JES2 now calls WLM to classify the job. Classification of the job occurs at this point for all jobs running under JES2 R4 once JES2 R4 functions have been activated, regardless of whether the system is running in compatibility mode or in goal mode. This contrasts with earlier releases, where jobs are classified after job selection. The service class assigned by WLM is recorded in the Job Queue Element (JQE) which is then added to the job queue. Once this process has completed the job can be sampled by the WLM state sampler.
The service class of a job can be changed prior to execution by WLM policy activation, which reclassifies all jobs, or by the JES2 reset command ($TJ). Once the job is executing, its service class may be changed via the MVS RESET command, and the change is also noted by JES2 in case the job is restarted. Changes to the service class prior to execution are communicated to SRM when the job is selected for execution.

In order to allow for gradual migration, the decision to manage initiation based on existing controls or based on WLM management is made on a job class basis. Those job classes managed manually are known as JES-managed job classes, and correspond to the behavior in prior releases. Those job classes whose initiation is managed by WLM, designated by specifying MODE(WLM) in the JES2 initialization deck JOBCLASS statement or via the modify job class command $TJOBCLASS(x),MODE=WLM, are known as WLM-managed job classes. Correspondingly, initiators processing jobs in a JES-managed job class are called JES-managed and initiators processing jobs in a WLM-managed job class are known as WLM-managed initiators.

Note that job priorities work differently for WLM-managed job classes. Management of queue delay by WLM requires strict first in first out (FIFO) queueing order, which job priorities would upset. Thus job priorities do not affect the job queueing order within WLM-managed job classes; they do continue to work as before for JES-managed job classes. Since many production job control language (JCL) systems use job priority to flag production batch work, job priority has

been added as a classification attribute to influence the choice of service class. Priority aging is not performed for WLM-managed job classes since doing so would require reclassification of all affected jobs, often to no benefit.

JES-managed initiators have only one operational change in behavior from prior releases: they cannot select jobs from WLM-managed job classes. They are still started and stopped by operator command, run and select work in both goal mode and compatibility mode, and each initiator consumes a job number since it is started under the JES2 subsystem. The job number is for the initiator itself, separate and distinct from the job number used by any job that the initiator may be running.

WLM-managed initiators select jobs only from WLM-managed job classes, select work only in goal mode, are started and stopped by WLM, and do not consume job numbers since they are started under the MSTR subsystem. If a system is switched into compatibility mode while WLM-managed initiators are running jobs, the initiators will drain and terminate. As a migration aid, the job class selection lists for initiators need not be changed when referenced job classes are changed between JES-managed and WLM-managed. If an initiator tries to select from a job class running under a different management scheme no job will be returned and the initiator will continue on down its job class selection list.

The installation should be careful not to mix JES-managed and WLM-managed job classes in the same service class. WLM allows doing so rather than failing jobs, but its management of initiators in such a case is unpredictable. When both initiation schemes are used for the same service class, the correlation between queue delay and the number of initiators WLM recognizes as servicing the service class is weakened. Bad data can theoretically lead to bad decisions, thus the recommendation against doing so.

Scheduling packages that depend upon the use of JES2 exit 14 will need updates when using WLM-managed job classes. JES2 exit 14 is not called by WLM-managed initiators. New JES2 exit 49 is called instead by both JES-managed initiators and WLM-managed initiators. Over the long term this change should be beneficial, since the interface to exit 49 is much simpler and should lead to far fewer dependencies upon JES2 internals than exit 14 typically requires.

New/Changed Measurement Data

Much of the value to be derived from this new support even when job classes are not turned over to WLM management comes from improved reporting of pre-execution job delays. Because WLM must differentiate between jobs that are truly eligible for selection from those that are ineligible but on the input queue, these times can now be more accurately reported. Reporting of the new times requires some JES2 migration actions, see "JES2 Migration", in order to be measured. Unless noted otherwise, the times are measured in both goal mode and compatibility mode to provide data for migration purposes. The following section describes the new data and how it is reported. Refer back to concepts introduced in "The (New) Secret Life of a Job" as needed.

Pre-Execution Job Delays

Delays caused by end users, such as TYPRUN= delays, are in general completely uninteresting from the system management point of view. Because the definition of response time used by SRM for batch work begins at the end of conversion, TYPRUN=JCLHOLD was already excluded from batch response times. Delays due to TYPRUN=HOLD however were included in batch response times; this has now been fixed. Once the JES2 R4 functions have been activated, TYPRUN=HOLD time will no longer be included in batch response times; indeed it will not be reported at all. For cases where ad hoc batch work has been given execution velocity goals because of the unpredictable response times caused by inclusion of TYPRUN=HOLD time in response time, this is a good time to reevaluate the use of response time goals for batch work. Other delays fall into several categories, described below.

Conversion time is the time after reader processing has completed and before the job is ready for initiation. This time is now measured and reported, but is not included in the response time for batch jobs. During conversion no state samples are reported to WLM sampling. The measured time is reported in the SMF type 30 record and SMF type 72 record, that is at both the job level and the service class/performance group level.

Queue delay is the time after conversion spent waiting only for an initiator to select the job. This time is of immediate interest to WLM, since it is the feedback mechanism by which the need for changes in the number of initiators is recognized. This is the time that WLM could eliminate if sufficient capacity is available for enough initiators to service the job class. While a job is accruing queue delay time, it also accrues queue delay samples. The measured time is reported in the SMF type 30 record and SMF type 72 record, that is at both the job level and the service class/performance group level.

Affinity delay is the time after conversion spent waiting for a required system and/or scheduling environment to become available. Scheduling environments act much like system affinities by restricting the systems on which a job can run. They are a new concept introduced with OS/390 Release 4 that allows the execution of jobs to be restricted to systems based on the availability of installation-defined resource names that might represent such things as software licenses, particular hardware, or a subsystem name. While a job is accruing affinity delay time, it also accrues Other samples. Thus the time will be included in the response time of the job, but not in the samples used to calculate execution velocity. The measured time is reported in the SMF type 30 record and SMF type 72 record, that is at both the job level and the service class/performance group level.

Note that use of system affinity requires that the designated systems be available for conversion, so in reality the only system affinity-related time accounted for in affinity delay time is the time when a system was available to convert the job, but became unavailable before the job began execution. Unlike the system affinity case, delays due to the unavailability of scheduling environments are not sensitive to the system which performs conversion.

Operational and scheduling delay is the time after conversion spent waiting due to job hold, job class hold, duplicate jobnames, and any other delays which fall into neither the queue delay nor

the operational and scheduling delay categories. While a job is accruing affinity delay time, it also accrues Other samples. Thus the time will be included in the response time of the job, but not in the samples used to calculate execution velocity. The measured time is reported in the SMF type 30 record and SMF type 72 record, that is at both the job level and the service class/performance group level.

The inclusion of user-induced duplicate jobname delay with operational delays is intentional, since JES2 introduced an option in OS/390 Release 3 to control duplicate jobname serialization. There have been requests already to separately report this time to assist in service level agreement (SLA) management by allowing jobs delayed because of duplicate jobnames to be excluded from the SLA criteria. This time could be reported separately if the various organizations generating requirements raise it as a formal requirement and give it a relatively high prioritization.

SMF 26: Job Purge
A new Workload Manager section is added, containing: the job class, the service class to which the job was first classified, the service class under which it last executed (different from the original if the job was reset for example), whether the job ran in a JES-managed or WLM-managed initiator, and whether the job was initiated normally or as the result of a $SJ command.

SMF 30: Job/Step Begin/End/Interval
The performance segment includes the scheduling environment for the job, conversion time, queue delay time, affinity delay time, operational and scheduling delay time, and indicators to determine if the job was reset, forced into execution via the $SJ command, or restarted. Jobs that are restarted will have one set of records for each time it begins execution, i.e. one set for the original execution and one set for each restart that reaches execution again. The new delay times from each execution of a restarted job are not cumulative. In order to arrive at the total times, the times in the records from each execution of the job must be added together. In practice it will probably to easier to ignore restarted jobs when post-processing records, and the restart indicator is provided with this in mind.

SMF 72: Service Class/Performance Group
The service class period and performance group period segments include conversion time, queue delay time, affinity delay time, and operational and scheduling delay time for the service class/performance group period.

SMF 99: SRM Goal Mode Decisions
Batch initiator queue statistics are included in a new section of the subtype 2 record, new trace codes are defined for batch support, and a subtype 6 record was added to provide summary data for modeling.

SDSF

SDSF support consists of two main items: addition of the management mode in effect for jobs on the DA and STatus panels, and the addition of a new action character, I, to display information about a particular job. Samples of the job status display and job information are shown below.

```
+-------------------------------------------------------------------------+
|Job status display sample                                                |
|                                                                         |
|JOBNAME  SRVCLASS WPOS  SCHEDULING-ENV   DLY MODE                        |
|BAT1280  BATCH1       0                   YES WLM                        |
|BAT1281  BATCH1       0                   YES WLM                        |
|BAT2283  BATCH2       0                   NO  WLM                        |
|BAT1282  BATCH1       0                   YES WLM                        |
+-------------------------------------------------------------------------+
+-------------------------------------------------------------------------+
|                            Job Information                              |
|                                                                         |
|Job name         BAT1155    Job class limit exceeded? NO                 |
|Job ID           JOB01140   Duplicate job name wait?  NO                 |
|Job schedulable? NO         Time in queue             N/A                |
|Job class mode   WLM        Average time in queue     N/A                |
|Job class held?  NO         Position in queue         N/A     of N/A     |
|                            Active jobs in queue      N/A                |
|                                                                         |
|Scheduling environment:                available on these systems        |
+-------------------------------------------------------------------------+
```

RMF

The example below shows the new format of transaction times in the WLMGL and WKLD reports of RMF monitor 1. Queue delay time, affinity delay time, conversion time, and operational and scheduling delay time are listed below execution time.

```
+-------------------------------------------------------------------------+
|      W O R K L O A D   A C T I V I T Y                                   |
|                                                                         |
| ---------------------------------------                                 |
| REPORT BY: POLICY=POLICY1 WORKLOAD=BATCH                                |
|                                                                         |
| TRANSACTIONS   TRANS.-TIME HHH.MM.SS.TTT                                |
| AVG     23.03  ACTUAL              3.883                                |
| MPL     23.03  EXECUTION           3.312                                |
| ENDED       6  QUEUED                571  <                             |
| END/SEC  0.00  R/S AFFINITY            0  <                             |
| #SWAPS     13  CONVERSION            310  <                             |
| EXECUTD     0  INELIGIBLE              0  <                             |
|                STD DEV             6.617                                |
+-------------------------------------------------------------------------+
```

Similar changes were made on the RMF monitor 3 reports which display response times, such as SYSRTD. In monitor 3 reports which did not have room for the new data, the existing Wait column was made cursor sensitive. Positioning the cursor on Wait and pressing the enter key causes RMF monitor 3 to display a pop-up with the same categories of time as are shown above. The following is an example from the SYSSUM report.

```
+-------------------------------------------------------------------------+
| RMF 2.4.0  Sysplex Summary                                              |
|                                                                         |
|Trans --Avg. Resp. Time-                                                 |
|Ended  WAIT EXECUT ACTUAL                                                |
|Rate   Time   Time   Time                                                |
|                                                                         |
|0.140 48.06  52.03  1.67M                                                |
|0.100 0.188  1.945  2.134 <                                              |
|0.000 0.000  0.000  0.000                                                |
|0.040 2.80M  2.95M  5.75M                                                |
|                                                                         |
+-------------------------------------------------------------------------+
```

The corresponding pop-up:

```
+-------------------------------------------------------------------------+
| RMF Response Time Components Data                                       |
|                                                                         |
|The following details are available for NRPRIME/1                       |
|Press Enter to return to the report panel.                               |
|                                                                         |
| Response Time Components:                                               |
|                                                                         |
|  Actual          : 2.134                                                |
|  Execution       : 1.945                                                |
|  Wait            : 0.188                                                |
|  - Queued        : 0.188                                                |
|  - R/S Affinity  : 0.000                                                |
|  - Conversion    : 0.000                                                |
|  - Ineligible    : 0.000                                                |
|                                                                         |
+-------------------------------------------------------------------------+
```

JES2 R4 functions were not yet activated on this system so new data is zero.

Syslog

The most obvious indications that WLM-managed initiators have started are the messages in syslog and on the console. The following syslog excerpt shows what you can expect to see when a WLM-managed initiator starts.

```
+-------------------------------------------------------------------------+
|IWM034I PROCEDURE INIT STARTED FOR SUBSYSTEM JES2 442                    |
|APPLICATION ENVIRONMENT SYSBATCH                                         |
|PARAMETERS SUB=MSTR                                                      |
|                                                                         |
|IEF196I         3 XXIEFPROC   EXEC   PGM=IEFIIC                         |
|                                                                         |
|IEF403I INIT - STARTED - TIME=10.01.51                                   |
|ICH70001I IBMUSER  LAST ACCESS AT 10:01:50 ON FRIDAY, JUNE 6, 1997       |
|$HASP100 BAT2016  ON INTRDR                          FROM JOB00090      |
|JOBHOLD2                                                                 |
|$HASP373 BAT1002  STARTED - WLM INIT  - SRVCLASS BATCH1   - SYS SY#A     |
+-------------------------------------------------------------------------+
```

Ignore the SYSBATCH in message IWM034I; you do not define it and cannot display or modify its definition. WLM defines SYSBATCH internally to provide a smooth migration path. The only other place you are likely to see this name is in an IPCS WLMDATA report. Note also that WLM-managed initiators are started using the same INIT proc as JES-managed initiators.

Operational Changes

While WLM-managed initiators reduce the need to actively manage job initiation, there are still constraints within which WLM must operate. Since operators lose the ability to control the maximum number of WLM-managed initiators or to force jobs into immediate execution by starting more initiators, mechanisms to do so that apply to WLM-managed initiators are provided. Likewise the need for orderly shutdown remains even when WLM-managed initiators are used.

Limiting the Number of Jobs

Most often the need to restrict the number of concurrently executing jobs derives from either a physical limitation such as the number of tape drives or from a service level agreement that specifically limits the number of jobs an organization can run concurrently. Typically this is accomplished by limiting the number of initiators started with the restricted job class in their selection list. This model is now implemented for WLM-managed and JES-managed initiators by specification of JOBCLASS(x) XEQCOUNT=(MAXIMUM=5) in the JES2 initialization deck or via the $TJOBCLASS(x) command. In the previous example no more than 5 jobs in the specified job class are allowed to execute concurrently within the multi-access spool (MAS), regardless of their service classes. See "Job Class Structure" for migration considerations when changing to use this facility.

Forcing Immediate Initiation

Since the $SI command cannot be used to start an initiator for a WLM-managed job class, the $SJ command is provided to enable the operator to force immediate initiation of a specific WLM-managed job regardless of goals or the job's position on the job queue. WLM will start an initiator on the system most likely to have enough CPU capacity to support additional work. This initiator will select the specific job targeted by the $SJ command and initiate that job. When the job completes, the initiator will terminate. Since the initiator will process only the targeted job, it does not give the operator the capability to manually add long term initiator capacity to a

WLM-managed job class. Likewise this command should be used only on an exception basis; additional overhead per job is incurred to start and stop the initiator, and only short term initiator placement data is used. If many jobs are initiated at once using $SJ, it is entirely possible to cause bottlenecks by not giving the capacity consumption feedback loop time to reflect recent placements before more are made to the same system.

Orderly JES2 Shutdown
In order to prepare for hardware or software maintenance, a system may need to be quiesced. Since WLM-managed initiators cannot be stopped via $PI a new command, $P XEQ, is provided for this purpose. $P XEQ announces your intent to perform an orderly shutdown of JES2: JES-managed initiators will cease selecting new work; WLM-managed initiators will drain and then terminate. $S XEQ allows selection of new work, and if running in goal mode with WLM-managed initiators it allows WLM to create initiators again. Both commands are single-system in scope; when WLM-managed initiators are being used WLM will likely start initiators elsewhere to compensate for the loss of capability on the system being quiesced.

Poly-JES
If multiple JES2 subsystems exist on a single image but belong to different MASes, WLM accepts queue delay samples from each of them.

Boss-JES
If multiple JES2 subsystems exist on single image and within the same MAS, WLM accepts queue delay samples from only one of them. If the primary JES2 is active, that member will contribute queue delay samples to WLM; if the primary JES2 is not active, IEFSSNxx is searched and the first JES member in IEFSSNxx and active in the MAS is allowed to contribute samples to WLM. JES2 has coined the term "boss JES2" for the one that contributes WLM state samples for its queued jobs. WLM-managed job classes belonging to other JES2 members will never have initiators started to service their job queues. If jobs must be run under a secondary JES2, the job classes should be JES-managed.

Migration Issues

There are a number of issues involved with exploiting the new WLM and JES2 functions, due primarily to existing shared resources such as the checkpoint dataset and WLM couple dataset. Management of sysplex-wide job queues by WLM and the possible parallel exploitation of scheduling environments adds to the mix.

WLM Migration

Effects on Execution Velocity

Jobs running in WLM-managed initiators contribute queue delay samples that are included in execution velocity calculations. Since these can contribute only delays, achieved velocities will be reduced versus jobs running in JES-managed initiators. JES-managed jobs will report equivalent queue delay samples in the SMF type 72 record that can be used to calculate the achieved velocity (shown below) for the same work if it were run under WLM-managed initiator and if WLM chose the same initiation scheme. Velocity goals for service classes running WLM-managed initiators should be adjusted accordingly.

```
+-------------------------------------------------------------------------+
|                                                                         |
|REPORT BY: POLICY=PRIMSHFT                                               |
|                                                                         |
|TRANSACTIONS    HHH.MM.SS.TTT                                            |
|AVG      0.03           3.475                                            |
|MPL      0.03           3.227                                            |
|                                                                         |
|VELOCITY MIGRATION:   INIT MGMT 92.3%                                    |
+-------------------------------------------------------------------------+
```

Effects on Response Time Goals

`TYPRUN=HOLD` time is no longer part of a batch job's response time once JES2 R4 functions are activated. This may necessitate adjustment of response time goals if they are currently used for batch work, or more likely it may allow the use of response time goals for batch work where it was not practical before. Since the user-caused delay is not included in the response time, response times should be less variable.

As long as enough completions are available, for the long term response time goals or discretionary goals should be used to manage batch service classes. If multi-period service classes are used, at least first period should be amenable to a response time goal once JES2 R4 functions are activated. Using response time goals in this way will minimize future migration actions by removing the need to recalibrate velocity goals after each environmental change.

When to Continue Using JES-managed Job Classes

When the depth of the job class queue is unrelated to the number of initiators servicing the queue, either a discretionary goal or JES-managed initiators should be used. If velocity goals or response time goals are used in cases where a very large number of jobs are released simultaneously, WLM's algorithms will project little incremental value in reallocating resources to help the work and the work may be delayed as a result. In cases where the batch jobs are the only work running, excess CPU and storage resource will still be used to support initiators. See "Critical Path Batch" for other considerations.

Of course the next logical question is: how large is 'a large number'? There is no specific magic number, no threshhold, no rule of thumb. Go back instead to the first sentence; the trouble is not from 'large' numbers of jobs, but from divorcing queue delay from the number of initiators. If the initiation approach for a workload amounts to dumping in a whole bunch of work and letting MVS sort through it all using the available capacity, that is not something whose initiation is managable via a response time goal or velocity goal; use a discretionary goal, possibly with a resource group minimum, or use JES-managed initiators to initiate it.

WLM Couple Dataset and Service Definition
[Planning] contains the complete list of constructs that cause the service definition to be upgraded to functionality level 004, along with migration checklists. From a toleration point of view, you should reformat the WLM couple dataset (CDS) using the OS/390 Release 4 level of the format utility as soon as the first OS/390 Release 4 image is running. Doing so allows all levels of WLM to install service definitions to the WLM CDS; not doing so prevents OS/390 Release 4 from installing a service definition or from executing a policy activation request initiated on the OS/390 Release 4 system.
Exploitation of scheduling environments or classification by job priority will cause the service definition to be upgraded to functionality level 004. Once this occurs, earlier levels of WLM will not be able to manipulate the service definition or execute a policy activation request initiated locally.

JES2 Migration

This is a summary of the relevant issues; [J2Mig] should be read thoroughly as well. *Do not underestimate this migration. JES2 usermods must be changed significantly in many cases.*

New Checkpoint and JQE Format
In order to support scheduling environments and maintain information about jobs such as the service class returned by WLM at the end of job conversion, the JQE has increased in size which in turn requires a reformat of the checkpoint dataset. When migrating to JES2 for OS/390 Release 4 using a warm or hot start, the reformatting must be manually triggered via the `$ACTIVATE` JES2 command; if a cold start is performed, the new format is automatically applied.

Warning!
All JES2 members in the MAS must be at an OS/390 Release 4 level before the checkpoint dataset is reformatted.

Backing out from the new format requires a cold start of a previous level of JES2. Running with the new format checkpoint is, in JES2 terms, called "Release 4 mode"; the old format is called "pre-release 4 mode".

New Services REQUIRED to Access JQEs and CATs
Due to internal changes, new services must be used to manipulate and/or access JQEs and CATs. Serialization changes and structural changes will also affect existing JES2 modifications.

Critical Path Batch
WLM batch management is not intended to replace job scheduling packages. It does not provide deadline management, critical path analysis, job dependency scheduling, et cetera. Jobs that must be guaranteed immediate initiation once released should be run using JES-managed initiators.

JES2 Commands

Many of the JES2 commands have changed to support the dynamic change of parameters related to WLM batch support and to generally enhance usability. There are also some new commands, mentioned previously, that can or must be used only with WLM-managed job classes. [J2Cmd] presents exhaustive details and [J2Mig] includes a table of the changed commands. For a basic starter set:

`$TJOBCLASS(x),MODE=`

Change job classes between JES-managed and WLM-managed modes.

`$DJOBCLASS`

Display the mode (JES-managed or WLM-managed) of a job class.

`$DJ`

Display the service class and scheduling environment for a job.

`$TJn,SRVCLASS=`

Change the service class for a job prior to execution.

`$SJ`

Force a WLM-managed job to be initiated immediately.

`$S XEQ/$P XEQ`

Begin or drain job selection.

Job Class Structure

Because jobs now have both a service class and a job class, and both are now used to manage or control them, the mapping between service classes and job classes needs to be re-examined. There are several cases to be wary of: use of execution limits and consistent initiation controls across a service class are the most likely sources of problems. On the positive side, it is likely that by exploiting scheduling environments you can eliminate some job classes completely. See [Planning] for details on scheduling environments.

While the execution limit (`JOBCLASS XEQCOUNT MAXIMUM`) has its place, the limit applies at a job class level and within the MAS. If more than one MAS is within the sysplex, this complicates matters. A more common problem will be having several different service classes running work in a single WLM-managed job class with an execution limit specified. Here the problem is that the limit is less granular than the WLM management. There will be times when WLM adds an initiator to reduce queue delay, but JES does not allow the initiator to execute another job because of the limit. The feedback to WLM is that the service class cannot take advantage of the initiators it has started, which is noted in the service class history. If competition for the limited slots from other service classes goes away, WLM still remembers that the service class it attempted to help before had some other problem. When multiple service classes compete for the slots in one job class with a maximum the historical data may have trouble differentiating between this problem and a genuine affinity problem, where jobs in the service class have affinity to other systems.

Inconsistent initiation controls refers to having a single service class run work in multiple job classes, where the job classes are a mixture of WLM-managed and JES-managed. Here the problem again is one of control scope, in that WLM has only partial control over the service class's queue delay. The objective is to have all jobs in a single service class run either in WLM-managed initiators or in JES-managed initiators, but never in both. Running with a mixture of initiation controls is not recommended and may lead to inconsistent results.

Initiator Management

In order to achieve some comfort level with handing over control of initiation to WLM, this section provides an overview of the conditions under which WLM might make changes to this new control. Readers unfamiliar with the concepts of donors and receivers in the WLM policy adjustment context should refer to [EffUse] and APAR OW25542 before continuing. What follows is not intended to detail every possible "what if" scenario or every twist and turn of the WLM implementation; there is far too little space for such a detailed treatment.

Making the Adjustments

Policy Adjustment

Every 10 elapsed seconds, WLM code examines the current state of the system, determines if any beneficial adjustments are possible, and if so it implements them. At a very simplified level: find out which work needs help (the receiver), figure out the receiver's resource problem, and see if work less deserving (donor) can give up some of the same resource. In the case of initiator management, the determination of the receiver service class period is unchanged from prior releases: it is based on the goals specified and how work is performing with respect to those goals. In order for initiators to be considered as a resource needed by the receiver, queue delay samples must be present and must be a significant cause of delay which WLM can address. Enough CPU and storage must be available, possibly by taking from donor service class period(s), to support the initiators that WLM wishes to start. WLM starts by assessing the effect of adding one initiator, and continues until it finds the minimum number of initiators projected to cause a meaningful change in the receiver's ability to achieve its goal. If enough CPU and/or storage can be found, the action is taken; if not, other resources are examined as in prior releases. There is no upper limit on the number of initiators WLM may request at once, but there are several levels of pacing to ensure that WLM does not flood the system with too many concurrent requests.

Resource Adjustment

Several times every 10 elapsed seconds (the exact interval varies according to hardware speed), WLM looks for resource imbalances. If any resources are found to be over-utilized or under-utilized, resources may be reallocated. In the case of WLM-managed initiators, if a service class period is experiencing queue delay one option available is to start additional initiators. WLM does ensure that enough CPU and storage are available to support the initiators that it starts.

Housekeeping

Every 10 elapsed seconds, WLM code examines the current state of the system, looking for exclusively-allocated resources which are no longer needed by the service class periods to which they are allocated. Initiators are drained if insufficient CPU exists to support them, if an excessive number of them are swapped out over the long term, or if they are idle over the long term. Any job executing in a draining initiator is unaware of the initiator's intention to terminate when it finishes. After the initiator is drained, it is eligible for reassignment to other service classes or for eventual termination if there is no demand for it over the long term (several minutes minimum).

Solving Affinity Problems

The JES2 and WLM sampling code cooperate to reflect the jobs eligible to execute on each system in the sysplex. WLM shares the data from each system with its peers so that each system has a view of where initiators for a given job class+service class need to be in order to address queue delay. WLM does not distinguish between system affinities and affinities due to scheduling environments.

It is possible to meet the goals of a service class while some queued jobs are unable to execute due to affinities. For example, system A could have enough capacity to keep the service class meeting its goals while jobs exist with affinity only to system B. If system B has no initiators for the job class+service class combination on system B, those jobs can never execute. Special code exists to understand cases like this and guarantee that some initiators get started on system B as long as enough CPU and storage exists to do so.

How Many Initiators?

To WLM, this is much like asking "What dispatching priority will this work get?" The answer is: it depends. The interaction between goals stated in the service policy, arrival/departure rates, and available resources is not simple. The number will vary over time, based on workload and resource availability. Better to be concerned with whether or not the goals are being met:

- If the goals are being met, you are done.
- If the goals are not being met, is initiation the bottleneck?
- If the goals are not being met and initiation is the bottleneck, is there sufficient CPU and storage available to support more initiators without impacting other goals adversely? If CPU or storage is the problem, more initiators will not help.

Initiator Placement

Given that WLM can start initiators to address queue delay and given that the job queue is really MAS-scoped (for simplicity assume one JES2 per image in the sysplex, all in the same MAS), the natural question is where in the sysplex initiators will be started. The overall philosophy is:

1. Avoid constrained systems, for example those with a recent storage shortage.
2. Utilize excess CPU capacity where it exists. Displace work with less important goals only if necessary.
3. When displacement is necessary, minimize the damage with respect to goals. In other words, if some work can be displaced and still meet goals, do that rather than displacing something else that would miss goals as a result.

Short Term

Short term placement covers two cases: initiators started in response to a $SJ command and those started to solve affinity problems not impacting goal achievement (see "Solving Affinity Problems"). Since the work to be started is limited in scope, placement is based on recent CPU capacity amongst the eligible systems. This is very similar to the existing routing functions for Sysplex Router and Generic Resource support, which all use the data returned by macro IWMWSYSQ in their decision-making.

Long Term

Long term placement is used by policy adjustment when starting initiators. It considers recent CPU capacity, and if work must be displaced local data from each system is gathered so that the impact to displaced work with respect to the policy is minimized. Data on these decisions is provided in the SMF type 99 records.

Balancing

Affinities and available CPU capacity are the primary influences on where work runs. In the absence of affinities WLM will tend to start initiators in such a way that the available CPU capacity will be equal across the sysplex. This does not imply that each service class will have its initiators spread equally across the sysplex. The existence of other work affects available CPU capacity, and therefore placement.

During testing of the placement algorithms, testcases were designed to spread initiators equally across two systems: they were not, because the jobs in the workload were all submitted on one system. The CPU required for conversion of the JCL stream made that system a less attractive choice and more initiators were started on the other system. When enough of the jobs had started to compensate for JES2's CPU, the placement was much more even. By far the best indicator of the effectiveness of the placement algorithm came when we noticed that the unused CPU service units differed by less than 3000 out of 165,000 per minute, or 1.8%. This is just a single example, other intervals during the same run were closer by a factor of ten.

Conclusion

For the first time in the history of MVS, WLM provides a direct connection between initiation policy and performance management. While it does not solve all batch management problems, it provides a base upon which to build. WLM initiator management and resource affinity scheduling together represent a significant step toward bringing the full power and sysplex-wide management of WLM to bear on batch workloads.

References

Further discussion of WLM batch management issues is available on the WLM web page at URL http://www.ibm.com/servers/eserver/zseries/zos/wlm . A document written by Mike Cox of the IBM Washington System Center treats some of the JES-related and operational issues in greater depth. Information on other WLM functions can be found on the web page and in conference proceedings from SHARE, MVS Expo, and CMG. Information on WLM functions provided in OS/390 Release 4 was presented starting at the August 1997 SHARE. The handouts presented at the session for this paper do contain additional information not covered here due to length restrictions.

1. [Planning] " OS/390 MVS Planning: Workload Management ", GC28-1761
2. [J2Cmd] " OS/390 JES2 Commands ", GC28-1790
3. [J2Mig] " OS/390 JES2 Migration Notebook ", GC28-1797, see Chapter 9
4. [J2ITG] " OS/390 JES2 Initialization and Tuning Guide ", SC28-1791, see Chapter 2 topic " Job Selection and Execution ",
5. [EffUse] " Effective Use of Workload Manager Controls ", Ed Berkel and Peter Enrico, CMG94

Acknowledgements

The author wishes to thank the following people for their constructive review and suggestions for this paper:
James Kilgore
Frank Soegeng
Peter Yocom
David Booz
Greg Dritschler
John Kinn
Chip Wood