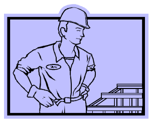


#26 zNibbler (z/OS' Workload Manager)
zTidBits Series

The workload manager (WLM)† component of z/OS is responsible for ensuring that customer response time objectives are met for the set of diverse applications and workloads that a customer may run. This includes making changes or adjustments on the local sysplex member level or LPAR, as well as redistributing work across members of a sysplex when needed, and is largely thought of as moving the work to the resources.

Instead of specifying low-level controls to tune system resources, WLM gives the system administrator the capability to specify goals for work in the system in business terms. The operative principle is that the system should be responsible for implementing resource allocation algorithms that allow these goals to be met. WLM is unique in offering externals that capture business importance and goals and implement them on behalf of the system administrator.

Two primary facilities that WLM provides should be introduced. The first is the ability to partition the universe of work requests into mutually disjoint groups, called *service classes*. This partitioning, referred to as *classification,* is based on the attributes of an individual work request, which might include the USERID that submitted the request, related accounting information, the transaction program to be invoked or the job to be submitted, the work environment or subsystem to which the request was directed, and so forth.

Installations are able to specify which service class is associated with each work request by specifying the value for one or more attributes and the corresponding service class. Defaults and other techniques may be used to group work requests into each service class. Each service class represents work requests with identical business performance objectives. To address the fundamental problem that the resource demands of most work requests are unknown at the outset and can vary depending on parameters that may be known only at execution time, there is a need to allow the business objectives to change on the basis of the resource demands of the work request. This is quite different from the requirement in other implementations that the resource demands be known in advance.

A service class comprises a sequence of *periods,* with a value defined by the installation to express how long a work request is considered to belong to each period. This "duration" is a measured amount of service consumed (service units (SUs)) that incorporates time spent actually running instructions on a processor, along with other components of service defined by the installation. Each work request starts in period 1 and is managed according to the first period goal (to be described in the next few paragraphs) until enough service is consumed to exceed the first period "duration." The work request is then moved to the second period and managed according to the second period goal, and so forth.

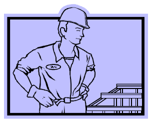
Each period has an associated *goal* and an associated *importance*, as alluded to above. Note that the durations may be assigned different values for distinct service classes, even when comparing the same period. In the same way, the goals for a given period in different service classes may be distinct. An installation may specify explicitly three major goal types for work requests. Certain activities associated with system work may be managed implicitly; these are accorded special treatment and do not require installation specification. The goal types provided by WLM are *response-time*, *discretionary,* and *velocity*. These types of goals are now described in turn.

Response-time goals indicate a desire for internal elapsed time to be, at most, a certain value. "Internal" refers to the fact that the time is measured from the point at which the work request is recognized by the system to the point at which the work request is considered complete. Note that elapsed time refers to wall-clock time and therefore includes delays when programs are not running on behalf of the work request. Use of wall-clock time is desirable, since it reflects the impact on a user awaiting completion of the work request.

The second goal type, discretionary, indicates that there is no business requirement for the work to complete within a certain predetermined elapsed time, and the system should use its discretion in giving resources to such work when it is ready to run. In an unconstrained environment, discretionary work will use available resources. In a constrained environment, discretionary work may be denied resources in favor of work requests with other goal types. Optional controls not described in this paper allow the installation to ensure that discretionary work makes progress in a constrained environment.

The third goal type is velocity. Work requests that are not considered discretionary and do not have a set response-time objective may nevertheless need further control to reflect the degree of delay that is tolerable once the work request becomes ready to run. Such work requests may be long-running (possibly "never-ending")

† **See: #32 zTibits (z/OS' WLM 101) for more information**



and want to run periodically or intermittently, during which time the work request must have access to resources. Velocity goals address this category of work requests. Many subsystem address spaces use this setting.

A final concept associated with periods, hich was mentioned previously, is that of *importance*. Importance is merely a relative ranking of work and is a factor only in constrained environments, where the algorithms must make choices as to whose goals will be attended to first when system resources are reallocated. The algorithms attend to the goals of work at the highest importance before attending to those at lower importance levels.

The concept of period was introduced to demonstrate a fundamental behavior of WLM on work that addresses the variability of resource demands. WLM does not require the system administrator to know these demands in advance. Goals are allowed to change on the basis of their cost. The term "period" is not used subsequently in order to avoid certain technical discussions and difficulties that are not central to the theme of this paper. The more general concept of "service class" is used in the remainder of the paper.
Example:

A service class representing TSO/E
work with multiple periods

Service Class		TSO
Period	1	
	Response Time	85% 0.5 second
	Importance	1
	Duration	400 Service Units
Period	2	
	Response Time	80% 1 second
	Importance	3
	Duration	1000 Service Units
Period	3	
	Response Time	60% 15 second
	Importance	4

A service class representing CICS transactions

Service Class		CICSHOT
Period	1	
	Response Time	0.5 second AVG
	Importance	1

A service class representing IMS transactions.

Service Class	IMSCAT1
Response Time	95% .3 Second
Importance	1

A service class representing IMS transactions.

Service Class	OIMSCAT3
Response Time	5 sec AVG
Importance	3

Batch Job Velocity

```
Service Class = BATCHX
Period 1
    Velocity      = 50
    Importance    = 3
    Duration      = 2500 SU
Period 2
    Velocity      = 15
    Importance    = 5
```

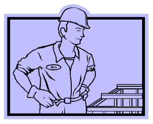
Batch Job Discretionary
Note: Last period

Service Class		DEVBATCH
Period	1	
	Response Time	80% 1 minute
	Importance	2
	Duration	2000 Service Units
Period	2	
	Response Time	80% 5 minutes
	Importance	3
	Duration	10000 Service Units
Period	3	
	Discretionary	

The WLM philosophy for **resource adjustment** is described in some detail in subsequent sections, but it is essentially a receiver– donor loop with respect to adjusting resources. The fundamental principle on which its success is based is that the system need not determine the optimal change at any given point. It is sufficient that the system make an improvement when adjustments are made. This principle allows WLM to avoid the trap of over analysis, where system overhead may balloon in search of optimal solutions. By working on only a single problem at a time, the algorithms leave intact resource allocations that are working well.

A number of benefits arise from the WLM philosophy of goal-oriented performance management. The most obvious of these benefits is the *simplification in defining performance objectives* and initialization states to the system. The system administrator is able to specify business objectives directly to the system in business terms. It is still the responsibility of the system administrator to ensure that each service class contains work with similar goals, business importance, and resource requirements in order to acquire the maximum benefit from WLM. Placing work with similar goals but diverse resource requirements into the same service class limits the ability of WLM to make effective resource tradeoffs, to correctly project resource needs, and to project the effects of resource adjustments.

continued



At the outset, the system administrator does *not* have to understand low-level technical controls. There is no need to adjust dispatch controls. For example, the system administrator does not have to understand tradeoffs for setting dispatch priorities when a machine has a single very fast processing engine vs. a single slower engine vs. multiple slower engines vs. multiple very fast engines.

Additionally, the business policy defined to WLM handles mixed workloads, i.e., interactive, batch, transaction processing, and data mining (analytic) environments. The system is responsible for resource management of work in execution and for the management of delays and their impact on attaining goals. There is no need to partition the images or nodes of the parallel environment for each separate workload. The system administrator does not have to specify the resource demands of work in advance. Effective use of capacity is ensured by the management algorithms.

WLM policy-adjustment algorithm

The WLM subfunction responsible for allocating computing resources based on the WLM policy is called the *policy-adjustment algorithm*. It is the job of the policy adjustment algorithms to determine whether there are service classes missing goals and to decide which, if any, resource reallocations are appropriate to help one of these classes to meet its goals. The functions of the policy adjustment will not be fully described in this paper. Although, these functions, which are relevant to WLM involvement also in IRD LPAR CPU management, are briefly summarized at the conclusion of this paper.

State sampling

The first step in solving the performance problem of a service class is to find out why the work is being delayed. Many delays can be measured quite precisely, but the cost is prohibitive. WLM solves this problem with state sampling. Four times a second, WLM samples *every* work unit in every system being managed. From these samples, the WLM builds a picture of the work in each service class. It learns where each service class is spending its time. It learns how much each class is using each resource, and how much each class is delayed waiting for each resource. The samples are aggregated over time, and from this picture of the work in each class, WLM can determine what to do.

Performance index

A fundamental problem with trying to meet performance goals and making tradeoffs among different work with different goals is knowing how well work is proceeding with respect to its goals and with respect to other work. The solution used by WLM is the performance index. The calculation of the performance index for a class with a response-time goal is as follows:

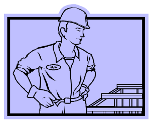
$$\textbf{performance\_index} = \frac{\textbf{actual\_response\_time}}{\textbf{goal\_response\_time}}$$

It is a calculated measure of how well work is meeting its defined performance goals. The performance index allows comparisons between work with different goals. A performance index of 1.0 indicates that the class is exactly meeting its goal. A performance index greater than 1.0 indicates that the class is performing worse than its goal, and a performance index less than 1.0 indicates that the class is performing better than its goal.

Policy-adjustment framework

The policy-adjustment algorithm is invoked every ten seconds to assess reallocation of system resources to better meet performance goals. This period of ten seconds is referred to as the *policy interval*. The effects of the resource reallocations made during one policy interval are observed in subsequent policy intervals and function as a feedback loop for continuous adaptive policy adjustment.

The first step is to choose the most eligible **receiver** service class. This is defined to be the most 'important' service class missing its goals. If there is a tie with respect to importance level, the service class with the highest performance index is chosen. The next step is to determine which resource is the biggest source of delay for the receiver. This is done by searching the state samples for the receiver service class to find the delay samples with the largest count for the receiver. The resource corresponding to these delay samples is considered the receiver's biggest bottleneck. Once a bottleneck has been chosen, a resource-specific *fix routine* is called to attempt to resolve the bottleneck by reallocating more of the bottleneck resource to the receiver service class. The WLM policy-adjustment framework is



designed to be extendable. New resources can be *added* to WLM goal-oriented management by adding a fix routine for that resource to the policy adjustment framework.
The only requirements are the following:

1) A delay that indicates a lack of the resource must be able to be sampled
2) A control variable controlling access to the resource must be able to be defined
3) A relationship must exist between the control variable and the resulting delay samples.

The job of the fix routine is twofold**. First**, it must determine whether the performance of the receiver can actually be improved by providing more of the bottleneck resource. It makes this determination by projecting how much the performance index of the receiver will improve if the receiver is given more of the bottleneck resource.

For a resource reallocation to be considered worthwhile, it must result in a *minimum* performance index improvement or the *elimination* of a minimum number of delay samples. This minimum required improvement to the receiver is called *receiver value*. The receiver value criteria are designed to reject very small improvements. The reason for rejecting actions having too little receiver value is to avoid making changes that yield only marginal improvements. Instead, the policy-adjustment algorithm goes on to select and assess another bottleneck for the current receiver or to select a new receiver. The receiver value criteria also indicate to the fix routine the point at which it has given the receiver enough help. These criteria keep one system in a sysplex from trying to solve all of the performance problems of a class when the class is running on more than one system. The criteria also keep multiple systems in the sysplex from trying to solve all parts of the problem simultaneously and running the risk of making too great a correction.

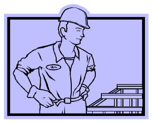
The **second** job of the fix routine is to project the impact on other work of providing additional resource to the receiver and determining whether the overall impact of the reallocation is a "good tradeoff" on the basis of the WLM policy. The fix routine must project how the performance index of each service class affected by the resource reallocation will change. For example, if the action being considered is to raise the CPU dispatch priority of the receiver, every service class with a CPU dispatch priority between the old priority of the receiver and its proposed priority inclusive is potentially affected by the change and must have a new performance index *projected*. The service classes whose performance will be adversely affected by the resource reallocation are known as **donor** service classes.

Once a performance index *projection* has been done for each donor, the next step is to determine whether the resource reallocation is a good *tradeoff* with respect to the receiver and each donor. This decision is based on the importance and performance index of the receiver and donors. For example, if the donor is less important than the receiver and the receiver is missing its goal, the resource reallocation is considered a good tradeoff with respect to that donor. On the other hand, if the donor is more important than the receiver and is missing goals, or if the action would cause the donor to miss its goals, the action is not a good tradeoff. If the receiver and donor are equally important, the action is considered a good tradeoff if the improvement in the receiver performance index is at least as large as the increase in the donor performance index and the action causes the performance indexes to converge. If the resource reallocation is found to be a good tradeoff with respect to the receiver and each donor, the action is taken. Otherwise, the fix routine returns to the main policy-adjustment algorithm so that another bottleneck for the same receiver can be addressed or another receiver chosen.

LPAR CPU management overview
Intelligent Resource Director (IRD) LPAR CPU management extends WLM goal oriented resource management to allow for dynamic adjustment of logical partition processor weights. This function moves CPU resource to the partition with the *most* "deserving" workload based on the WLM policy, and allows the system to adapt to changes in the workload mix.

The processor weight "*exchange*" is done among logical partitions in an LPAR cluster, which is the set of z/OS LPAR servers belonging to the same sysplex and running on the same 'physical' machine. WLM keeps the total processor weight of the LPAR cluster constant, so that partitions that are not part of the LPAR cluster are unaffected.

continued



Logical CPU management

A companion function to LPAR weight management is logical CPU management. This function *optimizes* the number of logical CPUs online to a given logical partition on the basis of the partition's current processor weight and current CPU resource consumption.

There are *two primary reasons* for doing this optimization:

1. If WLM increases the processor weight for a partition, the partition will be unable to use the amount of CPU Resource represented by the new weight unless it has sufficient logical CPUs online. For example, if a partition of a ten way physical multiprocessor is given a weight representing 50% of the physical capacity of the machine, it must have at least five logical CPUs online. If the partition has only four logical CPUs online, it is able to use only 40% of the physical capacity of the machine.

2. This optimization is based on the mechanism PR/SM uses to dispatch the logical CPUs of a partition on the physical CPUs of the machine. PR/SM evenly divides the processor weight of a partition among the logical CPUs online to the partition. The more logical CPUs online to a partition, the lower the processor weight of each logical CPU. This effectively makes each logical CPU less powerful. But, workload performance improves as the power of an individual logical processor increases.

– – –

REF: W. J. ROONEY ET AT.