Intelligent Resource Director

by W. J. Rooney
J. P. Kubala
J. Maergner
P. B. Yocom

Intelligent Resource Director (IRD), a feature of the IBM eServer zSeries™ processors and the z/OS™ operating system, manages multiple heterogeneous workloads with various business priorities toward achieving their goals. It establishes a more synergistic relationship with the Workload Manager (WLM) component of z/OS and the zSeries hardware, and augments the adjustments that WLM makes to local sysplex members by managing the set of logical partitions on a particular central processing complex (CPC) that are part of the same sysplex, known as a logically partitioned (LPAR) cluster. This paper describes the three primary areas that IRD manages: LPAR CPU management, channel subsystem priority queueing (CSSPQ), and dynamic channel path management (DCM).

Introduction

IRD [1] further extends the lead of the IBM eServer zSeries* processors in managing multiple heterogeneous workloads with various business priorities. Through its use, a more synergistic relationship is established between z/OS* and the zSeries (formerly S/390*) hardware with respect to the allocation of resources among logical partitions. The WLM component of z/OS is responsible for ensuring that customer policy goals 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, as well as redistributing work across members of a sysplex when needed, and is largely thought of as moving the work to the resources.

With IRD, WLM augments the adjustments it makes to local sysplex members by managing the LPAR cluster. This can be thought of as moving the resources to the work.

Processor Resource/Systems Manager overview

The IBM Processor Resource/Systems Manager* (PR/SM*) currently supports the creation of up to 15 logical partitions on a single CPC. Each of these logical partitions is fully capable of running an operating system independently. The following are all supported in a logical partition: z/OS (with or without Parallel Sysplex*), VM, VSE/XA, TPF, UNIX** applications under UNIX System Services (USS), and Linux**.

PR/SM allows granular levels of resource allocation. Storage can be assigned to logical partitions in increments of 1 to 128 megabytes, rather than gigabytes, depending on processor model and total available memory. I/O channel paths can be shared by logical partitions to fully utilize the bandwidth of the IBM Enterprise Systems Connection (ESCON*) and Fiber Connection (FICON*) channels. The PR/SM dynamic reconfiguration capability offers allocation of additional resources to partitions with demanding workloads without disruption to the workloads in the logical partitions.

PR/SM management of central processor (CP) resources allows for the use of either dedicated or shared CPs. Although CPs can be dedicated to logical partitions, the real power is in the effective use of shared processors. With shared processors, when a workload in one partition

0018-8646/02/$5.00 © 2002 IBM

goes idle, the available processing time is automatically redistributed to other partitions with no intervention.

Logical partition processor weights are a priority policy for logical partitions based on a user's dispatching priorities. They are specified when allocating the resources of the machine to its workloads. Using processor weights with shared processors, the user can use subprocessor granularity for allocating machine resources to workloads. Each logical partition is treated as a separate workload, managed against the processor weight policy. Processor weights define relative priorities of logical partitions for determining which logical partition receives the resource when there is contention for that resource. When there is no contention for the resource (i.e., all logical partitions are not, at the moment, trying to use all of the CP resource they are entitled to), the other logical partitions automatically fill that "white space" and use that excess capacity. Even the redistribution of the "white space" is done in accordance with the processor weight policy. When the logical partition that was under-utilizing its entitled resource requires it again, the resource moves back to it in accordance with the processor weights. The policy can be dynamically updated, with changes taking effect immediately.

The z/OS operating system running in a logical partition provides yet another level of heterogeneous workload management. Multiple workloads within a z/OS logical partition are given an even finer granularity of workload management via the WLM component of z/OS. The priority of the logical partition with respect to other logical partitions is managed by PR/SM. The priority of the individual workloads and applications within the logical partition is managed by WLM. IRD creates a synergy between the management done by PR/SM and the management done by WLM, producing a fully dynamic and automatic resource management capability that is unmatched in the industry. The detailed knowledge of WLM can be used to redirect the machine (PR/SM) resource allocations according to workload priorities across a set of logical partitions on a CPC called an LPAR cluster.

Workload Manager overview

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.

The z/OS operating system can function in one of two modes from a performance-management perspective. WLM *goal mode* allows a system administrator to state goals for the work in the system, and WLM is responsible for allocating computing resources to meet these goals. In WLM *compatibility mode* (sometimes referred to as compat mode), the system administrator must use low-level resource control to allocate computing resources to the work in the system. Compatibility mode exists to provide compatibility with the performance-management externals provided in earlier versions of the operating system. Goal mode is the focus of this paper. Except where compatibility mode is specifically referenced, this paper describes z/OS function available when running in goal mode.

Two primary facilities that WLM goal mode provides should be introduced at this point. 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 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 precise definition of when the clock starts or stops ticking to capture the elapsed time is documented in Reference [2] for each particular environment, and so is not elaborated in this paper.

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") 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.

A final concept associated with periods, which was mentioned above, 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. For a more complete description of WLM externals, see Reference [2].

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 overanalysis, 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.

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, e.g., interactive, batch, transaction processing, and data mining 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.

S/390 I/O overview

When the IBM MVS/Extended Architecture (MVS/XA*) was made available in 1983, the responsibility for selecting paths for driving I/O requests was moved from the input/output supervisor (IOS) portion of the operating system to a new component of the hardware known as the channel subsystem (CSS). This offloaded work that had previously been done by the central processing unit (CPU) to processing units in the channel subsystem known as I/O processors (IOPs) on bipolar processors and more recently referred to as system assist processors (SAPs) on the CMOS processors.

In order for this to function, the channel subsystem must know what paths are available to access the device which is the target of the I/O request. This is done by defining the I/O topology through a program known as the I/O configuration program (IOCP). The output of the IOCP, known as the I/O control data set (IOCDS) is then loaded into the CSS. Through this, the CSS can determine the set of paths that are available to access a device.

The MVS/XA I/O architecture allows for a maximum of eight paths to be defined to each device. When the XA I/O architecture was introduced, the method of physically attaching I/O devices to a System/370* processor was via the original equipment manufacturer interface (OEMI) or parallel interface. With the parallel interface, the set of paths which were defined via the IOCP was normally equal to the set of paths which were physically possible. That is, the channel subsystem typically knew of every path to a device that was physically cabled.

When ESCON architecture was introduced in 1990, and FICON architecture in 1999,[1] this all changed. With ESCON and FICON, channels are typically attached via a fiber optic cable to an ESCON or FICON director (or switch), which is in turn connected to one or more device control units.

A director may have as many as 248 external ports, each of which can be connected to either a channel or a control unit port. (For instance, the switch may be connected to 48 channels and 200 control unit ports.) This means that the number of paths physically available to access a control unit or device is much greater than the number of paths that may be defined (i.e., eight). In this case, there are 48 possible paths to a single control unit port, and if the control unit has two or more ports connected to the same switch, the number of possible paths would then be $48n$, where n is the number of ports.

The I/O configuration definition process is complex and requires significant skills. The process normally involves determining the number of channels required by a control unit to satisfy its bandwidth requirements and the number of other control units, if any, that can share that same set of channels.

There are availability considerations. For example, even if only a single channel is ever required by a control unit, two or more are normally defined to it in case of a failure somewhere along the path.

The configuration process tends to be iterative. A configuration is defined, measured, and analyzed. This is repeated until the configuration has the desired response time and availability characteristics.

Even if the configuration is defined perfectly the first time, the characteristics of the workloads on systems change. New workloads are added, some workloads grow more quickly than others, and new I/O devices are added to the configuration. Some workloads are less predictive than others. Some may vary slightly from week to week, and others may have sudden and unpredictable spikes in demand. These factors combine to create a situation in which an I/O configuration that allowed a user's response-time goals to be met last week is inadequate to do so this week. To make matters worse, the user may actually have sufficient I/O resources, just not where they are currently needed.

In 1991, dynamic I/O reconfiguration was introduced. This allowed the I/O configuration to be dynamically modified. Devices could be added to or removed from the I/O configuration, and the attributes of the devices could be changed, all without requiring that the system be restarted. Even devices that were currently in use could have certain characteristics changed. For example, a path could be added to or removed from a set of devices without affecting the applications that were currently using those devices.

IRD extends the tools already available to WLM to manage I/O by providing two new functions: dynamic channel path management and channel subsystem priority queueing. When used with WLM, these new functions reduce the time and skills required to define an I/O configuration, while at the same time allowing WLM to modify the I/O configuration in response to the demands of the workloads.

LPAR CPU management

Problem statement

As discussed previously, one of the resources that can be allocated to a logical partition is the processor weight of the partition. Weight determines the portion of the machine's shared CPU resource allocated to a logical partition when there is competition for CPU among two or more partitions. It is the job of the system programmer to set processor weights so that the workloads running in

[1] FICON bridge was introduced in 1999, followed by FICON native in 2000.

the logical partitions receive the CPU they require. If the workload mix changes, the weights have to be reevaluated and manually changed. Properly adjusting these weights requires a worker to monitor workload performance and understand the business goals and tradeoffs for these workloads. An alternative would be to configure enough CPU capacity to allow for peak workload demands so that no critical work is ever delayed, but this would be expensive. The goal of IRD LPAR CPU management is to simplify this configuration task by automatically managing physical processor resources to allow for high utilization of physical CPU capacity, while ensuring that performance objectives are met at times of peak demand.

LPAR CPU management overview

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 images 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.

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. First, 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.

The second reason for 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. Some workload performance improves as the power of an individual logical processor increases. The extreme example of this would be a single tasking workload, which can take advantage of only a single CPU.

On the basis of these factors, the optimal number of logical CPUs to have online to a partition is the smallest number of CPUs that can provide the CPU capacity required by the partition.

WLM CPU weight-management configuration

IRD LPAR CPU management works for uncapped shared processors only.[2] Members of the sysplex with dedicated CPs or capped shared CPs are not managed by WLM, but they are tolerated. WLM is allowed to change the distribution of the weights in the LPAR cluster, but the partitions outside the cluster cannot be changed, and the total of the weights in the cluster cannot be changed.

To define logical partition controls for managing CPU weights in an LPAR cluster, the following definitions are needed for each logical partition:

- *Initial weight*: The amount of processing weight the logical partition is initially assigned. This defines the amount of processing weight that is added to the LPAR cluster when the logical partition joins the cluster. It also defines the amount of weight that must be removed from the LPAR cluster, regardless of where it has been redistributed, when the logical partition leaves the cluster.
- *Current weight*: Set initially from initial weight. This is the current setting of the processing weight for a logical partition, taking into account changes made by WLM.
- *Minimum weight*: A boundary condition below which WLM cannot adjust the current weight of the logical partition.
- *Maximum weight*: A boundary condition above which WLM cannot adjust the current weight of the logical partition.

The specification of minimum and maximum weights is optional. However, all shared logical partitions are required to have a minimum of 1 for a current weight. Also, the current implementation limits weights to a maximum of 999. These limits are used implicitly when no limit is explicitly specified.

PR/SM provides interfaces with software to join an LPAR cluster, as well as information about the configuration of the LPAR cluster container. There is also a mechanism to update the configuration of the LPAR

[2] WLM can enforce a defined capacity for variable Workload License Charge products running in a logical partition, using what is sometimes referred to as a "soft cap." To do this, the hardware console is not used to turn on capping for the logical partition. WLM turns capping on and off via software as needed to enforce the defined capacity. The details of how this is done are beyond the scope of this paper, but it is compatible with LPAR CPU management.

cluster container, i.e., change the current weights of the logical partitions within the LPAR cluster.

The software in a logical partition "decides" to join an LPAR cluster when it is initialized (IPLed[3]). As logical partitions join an LPAR cluster, their initial weight is added to the pool available for WLM to manage within that LPAR cluster. WLM running in the logical partitions can then redistribute the weights, respecting any specified or implied minimum or maximum weights for the logical partitions within that LPAR cluster.

When a logical partition is system-reset, re-IPLed, or deactivated, it is removed from the LPAR cluster by PR/SM. When a logical partition is removed from an LPAR cluster, processing weight equivalent to the initial weight of that logical partition is removed from that LPAR cluster. If the current weight of that logical partition is no longer equal to its initial weight, a redistribution of weights among the remaining logical partitions in that LPAR cluster is needed, and this can cause the current weights of the remaining logical partitions in the cluster to increase or decrease. This redistribution is performed proportionally to the current weights of the remaining logical partitions at the start of this process.

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 are fully described in Reference [3]. These functions, which are relevant to WLM involvement in IRD LPAR CPU management, are briefly summarized in the following sections.

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.

[3] IPL: initial program load.

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:

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

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 following pseudocode outlines the steps taken to determine what (if any) resource reallocation should be done:

```
Main_Loop: Do until a resource reallocation has been
done or no more potential receiver service classes exist.
  Choose the next most eligible receiver service class.
  Do until a resource reallocation has been done or no
  more bottlenecks for the receiver.
    Find the next biggest bottleneck for the receiver.
    Call "Fix Routine" for bottleneck.
      Determine how much of the bottleneck resource
      the receiver needs.
      Find the most eligible donor of the bottleneck
      resource.
      Determine whether reallocating resource of donor
      to receiver is a "good tradeoff."
      If action is a "good tradeoff," reallocate resource
      and exit Main_Loop.
  End.
End.
```

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; and 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.

CPU dispatch priority management

Before discussing the WLM processor weight-management algorithm, a brief discussion of the WLM CPU dispatch priority-management algorithm is in order, since the weight-management algorithm is based on the same underlying concepts as dispatch priority management.

Maximum processor demand

The first problem with projecting the effects of dispatch priority changes is that the inherent processor demand of the work units in a service class cannot be measured directly. If a class consumes x amount of CPU time when it runs at dispatch priority A, it cannot be assumed that it will still consume the same amount of CPU time when it runs at a higher or lower priority or with a more or less competing demand. WLM solves this with an algorithm to project the consumption of a class at any dispatch priority, known as the *maximum processor demand*.

Maximum processor demand is defined as the theoretical maximum percentage of total processor time available to a logical partition that work units in a service class can consume if the work units experience no processor delay, viz.,

$$Maximum\_demand\_percentage = \frac{number\_of\_work\_units \times processor\_using\_samples}{(total\_samples - processor\_delay\_samples) \times number\_logical\_CPUs}.$$

The term *processor_using_samples*/(*total_samples* − *processor_delay_samples*) measures the percentage of time that the average work unit would execute on a CPU if it were to experience no CPU delay. Multiplying by the number of work units converts the average to a total demand. Dividing by the number of logical CPUs adjusts for the available CPU capacity. Maximum demand is calculated for each service class and accumulated for all of the service classes at each priority.

Wait-to-using ratio

A second basic concept is the CPU wait-to-using ratio, which is the ratio of CPU delay state samples to CPU using state samples. The wait-to-using ratio is a measure of how competition for CPU is affecting the performance of a given service class. The more CPU competition a service class has, the higher the wait-to-using ratio of the service class. Competition from work with higher CPU dispatch priority has a larger impact on the wait-to-using ratio than work at the same CPU dispatch priority. Lower priority has little to no impact on the wait-to-using ratio.[4] WLM measures the wait-to-using ratio both at the service class level and the total wait-to-using ratio for all work at a given CPU dispatch priority.

CPU dispatch priority decisions

When the biggest bottleneck for a service class is CPU delay, the policy-adjustment algorithm calls the CPU dispatch priority fix routine. To help the receiver service class, the CPU dispatch fix routine projects the effect of a combination of potential CPU dispatch priority changes. These changes include increasing the priority of the receiver and lowering the priority of other service classes that have an equal or higher dispatch priority. If a combination of potential CPU dispatch priority changes is found that gives the receiver enough benefit to pass the receiver value test and is considered a good tradeoff with respect to the donor service classes, WLM considers the set of priority moves to be the solution to the bottleneck and implements the service class dispatch priority tradeoff changes. On the other hand, if a combination of dispatch priority changes is found not to be a good tradeoff, the moves are abandoned, and a new set of dispatch priority moves is evaluated. If no more potential dispatch priority changes can be found, the receiver bottleneck cannot be solved by dispatch priority changes, and WLM will consider increasing the processor weight of the current logical partition.

A good predictor of the wait-to-using ratio at a given priority is the combination of the total maximum processor demand of service classes with a CPU dispatch priority higher than that priority and the maximum processor demand of service classes at that priority. The total maximum processor demand at higher priority has the strongest relationship with the wait-to-using ratio. To project the effect of CPU dispatch priority moves, WLM builds a table called the processor data table (PDT). This table contains one entry for each CPU dispatch priority that WLM uses. Each of these entries contains the measured wait-to-using ratio for service classes at that priority, the sum of the maximum processor demands for service classes at that priority, and the sum of maximum processor demands for the priority and all higher priorities. From this table WLM learns the relationship for each priority of that priority's wait-to-using ratio, the total maximum processor demand at work at higher priorities, and the maximum processor demand at the priority.

WLM also uses this table to project the effect of a priority change. For each potential service class CPU dispatch priority change, WLM subtracts the maximum processor demand of the service class from the maximum processor demand fields in the PDT for the current priority of the class and adds this value to the fields for the new priority of the class. Then, using the learned relationship between processor maximum demand and the wait-to-using ratio of a priority, WLM projects the new wait-to-using ratio for each priority affected by the priority move. From the change in wait-to-using ratio for each service class that will receive worse CPU access after the change, WLM projects the amount of additional CPU delay time the service class will experience. For the receiver service class, WLM projects the reduction in CPU delay time it will experience. From the change in CPU delay times, WLM calculates a new performance index. On the basis of the new performance index for each affected service class, WLM determines whether the benefit to the receiver meets the receiver value test and whether the action overall is a good tradeoff.

Processor weight-management algorithms

Data collection

WLM processor weight management functions by using a peer-to-peer relationship between the systems in the LPAR cluster. There is no master WLM instance making decisions for the whole LPAR cluster. Instead, WLM on each system determines whether work running on its system could be helped by increasing the processor weight of that system. That instance of WLM then determines the best system in the cluster to give up weight, and whether the weight reallocation is a good tradeoff. For a given WLM instance to make a judgment on whether a weight reallocation is a good tradeoff, it must be able to project the impact of the weight change on work running on the

[4] Because z/OS uses a time-slice-based dispatching, low-priority work can cause some delay to higher-priority work.

tradeoff system whose weight is being reduced. To make this projection, each WLM instance in the LPAR cluster must have access to data describing the performance of each service class on every member of the LPAR cluster. To make this data available to all members of the LPAR cluster, WLM uses a coupling facility. (A coupling facility is a shared-memory device that can be shared by members of a z/OS sysplex.) See Reference [3].

WLM creates one data structure in the coupling facility for each LPAR cluster in the sysplex: the LPAR cluster structure. The structure for a given LPAR cluster is uniquely identified by including the machine serial number as part of the name of the data structure. The sysplex name does not have to be included as part of the data structure name, because a coupling facility cannot be shared across sysplex boundaries.

Within the LPAR cluster structure, there is one entry for each member of the LPAR cluster called the LPAR data entry (LDE). The LDE contains performance data for each service class with work active on the associated system. This data includes the state samples of the service class, its local performance index, its maximum processor demand on the associated logical partition, and its CPU dispatching priority on the associated system. The LDE also contains the processor data table (described above) for the associated system. At the beginning of the policy interval, each system writes its LDE to the coupling facility and reads the LDE of every other member of the LPAR cluster into local storage for easy access in later processing.

Receiver processing

The LPAR processor weight fix routine is called when the CPU dispatch priority fix routine is unable to take an action to help the receiver service class, either because of a lack of receiver value or because no potential CPU dispatch priority change is a good tradeoff. The LPAR processor weight fix routine determines whether the receiver CPU delay bottleneck can be addressed by raising the partition processor weight and thereby increasing the CPU capacity of the partition. The approach to determining whether increasing the weight of the partition is the appropriate action to address the receiver bottleneck is the same as the other WLM resource-management algorithms—the action must improve the receiver performance enough to meet the receiver value test, and the action must be a good tradeoff with respect to all donor service classes. In this case the impact to the donor service classes comes from removing CPU resource from work running on the partition whose weight is reduced.

The first step is to determine how much additional weight must be given to the partition to help the receiver enough to meet the receiver value test. WLM considers raising the weight of the partition by a fixed percentage of the average weight of the members of the LPAR cluster. To project how much the weight increase will improve the performance index of the receiver, WLM uses an approach similar to what is done to project the effect of a CPU dispatch priority change. First WLM adjusts the maximum processor demands in the PDT on the basis of the proposed increase in the partition weight. Maximum processor demand represents the percentage of the overall CPU capacity of the logical partition that a service class could use if unconstrained. Raising the capacity of a partition by raising its processor weight in turn lowers the maximum processor demand of each service class proportionally. WLM adjusts the maximum demand of each service class on the basis of the proposed weight change and then recalculates the total maximum processor demand values in the PDT on the basis of the new service class values. Once the PDT has been updated for the proposed weight change, WLM projects the performance index improvement to the receiver using the same algorithm that is used for CPU dispatch priority changes. If the improvement is not enough to pass the receiver value test, WLM will consider a weight increase of twice the original increment. If that increment is not enough to meet receiver value, WLM will try three times the original increment, and so on. The largest weight increment WLM will consider is 30% of the total weight of the LPAR cluster. If the projected improvement to the receiver of that weight increase is not large enough to meet the receiver value test, WLM abandons the attempt to help the receiver.

Donor selection

If a weight increase is found that improves the receiver performance enough to meet the receiver value test, the next step is to pick another member whose weight can be reduced by this amount. In choosing a logical partition from which to take weight, WLM attempts to take resources from the lowest-importance work. Therefore, WLM prefers to lower the weight of a partition running a large amount of low-importance work instead of a partition running only high-importance work. To understand the importance of the work running on each system, WLM maintains a table known as the service-by-importance table (shown as **Table 1**) for each member of the LPAR cluster. The table contains seven rows. The seventh row contains the total CPU service[5] used by discretionary work, which WLM considers importance 6. The sixth contains the total CPU service used by importance 6 work plus the contents of the seventh row.

[5] CPU service is a measure of CPU time used normalized for the speed of the CPU.

Table 1 WLM service-by-importance table.

Importance	
0	Total CPU service used by system
1	CPU service used by importance 1–6 work
2	CPU service used by importance 2–6 work
3	CPU service used by importance 3–6 work
4	CPU service used by importance 4–6 work
5	CPU service used by importance 5–6 work
6	CPU service used by discretionary work (importance 6)

The fifth row contains the total CPU service used by importance 4 work plus the contents of the sixth row, and so on. The first row adds in the CPU service used by system work (considered importance 0) and represents the total CPU service used by the logical partition.

In order to choose a partition whose weight to reduce, WLM converts the weight increment into an amount of CPU service required by the logical partition being helped using the expression

$$\frac{weight\_INC}{total\_weight} \times total\_CPU\_service\_capacity\_of\_machine.$$

WLM then searches the service by importance tables for each member of the LPAR cluster. The logical partition where the required service is available furthest down the partition's service by importance table is chosen as the candidate to have its weight reduced.

Donor projections

The next step is to project the increase in performance index for every service class with work on the logical partition whose weight is to be reduced, otherwise known as the *donor partition*. Because reducing the weight of the donor partition reduces its CPU capacity, the percentage of that capacity that each service class demands is increased proportionally by the weight reduction. Therefore, WLM recalculates the maximum processor demand of each service class on the donor partition. The new service class maximum processor demand values are then used to update the processor maximum demand fields in the PDT of the donor partition. From there WLM can project the performance index change for each service class using the same algorithm as used in the CPU dispatch priority management algorithm. On the basis of the performance index increase on the donor partition, WLM can determine whether the weight reallocation is a good tradeoff. If the reallocation is a good tradeoff, WLM calls PR/SM to adjust the partition weights of the receiver partition and the tradeoff donor partition. If the reallocation is not a good tradeoff, WLM repeats the search for a donor partition, excluding donors already tried. If no donor partition can be found, WLM abandons the weight-adjustment attempt and returns to the main policy-adjustment algorithm to try to address another bottleneck for the receiver or to help another receiver service class.

Adjustment intervals

Because adjusting logical partition processor weights can be a fairly dramatic reallocation of resources, WLM adjusts less frequently than it reallocates other resources. It changes processor weights only once per minute within a single LPAR cluster. However, there is one exception to the one-minute limitation on frequency, designed to ensure that action to help high-importance work is not delayed because of lower-importance work. Whenever a processor weight change is done, WLM "remembers" the importance and current performance index of the receiver. These values are stored in the LPAR cluster coupling facility structure so that they are visible to all members of the LPAR cluster. When WLM considers helping a new receiver by increasing the weight of its partition, and the new receiver has a higher importance or worse PI than the last receiver, WLM will make the weight move even if a minute has not passed since the last weight move.

Dynamic channel path management

Dynamic channel path management (DCM) allows z/OS to dynamically change channel path definitions to ESCON director-attached DASD control units in response to changing workloads, moving channel resources to the control units where they are required. When combined with WLM running in goal mode, DCM moves the channel resources to control units that are being used by business-critical workloads to help them meet their goals. When further combined with control unit priority queueing in the Enterprise Storage Server (ESS or "Shark"), channel subsystem priority queueing (CSSPQ; see a later section), and parallel access volume (PAV) alias tuning, the z/OS system becomes even more self-tuning and self-defining to meet workload goals.

Topology

During system initialization, DCM builds tables that represent the physical I/O topology. These tables include entries for each channel, ESCON or FICON director, and DASD control unit that is physically attached to and accessible by this system. The topology tables are used by DCM to determine what potential paths exist that DCM could add to a control unit in order to help it achieve its bandwidth requirements.

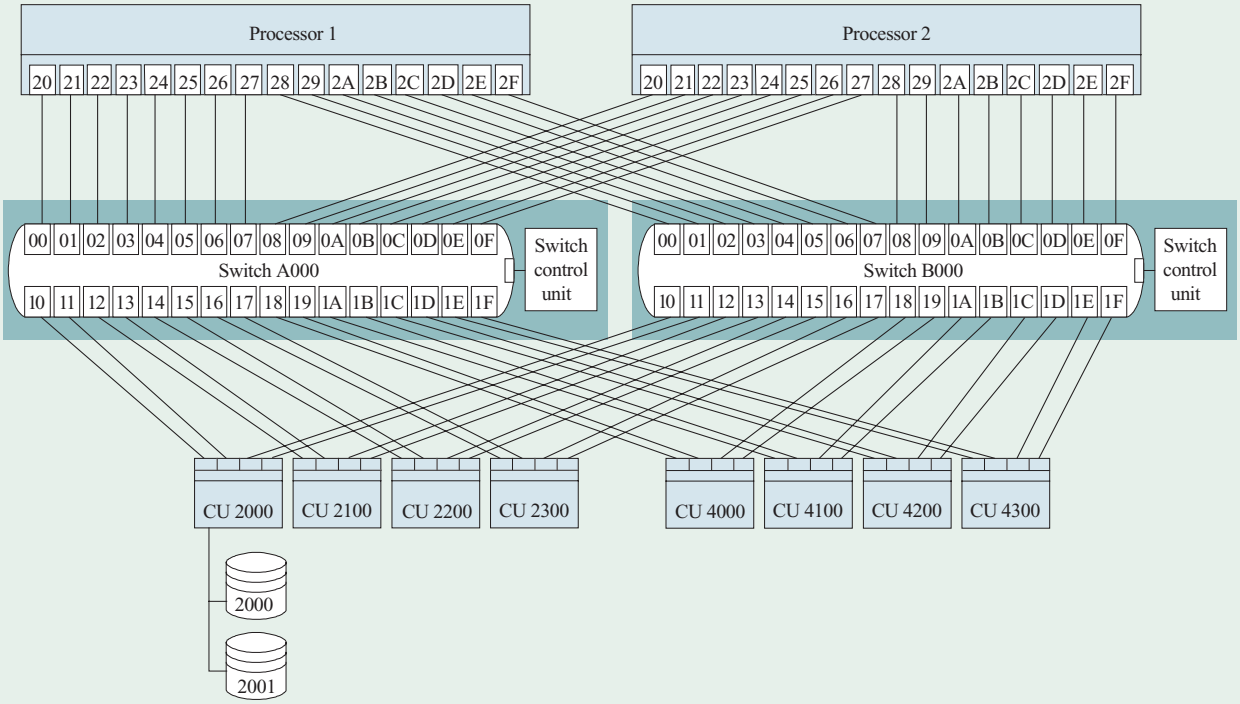

Figure 1

I/O topology.

The topology tables also include attributes associated with each entry. For example, a channel entry indicates whether the channel has been defined as managed or static,[6] while switch port entries indicate whether the port has been configured in such a way as to prevent DCM from using it to define a path. (The port may have been defined as "blocked," "prohibited," or "dedicated.")

DCM uses self-description commands, which are required to be supported by the ESCON and FICON architecture, to identify components of the I/O topology which have already been seen. For example, in **Figure 1**, DCM is able to determine that from Processor 1, channels 20 through 27 all connect to the same switch (A000), while channels 28 through 2F all connect to a different switch (B000). In addition, the devices attached to control unit 2000 can be accessed by Processor 1 via ports 10 and 11 on switch A000, as well as ports 10 and 11 on switch B000. With this configuration, DCM has 32 possible paths to the devices attached to control unit 2000 to consider: any of eight channels to port 10 on switch A000, eight channels to port 11 on switch A000, plus the 16 possible paths via switch B000. While the system is running, it is possible that the configuration may change. For example, someone may add a control unit to a switch by plugging its cables into unused ports on the switch.

To manage this situation, DCM maintains the tables by keeping the data current throughout the life of the system. In the case of the table which represents the switches, commands are sent to the switch control unit port every two minutes to read its key counter. If the value of the key counter is different from what it was on the previous query, the configuration of the switch has changed, and DCM updates its tables to reflect the current configuration.

Channel path cluster scoping

In order for DCM to dynamically move channel paths sensibly, it must understand a complete picture of the activity on the channel paths that it will move. The channel paths that can be moved by DCM are called managed channel paths. Traditionally, a customer defines channel paths in an I/O configuration as being shared by some or all of the logical partitions on a CPC. The scope of availability for these traditional channel paths is defined by the logical partition names themselves.

[6] Static channels are channels that are defined to control units in the traditional way and cannot be reassigned by DCM. Managed channels are channels that are owned, and therefore may be reassigned, by DCM. Both are defined using the z/OS hardware configuration definition (HCD) dialog. See Reference [5].

Table 2 Sample projected channel utilization.

Channel path ID (CHPID)	*Projected utilization (%)*
20	37
37	41
48	29
73	33

Table 3 Sample average channel utilization.

CHPID	*Projected utilization (%)*	*Average utilization (%)*
48	29	29
73	33	31
20	37	33
37	41	35

Managed channel paths are defined to belong to a specific LPAR cluster, not individual logical partitions. When any operating system is loaded into a logical partition, it does not immediately have access to any managed channel paths. All managed channel paths are automatically deconfigured from a partition when it is IPLed. It is only after the software in the logical partition declares that it is part of a specific LPAR cluster that it is allowed to configure the managed channel paths for that LPAR cluster online. With this process, only logical partitions that are part of the LPAR cluster can use the channel path, so the entire usage is understood by the LPAR cluster members in aggregate.

Subsystem I/O velocity

Once every interval (currently defined to be ten seconds), DCM goes through a process known as *data gathering*. During this process, DCM collects several measurements on DASD subsystems[7] and uses that information to calculate a new metric called the subsystem I/O velocity (IOV). The I/O velocity is conceptually a wait-to-use ratio on the channels serving a particular subsystem. It indicates how long I/O requests must wait for a channel to a subsystem, compared to how long I/O requests actually use those channels.

For example, assume that we have a subsystem which has accumulated eight seconds of connect time in the last ten seconds, and one second of pending time. Also assume that switch busy, control unit busy, and device busy are all zero. Then we would have an I/O velocity of $8/(8 + 1) = 0.89$.

[7] A subsystem is equivalent to a logical control unit (LCU). A physical control unit may contain several LCUs or subsystems. DCM operates on a subsystem basis; however, the more familiar term *control unit* is used throughout the paper.

If the pending time grew to three seconds and the connect time stayed the same, the I/O velocity would drop to 0.72. So the closer the I/O velocity gets to 1.00, the fewer the I/O requests that are waiting for channels to a subsystem. The closer it gets to 0.00, the more constrained the subsystem is for channel bandwidth. DCM calculates this I/O velocity for all DASD subsystems in the system. It then calculates a *default target I/O velocity*, looks at the I/O velocity actually being achieved for each of the managed subsystems, and attempts to get all of the subsystems close to the default target. A list of all subsystems whose I/O velocity falls below the target range is then passed to another phase of DCM called *imbalance correction*. Imbalance correction processes subsystems one by one, starting with the one furthest from target, and looks for changes that would help the subsystem reach its target by searching tables which represent the physical I/O topology built by DCM during system initialization and maintained since then. There are two types of changes that DCM looks for: paths that can be added to this subsystem and paths that can be removed from subsystems that share the path with the subsystem that requires more bandwidth.

Contention factor

In order to assess potential channel configuration changes, a metric is needed to allow us to calculate and then compare expected IOVs for each of the affected control units. One attribute required of this metric is that it must allow us to compare configurations with different channel utilizations and different numbers of channels. For example, is a configuration with four channels, each at 65% utilization, better than a configuration with three channels, each at 60% utilization? To determine this, DCM uses a *contention factor*, which is defined as the single-channel equivalent utilization yielding an equal probability that an I/O request to that control unit would wait. The contention factor is determined using the channel utilizations and resulting configuration of a proposed change.

To calculate the contention factor, DCM sorts all of the channels connected to each of the subsystems from lowest to highest utilization. It then calculates the contention factor based on the utilization of the first channel in the list, then the average of the first two channels in the list, and so on down the list. Subsequently, it uses the lowest contention factor found; this is done to reduce the effect of other subsystems which share one or more of the channels on the contention factor. For example, assume that subsystem 2200 is connected to four channels, and each channel has the projected utilization shown in **Table 2**. DCM then sorts them by increasing utilization and calculating the average utilization on the basis of the utilization of the first CHPID in the list, then

Table 4 Channel contention factors.

CHPID	*Projected utilization (%)*	*Average utilization (%)*	*Contention factor (%)*
48	29	29	29.0
73	33	31	14.7
20	37	33	8.8
37	41	35	6.3

the average of the first two channels in the list, and so on down the list, as shown in **Table 3**.

Each of the average utilizations is then applied to the graph shown in **Figure 2**, using the line representing the number of servers to arrive at an equivalent single-server utilization. Thus, in our example, we obtain the results shown in **Table 4**. DCM then uses the lowest contention factor, in this case 6.3, as the contention factor of the subsystem.

Next, the contention factor is converted into an I/O velocity. To do this, we use a table which is maintained by WLM. The table contains the historical IOV and contention factor samples acquired by WLM over time. Data is updated during data gathering, and old data is eventually aged out. There is one such table for each subsystem. **Figure 3** expresses graphically what the tables are attempting to achieve.

Using the contention factor of 6.3, an I/O velocity of 0.60 is obtained for the subsystem. This is then placed in a control block which represents a possible change as the projected I/O velocity. If there are other subsystems affected by this possible change, their projected IOVs are also calculated and placed in the control block.

Selecting the best option (IOV projection/availability/entropy)

Each of the possible changes may also affect the availability characteristics of the I/O configuration. If a path must be removed from a subsystem, DCM favors choices which do not leave all of the remaining paths with a single point of failure. When adding a path, DCM attempts not to select one that will have a single point of failure in common with all of the paths already defined for the subsystem.

A measure of configuration complexity is calculated for each of the possible changes. This measure of complexity is referred to an *I/O entropy index*. For example, consider a configuration in which subsystem 1 is connected to CHPIDs 23 and 24, subsystem 2 is connected to CHPID 24, and subsystem 3 is connected to CHPIDs 25 and 26. Assume that DCM has determined that subsystem 2 is below target and that it could add bandwidth either by adding CHPID 23 or 25. If it uses CHPID 23, the number

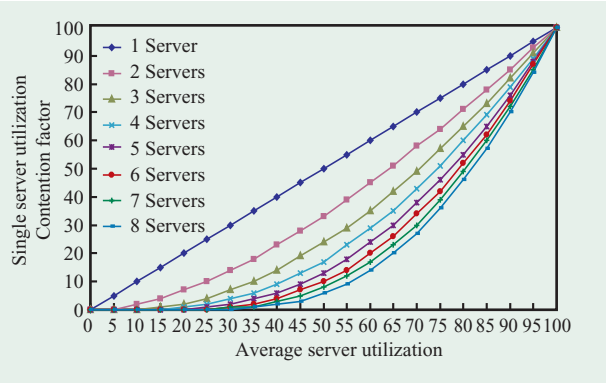

Figure 2

Equivalent contention factors for different numbers of servers.

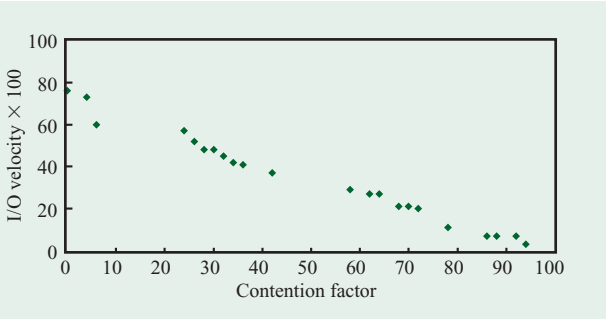

Figure 3

Historical data for estimating I/O velocities.

of interconnected subsystems and channels will not change, but if it adds CHPID 25, subsystems 1, 2, and 3 will be interconnected with CHPIDs 23, 24, 25, and 26. This is more complicated, and all other elements being equal, DCM would prefer the less complicated configuration.

Next, DCM projects the I/O velocity achieved by each of the possible changes. DCM also projects I/O velocities for any subsystems that are affected by the change. (An affected subsystem is any subsystem which shares a channel, either directly or indirectly, with the subsystem being helped.)

Finally, with each of the possible decisions evaluated, DCM considers all of the attributes and selects one to implement.

Balance mode vs. goal mode

DCM has two modes of operation, *balance mode* and *goal mode*.

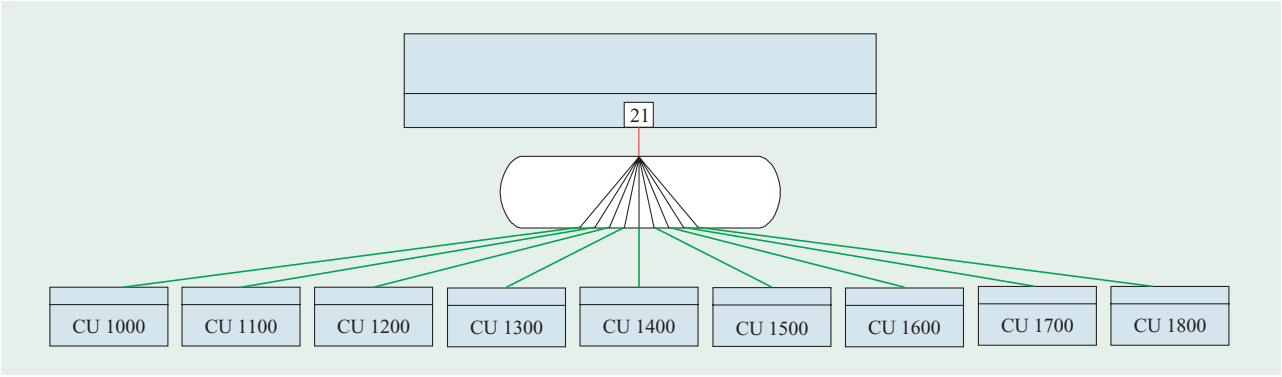

Figure 4

FICON-bridge-attached channel illustrating a configuration in which the bridge-attached channel is the limiting factor.

When WLM is operating in compatibility mode, DCM is said to be in balance mode. When WLM is operating in goal mode, DCM is also in goal mode. WLM does not have to be configured to use DCM. If managed channels and managed control units have been defined in the I/O configuration via HCD, and DCM is active, WLM will use DCM to help it achieve workload goals.[8]

In balance mode, DCM will determine the I/O velocity of all managed control units and calculate an average I/O velocity across them. This is then used as the target I/O velocity for all managed control units. DCM then scans all of the managed control units to identify any that are significantly below this target. If any are, DCM attempts to find a change it can make to the configuration that will move it closer to the target. DCM does this every interval, until all of the control units are within a certain tolerance of the target I/O velocity. When the system becomes overcommitted, the average I/O velocity may start to fall; when this happens, all of the control units are driven to what becomes an ever-decreasing target. Although this spreads out the available resource evenly, there may be business-critical work in the system that is using certain control units, while discretionary work is using other control units. In this case, it may be better to drive the control units being used by the business-critical work to a higher IOV at the expense of the control units being used by the discretionary work. This is where WLM goal mode is valuable.

When WLM is operating in goal mode, it uses the goals and relative importance of the different workloads running to set target I/O velocities for control units. If all work is meeting its goals, the I/O velocity is determined by DCM the way it is in balance mode. However, if WLM determines that work is not achieving its goals, and the reason is due to channel bandwidth, WLM can set an explicit IOV target for one or more of the workload control units. DCM then drives the control unit to the new target.

If there are several control units in the system with explicit IOV targets, and DCM is unable to achieve an explicit IOV target for one subsystem without adversely affecting another control unit with an explicit IOV target, DCM calls WLM so that it can decide which change, if any, to make on the basis of the relative importance of the work and the projected impact of the change to the affected workloads.

This is all done using the existing WLM policy definitions that current WLM goal mode users already have. DCM is just one more tool that it can use to achieve those goals.

Frequency of change

DCM considers changes to control unit definitions once every ten seconds. If there are multiple partitions which are members of the same sysplex, each partition considers changes every ten seconds. However, if one partition finds that another partition is already implementing changes, it skips this processing for this interval. During the interval, DCM attempts to help subsystems which are below their target velocity. Subsystems which have had channels added or removed from them are not eligible to have additional changes for another three intervals. This means that changes occur to a single subsystem at most once every 40 seconds.

The I/O velocity that is used to determine whether a subsystem is below target is actually a moving average of up to eleven intervals. This is done to dampen the subsystem I/O velocity and prevent frequent and unnecessary changes. In addition to the subsystem that is being helped, any subsystem that is affected by the change is also ineligible for changes for three intervals. An affected

[8] Although not required, it is recommended that the I/O priority management option in the WLM service policy be turned on.

subsystem is any subsystem which directly or indirectly shares a channel with the subsystem being helped.

FICON bridge channel support

In 1999 a new channel called the FICON converter (FCV) channel was introduced. The FICON converter (also known as the FICON bridge) channel can be attached via a FICON cable to an ESCON director which has the FICON converter feature. The FICON converter feature is a card placed in the ESCON director to handle the speed matching and buffering of I/O operations between a single FICON channel and one or more ESCON ports attached to control units. The FICON converter card contains eight internal processors that can support eight concurrent I/O requests. Since the capacity of the FICON link (shown in red in **Figure 4**) is several times that of the ESCON links on the attached control units (shown in green), several of the ESCON links can be active at the same time.

What this means to DCM is that, in contrast to a pure ESCON environment, several control units can be added to a FICON bridge channel before there is any significant degradation to control units already defined to the channel. For example, in Figure 4, if CU 1000 is the only control unit defined to FICON bridge channel 21, then if DCM were to dynamically add CU 1100, there would be virtually no impact on CU 1000, even if both control units were driving the ESCON port to its full capacity. Eventually, adding a control unit to the FCV channel will affect the other control units, and DCM will project this on the basis of the number of control units defined, the load on those control units, the other channels the control units are using, and the current FCV channel utilization.

Most modern control units allow themselves to be defined as multiple subsystems or logical subsystems, as shown in **Figure 5**. In cases such as this, if an additional subsystem were to be defined to the FCV channel, using the same port as an existing subsystem, there would be an effect to adding the new subsystem, even if the FCV channel were not nearing its capacity. This is because the control unit port would be the limiting factor.

Again referring to Figure 5, if control unit 2300 were defined to FCV channel 21 via port 01 (dotted line), control units 2200 and 2400 would be affected.

Channel subsystem priority queueing

Prioritizing I/O requests is not new for z/OS. When the system was known as MVS, I/O requests could be prioritized on device queues within the operating system. With the introduction of the Enterprise Storage Server* (ESS), WLM can also set priorities on I/O requests, which

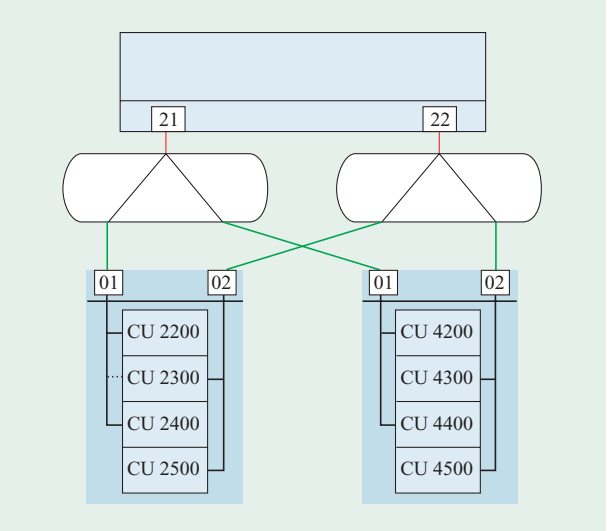

Figure 5

FICON-bridge-attached channel illustrating a configuration in which the control unit port is the limiting factor.

are then honored by the control unit. Channel subsystem priority queueing (CSSPQ) extends the ability to prioritize I/O requests by addressing one more place where queues can form, the channel subsystem.

WLM can set priorities on I/O requests, which are then used by the eServer z900 to schedule the work to resources within the channel subsystem. In this way customers may identify workloads that are the most critical, and z/OS will work with an eServer zSeries processor to allow the critical work greater access to channel subsystem resources. For an overview of the different kinds of I/O priority queueing, and how they differ from channel subsystem priority queueing, see Reference [6].

How it functions

The z/Architecture* specifies that I/O requests are passed from CPUs to the channel subsystem (CSS) by means of *start subchannel* (SSCH) instructions. These instructions are issued by the z/OS I/O supervisor (IOS) on behalf of applications or other system functions. These requests are copied into subchannel control blocks (SCBs), which are added to the tail of a particular work queue (WQ). Each SCB enqueued at the bottom of a work queue contains a field designating the I/O priority associated with this request. Since the CSS in zSeries 900 systems is implemented by one or more so-called system assist processors (SAPs), one WQ is associated with each SAP. The SAP is responsible for initiating I/O requests by dequeueing SCBs from the WQ and passing the request

Table 5 Overview of old and new algorithms.

A0.	*Old (base) algorithm without I/O priority queueing:*							
	each		time	take		**any**	**priority**	**top-of-Q**	element/request
A1.	*New I/O priority queueing algorithm—Option 1:*							
	each	fourth	time	take		**any**	**priority**	**top-of-Q**	element/request
	each	fourth	time	take	**first**	**highest**	**priority**	**initial**	element/request
	each	fourth	time	take	**first**	**highest**	**priority**	**redrive**	element/request
	each	fourth	time	take	**first**	**any**	**priority**	**initial**	element/request
A2.	*New I/O priority queueing algorithm—Option 2:*							
	each	second	time	take		**any**	**priority**	**top-of-Q**	element/request
	each	second	time	take	**first**	**highest**	**priority**	**any**	element/request

on to one or more channel paths designated to handle this request. If all of the channel paths controlled by a particular SAP are busy, the request is requeued at the bottom of the same or another WQ. Therefore, work queues contain not only requests (SCBs) just appended by any of the CPUs by means of SSCH instruction, but in addition SCBs which have been appended to the WQ due to an "all channel paths busy" condition. All other busy conditions (control unit busy, device busy, and switch busy) are not handled via WQs. It should be noted here that I/O priorities are observed only when an SCB is dequeued from a WQ.

I/O priority algorithm

Basic algorithm

The I/O priority mechanism is active when the corresponding global I/O priority switch is enabled via the system console. Otherwise, a strict first-in/first-out rule is observed which ignores the I/O priority value(s) specified by the program. The I/O priority mechanism must guarantee a reasonable fairness algorithm, so that on one hand I/O requests are handled according to their priority values, but on the other hand no I/O request will "starve" because of low(er) priority values assigned to it. Furthermore, the algorithm should be fairly simple in order not to impose too much latency during priority evaluation.

The algorithm works by first taking any (either initial or redrive) I/O request off the top of the work queue, then taking the oldest highest-priority initial request off the work queue, then the oldest highest-priority redrive request off the work queue, and finally the oldest initial request off the work queue, regardless of its priority. In doing so, this algorithm makes sure that I/O priority values as specified by software are well observed by the channel subsystem, but that starvation is totally avoided. The algorithm makes sure that

- High-frequency, high-priority requests do not monopolize the channels at the expense of others with lower priorities.
- High-frequency busy/redrive conditions do not block out initial requests (and vice versa). The alternate handling of initial or redrive requests is independent of any priority values.

While looking into design and implementation options, the basic idea was to change only one place, which deals with different I/O priority values. This is the channel subsystem interrupt handler (CSSIH), which runs in the SAP/IOP and selects all requests to be handled by the CSS on the basis of a given priority scheme. One of these requests is a signal work request (SIGW6) which asks the CSSIH to dequeue a subchannel (SCB) from the work queue header (WQH) for a start operation. The present implementation for start requests (without I/O priority queueing) dequeues the "top-of-queue" SCB and calls the appropriate service routine (start, halt, clear). This is the only place where a dequeue is done. Since the work queue is a FIFO queue, all other routines enqueueing SCBs do so by adding SCBs to the bottom of the queue. With the anticipated change, halt and clear requests (SIGW2) are not subject to I/O priority queueing, which means that they will continue to be handled in FIFO manner.

It was assumed that there was little room for highly sophisticated selection algorithms. If multiple queues or even sorted queues are not allowed, it is inevitable that the work queue must be searched—not necessarily every time—for SCBs with the currently highest priority on the work queue. **Table 5** presents the old and new algorithms for use whenever a subchannel control block (SCB) must be selected (dequeued) from the work queue.

Modeling the different algorithms

All three algorithms were modeled in a REXX program which could be executed on z/VM* as well as on OS/2*.

A minimum number of 5K to 10K start requests per simulation run were processed. With these numbers, multiple runs were executed which yielded acceptable mean values and standard deviations to make thorough decisions between the two algorithms and among various modifications of these two algorithms. The base algorithm, A0, did not require much attention because it does not attack the problem by nature.

The REXX model is a nondeterministic one. It generates and handles I/O requests on the basis of preselected probability values which are specified as parameters prior to the simulation run. Its main characteristics are the following:

- A preselected number of I/O requests (SSCHs) were generated—a typical number for short test runs was 1000, but more meaningful statistical results required 5000 or 10000. These requests were added to the work queue initially and either when the WQ ran empty or, in case of a non-empty WQ, with a preselected probability. The number of I/O requests added was a preselected parameter as well, allowing a significantly busy WQ to be generated to test the effectiveness of the different algorithms.
- Each I/O request being generated was given a priority between 0 and 7. The actual value chosen had an equal distribution between 0 and 7.
- At the point where the I/O request was being dequeued, the three different algorithms came into play. After selection of a particular I/O request, two preselected probability values determined whether or not this request could be successfully started (this simulated all channels busy on this SAP/IOP). The two different probability values were associated with initial or redrive requests, reflecting a possibly different behavior for initial/redrive requests for a given workload and I/O configuration.
- If the I/O request could not be successfully started, it was requeued again at the bottom of the WQ.
- After having the I/O request either started or requeued due to channel busy, a check was made to see whether or not the WQ had become empty, and if not, whether or not new I/O requests had to be generated.
- The process ended when the predefined number of I/O requests had been processed and the WQ had become empty.
- During the total simulation run, statistical information was gathered which was evaluated and printed at the end of the simulation run. This included information such as the number of simulation cycles, minimum/average/maximum number of cycles for I/O requests with a given I/O priority value being enqueued on the WQ, average WQ length being searched, and average WQ length total.
- All three algorithms were implemented in the REXX model. The one to be used in a particular simulation run could be selected by a specific parameter.

As indicated above, the simulation results of various runs using the same parameter settings differed considerably. Therefore, multiple runs were done for the most interesting parameter settings, and information such as mean values and standard deviation was calculated for the most interesting resulting figures in a separate spreadsheet. The REXX program, in particular the algorithm finally chosen, served as the base of the design for the actual changes to the CSSIH for the eServer z900 system.

Modeling results

The results of the modeling were as follows:

1. Both new I/O priority algorithms A1 and A2 showed significant improvement over the current FIFO algorithm A0. On average, high-priority requests were handled approximately three to four times faster than low-priority requests.[9]
2. Algorithm A2 showed slightly better results than Algorithm A1; however, we decided to select A1 for two reasons:
   - It is more robust in pathological cases (e.g., frequent redrives of high-priority requests while initial requests could probably be handled successfully on less-utilized channels).
   - If the SAP/IOP channel subsystem interrupt handler were to be replaced by an industry-standard real-time kernel on future systems, we would likely provide two separate work queues: one for initial requests (from CPUs) being handled on interrupt level and one for redrive requests (from the same or other SAPs/IOPs) being handled on task level.
3. Both I/O priority algorithms A1 and A2 provide a meaningful fairness algorithm, ensuring that low-priority requests do not starve.

How are the priorities set?

The I/O priority that is passed on the I/O request to the channel subsystem is set by WLM. There are no externals that a user can use to specify the priorities on I/O requests. Instead, WLM sets the priorities for I/O requests issued by a workload using this basic approach:

[9] *Note*: It would be easy to increase this factor by taking more high-priority requests than taking "any first" requests (as assumed in the "base" algorithm outlined above).

- System-related work is given the highest priority.
- High-importance work missing goals is given a higher priority than other work.
- Work meeting goals is managed so that lightweight I/O users have a higher priority than heavyweight I/O users.
- Discretionary work is given the lowest priority in the system.

In this way, CSSPQ can be regarded as another tool that WLM can use to help it achieve the goals for service classes as specified by the user.

Multiple logical partitions

By activating CSSPQ and then specifying an allowable range, the user can control how different partitions interact with it. If the user is running other operating systems on the same CPC that do not support CSSPQ (e.g., VM/ESA*, TPF, Linux, VSE/ESA*, and OS/390*), a default priority may be specified for these partitions.

For example, if the user has a VM partition that has work which is as important as the discretionary work in the z/OS partition, the user might set the VM partition to a priority range of (7–7), and on the z/OS partition set the priority range to (7–14). In this way, I/O requests from the VM partition are treated as equal to those of the discretionary work from the z/OS partition (as opposed to even lower if the priority is permitted to default to zero). Alternatively, the user can set the priority range for an OS/390 partition, running business-critical OLTP applications to (15–15), while the z/OS partition is set to (7–14). In this way, the OLTP work always has preference over the z/OS partition, but the z/OS partition can still make tradeoffs across the workloads it is managing.

The user does not even have to be running z/OS with WLM in goal mode to get some benefit from CSSPQ. If the user has three workloads, one of high importance, one of medium importance, and another of low importance, the priority ranges may be set, for example, to (8–8), (7–7), and (6–6) respectively, and they will be managed accordingly. However, they will not get the benefit of WLM monitoring service classes to their goals and adjusting the CSSPQ dynamically to help achieve those goals.

It should be noted, though, that partitions which are members of the same z/OS LPAR cluster should have all their ranges the same. In this way WLM is able to manage service classes across the cluster in a consistent way.

Dynamic channel path management

CSSPQ complements the other I/O portion of IRD, dynamic channel path management (DCM). DCM allows the system to dynamically redefine channel paths to control units in order to more efficiently use the channel resources that are available to the processor. The assignment of channels to control units by DCM is done with the cooperation of WLM, when it is running in goal mode. This allows DCM to give additional channel paths to control units which are being used by service classes that require such additional bandwidth to achieve their goals. However, these additional channel paths are available to all work in the LPAR cluster (even the low-priority work), so adding channel paths may not produce the desired results if low-priority work just consumes the added bandwidth. This is where CSSPQ helps. By assigning the more important work a higher priority, we are assured that it will receive greater access to the available channels.

FICON channel algorithms

CSSPQ is implemented completely differently for FICON channels. Unlike an ESCON channel, which rejects an I/O request if it is busy processing earlier requests, FICON channels have the ability to multiplex several I/O operations at one time. In fact, up to 32 channel programs may execute concurrently. As a result, queues are rarely created within the channel subsystem, so prioritizing them has little value.

Instead, the microcode within the FICON channel itself has been modified to honor the I/O priority. As more and more I/O requests arrive at the channel, they are sorted for execution in priority order. The channel achieves this by creating several queues and creating a circular queue of these queues, with a cursor that points to the queue currently being serviced. As requests are being serviced from the queue indicated by the cursor, the highest-priority information units (IUs) are entered into the next queue, the second-highest-priority IUs are entered into the queue after that, and so on down, with the lowest-priority work being entered into the furthest queue. When all of the entries in the current queue are satisfied, the cursor is changed to point to the next queue, effectively rotating the circular queue and making the queue with the highest-priority IUs the current queue. In this way, higher-priority I/O requests receive greater access to channel resources, but without starving the lower-priority requests.

Activating channel subsystem priority queueing

For a description of how channel subsystem priority queueing is activated, see Reference [6].

Concluding remarks

MVS[10] has always been an industry leader in managing multiple heterogeneous workloads, with various business priorities, in a single server. With the introduction of PR/SM, S/390 permitted a user to logically split physical resources such as processors, memory, and I/O channels

[10] MVS evolved into OS/390, which in turn evolved into what is now known as z/OS.

across multiple images. Each image could provide a complete, self-contained, computing environment, sharing, yet still isolated from, other images in the same processor.

When WLM was introduced in MVS 5.1, this management was taken to another level by permitting users to specify their performance goals in business terms. Instead of indicating to the system how much of each resource a workload should receive, with the hope that the user will obtain the desired responsiveness, goals are identified for the workload, such as those associated with response time, and the system adjusts the resource levels to achieve those goals. WLM also permits the user to specify the relative importance of workloads. In situations where the system is overcommitted, WLM uses this information in order to ensure that the user's most important work continues to achieve its goals, even at the expense of less important or discretionary work.

While other platforms attempt to duplicate this approach, z/OS again takes workload management to a new level. By establishing a more synergistic relationship between z/OS and the zSeries hardware with respect to the allocation of resources among logical partitions, WLM can now not only make changes or adjustments on the local sysplex member level and redistribute work across members of a sysplex, but it can also redistribute hardware resources among partitions. What this means to customers is that z/OS will continue to achieve the business goals of the workload in even more adverse situations than was possible in the past, and to achieve this while expending less systems management skill.

Acknowledgment

The authors would like to thank the reviewers of this paper for the insightful comments they provided. They would also like to thank Steve Hamilton for his constructive editing comments.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of The Open Group or Linus Torvalds.

References

1. IBM Corporation, *z/OS Intelligent Resource Director*, Order No. SG24-5952-00; see *http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245952.pdf.*
2. IBM Corporation, *z/OS V1R2.0 MVS Planning: Workload Management*, Order No. SA22-7602-02; see *http://publibfi.boulder.ibm.com/epubs/pdf/iea2w110.pdf.*
3. J. Aman, C. K. Eilert, D. Emmes, P. Yocom and D. Dillenberger, "Adaptive Algorithms for Managing a Distributed Data Processing Workload," *IBM Syst. J.* **36,** No. 2, 242–283 (1997); see *http://www.research.ibm.com/journal/sj/362/aman.html.*
4. IBM Corporation, *eServer zSeries 900 Processor Resource/Systems Manager Planning Guide*, Order No. SB10-7033-00; see *http://publibfi.boulder.ibm.com/epubs/pdf/b1070330.pdf.*
5. IBM Corporation, *z/OS V1R1.0 Hardware Configuration Definition User's Guide*, Order No. SC33-7988-00; see *http://publibfi.boulder.ibm.com/epubs/pdf/cbdzug00.pdf.*
6. IBM Corporation, *Hot Topics—A z/OS Newsletter*, Issue 4, February 2001, Order No. GA22-7501-00; see *http://www.s390.ibm.com/os390/bkserv/hot_topics.html.*

Received September 21, 2001; accepted for publication February 8, 2002

William J. Rooney *IBM Server Group, 2455 South Road, Poughkeepsie, New York 12601 (wrooney@us.ibm.com).* Mr. Rooney is a Senior Technical Staff Member in the zSeries Software System Design Department. He received a B.S. degree in computer science from the New York Institute of Technology in 1978 and an M.S. degree in computer science from Syracuse University in 1984. During his 24 years of service for IBM, he has worked on device allocation, software litigation, Parallel Sysplex prototyping, and porting of UNIX applications to OS/390 UNIX System Services. He is currently responsible for design of I/O functions, and is the lead designer for dynamic channel path management.

Jeffrey P. Kubala *IBM Server Group, 2455 South Road, Poughkeepsie, New York 12601 (kubes@us.ibm.com).* Mr. Kubala is a Senior Technical Staff Member in the LPAR Design and Development Department and is currently the technical team leader for the zSeries LPAR hypervisor. He received a B.S.E. degree in computer engineering from the University of Connecticut and joined IBM in 1981. Since then, he has worked on compiler design and development, OS/390 Hiperbatch, and S/390 and zSeries logical partitioning. In addition to his role as the zSeries LPAR hypervisor technical team leader, he is actively engaged with the iSeries and pSeries hypervisor teams as a technical consultant.

Juergen Maergner *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (jmaergner@de.ibm.com).* Mr. Maergner is a Senior Engineer, working on future I/O attachments for zSeries systems. He received a master's degree in electrical engineering from the Technical University of Berlin, Germany, and joined IBM in 1970. Since then, he has worked primarily in the area of channel subsystem design and I/O attachments for S/370, S/390, and z/900 systems. He is currently active in the standards community as chairman of both national and international standards committees in the fields of interconnection of computer systems and attached equipment.

Peter B. Yocom *IBM Server Group, 2455 South Road, Poughkeepsie, New York 12601 (yocom@us.ibm.com).* Mr. Yocom is a Senior Technical Staff Member in the WLM/SRM Design and Development Department. He received B.S. and M.S. degrees in computer science from Rensselaer Polytechnic Institute in 1985 and 1986, respectively. He subsequently joined IBM, where he has worked on the real storage manager, the I/O supervisor, and workload manager components of z/OS. He is currently the lead designer for the z/OS workload manager component.