z/OS support for the IBM TotalStorage Enterprise Storage Server

by A. S. Meritt
J. A. Staubi
K. M. Trowell
G. Whistance
H. M. Yudenfriend

The IBM TotalStorage™ Enterprise Storage Server® provides unique capabilities for eServer™ zSeries™ and S/390® environments. We describe two such capabilities, Parallel Access Volume and I/O Request Priority, and discuss the algorithms and mechanisms used to implement and manage them. We show how the new functions deliver significant value to customers, particularly in the areas of self-optimizing management of resources tied to customer goals and reduced customer configuration planning.

The IBM TotalStorage* Enterprise Storage Server* (ESS) was introduced in 1999.[1] Although ESS can be used with other operating systems, we focus here on its new capabilities that are supported by the software for the eServer* zSeries* and S/390* environments (for brevity, we use "zSeries" to refer to both the eServer zSeries and the S/390 environments). Many of these are latent capabilities and provide no value to the customer by themselves; software support is required in order to take advantage of these functions. We describe in this paper two ESS functions, Parallel Access Volume (PAV) and I/O Request Priority (IORP), as well as the related capability, multiple allegiance (MA). Although MA does not require software support, it is required for PAV and interplays with both PAV and IORP. We discuss z/OS* support for managing these capabilities, and in particular we describe the algorithms and metrics used by the z/OS Workload Manager (WLM).[2]

The workload to be processed consists of various types of work, with various completion and resource requirements. With WLM, the administrator defines performance goals and assigns a business importance to each goal. The goals for work are specified in business terms, and the system determines the resources, such as CPU and storage, to be dedicated to this task in order to meet its goal. WLM continuously monitors the system and adapts to changes in workload and configuration in order to meet the specified business goals.

The support for PAV and IORP is designed not only to deliver better performance, but also to support the requirements associated with each workload, consistent with the philosophy of WLM. The z/OS functions are designed to enhance the self-configuration and self-tuning properties of the system according to the autonomic computing vision.[3]

The remainder of this paper is structured as follows. In the next section we give an overview of the three main capabilities that are the focus of this paper: MA, PAV, and IORP. In the section that follows we describe z/OS support of aliases, the key mechanism that makes PAV work. Then, two sections are dedicated to describing, respectively, WLM dynamic alias management, and WLM I/O priority management. Next we describe ESS I/O queuing and priority management, and then we discuss some field experience that shows the benefits of the capabilities introduced above. We summarize our work in the final section.

0018-8670/03/$5.00 © 2003 IBM

Overview

Historically, disk storage was based on physical disk drives that could process only one I/O request at a time.[4] Moreover, in an environment with multiple hosts and shared volumes of data storage, access to a volume was limited to at most one I/O request at a time *from all hosts*. This resulted in the software in a system queuing I/O requests for a volume whenever *any* host had one request active; this is the normal operating mode when the device can only do one thing at a time. This scheme works in multisystem environments with shared volumes, and operates without the need for direct communications between systems.

The introduction of cache controllers and RAID (redundant array of independent disks) changed this, allowing the simultaneous processing of multiple I/O requests. With the single-threaded operation described above, a request that encounters a cache miss in the control unit continues to tie up the volume until it is completed. Other requests for the same volume, from the same system or other systems, whose data may be resident in the cache, cannot start because the volume remains in use.[5]

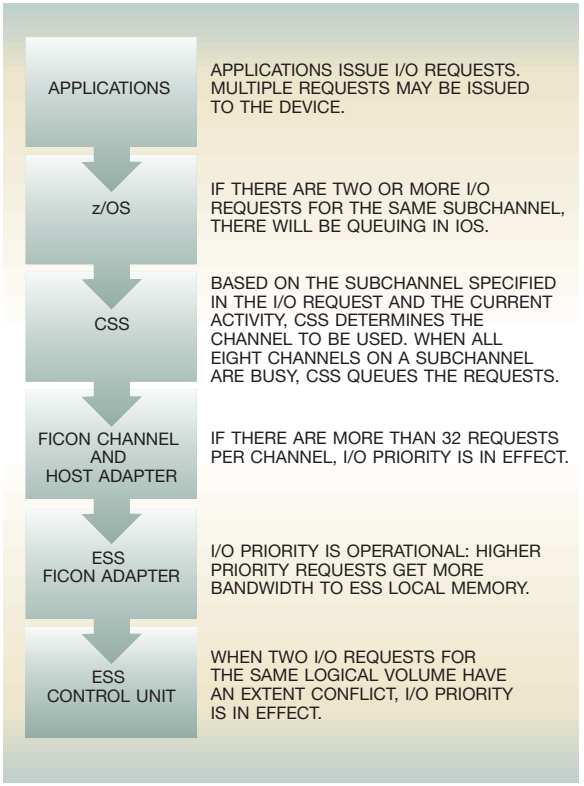
RAID breaks the one-to-one association of volumes with devices. A *logical volume* is now the addressable entity presented by the controller to the attached systems. The RAID unit maps the logical volume across multiple physical devices. Similarly, blocks of storage on a single physical device may be associated with multiple logical volumes. Because a logical volume is mapped by the RAID unit across multiple physical devices, it is now possible to overlap processing for multiple cache misses to the same logical volume, because these can be satisfied by different physical devices.

Multiple allegiance. Multiple allegiance (MA) is the capability to support I/O requests from multiple systems—one per system—to be concurrently active against the same logical volume if they do not conflict with each other.[1] Conflicts occur when two or more I/O requests require access to overlapping extents (an extent is a contiguous range of tracks) on the volume, and at least one of the I/O requests involves writing of data. Requests involving writing of data can execute concurrently with other requests as long as they operate on nonoverlapping extents on the volume. Conflicting requests are internally queued in ESS. Read requests can always execute concurrently regardless of their extents. Without the MA capability, ESS would generate a busy indication for the volume whenever one of the systems issues a request against the volume.[5] This would cause the I/O requests to be queued within the channel subsystem (CSS).

Parallel Access Volume. Before PAV (Parallel Access Volume) was available, the operating system allowed only one request at a time for each volume. Thus, when there was an active I/O request to a volume, its UCB (unit control block) was flagged as busy. In order to support PAV, there are multiple unit addresses associated with the same logical volume, and each such address is associated with a corresponding subchannel within the zSeries CSS.[4] These additional unit addresses are known as *PAV aliases*, or simply *aliases*. Thus, a PAV volume is represented by a *base* address and possibly one or more aliases. Because the zSeries I/O architecture permits a unit address and its associated subchannel within CSS to handle only a single request at a time, PAV supports multiple concurrent I/O requests from the same system against the same logical volume.

Aliases can be associated with logical volumes in two ways. With *static* assignment of aliases, the administrator specifies the aliases to be associated with a particular volume; this assignment remains operational until a reconfiguration is manually performed (static aliases were supported starting with OS/390* Version 1 Release 3). Static aliases work well for stable, well-defined workloads, including most performance benchmarks. But modern workloads are becoming more and more dynamic, with less predictable access patterns, in which case dynamic management of aliases is preferred. (The WLM dynamic alias management function is supported starting with OS/390 Version 2 Release 7 and all z/OS releases.) With dynamic alias management, the alias addresses are managed as a pool, and the aliases in the pool are available for allocation to volumes, rather than being statically associated with specific volumes. The number of aliases associated with any particular volume is dynamically adjusted by WLM either to achieve the workload goals, or to improve overall efficiency when all workloads are achieving their goals. WLM manages aliases as a resource; as work shifts dynamically from one volume to another, the aliases needed to handle that work are also dynamically shifted. This reduces or eliminates the need to manually relocate data sets in order to manage contention, especially in combination with dynamic channel path management (DCM) and the Intelligent Resource Director (IRD).[6]

Figure 1 I/O request flow within z/OS on zSeries and ESS with FICON channel



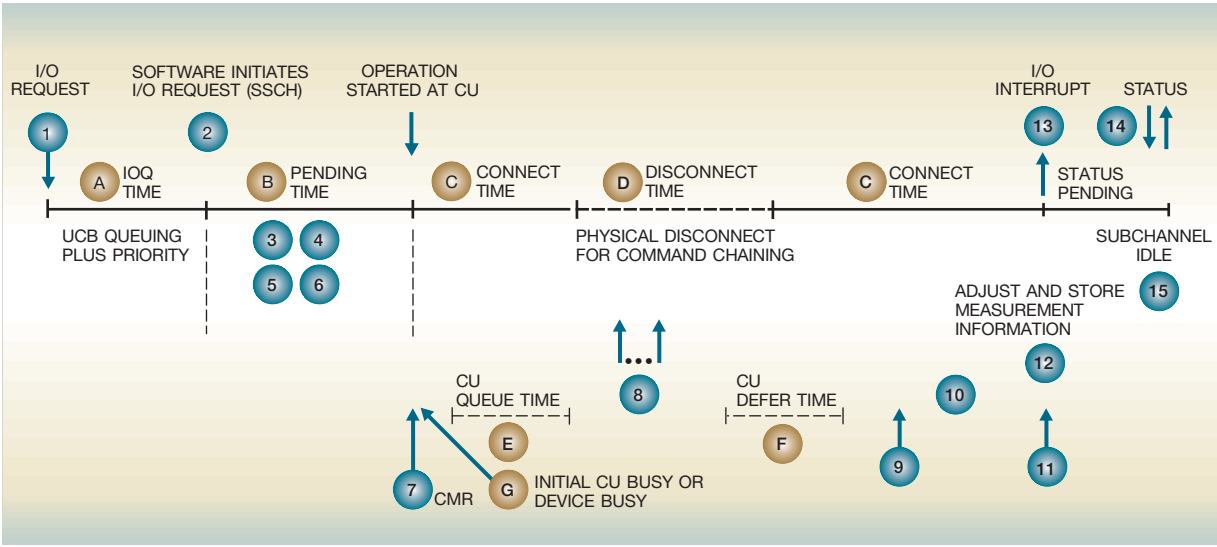
I/O request priority. IORP is a means for associating a priority with each I/O request. With PAV, ESS architecture allows more work to be sent to it. Indeed, there are usually multiple physical interfaces between each system to ESS. The FICON* (Fiber Connection) channel also allows multiple I/O requests to execute concurrently on the same physical interface.[7] (Fibre Channel Connection Architecture is a channel-to-control-unit interface architecture based on the Fibre Channel standard.[8])

The priority value is used when ESS is dealing with internal queuing; this occurs for cache misses and for extent conflicts. For those cases, IORP is used when selecting requests to be serviced that are no longer in conflict, or when scheduling cache miss activity. It is also used when selecting requests for access to previously busy resources, such as reconnecting on ESCON* (Enterprise Systems Connection) interfaces, which support only a single active request. (S/390 Enterprise Systems Connection architecture is a channel-to-control-unit-interface architecture.[9]) FICON interfaces support multiple active I/O requests and also use priority. ESS can take advantage of the multiplexing capability of FICON by immediately processing requests according to their priority when the cause of the queuing has ended; that is, data have been staged following a cache miss, or an extent conflict is resolved. Priority also affects data transfers on FICON; ESS uses the priority value to manage its data transfer queue on FICON. Higher priority requests get more of the link bandwidth than lower priority requests. In addition, with FICON interfaces, MA and PAV permit higher priority requests to continue to be sent to ESS from multiple systems or a single system, respectively, beyond the number of physical interfaces available. In this case, priority is used by ESS when workload contention exists between the ESS FICON adapter and the ESS control unit.

Figure 1 illustrates the flow of I/O requests in a zSeries machine with z/OS, FICON, and ESS. Multiple applications within an operating system image may issue I/O requests concurrently. The I/O supervisor (IOS) component of z/OS queues the work for each target logical volume in order of priority. As many of these requests are started as there are PAV aliases available. They are initiated by sending them to the CSS.[4] The CSS will order them by priority on its internal work queue. Once an available channel path is found, the I/O request is sent to it, where again it is started in priority order. The I/O requests are transported through the I/O fabric (e.g., Fibre Channel[7]) to the control unit where the higher priority work is allocated greater portions of the bandwidth available on the FICON channels.

With the ability to push more work out to ESS, there is less opportunity for queuing within the software, and thus less opportunity to exploit priority. Instead, z/OS with ESS now provide end-to-end management of I/O priorities, coordinated across the multiple systems of a sysplex. Priorities are now used in the operating system, CSS, the FICON channel, and ESS; the priority values specified for these do not have to be the same (IORP in ESS is supported starting with OS/390 Version 1 Release 3 and in all z/OS releases). IORP provides a means for the software to associate a priority value with I/O requests passed to ESS. This priority value has a larger scope than the priority value used earlier; the scope of the earlier priority value was a single volume within a single operating system, whereas this priority value is global, spanning all volumes and including requests from all attached systems.

Figure 2 I/O processing metrics with zSeries, ESS, and FICON channel



Monitoring I/O operations. Figure 2 shows the sequence of events that typically occurs during the execution of an I/O operation. The lettered entities represent the various metrics associated with I/O operations. The numbered steps in the figure are detailed below (the term "control unit" below refers to ESS).

1. An I/O request is issued by an application or subsystem. The request arrives at the IOS and is queued at the (PAV-base) UCB representing the logical volume.
2. The SSCH (start subchannel) instruction is issued against the first subchannel that becomes available. An I/O priority is passed to CSS with the I/O request for use by CSS, channel, and control unit.
3. The I/O request is received by CSS and placed on a work queue with the assigned I/O priority.
4. A channel path that has available resources to execute the I/O request is selected. The channel will execute higher priority requests ahead of lower priority requests.
5. The channel begins processing of the channel program by opening an outbound exchange.[7] The request may encounter contention delays in the switching fabric.
6. The channel program is streamed out to the destination control unit using the channel program pipelining feature of the FICON protocols for up to 16 information units[7] (IUs).
7. The control unit acknowledges that the first command has begun execution by sending back a command response (CMR) to the first command.
8. For read commands, data begin to arrive from the control unit.
9. The IU pacing function of FICON allows the control unit to use CMR to signal the channel that the channel program has reached the point where the next set of IUs should be sent to the control unit.
10. The channel responds to the IU pacing CMR by sending another eight IUs.
11. The channel program execution is complete and ending status is sent from the control unit to the channel.
12. CSS receives the ending status from the device, makes the appropriate final calculations, and stores all the measurement data into the appropriate fields.
13. The I/O interrupt is presented to the software and the subchannel/UCB changes to the status-pending state.
14. The software retrieves the ending status via the TSCH (test subchannel) instruction.[5]
15. The subchannel/UCB changes to the idle state.

The measurements captured during the execution of the channel program are as follows.

- *IOSQ (A)* is the time that an I/O request is queued in software waiting for the subchannel associated with the device to become idle.
- *Function-pending time (B)* is the time interval between the acceptance of the start function (or resume function if the subchannel is in the suspended state) at the subchannel and the acceptance of the first command associated with the initiation or resumption of channel-program execution at the device.[5]
- *Device-connect time (C)* is the sum of the time intervals in which the device is logically connected to a channel path while the subchannel is active and the device is actively communicating with the channel path.
- *Device-disconnect time (D)* is the sum of the time intervals in which the device is logically disconnected from CSS while the subchannel is subchannel-active. The device-disconnect time also includes the control-unit-defer-time intervals reported by the device during the I/O operation.
- *Control-unit-queuing time (E)* is the sum of the time intervals measured by the control unit in which the device is logically disconnected from CSS while the device is busy with an operation initiated from a different system.
- *Control-unit-defer time (F)* is the sum of the time intervals measured by the control unit in which the device is logically connected to CSS during an I/O operation but is not actively communicating with the channel because of device-dependent delays in channel program execution. Control-unit-defer time is subtracted from the device-connect-time measurement and is added to the device-disconnect-time measurement reported for the operation.
- *Initial device or control unit busy (G)* is the time spent waiting for device and control unit busy to subside and is accumulated as part of function-pending time.

Prior to PAV, multiple requests for a volume from a single operating system would result in the OS queuing all requests after the first one was scheduled to the volume; the time that requests were queued within the operating system was reported as I/O supervisor queue (IOSQ) delay in Resource Measurement Facility[9] (RMF*) reports. PAV extends the ability to push the work requests for that volume out to ESS rather than leaving the work queued within the operating system.

The queuing time spent by an I/O request within the CSS is reported as pending time (PEND) in reports from RMF or other performance monitors. ESS will subsequently generate a no-longer-busy indication, which causes CSS to reinitiate the request. MA effectively eliminates the queuing in the CSS that resulted from multisystem contention against a volume. If requests do conflict, they are queued within ESS; this represents a shift in PEND delay from the CSS to ESS and is reported back as control-unit-queuing time.[4,9]

These device measurements, such as PEND time and control-unit-queuing time, are now critical for workload management. Over the years S/390, and now the zArchitecture*, has built into the system a number of features that have allowed efficient management of multiple workloads. The z/Architecture features for I/O include the channel path measurement facility (CPMF), which gathers performance data on channel path resources, and Channel Monitoring Mode, which allows, through the creation of channel measurement blocks (CMB), the collection of data on I/O resource usage and I/O contention at the individual device level. In addition, RMF provides detailed reporting on other I/O-related statistics and resource contention for capacity planning and problem analysis. These facilities also support accurate accounting and billing by tracking the consumption of I/O resources. WLM has been frequently upgraded in order to exploit the zSeries I/O instrumentation capabilities. The current objective is to create systems that are self-tuning, require fewer specialized skills to perform installation planning and configuring, and optimize the use of I/O resources. The measurement facilities in zSeries provide the metrics needed in order to implement these autonomic capabilities.[3]

Benefits from PAV and IORP. The queuing of requests in CSS and software, reflected in PEND and IOSQ, is an indication of the maximum throughput achievable through storage subsystems. With ESS, those queues have been substantially reduced or eliminated. MA and PAV break the serialization of access to a volume, permitting the simultaneous processing of requests by ESS. If there is no conflict, these may execute concurrently rather than serially; if there is a conflict and the requests are queued within ESS, IORP determines the order in which these requests are serviced. Total response time is reduced for requests that do not conflict because they execute concurrently rather than serially. Response time is reduced even for requests that do conflict and thus execute serially, because there is less overhead when ESS accepts and queues the request, rather than having the software queue the requests. Providing shorter response times improves productivity, per-

mits batch work to be completed in less time, provides for better handling of peak or unpredictable I/O workloads, reduces the need for performance tuning by reducing or eliminating I/O bottlenecks, and reduces the need to replicate data for performance reasons.

Consider the case of heavily updated DB2* databases. In DB2 applications, deferred writes or checkpoint processing can cause multiple requests to be queued within software, driving up IOSQ time. DB2 accumulates changed data in its buffers until the deferred write threshold is reached, at which time many random updates are written to the databases. In the past, when these long-running writes were performed, they serialized access to the volume from all attached systems. If a new transaction needed to read data, even data in cache, it may have been delayed by writes that were already active or queued. These delays showed up as queuing time in an RMF report. IORP allows the read requests to get to the head of the queue, and PAV makes it possible to start the read I/O on the first available alias, even if writes are taking a long time, and MA permits the read to start even if the writes are occurring from another system. Thus, PAV ameliorates the impact some DB2 utilities have on the transaction response time, for example, utilities that run concurrently, such as copies and database reorganization.

z/OS PAV support

The z/OS support for PAV aliases includes how they are defined and initialized and the mechanisms for dynamically changing the association of aliases with base volumes; these are performed by the IOS component of the z/OS operating system. The algorithms and metrics that utilize these mechanisms will be discussed in the section "*WLM dynamic alias management*."

Aliases are not directly visible to applications, middleware, or even most system components that perform I/O. This is a critical aspect of the design because it permits the system to change the association of a base to an alias in a way that is transparent to users of the volume; these users are only concerned with accessing the data, not with the mechanisms that enable such access.

Defining the I/O configuration. zSeries systems with ESCON or FICON require that the I/O configuration be defined to the processor and the operating system.[8] This step allows the administrator to control and customize various features. User-friendly names can be associated with I/O resources for the purpose of access control, resource monitoring, and event reporting. Security policies can be enforced by limiting the resources accessible by the machine, the logical partition (LPAR), and the operating system. Bandwidth can be managed by designating which I/O adapters may be used by which control units and I/O devices.

Aliases and their dynamic association with base volumes are transparent to applications.

The definition methodology was extended to include PAV-related features such as specifying base and alias addresses. An attempt was made to preserve the consistency with existing concepts and tools, to minimize additional work for the administrator, and to allow multisystem configuration changes without impairing the accessibility of the volume to any running applications. Therefore, the association of aliases with base volumes is not explicitly defined by the administrator. Instead, base volumes and aliases are defined, and the operating system dynamically discovers the relationships between the base volumes and their aliases, as follows.

First, the administrator defines two new logical device types to the operating system, base devices (e.g., 3390B) and alias devices (e.g., 3390A). These definitions must match the configuration customization of the ESS setup by the StorWatch* ESS Specialist.[10] (The StorWatch ESS Specialist is a Web application that provides the interface for customizing and controlling ESS.) A logical subsystem (LSS) is a logical structure internal to ESS that consists of up to 256 addresses to be used for base volumes and aliases.[1] An alias can only be associated with a base within the same LSS. ESS supports up to 16 logical subsystems for zSeries. Bases and aliases are managed within the scope of each LSS. Thus, the administrator is designating a certain number of devices based on the amount of data that needs to be held by the LSS—base volumes are currently limited to about 27 GB.

Next, the customer designates the number of aliases needed for accessing those base devices within each

LSS. Each alias device allows an additional concurrent I/O request to be issued to the base with which the alias is currently associated. To the software, they appear as distinct UCBs that attach to their own subchannel. The CSS is not aware of any distinction between base devices and alias devices. The ability to determine the number of aliases to use is analogous to the ability to specify the queue depth for command tag queuing for Small Computer System Interface (SCSI) attached devices[11] (attached via the Fibre Channel Protocol, FCP).

Initial assignment of aliases. The initial setup using the StorWatch ESS Specialist designates the number of alias devices for each base device within the LSS. z/OS discovers the specific association of PAV alias devices to PAV-base devices only after ESS undergoes its initial microcode load; subsequently, the association may be dynamically changed numerous times by different z/OS systems. Therefore, the customer has no practical way of specifying any specific binding to z/OS; z/OS must be able to discover any association that may exist. The discovery of the existing binding of aliases to their respective base devices is performed during system initialization.

A base device is initialized by the system like any other on-line disk device; the same processing also occurs when a base device is varied on-line after system initialization. An additional step, however, is required for base devices: a command is issued to ESS informing it that the device is a PAV device. When alias devices are initialized, there is no association between an alias and a base in the software configuration definition; the association must be discovered. Self-description information is read for each alias defined to the system, to allow the software to determine to which corresponding base the alias is bound.[12] All zSeries devices that attach to ESCON and FICON interfaces are required to support the device self-description architecture.[12]

Once system initialization is complete, all alias devices should be bound to their corresponding base device, as shown in Figure 3. Each base and its set of bound aliases can be thought of as a single logical unit to which I/O requests are initiated; a request to start I/O to a PAV volume will cause IOS to initiate the request to the base or any of its bound aliases.

The middle column in Figure 3 shows the subchannels that have been defined for the configuration and reside in the machine storage within the channel subsystem. These subchannels contain the physical addressing information that is used by the system to communicate with a specific unit address in ESS. For each alias UCB in the operating system (e.g., alias UCB 2077), the configuration data are read from ESS for that unit address, and the information is used to bind the alias UCB to the correct base UCB (see the section "Binding an alias to a base"). The right column in Figure 3 shows a zSeries-oriented LSS in which a UA mapping from base and alias device UAs to the corresponding logical volume (e.g., V1) is illustrated. (ESS LSSs can be either zSeries-oriented, denoted by ECKD* (extended count key data), or SCSI-oriented.) The software issues a read configuration data command to ESS (bottom blue arrow in Figure 3) and receives a configuration data record that contains enough information for the operating system to determine the base devices with which the alias device should be associated.

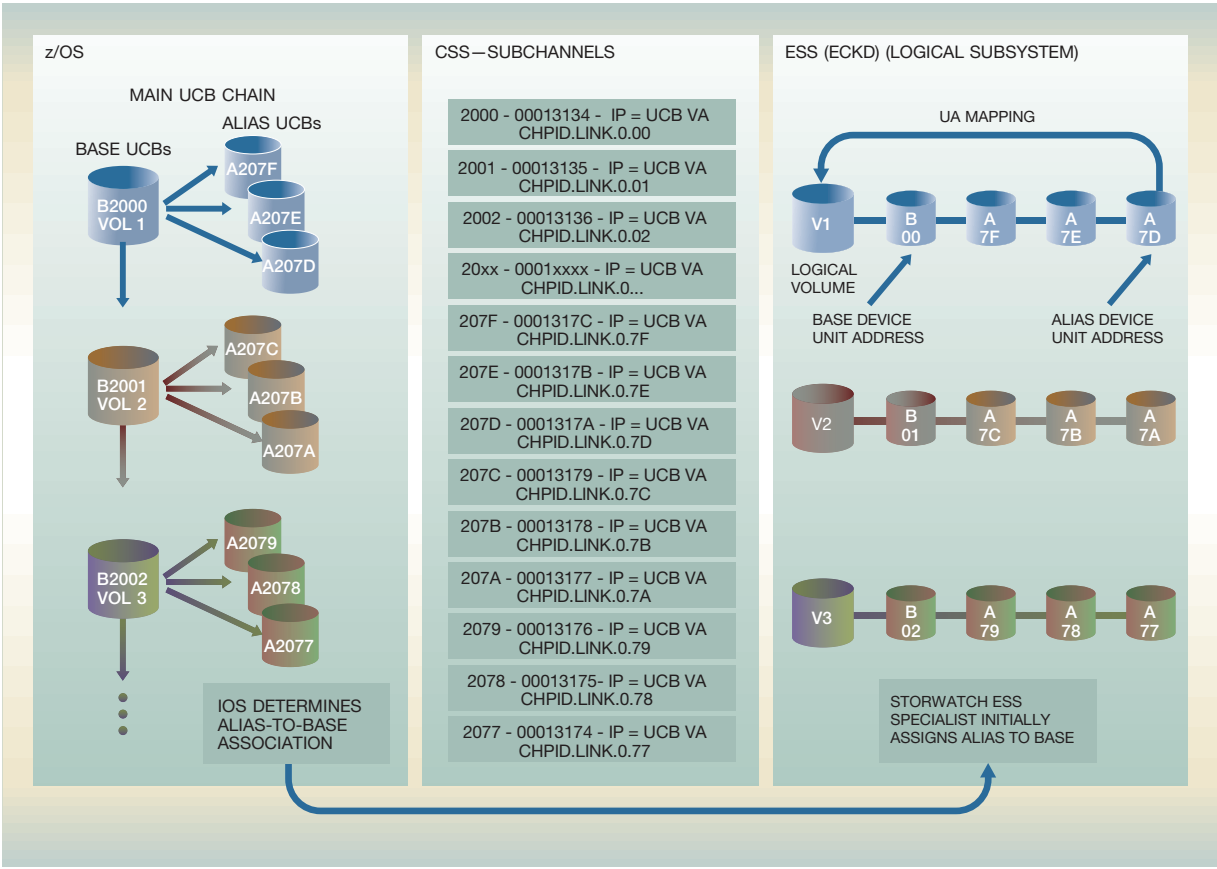
Binding an alias to a base. Device self-description was originally created to allow the operating system software to validate that the physical configuration matched the configuration definition in the CSS, in order to guarantee data integrity. This zSeries feature in combination with the reset event architecture[4] allows for the real-time notification of configuration changes and the checking needed to make sure that errors are not introduced into the configuration while the device is in use.

The operating system also takes advantage of the device self-description data to dynamically discover which alias devices currently bind to a corresponding base device to form a logical volume. The data structure corresponding to a logical volume is used to access the data on the device, to report errors, and to aggregate measurement data for accounting, hot spot analysis, capacity planning, and WLM dynamic alias management.

When IOS initiates self-description processing for a device, the read configuration data (RCD) command is issued to retrieve the configuration data record[12] (CDR) of the device. For an alias, this contains the unit address of the base with which this alias should be associated.

To bind the alias to its base, the system must first find the correct base by examining the CDRs for base devices in the same LSS as the alias, and by matching unit address fields. After the correct base has been found, the alias is bound to the base by adding the alias to the chain of alias control blocks associated with this base. Because the alias represents the same

Figure 3 PAV initial assignment of aliases



volume as the base, other information from the base is also copied to the alias control block, such as current path status.

In addition, timestamp information is updated so WLM knows when an alias was last bound to a base. Notification is provided for monitor programs, such as RMF, that an alias has been bound to a particular base, since this affects how they accumulate and aggregate performance information.

The alias devices are transparent to applications and middleware, which always allocate to the base device. The transparency of alias devices affords the operating system the flexibility to use whatever resources it chooses to keep track of state information in the most efficient ways possible. For example, in z/OS, devices are represented to the software by a UCB. Some applications are sensitive to whether the UCB has a 3-digit device number representation, or a 4-digit device number representation. Additionally, some applications require the UCB to reside in 24-bit addressable system storage instead of 31-bit or 64-bit storage. Because alias UCBs are in general not visible to the applications, the customer can choose to use 4-digit device numbers and 31-bit storage to minimize impact on system constraints. In fact, PAV devices allow the customer to combine many small volumes (e.g., the 3GB 3390 Model 3) into a larger volume (the 3390 Model 9 has up to 27 GB) with multiple aliases. The result is to consume less 24-bit virtual storage (one UCB for the base).

Bind events. After system initialization, bindings can be done in one of two ways. PAV devices can be dynamically added to the configuration. Then, the bind process described earlier is initiated immediately in order to enable the use of the new PAV alias.

Alternatively, WLM can manage the alias devices. WLM interfaces with IOS in order to gather information on PAV aliases, which it then uses to determine what PAV bases will benefit from additional alias devices. IOS is instructed to transfer a bound alias from one base to another, or attach an unbound alias to a base, in order to react to workload needs.

Unbinding an alias from a base. Unbinding an alias from a base removes the association between an alias and a base. Following the execution of an unbind for an alias, IOS no longer sees that alias associated with any logical volume. When WLM is managing aliases, this alias may be transferred to another base, as needed.

A request to unbind a base causes all aliases currently bound to that base to become unbound. When unbinding an alias from a base, the alias UCB is removed from the UCB chain of aliases associated with this base. In addition, timestamp information is updated so that WLM can determine when an alias was last unbound from a base. In addition, notification is sent to monitor programs, such as RMF, that an alias has been unbound from a particular base, so that the monitors stop including this alias with the performance information for the base.

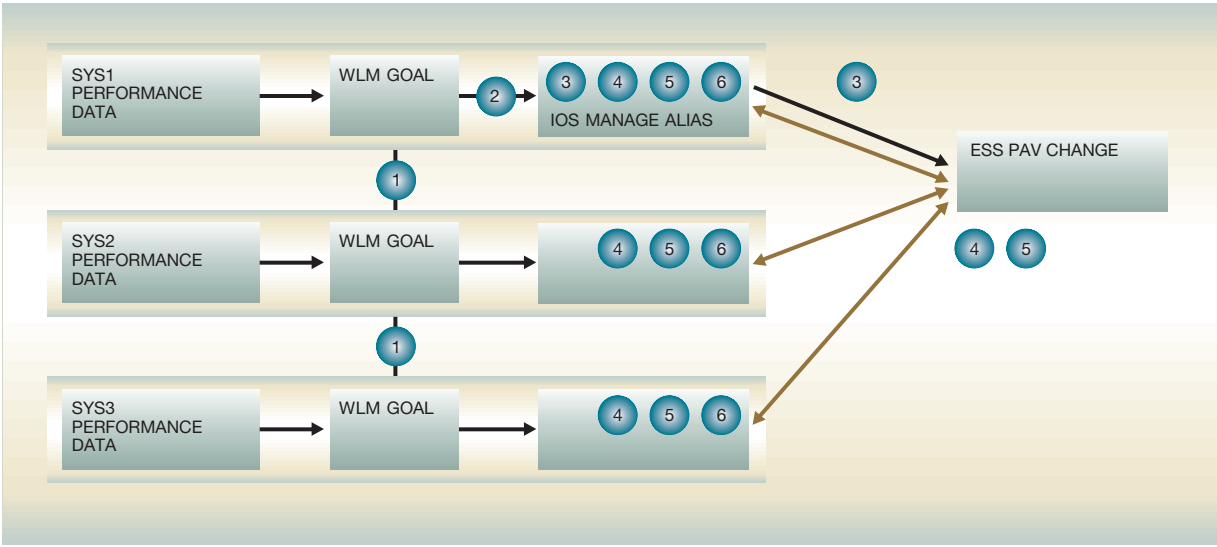
Unbind events. Unbinding can occur either when a base device is taken or forced off-line (by the operator or by the system), or when WLM manages alias devices. Depending on workload needs, a bound alias may be transferred from one base to another, or an unbound alias may be bound to a base. In the course of transferring a bound alias to a new base, the alias will make a transition through an unbound state before IOS binds the alias to the new base.

Dynamic changes to PAV devices. The alias-to-base relationships can be dynamically changed in one of two ways. First, the ESS Specialist provides an interactive Web-based user interface for ESS customization, where the alias-to-base relationships can be altered.[10] This can be done while the logical volumes are in use; this is a critical requirement to avoid having to shut down the application and thus impact the users. Whenever an alias is transferred from one base to another, ESS first quiesces the I/O activity to the alias. If there is an active request to the logical volume through the alias, ESS waits for the request to finish. After the alias is idle, further requests to access the alias are held in abeyance by ESS. Next, the alias is attached to the new base. ESS notifies all sharing systems that a change has occurred for this alias by presenting an unsolicited device-state-transition (DST) interrupt[7,13] (a special status combination); this also flushes any requests against the alias that ESS held in abeyance. This unsolicited interrupt is ensured to be presented to each sharing host before the alias will accept an I/O request from that host, in order to ensure that all sharing systems are aware of the change regarding the binding of this alias. Upon acceptance of the DST interrupt, the operating system is responsible for quiescing the I/O activity to the logical volume until the device self-description data can be reread to determine the current alias binding. If the alias binding has changed, the alias is unbound from the base (logical volume) and then re-bound to the correct logical volume (base UCB). This synchronizes this operating system's view of the alias-base relationship with ESS when changes may have been initiated by other systems or the ESS Specialist.

All events that could indicate the loss of this interrupt are visible to zSeries operating systems. These events result in the operating system assuming that a DST may have been lost and cause the reverification of the alias-to-base bindings. Finally, as a last safety measure, each I/O request to the logical volume includes a token representing the base device for the logical volume. ESS verifies that the z/OS view of the base is consistent with the current base association in ESS. No I/O request will be processed when the token provided by software does not match the actual state. If the software token does not match, the request is rejected, indicating a mismatch has occurred. The operating system reacts to this by rebuilding the association of this alias to the base. Failure to recognize when base-to-alias bindings have changed results in data corruption.

The alias-to-base relationships can also be dynamically changed, under the control of the system software. This is the mechanism used by WLM dynamic alias management. WLM, based on workload requirements, interfaces with IOS in order to either transfer a bound alias from one base to another, or bind an unbound alias to a base. Once a transfer request is honored, the Manage Alias command is issued to instruct the ESS to transfer an alias from one base to another. When the Manage Alias command is issued from a system, steps are performed by ESS similar to the case in which an alias transfer was initiated through the ESS Specialist. Thus, I/O activity is held in abeyance until the alias is idle; systems sharing the device are informed that the alias has changed state, including the initiator of the Manage Alias

Figure 4 WLM dynamic alias management in a sysplex enviroment



command. The appropriate systems are notified of the change so that they can resynchronize their tables representing both the original and new base volumes. Figure 4 summarizes the WLM dynamic alias management process.

Shown in Figure 4 are steps 1 through 6 as follows:

1. WLM detects that a service class is missing goals because its I/O requests are suffering IOS queue delay. One WLM uses its sysplex-wide view of devices and its local view of service class performance to decide to move a PAV alias to another PAV base.
2. Only one system (top WLM) requests that the local IOS move a PAV alias from a donor base device to the required receiver.
3. IOS issues the Manage Alias I/O-command request to the ESS to move the alias from one base to another.
4. Before any new I/O requests can be executed by the alias, ESS informs all the systems that have aliases enabled that the PAV alias is associated with a new PAV base.
5. Each IOS obtains the current configuration data of the alias device.
6. Using the current configuration data, each IOS updates the software configuration in order to associate the PAV alias UCB with the correct PAV base UCB.

Initiating an I/O request. When an application issues an I/O request to a volume, IOS may direct the request either to the base or to any alias bound to the base. Certain conditions may temporarily preclude the use of aliases for I/O requests: during processing of a device reserve or release, during various I/O recovery actions, or when the device state indicates that an alias association with the base may have been changed (a DST interrupt is being processed, as previously described). In order to initiate an I/O request, IOS scans the base and its list of bound aliases and looks for the first alias that is not currently active. If such an alias is found, priority values are assigned by WLM—including the priority value to be sent to ESS for the I/O request and the priority value to be sent to CSS.[6] The start subchannel (SSCH) instruction is issued to the alias subchannel.[4] If an I/O request cannot be started (e.g., all the aliases are active), the request is queued on the IOS queue—this queuing is tracked in IOS queuing time (IOSQ) and used by WLM as a metric for changing alias associations.

WLM dynamic alias management

The z/OS Workload Manager (WLM) can be used to dynamically manage the assignment of alias devices to base devices and thus ensure the concurrency in I/O processing provided by PAV is available where it is most needed.[2] This section describes the current

implementation of the WLM algorithms for dynamic alias management with ESS.[14]

ESS supports up to 16 LSSs for zSeries[1]; the combined number of alias and base devices in each LSS is limited to 256. The pool of alias devices for an LSS is managed by WLM. Because aliases can be viewed as resources within ESS shared by multiple systems, the WLM management of aliases is at a sysplex level.[15] The WLM policy specified by the administrator determines whether dynamic management of aliases for a z/OS sysplex is to be activated. With WLM managing aliases, the customer does not need to determine how many aliases to assign to each base device and does not need to manually change this assignment when workloads change. WLM performs these tasks automatically. WLM uses two mechanisms to manage the number of aliases. Both these mechanisms use sysplex-level device activity information to drive the decision to transfer an alias from one device to another.

The first mechanism, based on the *goal algorithm*,[16] attempts to allocate additional aliases to a base device (the *receiver* device) that is experiencing IOS queuing delay and is servicing a workload that is missing its customer-specified goal in the WLM policy.[2,17] A *donor* device is found among the devices used by workloads that have equal or less business importance than the receiver device. If such a donor device is found, then an alias can be transferred from it to the receiver even if it means increased IOS queuing for the donor. However, if the donor device is used by a workload of equal importance to the receiver, the donor device must not suffer increased queuing as a result of losing the alias. These restrictions simplify the algorithm because then WLM does not need to predict how much an alias transfer will help or hurt the attainment of business goals. An alias transfer requested by the goal algorithm does not necessarily decrease overall IOS queue delay. It does help a higher importance workload do better, but it could be at the expense of a lower importance workload, which is consistent with the philosophy for other WLM-managed resources.[16]

The second mechanism, based on the *efficiency algorithm*, transfers alias devices from low contention base devices to high contention base devices without regard to the business importance of the workloads involved. The objective is to minimize the total IOS queuing within an LSS. This algorithm transfers an alias only when the goal algorithm did not result in an alias being transferred, for reasons having to do either with the receiver or with the donor device.

The goal algorithm is invoked more frequently than the efficiency algorithm. This gives the attainment of business goals precedence over simply reducing total IOS queuing. High contention base devices are those that have significant IOS queue delay. Low contention base devices are those that have no significant IOS queue delay and can give up an alias with no increase in queue delay. The efficiency algorithm is conservative because it will not make a transfer that could potentially cause more harm to the donor than benefit to the receiver. The WLMs in the sysplex act as peers with respect to the efficiency algorithm: any of them may use the sysplex-level PAV device data to initiate alias transfers regardless of who is using the devices.

Sysplex view of device activity. Because moving an alias from one base device to another affects every system using those devices, the algorithm making that determination should have a sysplex view of the logical volume, including the appropriate performance data. To build a sysplex view, each z/OS system in the sysplex broadcasts its local performance data to all other z/OS systems. A peer approach is used by WLM for managing aliases under the efficiency algorithm. Any system in the sysplex is able to make alias transfer decisions based on the sysplex view. The goal algorithm is "locally-oriented" because each system tracks the PAV devices that are causing the most IOS queue delay for each service class on the local system.

Both the goal algorithm and the efficiency algorithm are invoked periodically, albeit with a different period. The invocation will result in action only if the time since the last action is greater than a specified value. It has been observed in laboratory testing that one system in the sysplex tends to take over most of the adjustment decisions. For example, if on a particular system the efficiency algorithm is invoked 60 seconds or more since the last time the efficiency algorithm ran, the first system to detect that the 60 seconds have expired is likely to be the same system every time. This works very well because each system has the same view of the PAV device activity data, and any of the peer WLMs can make alias transfer decisions on behalf of the sysplex, even if it is not itself using ESS at the time.

Efficiency algorithm. We describe here the WLM efficiency algorithm in more detail. Although the goal

algorithm is invoked before the efficiency algorithm, we discuss the efficiency algorithm first in order to introduce some new concepts that are more easily described in this context.

The WLM efficiency algorithm takes aliases from base devices with low contention and transfers them to base devices with higher contention, the contention being measured by the IOS queuing delay. The result is to minimize overall IOS queuing against an LSS. In a sysplex, invocations of the algorithm are scheduled at least 60 seconds apart. Note that this algorithm is invoked less frequently than the goal algorithm, for which the period is about 30 seconds.

A table listing all base volumes, as well as the unbound aliases, in an LSS, sorted by IOS queue length, is used to find the most "needy" receiver devices as well as the "richest" donor devices (unbound aliases are treated as having zero queue length). When a receiver-donor pair is identified, IOS is instructed to unbind the alias from the donor device and bind it to the receiver device. The efficiency algorithm can transfer multiple aliases in one pass but will not transfer more than one alias to or from a particular base device more often than once per cycle.

Searching for a receiver-donor pair. WLM creates a table listing all base devices in an LSS. The table is sorted in descending order by sysplex IOS queue length. For ties on IOS queue length, the table entries are in ascending order with respect to the number of aliases assigned to the base device. Devices at the top, at the high-IOS-queue-length end of the table, are potential receivers of aliases. A device must have a sysplex IOS queue length above a threshold of 0.5 to be considered as a receiver; for example, an average queue length of 0.5 would be calculated when an IOS device queue is sampled 40 times and when, on 20 of those samples, there was one request queued. The value of 0.5 was empirically chosen after experimentation. Note that if no receiver device is found on the first pass, that is, if there are no devices with an IOS queue length greater than 0.5, a second pass is made in which receiver devices with lower levels of queuing are considered.

Devices at the bottom, at the low IOS queue length end of the table, are potential donors. Unbound aliases are placed at the very end of the low-IOS-queue-length end of the table. Unbound aliases are the first choice as donors because they represent unused resources, possibly unbound from base devices now off-line and therefore idle. The efficiency algorithm attempts to transfer one alias from a donor to a receiver, but only if taking the alias from the donor is not expected to cause a significant increase in queue delay for the donor device. Figure 5 shows the WLM PAV data table, the sorted table used for searching for receiver-donor pairs.

Figure 5 The WLM PAV data table for a sysplex

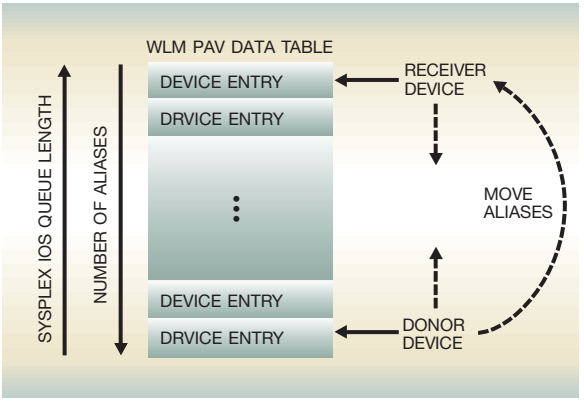


Table 1 Deciding on the suitability of a device to be a donor of aliases

Number of Remaining Aliases	Acceptable Utilization
1	20%
2	35%
3	47%

Assessing the impact on a donor device. The efficiency algorithm takes an alias from a donor device only if the action will not result in a significant increase in IOS queuing for the device. The more aliases the device has, the higher its utilization can be and still remain an acceptable donor. Table 1 shows an example on how the decision is made on the suitability of a donor.

According to this table, when a potential donor is taken down to one alias, its utilization must be below 20 percent. When the potential donor is taken down to two aliases, the current utilization must be below 35 percent. The table was generated from a multiserver queuing model used to estimate the impact on device queuing based on the number of aliases and the average utilization per alias. The uti-

lization figures in the table were selected to ensure minimal impact on the performance of the donor device. Here, the definition of minimal impact on queuing is the point at which the probability of an I/O request encountering the base and all aliases in use is less than 20 percent. Thus, if the projected impact of removing an alias keeps the device below the utilization that would cause a 20 percent chance of queuing, then the alias can be transferred.

Goal algorithm. In many cases, the efficiency algorithm is sufficient to optimize alias placement in a logical subsystem and minimize overall IOS queue delay. However, there is one obvious case that the efficiency mechanism cannot handle. This is the situation in which an alias needs to be taken from a high activity device being used only by work of lower importance and given to a high activity device that is delaying work of high importance. For example, suppose there is a transaction processing workload of high importance being delayed by IOS queuing on a device, and there is a second device in the same logical subsystem that has multiple aliases and is being used only by a batch workload of lower importance. Then aliases should be taken from the second device and transferred to the first in order to decrease the queuing delays for the first device, even at the expense of increased queuing delays for the second device.

The goal algorithm is designed to handle such cases. Moreover, the goal algorithm runs before the efficiency algorithm in order to ensure that when the number of donor devices is limited, goal-attainment alias transfers take precedence over efficiency transfers. Because the service-class-delay information used by the goal algorithm is only available to the local system, it is the local system that will make the alias transfer decision in order to help a local service class that is missing its goals.

The goal algorithm runs every 30 seconds on each system in the sysplex. Because the goal algorithm runs more frequently than the efficiency algorithm, it can be more responsive to situations where work of high importance is missing its goal. Specifically, the algorithm looks for a service class missing its goal and also experiencing a significant amount of IOS queue delay on the local system. For each such service class, an attempt is made to find additional aliases for up to three devices among those contributing the most to the delay of the service class on the local system. For each one of these three devices, the algorithm attempts to find a donor device.

Although the search for a donor is performed in the same order used in the efficiency algorithm, only devices used by service classes of the same or lower importance can qualify as donors. The device-to-service-class mapping is used to make this determination. If the donor device is being used by less important service classes than the receiver, the transfer is made even if IOS queuing will increase on the donor device. If the donor is being used by work of equal importance, a more conservative approach, similar to that used in the efficiency algorithm, is used. The alias transfer is made only if there will be no increase in IOS queuing on the donor.

Pacing of alias transfers. Alias transfers are paced in order to prevent overreaction to transient fluctuations in I/O workload. Any particular base device gets no more than one alias action, either added or removed, per minute. The exception is when an alias is needed by the goal algorithm to help a more important service class than the last service class helped in the subsystem. In this case, the minimum time between alias transfers is lowered to 30 seconds. This allows a more important workload to get first chance at taking aliases from devices that have had a recent alias transfer. If the pacing interval were one minute in all cases, the efficiency algorithm on one system in the sysplex could always be the first to run after the minute had expired; it could transfer several aliases from a donor base device and delay the goal algorithm on another system from helping work that is missing its goal.

Limiting the number of aliases. There is no prescribed maximum or minimum number of aliases that can be assigned to a base. The adjustment algorithms heuristically determine the optimal number of aliases needed to reduce or eliminate IOS queuing. This means WLM needs to recognize the point at which adding more aliases will not improve I/O performance. Additional aliases do not help if the IOS queuing would simply be replaced by increased channel, control unit, or device contention. WLM will not add aliases to a device if its average pending time, control unit queuing time, or disconnect time are over a threshold. The current thresholds are 20 milliseconds for average control unit queuing plus disconnect time and 20 milliseconds for average pending time. (These threshold values were empirically derived based on experimentation.)

Handling of the pending time threshold is different from handling of the CU queuing/disconnect time threshold. Average pending time for a PAV device

could be high on one system and low on another if it is caused by a channel constraint local to a system. In this case, aliases would still be added to reduce IOS queuing for the system with low pending time even though the other system has high pending time

The automatic management of aliases by WLM saves the customer the need to mannually adjust the use of this resource when workload changes.

and cannot benefit from the additional alias. A pending device reservation is handled as a special case. WLM will not take action against delays that are a result of pending device reserves on the volume because adding an alias will not decrease this type of IOS queuing or pending time.

Administrator control of PAV management. There are two ways for the administrator to control dynamic alias management, either a sysplex level option in specifying the WLM policy[2] or a device level option in the z/OS Hardware Configuration Definition[8] (HCD). The sysplex level option in the WLM policy is a YES/NO corresponding to whether dynamic alias management is active (for the entire sysplex) or not. The default is NO so that the administrator has a chance to prepare the configuration and upgrade to the required software release levels before switching on the dynamic alias management. The device level option in HCD enables or disables dynamic alias management for a particular PAV base device. A device is enabled by default.

The device-level option in HCD is used to stop WLM from adjusting the number of aliases for a device that is shared by systems that do not support dynamic alias management. Such systems include systems earlier than OS/390 Version 2 Release 7, systems running in WLM compatibility mode, systems from another sysplex, or systems running an operating system other than z/OS, such as VM. If a customer were to allow dynamic alias management for such shared devices, the WLM alias transfer decisions would factor in only the device usage by the systems that are participating in dynamic alias management. These decisions, based on a partial view of device usage, are at risk of causing less than optimal alias allocation. The HCD device-level option is also used to disable dynamic alias management for base devices that are off-line to some systems in the sysplex. If a device is on-line to some systems in the sysplex but off-line to others, WLM does not make valid alias transfer decisions.

Details on PAV metrics. The measurements used to drive the PAV adjustment algorithms are accumulated across a 60-second interval on each system in the sysplex. Thirty-second measurements are used if the goal algorithm needs to make an alias adjustment to help work of high importance before the full minute has elapsed. The values maintained are as follows.

- Sysplex IOS queue length—IOS queuing on a base device is measured by WLM for dynamic alias management because it is the indicator that the device has more outstanding requests than its current number of aliases can handle concurrently. The average IOS queue length for each base on a single system is calculated by sampling the IOS queue every quarter second and averaging the queue length across the samples. The sysplex IOS queue length for each base is the *sum* of the average queue lengths on each system in the sysplex. The sum is used rather than an average across the systems to ensure WLM properly handles the case where only one system out of many in a sysplex has a nonzero queue length. Using a sum ensures appropriate action is taken for the device to relieve the contention within that one system. Averaging the queue length across the systems, especially in a large sysplex with many systems, could dilute the value so much that WLM would not recognize that action is needed.
- Total service time—The device utilization over the measurement interval is calculated for each system based on the total service time on the base device and all its current aliases. The sysplex device utilization is calculated as the maximum utilization observed by any system in the sysplex. This utilization is then used in a formula to determine if removing an alias would increase IOS queuing for a device. See "Assessing impact on a donor device" in the subsection "Efficiency algorithm" for more information.
- Average device delay time per request—This value includes control unit queue time and disconnect time. It is compared against the threshold of 20 milliseconds. If the sum is higher than the threshold, it indicates the device cannot benefit from additional aliases, and WLM stops transferring aliases to the device.
- Average pending time per request—This value

represents channel contention. It is compared against a threshold of 20 milliseconds. If this value is higher than the threshold on the local system, it indicates the device cannot benefit from additional aliases.
- Device-to-service-class mapping—This is an array indicating which service classes are using which PAV devices during the minute being measured. Each service class is assigned an importance level in WLM policy. The goal algorithm determines the most important service class using a device and ensures that a donor device has a lower importance than or equal importance to the receiver device. Certain special I/Os are ignored for the purposes of creating this mapping so that the mapping represents only application-level I/O activity.

The automatic management of alias devices by WLM makes the most efficient use of this valuable resource in ESS under changing workload conditions. Letting WLM manage aliases saves the customer from having to continually make manual alias adjustments in order to match resources to work requirements. Because WLM takes into account the business priority of the work in the z/OS sysplex, it can make alias adjustments to ensure that business-critical work gets the resources it needs to meet its goals, and that overall IOS queuing is minimized.

WLM I/O priority management

Dynamic management of I/O priorities by WLM complements dynamic alias management. The greater parallelism provided by PAV has the effect of transferring the queuing of requests from the software (IOS) to the channel (CSS) or to ESS. Further improvement in achieving workload goals can be obtained by enabling WLM to control the order in which the competing requests are handled within ESS.

Manual control of I/O priority has been available in z/OS and OS/390 for years. I/O priority management by WLM was introduced in OS/390 Version 1 Release 3 for IOS queuing within a single system. When ESS was released, I/O priority management was extended to queuing within ESS for handling, for example, extent conflicts and reconnects after cache misses. The priority values used for IOS queuing are passed now to ESS to be used in queuing situations. In both cases, the queuing is at the volume level. More recently in z/OS Version 1 Release 1, the Intelligent Resource Director (IRD) introduced CSS priority management.[6] The objective is to manage I/O priority end-to-end so that the most important work in the system has the best possible chance of meeting its goal. In this section we describe I/O priority management in both IOS and ESS.

Why manage I/O priorities? In the earliest releases of WLM, I/O priorities were not directly managed, but instead were set to equal the CPU dispatching priorities. This worked fairly well because interactive workloads usually have a higher dispatching priority than batch workloads. However, it is easy to envision cases where a service class has few CPU delays but is missing its goals because its I/O priority is too low, perhaps because it is competing with work in another service class for the same device, and the other service class is dominating the device. In such a case, the I/O priority should be different from the dispatching priority.

The setting of I/O priorities is done at the sysplex level in order to handle the case where devices are shared among systems in a sysplex, and there is queuing within ESS. The mechanism used to manage I/O priorities is similar to dispatching priority management with some notable exceptions. First, only eight priority values are used because management of I/O requires fewer levels than dispatching. Second, unlike dispatching priority, I/O priority is considered a sysplex-wide attribute. That is, a service class has the same I/O priority across the sysplex. Therefore, when considering I/O priority changes, WLM takes into account the effect of the change on the work across the sysplex.

In managing I/O priorities we use the concept of a *device cluster*, a set of service classes competing for the same set of devices (a more accurate name would be *service class cluster based on device usage*). Each device is part of a single device cluster and each service class period is associated with a single device cluster. This allows I/O priorities to be set independently for service classes that are not competing for the same I/O resources.

The eight I/O priorities used by WLM are given in Table 2.

Sampling. I/O priority management uses *I/O-using* samples and *I/O-delay* samples in order to identify a service class that is missing its goal due to I/O contention, and that can be helped by increasing its I/O priority relative to another service class. I/O-using samples are the samples in which a work unit is making use of a device and could be causing delay to other work. I/O-delay samples are the samples in

which a work unit is delayed when it tries to use a device that another work unit is using. I/O-using time (computed from the I/O-using sample count) consists of device connect time, reported by CSS. I/O-delay time is the combination of IOS queue time, CSS device pending time, and control unit queuing time. In the original design, disconnect time was counted as part of I/O-using time because it was considered to be productive time. An I/O request may be disconnected from the channel for a variety of reasons, such as an extent conflict or a cache miss during data transfer from disk to cache. However, very high disconnect times usually represent high contention in the device, and not a large amount of useful work being done. Therefore, disconnect time has been dropped from the I/O-using calculation. Some customers observe very high disconnect times due to device constraints, and in these cases, it is not appropriate to count disconnect time as productive time. Doing so would make a service class look as if it were doing well when in fact it was suffering high delay. This would prevent WLM from helping the service class when needed.

There are two approaches to gathering I/O samples. Traditional sampling is used for measuring delay due to IOS queuing. In addition to sampling IOS queue length, WLM also tracks the devices each service class is using in order to determine which groups of service classes are competing for the same I/O devices (device cluster).

An alternate method is used to gather samples representing I/O-using time (device connect time) and the portion of I/O delay samples represented by subchannel-pending and control-unit queuing delays. Instead of using direct sampling, WLM derives equivalent sample values from measured times which are reported by the channel subsystem to the operating system. These are the same measurements reported in RMF device reports.

Tracking device clusters. Managing I/O priorities is different from managing other resources in that work doing I/O does not necessarily compete with other work in the system doing I/O. If application X is using a set of devices that is disjoint from the set of devices used by application Y, the I/O for X has minimal effect on the I/O for Y. This is different from managing CPU priority, for example, where all the work in the system competes for the same pool of CPU resource.

Table 2 I/O priorities used by WLM

Value	Description	Used By
FF	SYSTEM service class	System tasks
FE	SYSSTC service class	High-importance application
F9 to FD	Managed priorities	Service classes with goals
F8	Discretionary work	Low-importance work

Thus, when WLM considers changing the I/O priority for a service class, it needs to know what other work will be affected by this change, and for this it uses the previously defined concept of device cluster. The device clusters are defined based on sysplex-wide I/O samples for each service class. One of the systems in the sysplex, usually the first to initialize, takes the responsibility for defining the device clusters and broadcasting the information around the sysplex. The other systems send their samples to the "clustering" system once a minute. Every 10 minutes, the clustering system consolidates the samples from each system, creates the device clusters, and broadcasts them. Each system then has the same sysplex-wide view of I/O usage when making I/O priority decisions that affect other systems in the sysplex. Because any I/O priority change is also broadcast, I/O priorities for service classes remain consistent across the sysplex.

Table 3 shows the combined number of I/O-using samples and I/O-delay samples (taken every 0.25 seconds) for each one of five service classes and a set of devices. The use of device 500 by Class 3 is considered "insignificant." The device clusters to be defined based on these samples are (1) Classes 1, 2, 3, and (2) Classes 4, 5.

Adjusting I/O priorities. Every 10 seconds, WLM invokes its policy adjustment logic.[16] It looks at each service class period, starting with the most important, until it finds a service class period that is missing its specified goal. This service class period is known as the receiver. If the most important bottleneck is I/O delay, then WLM considers making an I/O priority change to help the receiver meet its goal. The action taken is to either raise the priority of the receiver service class period or lower the priority of a competing service class period, known as the donor. Both the donor and the receiver periods must belong to the same device cluster. Before making a priority change, WLM estimates the performance gain to the receiver as well as the performance cost to the donor. WLM makes the change only if the value

Table 3 Samples collected for a set of service classes and a group of devices

Service Class	Dev 200	Dev 201	Dev 202	Dev 500	Dev 501	Dev 502	Dev 503
Class 1	100	150	150	0	0	0	0
Class 2	0	90	100	0	0	0	0
Class 3	0	100	100	5	0	0	0
Class 4	0	0	0	100	100	100	100
Class 5	0	0	0	0	150	0	150

to the receiver is sufficient, and if it will not cause more important work to miss its goal. Projections are in terms of the number of using and delay samples to be expected following the change. For example, when the I/O priority of a receiver is raised to be equal to or above that of a donor priority, the using samples of the receiver will go up and its delay samples will go down. The result is that the receiver period will be closer to meeting its goal.

WLM's management of I/O priority complements its dynamic management of aliases. Although PAV increases parallelism for I/O requests and reduces a major cause for I/O delays, there still can be contention for resources at some level, either in the operating system, in CSS, or in ESS. I/O priority management ensures that when there is contention, the work is processed in the appropriate order; that is, the work with the highest business importance receives preferred access to the I/O resources as needed to meet its performance goals.

I/O priority within the ESS FICON adapter. The final part of the end-to-end priority management provided is within ESS. Without ESS supporting the end-to-end I/O priority management, discretionary I/O requests would be managed and executed within ESS with the same priority as system I/O requests; when contention exists, this can seriously impact the performance of system I/O requests, as well as all the service classes being managed by WLM (see Table 2 for I/O priorities used by WLM).

ESS uses I/O priority in two of its functional areas: the FICON adapter and the ESS control unit. I/O priority is used at the FICON adapter when contention exists between the FICON adapter and ESS control unit, and it is used for managing data transfers on the FICON interface. It is also used within the cluster controller when an extent conflict exists for the logical volume, independent of the type of path. We focus here on the handling of priority for I/O requests within the ESS FICON adapter.

ESS FICON adapters can perform multiple concurrent I/O requests between the adapter and cluster controller, and between the adapter and FICON channel. Because of this, FICON adapters provide a very effective additional place where the priority is used when workload contention exists.

The FICON adapter uses the I/O priority value sent from z/OS for the duration of the request (i.e., a FICON request can consist of a chained set of commands) in order to (1) allow a high priority request to proceed before lower priority requests, and (2) prevent extended aging of low priority requests within the FICON adapter. When the initial command for the I/O request is received and during command execution of each subsequent command in the chain, the command will be queued in priority order on the adapter-to-cluster controller queue between the FICON adapter and the appropriate ESS cluster controller. When the adapter-to-cluster controller interface is free, the adapter will select the next element of work from the active queue. After the command request has been passed to the cluster controller, the FICON adapter selects another element of work (a command from another I/O request) and passes it to the appropriate cluster controller, providing the adapter-to-cluster path is not busy. The selection of work elements from the active queue continues until it is empty; at that point, all other elements of work of lower priority are promoted one priority level, and the highest priority nonempty queue will become the active queue. Work is processed as described above; when that priority queue is empty, the queue promotion process is repeated. After processing for a command is completed for one command in a chain, the next command in that chain is queued on the adapter-to-cluster queue based on the priority initially associated with that I/O request.

While the dequeuing process is proceeding (based on priority and the availability of the adapter-to-cluster path), new I/O requests can be received at the FICON adapter from FICON channels; this is in con-

trast to the way the ESCON adapter works, where only a single element of work could be handled per adapter. As a result, high priority work on FICON adapters will not be blocked by lower-priority long-running requests. These new I/O requests will be queued on the adapter queue based on the priority value sent with them. If no priority value is sent, the I/O request in placed on the lowest priority in the queue; automatic priority promotion ensures that the I/O request will execute.

Field experience

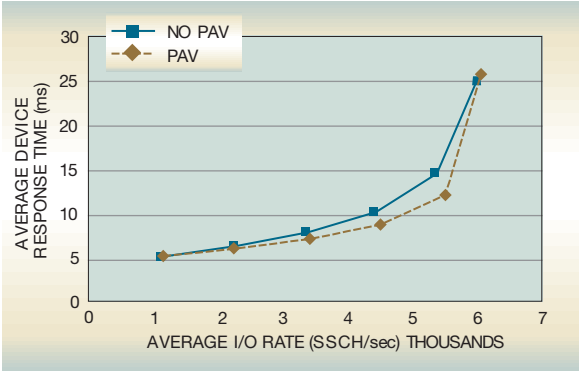
In this section we discuss field experience with PAV and IORP and present some related experimental data.

MA and PAV. For each volume managed by ESS, I/O requests may be overlapped: MA supports one I/O request from each of the attached systems, whereas PAV permits multiple requests from the same system. MA is beneficial in environments where a volume is shared by multiple systems, such as a Parallel Sysplex*.[15] The overlapped execution of requests reduces the response time. The situations described below apply to multisystem environments sharing a volume, single systems with multiple requests against a volume, and the combination of these.

Consider a common situation in which a cache miss is followed by a request for data resident in cache. Before MA and PAV, the second request had to wait for the cache miss to be serviced, resulting in delays of milliseconds, the time typically required for a simple cache miss to be resolved. PAV allows subsequent requests to be serviced while the control unit stages the data for the miss to cache from the physical disks, so the cache-hit request is no longer delayed behind the longer-delay cache miss. Similarly, requests for small amounts of data, which may be in the cache, need not be delayed by a long-running sequential transfer.

Figure 6 shows some experimental data from a system running a standard database workload using multiple volumes spread across several RAID arrays.[18] This is a typical "cache-hostile" z/OS workload. The I/O rate increases as additional users are added, each accessing his or her own "working set" in the large database stored across many logical devices. Figure 6 shows the performance improvement with PAV. The solid curve represents the behavior without PAV, and the dashed curve shows the behavior of the database volumes with aliases. The graph also illustrates

Figure 6 Cache-hostile database workload with and without PAV



that PAVs cannot increase the aggregate maximum capabilities of an I/O subsystem. PAVs will improve the response time for I/O requests by reducing IOSQ, the time queued in the IOS component of z/OS for volumes with aliases, until some other resource in the subsystem becomes the limiting factor. The limiting resource could be the set of channels, the number of aliases available, or the internal bandwidths of other resources, such as device adapters or RAID ranks. As these other resources saturate, the I/O response time will increase in spite of PAVs, appearing as either device pending time (PEND), time queued in either the channel subsystem or in the control unit, or device disconnect (DISC) time, usually the time spent waiting for a cache miss to be resolved. As queue depths continue to increase, IOSQ time may reappear, as well.

Sequential data can experience significant improvement from PAVs, either as increased data rate or decreased elapsed time for multiple job streams. That translates into reduced batch windows. In one QSAM (queued sequential-access method) experiment, multiple read and write streams were directed to different data sets on the same logical volume. In Table 4, there were six jobs reading or writing data sets on a single logical volume. The number of aliases for the volume varied from 0 (no PAV) to 4; the workload was for 27 KB records with BUFNO=5. The jobs were designed to make sure the data were read and written all the way to the disks in the array group that contained the logical volume. For these sequential cases, we see the limiting sequential capability of a single RAID5 rank near 40 MB/sec.

Table 4 QSAM Experiment 1 involving six simultaneous jobs writing or reading different data sets on a single volume

	Number of Aliases	MB/sec
Write	0	12.34
	1	24.66
	2	31.7
	3	39.69
	4	40.56
Read	0	13.5
	1	29.96
	2	33.46
	3	41.68
	4	42.2

Table 5 QSAM Experiment 2 involving one data set on a single logical volume with eight read streams

Number of Aliases	MB/sec	MB/sec/stream
0	13.6	1.7
1	27.28	3.41
3	53.54	6.69
6	83.52	10.44

In a subsequent experiment (see Table 5) a volume with six aliases, in addition to the base address, was defined. Over 80 MB/sec were read from a data set; the workload had 27 KB records with BUFNO=5. In this case, the eight channels to the logical device were the limiting resources in the subsystem.[18]

Experiments to demonstrate the value of PAV were performed using the Commercial Batch Work Load (CB84) of the *Large Systems Performance Reference*[19] (LSPR) on z/OS Version 1.2 with ESS Model 800. The goal of the measurements was to determine how large volumes performed as an increasing number of CB84 Batch 3390 Model 3 work volumes were consolidated. The IBM LSPR method is designed to provide relative processor capacity data for System/390 and zSeries architecture processors, both IBM and IBM-compatible. The CB84 workload is a moderate commercial batch job stream consisting of 130 jobs, with 610 unique job steps. The work done by these jobs includes various combinations of compile, link-edit, and execute steps. Utility jobs, primarily for data manipulation, are also included. The CB84 job stream is replicated to assure a reasonable measurement period, and the job queue is preloaded. Enough initiators are activated to ensure a high steady-state utilization level, approaching 100 percent. The measurement is started when the job queue is released, and ended as the last job is completed. Each job in the job stream uses its own data sets in order to eliminate data set contention.[19] On average, each CB84 database volume contained about 380 MB of data. Ultimately, 70 3390 Model 3 volumes were consolidated, from different LSSs onto the single 27-GB volume, before the 27-GB-disk space was exhausted, and the large volume continued to perform well.

As shown in Figure 7, the large volume response time does increase as the SSCH rate to the volume increases. However, while the I/O rate to the volume increases twentyfold, the response time increases only by a factor of two, and there is no negative effect on system transaction rate.

IORP. MA and PAV provide the capability for ESS to support more work by eliminating the volume as a serialization point. z/OS support for ESS provides the means to feed more I/O requests to ESS. As ESS is fed more work, it becomes increasingly important to provide a means for the CSS and ESS to understand the priority of the work and to exploit that priority.

IORP can also be used to greater advantage with FICON channels than with ESCON channels, because it can help prevent more important work from being delayed behind less important work. When a long-running I/O request is started on an ESCON channel, it blocks additional work on the interface until it completes; work with a higher IORP must wait until the active request is completed before that ESCON channel can be used. Although contention can still occur for a FICON channel, FICON eliminates the I/O interface as a serialization point and hence complements the ability to move more work between zSeries and ESS; new requests can be sent to ESS while other requests are already active on the interface, and ESS can resume previously queued requests that become ready for execution without waiting for the interface to become available. The value of multiple simultaneous data transfer on a FICON interface is enhanced by ESS recognizing priorities.

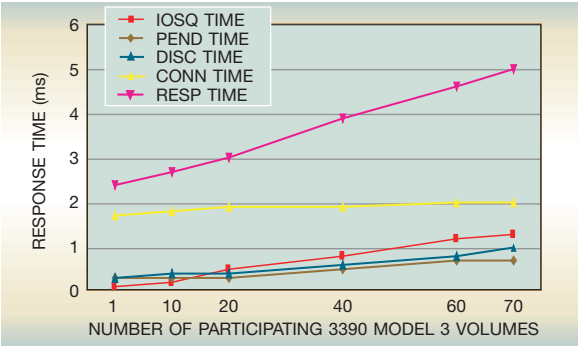
MA, PAV, and IORP should not be viewed as totally separate, isolated capabilities; rather they should be viewed as a set of related complementary enhancements, further enhanced by other capabilities such as FICON and DCM, and all brought together by z/OS support and WLM management. IORP allows WLM to assign more important work a higher priority and less important work lower priorities to affect the re-

sponse times of I/O requests, given the I/O resources that are available at any specific instant. This makes the most effective use of the resources, given the business goals. WLM dynamic alias management allows the system to dynamically move additional I/O resource to or from a logical volume in order to increase or decrease the number of concurrent I/O operations that can be performed on that volume. If important work is suffering because of a lack of available bandwidth, DCM can be used to increase or decrease bandwidth by redefining the use of the existing channel path in the configuration. All these capabilities work together, coordinated by WLM, allowing the system to tune itself to meet workload goals.

The administrator's view of storage management. In the past, when serialization of I/O activity to a volume could cause a bottleneck, administrators invested effort into relocating or replicating data sets, or both. As they attempted to spread the data across different volumes, they had to anticipate the workload demands against the data, or else one or more volumes could remain a bottleneck. Workload activity is generally not uniformly spread across volumes, and thus some volumes have higher activity than others. However, because workloads are dynamic, the spots of high activity shift over time from volume to volume. To deal with this changing environment in the past, volumes were kept at low utilizations so that they could absorb increased workload if and when that occurred; this resulted in inefficient use of storage capacity, and increased the complexity of the configuration process. PAV and MA have significantly ameliorated these problems, especially with WLM dynamic alias management. Alias resources are now managed automatically to handle the changing workload.

The configuration planning and data placement activities previously undertaken were intended to avoid hot spots or bottlenecks. They required both time and skills on the part of the customer staff. With the new functionality provided through MA, PAV, and IORP, people have been spending less time analyzing device configurations or data placement. As the focus of workload activity moves from device to device, WLM dynamic alias management manages resources toward achieving the specified goals, thereby preventing any individual device from becoming a hot spot; as the device gets "hot," resources are moved to "cool" it if it is affecting workload goals. Alias resources are also reallocated in order to improve overall efficiency, even when goals are all being achieved. Thus, the dynamic alias management has reduced the need for customers to analyze their workload and data access patterns. In effect, the granularity that the customer must deal with is decreased from the volume level to the subsystem level; instead of ensuring enough resources are configured for each individual volume to handle peak workloads, the customer only needs to configure enough resources to handle the maximum peak workload for the entire subsystem.

Figure 7 Response time of a large volume vs the number of participating 3390 Model 3 volumes consolidated



For database applications, the storage management tasks usually associated with achieving high levels of parallelism are also simplified. Administrators no longer need to manually manage DB2 partitions; instead, the operating-system storage-management facilities can be used to place the DB2 partitions, and WLM management of PAV can support high parallel query throughput regardless of the particular data placement. In fact, multiple DB2 partitions can even be placed on the same volume; with PAV, when a parallel query scans multiple partitions simultaneously on the same volume as opposed to scanning a single partition, ESS shows virtually no degradation.

PAV has enabled the use of larger volumes. Larger volumes mean more storage capacity with less storage or configuration management. Use of larger volumes enables larger database partitions, resulting in fewer partitions needed for a database of a given capacity; this results in less effort for planning and configuring the database. Larger volumes have also resulted in fewer "out of space" conditions. Before PAV, performance was one of the major inhibitors to the use of larger volumes; because requests against a vol-

ume were performed serially, the achievable access rate against a large volume was limited. In order to obtain a specified access rate against data, customers needed to spread the data across multiple volumes. By doing away with this serialization, PAV allowed higher access rates to be achieved. For example, to provide 27 GB of capacity using 3390 Model 3 volumes of 3 GB each, nine separate volumes would be required, including nine separate volume labels, unit addresses, and subchannels; this permits a maximum of nine requests to be active concurrently from the system, but only one active request within each 3-GB volume. With PAV, customers now provide one base address and subchannel for a single 27-GB volume, and subchannels are assigned for aliases as the workload requires. This results in the number of alias subchannels being associated with the base from zero to nine, or larger. Furthermore, concurrent requests are now serviced within 3-GB extents of the larger volume. When necessary, an even greater level of concurrency can be achieved. As a result, as long as an adequate number of alias subchannels is provided, performance is no longer an inhibitor to large volume sizes.

Conclusion

The z/OS support for PAV and IORP unlocks the value of these ESS capabilities, which in turn provides value to customers. The functions complement each other, as well as other I/O capabilities such as CSS priority, FICON priority, and DCM. The z/OS support, which is self-configuring and helps the system to be self-monitoring and self-tuning, is consistent with the autonomic computing vision.[3] The z/OS support is global in scope, coordinated across multiple logical partitions and multiple machines within a Parallel Sysplex.

Acknowledgments

The authors would like thank Peter Yocom for his suggestions and comments during the writing of this article and Mark Wiesnewski for the experimentation data.

*Trademark or registered trademark of International Business Machines Corporation.

Cited references and notes

1. *IBM TotalStorage ESS Introduction and Planning Guide*, GC26-7444, IBM Corporation (2002).
2. *z/OS MVS Planning: Workload Management*, SA22-7602, IBM Corporation (2002).
3. P. Horn, *Autonomic Computing: IBM's Perspective on the State of Information Technology*, IBM Corporation (October 15, 2001); http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.
4. M. Bohl, *Introduction to IBM Direct Access Storage Devices*, SR20-4738, IBM Corporation (1981).
5. *IBM z/Architecture Principles of Operation*, SA22-7832, IBM Corporation (2001).
6. W. J. Rooney et al., "Intelligent Resource Director," *IBM Journal of Research and Development* **46**, No. 4/5, 567–586 (2002).
7. *Fibre Channel Single Byte Command Code Sets-2 Mapping Protocol* (*FC-SB-2*), T11/Project 1357-D/Rev 2.1, INCITS (December 2000); http://t11.org/index.htm.
8. *z/OS RMF User's Guide*, SC33-7990, IBM Corporation (2002).
9. *z/OS HCD User's Guide*, SC33-7988, IBM Corporation (2002).
10. *IBM TotalStorage ESS Web Interface User's Guide*, SC26-7448, IBM Corporation (2002).
11. *Information Technology—SCSI Architecture Model-3*, T10 Project 1561-D, Revision 4, INCITS (2002); http:/t10.org/index.htm.
12. *IBM Enterprise Systems Architecture/390: Common I/O-Device Commands*, SA22-7204-01, IBM Corporation, (April 1992).
13. *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202, IBM Corporation, (August 1992).
14. In the future, IBM might decide to change or "tune" the WLM algorithms for dynamic alias management with ESS without notice.
15. "S/390 Cluster Technology: Parallel Sysplex," by J. M. Nick, et al., *IBM Systems Journal* **36**, No. 2, 172–201 (1997).
16. J. Aman et al., "Adaptive Algorithms for Managing a Distributed Data Processing Workload," *IBM Systems Journal* **36**, No. 2, 242–283 (1997).
17. "Workload Manager/System Resources Manager for OS/390," http://www.s390.ibm.com/wlm/.
18. B. J. Smith, "Parallel Access Volumes for OS/390," *Computer Measurement Group 1999 Proceedings*, Reno, NV (December 1999).
19. *Large Systems Performance Reference*, SC28-1187-08, IBM Corporation (2002).

Accepted for publication January 12, 2003.

Allan S. Meritt *21 Sutton Park Road, Poughkeepsie, New York 12603 (asmeritt@optonline.net).* Mr. Meritt recently retired after 32 years with IBM. He was an IBM Distinguished Engineer in the former Server Group and was a member of the IBM Academy of Technology. He joined IBM in 1970 after receiving his M.A. degree in physics from the University of Texas at Austin. He received IBM Corporate Awards for his contributions to the 370-XA I/O architecture and software design, and for his contributions to ESCON architecture. He participated in the task force that defined ESS, as well as a number of subsequent corporate assessments of ESS. He has also received awards for his contributions to FICON and for the concept of Parallel Access Volume (PAV) implemented by ESS.

John A. Staubi *IBM Systems Group, 2455 South Road, Poughkeepsie, New York 12601 (jstaubi@us.ibm.com).* Mr. Staubi is an advisory software engineer in the z/OS development laboratory. He joined IBM in 1991 after receiving his M.S. degree in computer science from Northeastern University in Boston, MA. He has been in the z/OS development and service area his entire IBM career, involved in many programming projects as a developer,

tester, development team leader, and designer. Mr. Staubi was instrumental in providing the z/OS support for the IBM TotalStorage project. Additionally, he received an Outstanding Innovation Award for his work in zSeries software development.

Kenneth M. Trowell *IBM Systems Group, 5 Tallara Place Terrey Hills, NSW 2084, Australia (kennetht@us.ibm.com).* Mr. Trowell is a Senior Technical Staff Member. He joined IBM in the United Kingdom in March 1965. He has worked on every generation of IBM mainframe products from System/360™ to the present zSeries including, most recently, the system design for FICON Cascading. Mr. Trowell has received numerous awards including an Outstanding Technical Achievement Award for his contribution to System/370-XA.

Gail S. Whistance *324 Dewitt Mills Road, Kingston, New York 12401 (whist@hvi.net).* Ms. Whistance recently retired after 30 years with IBM. She was a senior software engineer in the former Server Group. She joined IBM in 1972 after receiving her B.S. degree with high honors in mathematics and English education from the University of Illinois at Urbana. She contributed to the early development of the System Resources Manager component of the MVS™ operating system and more recently, the Workload Manager. She was responsible for verifying the Workload Manager algorithms that manage the IBM TotalStorage ESS Parallel Access Volume for ESS resources.

Harry M. Yudenfriend *IBM Systems Group, 2455 South Road, Poughkeepsie, New York 12601 (harryy@us.ibm.com).* Mr. Yudenfriend is a Distinguished Engineer. He joined IBM in 1980 after receiving his B.S. degree in computer science from Columbia University, School of Engineering and Applied Science. He is the chief architect for I/O on z/OS. Mr. Yudenfriend received an Outstanding Innovation Award for his contributions to Parallel Access Volume for ESS and an Outstanding Technical Achievement Award for the architecture and design of FICON. He was named an IBM Master Inventor in December 2001.