Iain Neville

JES2 Performance and Availability Considerations

This paper provides insights into JES2 performance and includes tips on how to monitor and tune your environment. It discusses the key components of the JES2 infrastructure in the context of performance goals. The content provides guidelines but does not offer clear cut recommendations. It is assumed that readers of this paper have the technical understanding and knowledge to manage an existing JES2 environment.

This paper discusses JES2 processor utilization, checkpoint and spool considerations, workload manager (WLM) managed initiators, scalability, command processing, and identifying issues with performance degradation. It also discusses performance tools that are available for analyzing performance-related data.

JES2 processor utilization

The processor utilization of JES2 is not easy to tune. However, there are some general factors that have a direct impact on the activity of the main address space:

► The number of active JES2 processes and frequency of the following:

- Initiators, started tasks, and Time Sharing Option (TSO) logon requests
- Generation of SYSOUT data sets
- Printers, punches, and readers
- Network job entry (NJE) connections, JOB, and SYSOUT transmitters and receivers
- JES2 commands (see "JES2 command processing" on page 15)
- Priority aging of jobs or output

► JES2 queue structure, size, and work selection criteria

► Spool buffer sizes (see comments on BUFSIZE in "JES2 SPOOL volume allocation" on page 10)

► Transaction program (TP) buffer sizes (bigger is better)

► Number of JES2 SAF calls to Resource Access Control Facility (RACF®) or similar product

► Exploitation and efficiency of JES2 exits

► Currency of software maintenance (always check the PSP bucket)

► Unnecessary overhead writing System Management Facility (SMF) records or journaling

For information about analyzing these resources, see "Performance tools" on page 18.

JES2 checkpoint considerations

The JES2 *checkpoint* is the general term used to describe the checkpoint data set that JES2 maintains on either direct access storage devices (DASD) or a coupling facility. The checkpoint data set (regardless of whether it resides on a coupling facility structure or a DASD volume) contains a backup copy of the JOB and OUTPUT queues, which contain information about what work is yet to be processed and how far along that work has progressed. The checkpoint locking mechanism is used to serialize access when any change is required to the JOB or OUTPUT processes that changes the content of the queues (for example, a change in phase or processing status).

Similar to the spool data sets, the checkpoint data set is commonly accessible by all members of the multi-access spool (MAS) complex, but only one member has control (access) of the checkpoint data set at any one time. Furthermore, the checkpoint data set provides communication among all members of the configuration about jobs and the output from those jobs.

The JES2 checkpoint cycle

The JES2 checkpoint cycle is the same whether it is a single-system environment or an MAS complex. The checkpoint cycle is primarily made up of the following components:

► HOLD time

The HOLD time is the amount of time that a system is allowed to use the checkpoint. This time allotment is set by the HOLD parameter on the JES2 MASDEF statement. You can change the value dynamically using an operator command or the online System Display and Search Facility (SDSF) interface. In reality, the HOLD parameter is a minimum hold value. If the HOLD parameter is set lower than the time it takes for the primary write to

complete, the hold time is extended until the primary write completes. Also, it might be possible to have a unit of JES2 work become processor intensive and prevent the checkpoint processes from ending the hold period.

► DORMANCY time

The DORMANCY time is the amount of time that a system refrains from asking for the checkpoint. You set this time allotment using the DORMANCY parameter on the JES2 MASDEF statement. You can change the value dynamically using an operator command or the online SDSF interface.

► Processing time

The processing time is the amount of time that it takes to perform the mechanical functions of passing the checkpoint among the members in the complex. This time is primarily a function of the amount of time to do DASD reads and writes of the checkpoint data set information. You can change this time allotment using DASD cache controllers or coupling facility list structures to speed up the I/O response times.

► Free time

The free time is the amount of time when no member is using or obtaining the checkpoint. JES2 acquires, uses, and frees the checkpoint on a time slice basis which is set by the installation. The checkpoint cycle time then refers to the total time it takes for each processor in the complex to obtain ownership of the checkpoint, hold it, and then release control. When looking at one member of the complex, the cycle time is the elapsed time from the start of READ1 to the next READ1 on the same member.

The checkpoint mechanism is as follows:

► *Reserve*. JES2 issues a reserve to CKPT1. The reserve is always issued to CKPT1, even if both CKPT1 and CKPT2 are in use or if dual or duplex mode is used. When the reserve is satisfied, the processing continues.

► *Read 1*. After the reserve has been satisfied, JES2 obtains the software lock and reads the first track of the checkpoint data set.

► *Read 2*. If running in duplex mode, this I/O reads all of the changed 4 KB pages that reflect down-level system activity. If running in dual mode, this I/O reads the remainder of the change log and the changed 4 KB pages from any N-2 systems in the complex. This portion of the cycle then updates the storage queues of the system that is holding the checkpoint.

► *Primary write*. The primary write operation is used to bring the down level DASD copy of the queues up to date. In duplex mode, the down-level data set is the duplex or alternate data set. In dual mode, the down-level data set is the data set which was not read from.

► *Intermediate write*. The intermediate write activity updates the in-storage queues with new changes from the system and brings the DASD copy up to date. In duplex mode the 4 KB pages are changed. Every tenth intermediate write is also written to the down-level DASD data set. In dual mode, the change log is used, and the 4 KB pages are written only if the change log is filled.

► *Final write*. The final write is the same as an intermediate write, but after the final write starts, no further changes to the in-storage queues are allowed.

► *Lock release*. Releases the software lock.

► *Release*. Releases the hardware reserve.

There are two methods of operating a multi-access spool (MAS) configuration to provide individual members access to the checkpoint data set:

► Contention driven checkpoint data set access

Each member competes for the checkpoint data set control. If DORMANCY=(0,100) is specified on the MASDEF initialization statement, a member attempts to regain access as soon as a processor control element (PCE) requests it. Some forms of contention-driven access can provide benefits. Setting a low HOLD value (HOLD=10) and a low minimum dormancy value can benefit a response-oriented configuration where members are of equal size and strength. This type of checkpoint control has the advantage of ensuring that a member is always requesting the checkpoint. As a result of the low HOLD value, no member retains the data set for too long. However, contention-driven access has the disadvantage of causing potential member lockout and severe member degradation in an MAS environment.

► Controlled checkpoint data set access

This is the recommended method of checkpoint access in an MAS environment. It is a controlled environment where each member is allowed limited and regulated data set access. Using the queue control initialization parameters, HOLD and DORMANCY, on the MASDEF initialization statement controls how individual members contend for checkpoint data set control and the length of time each phase takes to complete. This paper concentrates on this method of checkpoint access.

Checkpoint-related useful time

When discussing the JES2 checkpoint cycle, it is important to understand the concept of checkpoint-related useful time. Think of the checkpoint as in two states: a usable state, where changes to the queues can be made, and an unusable state, where changes cannot be made to the queues.

From the perspective of a single member in a complex, checkpoint-related useful time is the time in a checkpoint cycle, which starts at the beginning of the primary write and ends at the beginning of the final write. Checkpoint-related non-useful time is composed of all other portions of the checkpoint cycle, namely the time it takes to perform Read1 and Read2 functions and to complete the final write.

It is important to examine the checkpoint cycle from a single members perspective as well as the useful time of the entire complex. Just because one member is unable to access the queues another member might be performing useful work. Therefore, overall useful time for the complex should be appreciably higher than a single member in the MAS. Checkpoint-related useful work should not be confused with JES2 work. Processes within JES2 which do not need to access the checkpoint will run uninterrupted even during a time frame when the checkpoint is in a non-useful state. The most obvious example of this situation is a printer. After the queues have been updated to assign an output element to a printer, the printer no longer needs the checkpoint and can obtain the data from the spool even though the checkpoint is owned by another member of the complex.

Complex useful time

During a checkpoint cycle it is important to ensure that some member of the complex is using it. It is possible to have no member requesting the checkpoint when the DORMANCY values are too high on all members in the complex. As a result, all of the systems are in some portion of their DORMANCY period, and no system is requesting the checkpoint even though JES2 checkpoint related work exists.

Optimizing the checkpoint cycle

You can use the following mechanisms to leverage the checkpoint mechanism and improve JES2 throughput and performance:

► Optimizing throughput

This method has the objective to maximize the useful time by specifying values for HOLD and DORMANCY that can be obtained through the formula in Example 2 on page 7. These parameters directly affect the amount of time a system can use to make updates to the queues. Therefore, there is a direct correlation between the specification for HOLD and the amount of useful time. After you have decided which is your desired cycle time, you can use the formula to derive the values for the two JES2 parameters and you can eventually increase the value for HOLD up to the point that both values will still fit within the cycle. This round-robin approach might break when the number of JES2 members increase. At this point, the suggested way is to switch more to a contention mode approach.

► Optimizing response time

This approach is for installations where some JES2 functions have a need of a response time smaller than the cycle time. By reducing the amount of time spent in doing the mechanical functions inherent in the checkpoint cycle, the I/O operations can be made to happen more quickly and the amount of useful time can increase. The use of the latest DASD technology with cached controllers and FICON® attachment or coupling facility list structures can optimize the I/O to the checkpoint data. If you need to further reduce the response time for some or all of your JES2 members, then you can use the contention-driven checkpoint data set access that is described in "The JES2 checkpoint cycle" on page 2. As an alternative, you can also reduce the HOLD times on all systems and reduce the DORMANCY times on the critical system to a smaller number (not necessarily zero).

HOLD, DORMANCY, and LOCKOUT guidelines

These three parameters of the MASDEF initialization statement are significant for the performance of your MAS. These are also specific to the environment. There are no hard and fast recommendations to follow. Figure 1 on page 6 provides a chart with guideline values. However, these are only starting points. SDSF provides good, real-time checkpoint performance data and should be used to adjust the HOLD and DORMANCY parameters dynamically. This adjustment has instant effect and reflects real-time values as opposed to an average over a stated period.

Setting the HOLD and DORMANCY parameters ensures that no member holds the checkpoint longer than it needs to (the HOLD setting) and that members allow others access to the checkpoint and do not maintain the checkpoint (DORMANCY=(x)). The other parameter on DORMANCY controls a member from becoming lazy (DORMANCY=(,x)). Tuning the checkpoint really starts with getting a good HOLD value for each system.

So, why not make the HOLD value very small so that no member keeps the checkpoint longer than necessary? To understand the concern when doing so, you must understand how JES2 works.

Many JES2 processes involve updating checkpoint control blocks, then writing data to spool, and then making final updates to checkpoint control blocks. In general, this process takes two checkpoint cycles to complete. However, in some cases, if the checkpoint is held long enough, the process can be done in one checkpoint cycle. If it is done in one checkpoint cycle, the process takes less than a second and less data is written to the checkpoint. If it must take two cycles, then it can take two or more seconds to complete and can write twice as much data to the checkpoint. So, short holds (of, say, 0 to 10) can cause various

processes to take too long to complete (too much checkpoint thrashing). On the other hand, a hold that is too long means that a member is holding the checkpoint and not doing anything, thus wasting a shared resource.

HOLD and DORMANCY starting values

SYSTEM WORKLOAD	SINGLE SYSTEM	TWO MEMBERS	THREE MEMBERS	FOUR MEMBERS	FIVE OR GREATER MEMBERS
BATCH, NJE, RJE, TSO, PRINT	HOLD=60000 DORM= (5,500)	HOLD=50 DORM= (50 ,500)	HOLD=40 DORM= (80 ,500)	HOLD=30 DORM= (90 ,500)	HOLD=20 DORM= (100 ,500)
HEAVY SSI USAGE (Limit to as few members as possible)	HOLD=60000 DORM= (5 ,500)	HOLD=80 DORM= (20 ,500)	HOLD=80 DORM= (20 ,500)	HOLD=80 DORM= (20 ,500)	HOLD=80 DORM= (20 ,500)
LITTLE JES2 ACTIVITY	HOLD=60000 DORM= (5 ,500)	HOLD=30 DORM= (80 ,500)	HOLD=20 DORM= (100 ,500)	HOLD=20 DORM= (100 ,500)	HOLD=20 DORM= (100 ,500)

Figure 1   HOLD and DORMANCY starting values

The most common problem that is experienced with the checkpoint lock is when one member holds the lock, preventing other members from doing useful work. This is often when the owning JES2 address space is not running due to a failure or having not been partitioned out of the sysplex. Other reasons could be error recovery (dump processing) or some other type of failure.

Checkpoint cycle time metrics

Cycle time is the average HOLD time plus the average DORMANCY time in your complex. As cycle time increases, JES2 overhead tends to go down but so does overall JES2 responsiveness. Longer cycle times often indicate that JES2 is sluggish. On the other hand, with a shorter cycle time, JES2 appears to be much faster. In such cases, commands respond quickly, printers pick up work sooner, and so forth. However, there can be increases in JES2 processor consumption.

A general recommendation is to work towards a cycle time of around two seconds. To calculate what you are actually realizing, you can use the following command:

```
$D PERFDATA(CKPTSTAT)
```

To reset the PERFDATA table, use:

```
$T PERFDATA(CKPTSTAT),RESET
```

Note that this command is system specific and only reports on the member from which the command is issued. Example 1 shows sample output from this command.

Example 1   Sample output from $D PERFDATA(CKPTSTAT)

```
$HASP660 $DPERFDATA(CKPTSTAT)
$HASP660 CKPT PERFORMANCE STATISTICS - INTERVAL= 1:50:04.378501,
$HASP660 AVGHOLD=0.504059,AVGDORM=0.935134,TOT$CKPT=1037280,
$HASP660 WRITE-4K=47603,WRITE-CB=0,OPT$CKPT=0,OPT4K=88304,
$HASP660 IO=R1,COUNT=4574,AVGTIME=0.007728,
$HASP660 IO=R2,COUNT=4572,AVGTIME=0.004771,TOTAL4K=24816,
$HASP660 TOTALCB=0,
$HASP660 IO=PW,COUNT=1023,AVGTIME=0.029629,TOTAL4K=14640,
$HASP660 TOTALCB=0,
$HASP660 IO=IW,COUNT=6239,AVGTIME=0.010084,TOTAL4K=41381,
$HASP660 TOTALCB=0,
$HASP660 IO=FW,COUNT=4575,AVGTIME=0.005397,TOTAL4K=6222,TOTALCB=
$HASP660 0
```

Adding together AVGHOLD and AVGDORM establishes the average cycle time on this member in the MAS. Tuning is now a matter of increasing the HOLD on the member or members that are most out of line with the HOLD setting and decreasing the member or members that are closest to their HOLD setting. If all members are significantly greater than what was set, then your goal might not be attainable. A well-tuned system has all members pretty close to their HOLD values, and a cycle time that is near your goal.

Another mechanism is to use your desired cycle time in the formula to calculate your starting HOLD and DORMANCY values as shown in Example 2. Example 2 uses the output from Example 1. Because HOLD and DORMANCY are measured in hundredths of a second, the figures have been converted accordingly.

Example 2   Calculating starting values for HOLD and DORMANCY

```
Input to formula:
R2 = Average READ2 time - rounded to 0.005 seconds
FW = Average Final Write time - rounded to 0.005 seconds
MAS members = 6

HOLD = (desired_cycle_time-((R2+FW)*number_of_member))/number_of_members
(200-((0.5+0.5)*6)/6 -> (200-6)/6 -> 32 value for HOLD

DORMANCY =(HOLD+R2+FW)*(number_of_member-1)
(32+0.5+0.5)*(6-1) -> 33*5 -> 165 value for dormancy

The second value in dormancy is really just a measure of how long you want to wait if there
is nothing to do. A starting recommendation is to make this 2 to 3 times the desired cycle
time (or split the difference and do 2.5).
200*2.5 = 500
So in our example, HOLD=32, DORMANCY=(165,500)
```

Example 2 shows an initial evaluation for the HOLD and DORMANCY value for the members in the MAS. These values should be re-evaluated if the JES2 workload is not distributed homogeneously across all the members of the MAS. If the installation has heavy JES2 members and light JES2 members, one way to evaluate the HOLD and DORMANCY is to group the similar members logically and use their values in the formula as thought they were a logical entity. This grouping should provide more accurate initial values for HOLD and DORMANCY for the members.

In such an installation, you might want to use a more aggressive HOLD and DORMANCY setting, with shorter HOLD and DORMANCY times on all systems but with a slightly longer HOLD time and shorter DORMANCY times for the members where the heavy JES2 workload is running. This is especially true with CKPT1 on a coupling facility (CF) but still applies when using DASD, as most requests are handled first in, first out (FIFO).

Typical reported symptoms

The following messages are typical of poor management of HOLD, DORMANCY, and LOCKOUT settings or are symptomatic of a system problem:

```
$HASP263 WAITING FOR ACCESS TO JES2 CHECKPOINT
$HASP9207 JES2 CHECKPOINT LOCK HELD
```

When issued together, these messages can be due to JES2 spending a lot of time processing a specific request or because JES2 is a victim of another problem. A useful command to help diagnose this is the following:

```
$D PERFDATA(EVENT)
```

This command tracks the processes that took more than five seconds to complete. The event table tracks the latest 100 events on a member since JES2 was started or since the table was reset using the `$T PERFDATA(*),RESET` command.

When `$HASP263` is issued without a corresponding `$HASP9207` message, this is normally a tuning concern. Increasing the LOCKOUT parameter can relieve the issue. The LOCKOUT parameter reflects the time between a member being denied access to the shared queues and the time the controlling member is assumed to be down. The `$HASP263` commands is issued when this time allotment is met. The key consideration for LOCKOUT is to tune it in such a way that it indicates a real issue. Start high, and then lower it if there are problems and you do not see the messages when you wanted to. You change the LOCKOUT parameter using the `$T MASDEF` command.

JES2 checkpoint allocation guidelines

Access to checkpoint data is governed by HOLD and DORMANCY settings across the MAS. These are specified in hundredths of a second. DASD I/Os are now in the region of one to two milliseconds with cached controllers and FICON attachment. This technology assists the checkpoint useful time. However, it should be noted that functionality such as parallel access volumes (PAV) and multiple allegiance (MA) provide no benefit to checkpoint performance due to the locking mechanism that is used to access checkpoint data.

As the checkpoint data set is frequently referenced, particularly in MAS configurations, it should not reside on any volume that contains high-usage data sets. For example, performance can be seriously degraded if you allow your checkpoint data set to share the same volume as that of your spool. Only low-usage data sets (if any) should be allocated on the same volume as the checkpoint data set.

The coupling facility is faster for read operations than cached DASD but slightly slower for writes when comparing the JES2 checkpoint I/O operations. However, connectivity speed is not the main benefit. Placing a checkpoint on a coupling facility structure ensures equal access among MAS members and provides a better use of system resources than those available through DASD. JES2 uses the coupling facility lock to serialize access to the checkpoint data set. This coupling facility lock is better than the hardware RESERVE and RELEASE macros that are required for DASD because it ensures all members fair access to the checkpoint through FIFO queuing. The coupling facility lock affects only the checkpoint data set, while the hardware RESERVE macro locks the entire volume upon which the DASD checkpoint resides. The more members added to the MAS, the bigger the benefit of the coupling facility structure.

In the coupling facility configuration, it is the cross-system extended services (XES) serialized list processing that manages an ordered FIFO request queue for obtaining the lock needed to satisfy the checkpoint requests. There is a lock table in the coupling facility structure associated with a serialized list. The lock table indicates the lock holder, and in the absence of any contention, the lock in the CF is sufficient and no additional management is needed. When contention for the lock occurs, the coupling facility detects the contention and invokes XES to manage the contention. XES then manages the queue of requests in a FIFO order.

> **Tip:** We generally recommend that a CF structure be adopted for the primary checkpoint (CKPT1) when you have five or more members in your MAS.

When you review the allocation of checkpoint data, your interests are not solely performance related. The prime objective is availability. You should adopt dual or duplex mode. Duplex mode must always be established when using a coupling facility structure. We do not recommend placing both checkpoints on coupling facility structures or placing the primary checkpoint on DASD and the secondary checkpoint on a coupling facility structure. If both checkpoints reside on coupling facilities that become volatile (a condition where, if power to the coupling facility device is lost, the data is lost), your data is less secure than when a checkpoint data set resides on DASD. If no other active MAS member exists, you can lose all checkpoint data and require a JES2 cold start. Placing the primary checkpoint on DASD while the secondary checkpoint resides on a coupling facility provides no benefit to an installation.

Dual mode can add an additional 10% in JES2 processor cycles. Dual mode might use more processor, but it generates less I/O and less data transfer then duplex mode.

> **Note:** All combinations of checkpoint data sets on DASD and coupling facility structures are supported as potential error recovery scenarios through the checkpoint reconfiguration dialog.

Checkpoint allocation summary

You can be use the values in Table 1 as starting points for checkpoint allocation.

Table 1   Starting values for checkpoint allocation

Members	CKPT1	CKPT2	MODE
1	DASD	DASD	DUPLEX
2	DASD	DASD	DUAL
3	DASD	DASD	DUAL
4	DASD	DASD	DUAL
5+	CF	DASD	DUPLEX

Coupling facility structure duplexing

The introduction of coupling facility structure duplexing with CFLEVEL12 on zSeries® processors with z/OS 1.4 and above gives recommended availability benefits. There is no significant performance overhead to JES2 itself, and it will avoid the operational use of the JES2 checkpoint reconfiguration dialog.

However, there are other overheads to be aware of with the enablement of CF structure duplexing (z/OS, CFCC, and CF link utilization). For more information, refer to:

http://www.ibm.com/servers/eserver/zseries/library/techpapers/pdf/gm130103.pdf

JES2 Spool considerations

Spooling is the process by which the system manipulates its work. This includes using storage on DASD as a buffer storage to reduce processing delays when transferring data between peripheral equipment and a program to be run. The spool also refers the direct access device that contains the spool data sets. Spooling is critical to maintain performance in the z/OS-JES2 environment. JES2 attempts to maximize the performance of spooling operations, which ultimately benefits the throughput of work through the JES2 installation. After JES2 reads a job into the system, JES2 writes the job, its job control language, its control statements, and its data to a spool data set until further processing can occur.

It should be noted that access to the spool is not wholly governed by the checkpoint. You only need to serialize on the checkpoint if you modify the content of one of the queues. Many JES2 processes do not require checkpoint or require it very infrequently. Examples of this are printing, JOB execution, and NJE transmissions. Consequently, concurrent read and write activity are constantly requested to the spool data sets.

JES2 SPOOL volume allocation

You can define from 1 to 253 spool volumes to use as JES2 spool. This is established by the SPOOLNUM parameter on SPOOLDEF. Over allocating this value will give a slight performance overhead as the job queue elements (JQE) control block increases by four bytes for each additional 32 spool volumes.

JES2 allocates spool space for a job by dividing each spool volume into track groups. A track group consists of one or more DASD tracks. To ensure spool space is not wasted, performance is not degraded, and sufficient spool space is catered for, take care with the following parameters of SPOOLDEF:

- ▶ BUFSIZE, which defines the length of each record
- ▶ TGSPACE, the maximum number of track groups required before the next COLD start
- ▶ TGSIZE, the number of buffers in a track group
- ▶ TRKCELL, the number of spool records on a spool track

> **Tip:** Keep the default of BUFSIZE=3992. This maximizes DASD use as well as central and virtual storage usage. TRKCELL=4 is a good setting for 3380 and 3390 devices with this BUFSIZE setting. However, refer to the following for specification of these values: *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

For best performance, dedicate spool volumes (that is, do not share a volume with paging data sets or other data sets). With multiple spool devices, they should target different channels and control units on separate channels. The channel need not be dedicated, however, because JES2 channel use is low.

Formatting spool volumes

The `$S SPL,FORMAT` command results in extremely high channel utilization, but this is only a temporary condition which lasts for the duration of spool formatting. Consequently, this should be performed at a quiet time.

Choice of DASD device

JES2 spool performance can be significantly enhanced with the latest DASD technology. Storage subsystems such as the Enterprise Storage Server® (ESS) that introduced the 2105 control unit with benefits such as the following:

► FICON Express attachment with 2 Gbps link speeds.

► PAVs can be either static or dynamic (managed by WLM), and the benefits include:
  – Multiple UCBs per logical volume
  – Reduction in IOSQ time
  – Multiple users or jobs with multiple accesses to the same logical volume
  – Simultaneous reads and writes
  – Serialization of writes to the same data set extent

► MA. Benefits include:
  – Serialized, simultaneous access (reads and writes) of logical volumes by multiple hosts
  – Reduced device busy delays (PEND time)

► Large cache controllers and non-volatile storage providing quick channel access to data and high speed buffering of DASD reads and writes.

JES2 is still a single extent data set and has a very low read-to-write ratio. Prior to z/OS 1.4, PAVs would help with the reads but caused serialization on the writes. This resulted in IOSQ time being traded for PEND time. With z/OS 1.4 and higher, IBM® introduced a change in the way the contention to the spool data set with an input/output block extension (IOBE) is managed. This facilitates concurrent read and write to track groups within the same data set extent. As a consequence, the use of PAVs and MA will significantly benefit your spool performance and should be exploited.

Spool partitioning

Spool partitioning (or fencing) allows track group allocation across explicit volumes. Standard JES2 processing allows the allocation of track groups across any available spool volume. By fencing volumes, you can improve JES2 performance. Frequently, run system jobs can be isolated on separate volumes leaving the remaining volumes to be exploited by user-defined work. This can realize performance improvements for sysout intensive batch jobs. However, in general terms, performance will be better with FENCE=NO. Jobs may access all volumes, so increasing pathing capability and reducing the impacts of IOSQ and PEND time.

The main benefit of fencing is availability. Without it, the loss of a spool volume can result in the loss of all jobs. With it, jobs can be limited to one volume and isolation of the failure. With the common usage of RAID DASD, spool fencing might seen less beneficial with data striped across multiple physical volumes. However, this only caters to the physical residence of the data. There are potential failure scenarios at UCB level that continue to qualify the availability benefits of fenced volumes.

Spool fencing is enabled by the FENCE parameter on the SPOOLDEF initialization statement. Exit11 and Exit12 facilitate masks to limit access to the volumes based on jobname or jobclass. Alternatively, fencing can be used in association with spool affinity (see the next section, Spool affinity) to control the member selection criteria for fenced volumes.

Spool affinity

Spool affinity is an alternative (or addition) to fencing and provides a mechanism to logically partition the spool. This gives availability benefits. It facilitates the split of a large DASD spool pool into smaller pools for critical systems. It also provides a mechanism to have spool space online and ready to use if spool volumes start to fill up.

Spool affinity never stops spool space from being allocated. Affinities are used when resources are plentiful. The affinity is system based, not job based. Jobs may go through JES2 phases on different systems (for example, input, conversion, and output) with different spool SYSAFFS values. Such jobs can receive space from more volumes than if the job spends its life on a single system. This isolation can prove beneficial for systems with a tendency to be output intensive, so protecting the other members of the MAS.

> **Note:** Spool affinity is not controlled by a JES2 initialization statement. It is enabled when the spool volume is formatted ($S) or modified ($T):
>
> ```
> $S SPOOL(nnnnnn),FORMAT,SYSAFF=(memname,memname,memname)
> $T SPOOL(nnnnnn),SYSAFF=(memname,memname,memname)
> ```
>
> Multiple SYSAFF values can be coded as shown. The default is SYSAFF=ANY.

Further details of spool fencing and affinity, see *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

WLM managed initiators

The installation can choose between batch initiators managed by Job Entry Subsystem (JES) and batch initiators by job class managed by WLM. Both types of initiators can coexist. With WLM managed batch initiators, it is WLM who controls the number and placement of the initiator address spaces.

New initiators are started in circumstances such as the following:

► Service class goals are missed
► A system is under utilized and there are jobs waiting to be selected
► Jobs have an affinity to a system on which no initiator is available

Decision factors are:

► The available processor and memory resources of the system
► The service class importance
► The projected net value on overall goal achievement

Prior to z/OS 1.4, WLM stops initiators when the number of started initiators is 1.5 times of the long term average queue length, when a system runs short of processor or memory or when the last initiator was inactive for one hour.

With z/OS V1.4 and later, additional functionality is introduced. WLM improves the balancing of WLM managed batch initiators between the systems of a sysplex. On high use systems, the number of initiators can be reduced while more are started on low use systems. This enhancement improves the performance of the sysplex with better use of the processing capacity of each system. WLM attempts to distribute the initiators across all members in the sysplex to balance the use of the systems while taking care that jobs with affinities to specific systems are not hurt by WLM decisions. When the available processor capacity of a system decreases to less than 5%, WLM stops an initiator address space when the current system is observed as the system with the highest processor demand and when there is another remote system that has enough available resources to start a new initiator. This evaluation is done every 10 seconds. The order of decrease is to stop initiators serving lower importance service classes first.

Note: WLM has no concept of processor use, only the use of each z/OS image. Your processor might be running at 100% with images still managing WLM initiators that are running at less than 95%. In this instance, WLM takes no action on the image in question.

WLM increases the number of initiators on lower use systems. When large numbers of jobs are awaiting execution, up to five initiators can be started at once. This is compared to the increment of one that was used before z/OS 1.4.

Scalability

If you have many active MAS members and large batch processing overheads, it is best to understand the watermarks of JES2 scalability. In essence, the topics discussed in the following sections define the boundaries you must adhere to before you are forced to split the MAS.

Jobs queue and job output elements

The current limits for these control blocks are:

► The limit on the number of job queue elements (JQEs) is 200 000.
► The limit on the number of job output elements (JOEs) is 500 000.

Note: If you need to increase the number of JQEs beyond the current limit, you might want to ask the IBM support for a USERMOD that will allow your installation to increase the JOBNUM to 400 000. The Usermod will require a JES2 cold start.

These limits define the maximum number of each element that can be active in the MAS at any one time. Warnings are issued with the `$HASP050` message when these limits hit the warning thresholds you have set. See "Resource shortages" on page 17 for further details. Automation routines should react to these messages and purge non-critical output on a regular basis to ensure that limits are not breached.

Spool limits

The limit for JES spool space is a maximum of 253 volumes, each of 64 KB tracks in a single MAS. Note that JES2 spool volumes have more cylinders that are addressable. Because spool volumes are read by a number of vendor products (including SDSF), any change to how spool addressing operates affects these products. Specifically, JES2 supports both relative and absolute track addressing for spool.

The current spool addressing scheme uses an absolute track address. Because only two bytes are allocated for the track portion of the address, no spool data set can cross the 64 KB track boundary on a volume. By allowing the use of relative track addressing, JES2 can support a spool data set of up to 64 KB tracks in size, anywhere on a volume. This limit uses a track address that is relative to the start of the spool data set. The 64 KB track size limit is in part due to the limits on allocating a data set extent.

Note: With z/OS 1.7, the limit of a 64 KB extent is increased to one million tracks.

For more information regarding the establishment of relative spool addressing, refer to the SPOOLDEF initialization statement in *z/OS JES2 Initialization and Tuning Reference,* SA22-7533.

General scalability tracking

As your environment grows, you should understand how this growth affects the limitations of key operating system components. To track scalability of JES2 limits, use the **$JDHISTORY** command to display the history of resource usage and processor statistics over time. This data provides high and low use thresholds of the requested resources. The key elements are JQEs, JOEs, and track groups. This information should be used as capacity planning data to gauge whether you have significant growth and need to take any action. A good use of the command is

```
$JDHISTORY(JQES,JOES,TGS),HOURS=24
```

This command makes use of the filters introduced with APAR OA06186. You should use this command on a regular basis. You can store the data in spreadsheet format and portray it graphically to emphasize growth. Example 3 shows an example of the output of this command.

Example 3   Scalability reporting with $JDHISTORY command

```
$JDHISTORY(JQES,JOES,TGS),HOURS=24
$HASP9130 D HISTORY
$HASP9131 JES2 JQES     USAGE HISTORY
DATE     TIME      LIMIT    USAGE     LOW      HIGH  AVERAGE
-------- -------- -------- -------- -------- -------- --------
2005.020 12:00:00  20000     2555     2550     2555     2553
2005.020 11:00:00  20000     2550     2548     2551     2550
2005.020 10:00:00  20000     2548     2544     2548     2546
2005.020  9:00:00  20000     2544     2540     2544     2542
2005.020  8:00:00  20000     2540     2540     2541     2540
2005.020  7:00:00  20000     2540     2539     2540     2539
2005.020  6:00:00  20000     2539     2539     2540     2539
$HASP9131 JES2 JOES     USAGE HISTORY
DATE     TIME      LIMIT    USAGE     LOW      HIGH  AVERAGE
-------- -------- -------- -------- -------- -------- --------
2005.020 12:00:00  30000     2504     2498     2504     2502
2005.020 11:00:00  30000     2498     2496     2498     2498
2005.020 10:00:00  30000     2496     2494     2496     2495
2005.020  9:00:00  30000     2494     2491     2494     2492
2005.020  8:00:00  30000     2491     2491     2491     2491
2005.020  7:00:00  30000     2491     2490     2491     2490
2005.020  6:00:00  30000     2490     2490     2490     2490
$HASP9131 JES2 TGS      USAGE HISTORY
DATE     TIME      LIMIT    USAGE     LOW      HIGH  AVERAGE
-------- -------- -------- -------- -------- -------- --------
2005.020 12:00:00  52045    17819    17812    17819    17817
2005.020 11:00:00  52045    17812    17808    17813    17812
2005.020 10:00:00  52045    17808    17803    17810    17806
2005.020  9:00:00  52045    17803    17795    17803    17798
2005.020  8:00:00  52045    17795    17795    17796    17795
2005.020  7:00:00  52045    17795    17794    17795    17794
2005.020  6:00:00  52045    17794    17794    17795    17794
```

On this system, you can see that JQEs have been limited to 20 000 using JODBDEF, JOEs have been limited to 30 000 using OUTDEF, and there is a maximum of 52 045 track groups defined.

JES2 command processing

You should use caution when using the following JES2 commands because they can impact performance:

► JOB list commands (`$L JOB`). When using this command to display information pertaining to a large range of jobs, started tasks or TSO users.

► Commands that operate on the entire job or output queue (for example, `$TJQ(xx)`). With the availability of 200 000 jobs and 500 000 output elements, these commands can take a significant amount of time to complete and have an overhead to performance. Commands such as these should be broken up into ranges, such as:

    $TJQ(0-20000), $TJQ(20001-40000)

► Display history of JES2 resource usage (`$JDHISTORY`). This command has significant overhead because of the number of messages that can be issued and the console flooding that can occur. APAR OA06186 provides filters for use with this command to limit the overhead.

Identifying issues with JES2 performance degradation

One of the reasons this study was done is because JES2 provides no internal performance monitor. It is often difficult to determine if there is a JES2 performance issue. However, by looking at external processes, you can make judgements regarding JES2 performance and its ability to get to the checkpoint data set. In cases such as printers pausing, slow commands, or slow initiation of work, it is often lack of access to the checkpoint that is causing the delay.

The JES2 channel programming for the checkpoint is unlike typical user I/O. As such, much of the information from a resource measurement facility (RMF™) DASD report is often a misleading indicator of true JES2 checkpoint performance. Whether an RMF report shows the checkpoint as performing well is often immaterial to actual delays that are caused by waiting for access to the checkpoint. The same is not true of JES2 spool I/O. This channel programming is relatively straightforward, and the RMF DASD reports are generally good indicators of performance.

The following points characterize symptomatic evidence of a JES2 performance issue:

► Printers pause between data sets.
► TSO submit, cancel, and status commands are slow.
► TSO logons are delayed.
► Remote job entry lines time out.
► JES2 Waiting for Checkpoint message (`$HASP263`).
► Jobs take longer to run.
► JES2 uses more processor cycles.
► JES2 is unresponsive.
► Queues of jobs awaiting conversion, output, or purge are growing.
► JES2 commands take a while to begin or never end.
► Started tasks are slow to begin.
► `$HASP050` short of resources messages.

Identifying the rogue member of the MAS

In instances of heavy JES2 resource and processor use, it can be that one member of the MAS is the primary contributor to the activity. To identify the most active member or members, the simplest approach is to establish which has the most checkpoint activity (locks). To do so, you can use the following command:

```
RO ALL,$D PERFDATA(CKPTSTAT)
```

This command requests checkpoint statistics from all systems in the sysplex. Example 4 shows the output for one of these systems.

Example 4   JES2 checkpoint statistics

```
RESPONSE=SC64
 $HASP660 $DPERFDATA(CKPTSTAT)
 $HASP660 CKPT PERFORMANCE STATISTICS - INTERVAL=
 $HASP660 1212:32:35.265556,AVGHOLD=0.024942,AVGDORM=
 $HASP660 1.008878,TOT$CKPT=4574691,WRITE-4K=58387,
 $HASP660 WRITE-CB=0,OPT$CKPT=0,OPT4K=4225464,
 $HASP660 IO=R1,COUNT=4222075,AVGTIME=0.005975,
 $HASP660 IO=R2,COUNT=3634,AVGTIME=0.007285,TOTAL4K=
 $HASP660 28935,TOTALCB=0,
 $HASP660 IO=PW,COUNT=536265,AVGTIME=0.006529,TOTAL4K=
 $HASP660 24236,TOTALCB=0,
 $HASP660 IO=IW,COUNT=0,AVGTIME=0.000000,TOTAL4K=0,
 $HASP660 TOTALCB=0,
 $HASP660 IO=FW,COUNT=4222076,AVGTIME=0.004824,TOTAL4K=
 $HASP660 58387,TOTALCB=0
```

The value that you are interested in is TOT$CKPT. This value reflects the number of checkpoint accesses in that interval. This data should be compared for all systems to establish which member has the most activity.

After you have identified the rogue member, you might want to identify in which process (PCE) it is spending the most time by using the following command:

```
$D PERFDATA(CPUSTAT)
```

Example 5 on page 17 shows a subset of the output from this command. If one particular PCE is shown to be far busier than others, then you can balance the associated work across more members of the MAS to spread out the activity. You can find a list of PCEs, their names, and a description of their function in *JES2 Messages*, SA22-7537.

Example 5   JES2 processor statistics

```
$D PERFDATA(CPUSTAT)
$HASP660 $DPERFDATA(CPUSTAT)
$HASP660 CPU PERFORMANCE STATISTICS - INTERVAL=
$HASP660 1212:54:43.316277,CPU=25:03:05.456549,
$HASP660 PCENAME=CKPT,CPU%=98.70,CPU=24:43:38.310735,
$HASP660 TIME=29:01:05.132676,QSUSE_TIME=3.694285,
$HASP660 IOCOUNT=17433569,CKPT_COUNT=4240420,
$HASP660 PCENAME=XCFCMND,CPU%=0.68,CPU=10:16.585161,
$HASP660 TIME=12:16.381269,QSUSE_TIME=0.000000,IOCOUNT=
$HASP660 0,CKPT_COUNT=0,
$HASP660 PCENAME=TIMER,CPU%=0.08,CPU=1:14.690645,TIME=
$HASP660 1:24.822034,QSUSE_TIME=0.000000,IOCOUNT=0,
$HASP660 CKPT_COUNT=0,
$HASP660 PCENAME=MCON,CPU%=0.05,CPU=47.956397,TIME=
$HASP660 55.647866,QSUSE_TIME=0.004460,IOCOUNT=28,
$HASP660 CKPT_COUNT=192,
$HASP660 PCENAME=RESOURCE,CPU%=0.04,CPU=36.878716,TIME=
$HASP660 39.979062,QSUSE_TIME=0.000000,IOCOUNT=0,
$HASP660 CKPT_COUNT=0,
$HASP660 PCENAME=SNF,CPU%=0.00,CPU=3.889295,TIME=
$HASP660 4.379272,QSUSE_TIME=1.109888,IOCOUNT=0,
$HASP660 CKPT_COUNT=0,
$HASP660 PCENAME=HOPE,CPU%=0.00,CPU=1.322633,TIME=
$HASP660 1.728924,QSUSE_TIME=1.305222,IOCOUNT=11057,
$HASP660 CKPT_COUNT=56276,
```

Resource shortages

JES2 is very vocal in reporting shortages of its critical resources. You can monitor the **$HASP050** message for shortages in buffer pools and other resources by adjusting the warning thresholds to the level at which you want to be notified. Table 2 provides a list of the JES2 resources and the corresponding initialization statements that you can adjust.

Table 2   JES2 resources and relative initialization statements

Resource	Initialization statement
BSCB (BSC TP buffers)	BELOWBUF on TPDEF
BUFX (extended local buffers)	EXTBUF on BUFDEF
CKVR (checkpoint versions)	Number on CKPTDEF
CMBs (console message buffers)	BUFNUM on CONDEF
LBUF (logical buffers)	BELOWBUF on BUFDEF
JNUM (job numbers)	RANGE on JOBDEF
JQE (job queue elements)	JOBNUM on JOBDEF
JOE (job output elements)	JOENUM on OUTDEF
NHB (NJE header buffers)	HDRBUF on NJEDEF
SMFB (SMF buffers)	BUFNUM on SMFDEF
TG (spool track groups)	TGNUM on SPOOLDEF
TTAB (trace tables)	TABLES on TRACEDEF
VTMB (VTAM® buffers)	EXTBUF on TPDEF

Performance tools

JES2 has no internal or external performance monitor. You can use the following tools for analysis of performance-related data.

Resource Management Facility

The Resource Management Facility (RMF) provides useful reports or online analysis of JES2 performance. When operating in a MAS, note that you should summarize the statistics to give the complete picture. RMF monitor I and monitor III both provide useful reports, such as the following:

- ► RMF Coupling Facility report for analysis of delays when the checkpoint is resident on a coupling facility.
- ► DEVICE Activity report for analysis of I/O delays to devices that are allocated by JES2.
- ► CACHE Activity report for analysis of issues with JES2 devices on cached DASD.
- ► Subsystem report which provides an menu entry for JES2 subsystem delays.

The PERFDATA command

JES2 provides a command to display internal performance statistics (**$D PERFDATA**). The output of this command is provided *as is*. It is tailored for use by IBM JES2 development and can be requested as part of diagnostic procedures.

A summary of the displays is as follows:

- ► Checkpoint statistics:

  `$D PERFDATA(CKPTSTAT)`

- ► Processor statistics:

  `$D PERFDATA(CPUSTAT)`

- ► Event statistics:

  `$D PERFDATA(EVENT)`

- ► Initialization statistics:

  `$D PERFDATA(INITSTAT)`

- ► PCE statistics:

  `$D PERFDATA(PCESTAT)`

- ► QSUSE use and delay (this is a JES2 service used for checkpoint acquisition):

  `$D PERFDATA(QSUSE)`

- ► WLM sampling data:

  `$D PERFDATA(SAMPDATA)`

- ► JES2 subtask statistics:

  `$D PERFDATA(SUBTSTAT)`

- ► To reset the performance data statistics:

  `$TPERFDATA(*),RESET`

For further information regarding the use of the **$D PERFDATA** command, see:

`http://www.ibm.com/support/techdocs/atsmastr.nsf/Web/Techdocs`

JES2 monitor commands

With z/OS® 1.4, the JES2 monitor was introduced. The monitor runs in its own address space. The name is JES2MON, where JES2 is the name of the subsystem that is being monitored.

The JES2 health monitor is intended to address situations where JES2 is not responding to commands and where it is not possible to determine the issue. There are many possibilities about the cause of the issue. Some examples are:

► A JES2 command that is taking a long time to complete
► An error in a JES2 module or exit
► Checkpoint hangs

This monitor is not intended to be a performance monitor. However, it does provide useful information that you can use in performance analysis.

All health monitor commands have the JES2 command prefix, followed by a letter *J*. All commands that have the JES2 command prefix followed by a *J* are sent to the monitor command subtask. If the monitor does not recognize the command, it is routed to the JES2 address space for normal command processing.

The available commands are:

► `$JDSTATUS`, displays current status of JES2.

► `$JDJES`, displays information about JES2.

► `$JDMONITOR`, displays monitor task and module status information.

► `$JDDETAILS`, displays detailed information about JES2 resources, sampling, and MVS™ waits.

► `$JDHISTORY`, displays history information. Use caution with this command because it provides a significant amount of spooled output. APAR OA06186 provides filters for use with this command to limit the overhead.

► `$JSTOP`, stops the monitor. (JES2 restarts it automatically within a few minutes.)

JES2 traces

There are numerous JES2 traces that can help with performance analysis. In essence, running these traces does not have any significant overhead to the operation of JES2. So, you can run them with confidence. One exception is if your environment has substantial SYSOUT API (SAPI) requests (from an independent software vendor or home grown product). Tracing adds to this overhead. However, even in these circumstances, the impact is minimal.

You also have the option to turn on tracing without the trace logging function using the following command:

```
$T TRACEDEF,ACTIVE=YES,LOG=START=NO
```

This command places trace entries into buffers that are stored in ECSA but without the overhead of writing them to spool. This is especially useful for intermittent performance issues where the environment might go for hours without seeing a problem. Rather than always writing to spool, it simply writes to storage buffers and takes a dump when the JES2 performance issue is encountered. You can then analyze the trace information that is in the dump.

Noted that TRACE ID 17 is the only trace with a reduction program. When using any of the other traces, you must analyze the raw formatted data to determine what is occurring.

Because the primary concern is performance issues, the number of requests, the source address space, and the time to complete the requests are usually the first indicators of a problem.

The following are the most commonly used traces that are used for JES2 performance analysis:

► TRACE ID 17

The most commonly used trace for analyzing checkpoint performance problems. There is a sample reduction program supplied in SYS1.SHASSAMP(JES2T17A) which converts the trace entries into report format.

► TRACE ID 20

A $#GET performance trace. If output processing overhead is high, it could be because poor work selections are set.

► TRACE ID 27

A processing sysout (PSO) performance trace. This is particularly useful where a PSO application is in use and experiencing problems.

► TRACE ID 29

Particularly useful for analyzing sysout API (SAPI) performance problems. It traces all SAPI requests made to JES2 and the responses received. Often, it is the total number of trace entries written that highlights the problem.

► TRACE ID 30

A $#POST performance trace. This can be combined with trace ID 20 to get a picture of the general overhead of output selection overhead.

► TRACE ID 31

A $QGET trace which is particularly useful for job selection processing. If you are experiencing delays in jobs getting to execution, this trace can help identify the cause of the delays.

The person who wrote this Redpaper

Iain Neville is a zSeries Technical Specialist with IBM United Kingdom. He has 14 years of experience in zSeries and S/390® hardware and software technical support and consultancy. His areas of expertise include Parallel Sysplex®, z/OS, FICON, and zSeries hardware solutions. His responsibilities include pre-sales zSeries technical consultancy that supports numerous large financial institutions across the United Kingdom.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

This document created or updated on March 14, 2005.

Send us your comments in one of the following ways:
- ► Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- ► Send your comments in an email to:
  redbook@us.ibm.com
- ► Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server®	zSeries®	Parallel Sysplex®
@server®	Enterprise Storage Server®	RACF®
Redbooks (logo) ™	FICON®	RMF™
ibm.com®	IBM®	S/390®
z/OS®	MVS™	VTAM®

Other company, product, and service names may be trademarks or service marks of others.