JES3 White Paper

Written by: Edward E. Jaffe

Last updated: March 3, 2006

Contents

Introduction

IBM's Job Entry Subsystem (JES) is a required and strategic part of the z/OS operating system. IBM offers two JES choices: JES2 and JES3. JES3 is the considered the premium choice and incurs additional license fees.

Fifteen years ago, the differences between the JESes were more obvious than they are today. JES3 was originally developed to assist installations with the need to manage multiple MVS images.

Today, JES3 functions such as multi-system consoles, automatic tape sharing, dynamic initiators and workload balancing can be provided by the operating system itself and are, therefore, available to installations running JES2. This has left some JES3 installations wondering whether the premium they pay in licensing fees to run JES3 is still worthwhile.

The purpose of this document is to provide an up-to-date list of JES3's intrinsic features and an analysis of the associated additional licensing costs in order to help installations justify their JES3 strategy.

Every attempt has been made to present only factual information in this document. It has been reviewed by numerous individuals, including representatives from IBM's Washington Systems Center in Gaithersburg, Maryland. However, software environments and capabilities change over time and some of the information presented here may become incorrect or obsolete. If you believe you have discovered an error in this document, please e-mail the author at edjaffe@phoenixsoftware.com.

JES3 to JES2 Functional Comparison

Overview

At a high level, JES2 and JES3 perform very similar functions. They read jobs into the system, manage initiators, process output, purge jobs from the system, and so forth. However, there are significant differences in their processing paradigms, especially noticeable when running a multi-image configuration (referred to as a JESplex in this document).

JES2 processing is considered to be *independently controlled*. Each JES2 image processes its own job input, job scheduling and job output.

JES2 uses a contention-based paradigm for managing a multi-image workload. Each image "wakes up" periodically and attempts to serialize the JES2 checkpoint using a hardware serialization technique. The image that succeeds carries out job and output scheduling activities. The others go back "to sleep" waiting for another chance. There is no cooperative decision-making process. The order in which activities occur and on which images is random.

By contrast, JES3 processing is considered to be *centrally controlled*. One image is designated as the focal point for the entry and distribution of jobs and for the control of resources needed by the jobs. That image, called the global processor, distributes work to itself and the other images in the configuration, known as local processors. It is from the global processor that JES3 manages jobs and resources for the entire JESplex, matching jobs with available resources.

JES3 manages processors (images), I/O devices, volumes, and data. To avoid delays that result when these resources are not available, JES3 ensures that they are available before selecting the job for processing. Using a centralized approach, JES3 can make informed and complex decisions about job scheduling and placement.

JCL Processing

Detecting JCL Errors

When Job Submitted to Execute

JCL errors are detected by two different processes. The first process is known as JCL conversion. The second process is known as JCL interpretation. Most JCL errors are detected during this latter process.

Both JES2 and JES3 perform JCL conversion soon after a job is placed into the job queue.

JCL interpretation in a JES3 environment occurs immediately after conversion, providing instantaneous feedback when a JCL error is detected.

JCL interpretation in a JES2 environment doesn't occur until after the job is selected and placed into an initiator for execution. In an environment in which job selection takes time, this can negatively impact user productivity.

Figure 1 depicts this difference in JCL processing.

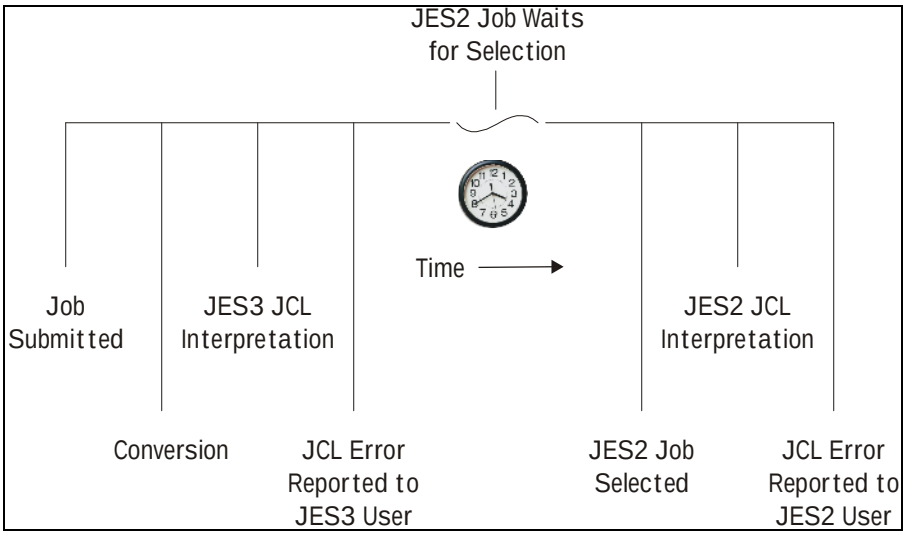

Figure 1 – Detecting JCL Errors

When Job Submitted for JCL Syntax Checking Only

When TYPRUN=SCAN is specified on the job card, the job is checked for JCL errors only. It is not initiated or executed.

Because JES2 does not call the MVS interpreter, it cannot check the JCL for parameter value errors and excessive parameters. Bad specifications, such as DISP=(,CATALG), will not be detected by JCL scanning in a JES2 environment. Such errors will be detected only after the job is selected and placed into an initiator for execution.

Because it invokes the MVS interpreter, JCL submitted to JES3 with TYPRUN=SCAN will receive parameter value and excessive parameters checking along with all of the other processing performed by JES2. Bad specifications, such as DISP=(,CATALG), will be detected immediately and can be corrected by the programmer before the job is submitted for execution.

Offloading JCL Processing

Most JES3 installations make use of Converter/Interpreter Functional Subsystems. A C/I FSS runs in its own address space and performs both JCL conversion and interpretation functions. The advantages to using C/I FSS address spaces are:

- C/I FSS address spaces offload much of the overhead caused by converter/interpreter processing from the JES3 address space. A dispatching priority (or goal) can be established for a C/I FSS that is lower than the priority assigned to JES3. This allows the high-priority JES3 address space to concentrate on important functions such as job scheduling (crucial to overall system throughput), without being impacted by relatively expensive but less important JCL processing.

- The MVS converter and interpreter can consume large quantities of the Scheduler Work Area (SWA) and storage in subpool 0. The use of C/I FSS address spaces provides considerable virtual storage constraint relief for the JES3 address space.
- Because it is a dedicated to a single purpose, a C/I FSS can process more jobs at a time than the JES3 address space can handle. There is no practical limit on the number of C/I FSS address spaces that may be defined to execute simultaneously in the JESplex, though it's likely an installation will define no more than it actually needs to effectively handle its peak job submission workload.

JES2 provides no way of offloading or lowering the priority of the JCL processing tasks it performs. All MVS conversion processing occurs within the JES2 address space itself. (As discussed earlier, MVS interpretation in a JES2 environment is not performed until a job is placed into an initiator. Offloading this aspect of JCL processing is not relevant).

Device and Data Set Allocation

JES2 performs no device or data set allocation of its own. It relies strictly on MVS allocation. A job's requirements are not known until JES2 selects the job for execution, and a system initiator begins the step allocation process. At each job step, MVS allocation attempts to satisfy the requirements for the step, in contention with every other job step currently executing on the same image or sysplex. If the requirements cannot be met, MVS allocation gives the operator the option of canceling the job or allowing it to wait for resources. Thus, in a JES2 environment, there may be jobs executing and other jobs waiting for resources.

The jobs waiting in MVS allocation hold critical resources (a system initiator, an address space, data sets, and possibly devices). Holding these resources longer than necessary makes it difficult to determine how many initiators should be started to keep the system fully utilized, because at any given time, an unknown number of initiators may be waiting in MVS allocation. Often, JES2 installations use ENQ monitors to detect lengthy contention situations and cancel one or all of the jobs involved in the conflict.

MVS allocation occurs for one job step at a time. If a data set is not found, a JCL error occurs and the job is terminated. Such a JCL error could occur hours after the job starts executing, creating unnecessary surprises and can be an impediment to user productivity.

With JES3 pre-execution setup, the resources (data sets, devices, and volumes) that a job requires are already set up when the job is passed to MVS for execution. There should never be an idle initiator caused by a job waiting for these resources. Setup occurs while a job is in the JES3 address space, and the only system resource used while the job is waiting is the JES3 queueing space. JES3 helps the system make maximum use of devices and allows jobs to run in a minimum amount of

time once they are passed to the system for execution. In addition, JES3 allocates devices for jobs with higher priorities before jobs with lower priorities. This helps ensure that high-priority jobs will not be delayed for resources held by low-priority jobs.

JES3's locate processing ensures all data sets referenced by a job exist before the job is scheduled. If any data sets are missing, a JCL error is reported. Locate processing occurs as part of converter/interpreter processing. This helps ensure jobs do not run for hours before ultimately failing with a JCL error. Once a job has undergone C/I processing, the submitting user can feel confident the job will not fail due to a JCL error, regardless of when it finally executes.

JES3's main device scheduler requests and verifies the mounting of the initial volumes a job requires on each device (e.g., tape devices) before a job can be selected for execution (unless deferred volume mounting is specified in the JCL). This further delays unnecessary use of an initiator until the operator has had time to find and mount the necessary tapes.

JES3 device setup is available only for devices that are defined in JES3's initialization stream. Devices that are unknown to JES3 will be allocated using normal MVS allocation as in JES2.

JES3 data set setup is available for SMS-managed data sets, whether or not the devices are defined in the JES3 initialization stream. JES3 uses an SMS callable API to perform setup for SMS-managed data sets. SMS ensures that JES3 knows which systems are eligible to run a job, based on the SMS volume and storage group status for any SMS-managed data sets required by the job.

Job Class Management

JES3 allows for up to 255 job classes. JES3 job class names may be up to eight characters in length, allowing for mnemonic, self-documenting names like PRODWORK, DEVWORK, IMSA, etc. to be used. JES3 job classes may be grouped together into up to 255 job class groups. Job class groups allow operators and system programmers to define and control many job classes in a single operation.

JES3 also provides job class constraints. These optional parameters influence the way jobs are scheduled. TDEPTH limits the total number of jobs in a class that will be scheduled simultaneously. TLIMIT allows the number of jobs running in one class to affect the number of jobs that can be scheduled in another class. These constraints are applied at a JESplex level. MDEPTH and MLIMIT are similar to TDEPTH and TLIMIT, but are applied to select z/OS images.

JES2 allows only 36 job classes (A-Z, 0-9) to be defined. Each job class is only one character in length. One class constraint (maximum execution count) was added with the WLM-managed batch initiator support. This parameter is similar in function to JES3's TDEPTH.

System Log Management

JES3 produces a JESplex-wide, merged system log, known as the DLOG.

JES2 produces a unique system log for each image in the JESplex, making chronological archiving and problem determination more difficult.

The Sysplex Operations Log (OPERLOG), a sysplex-wide merged system log, intended to replace the traditional system log, is infrequently used because:

- Browsing OPERLOG is slow when compared to browsing the traditional system log.
- OPERLOG is sysplex-wide in scope. When multiple JESplexes exist within a sysplex, log entries from other JESplexes are interleaved with log entries from the current JESplex.
- For a multi-image sysplex, OPERLOG requires the use of a coupling facility (CF). That is, OPERLOG will not work in a multisystem base sysplex.

Initiator Management

JES2 provides only static initiator management. Initiators are defined and controlled on an individual image basis.

JES3 provides JESplex-wide dynamic initiator management. Initiators are controlled indirectly by changing policy parameters used by JES3's generalized main scheduler (GMS).

Workload manager (WLM) batch initiator management is intended to alleviate many of the difficulties associated with planning and controlling JES initiators. Its main benefit to JES2 installations is dynamic initiator management itself. For JES3, the primary benefit is the introduction of real-time workload analysis to dynamic initiator management.

Even when using WLM-managed batch initiators, JES3 provides superior functionality. For example:

JES2 provides no way to limit the number of jobs initiated on any particular image. JES2 installations typically resort to assigning a default WLM scheduling environment to every batch job entering the system. By manipulating the state of a resource referenced by every scheduling environment, they can achieve the result of completely protecting an image from WLM-managed batch job initiations.

JES3 handles this problem much more directly. WLM-managed batch initiators for a job class group will not be started on any image where the job class group is disabled. Disabling a job class group on an image can be done with a single command. To make the change permanent, the initialization stream can be updated to indicate no execution resources should be created for that job class group on the specified image.

If an "all-or-nothing" approach is not acceptable, flexible rules for influencing WLM-managed batch initiations may be specified using JES3 class constraints, described in the section entitled "Job Class Management."

Workload Balancing

It is competition, not cooperation, which best describes JES2's shared spool algorithm. While a JES2 member holds the checkpoint lock, it attempts to schedule all eligible work until it runs out of available resources, there is no more work to be scheduled, or it has held the checkpoint lock for too long of a time. This algorithm can result in workload-scheduling imbalances.

For example, if there are ten idle initiators per system and ten jobs are submitted at the same time, those ten jobs will usually be converted and initiated on the same system, regardless of the available capacity of other systems in the JESplex. This pattern can be observed even when WLM batch initiator management is used. Remember, JES is responsible for assigning jobs to ready initiators. WLM simply controls the number of available initiators on each system for each service class. As long as there are enough idle initiators to satisfy the needs of the jobs waiting for execution, a JES2 image will attempt to initiate them all on the current system.

By contrast, JES3 distributes workload across the JESplex in a balanced manner. The decision as to where and when to schedule a job is made by JES3 running on the global processor, which is aware of exactly which jobs are currently running, the systems they are running on, and how many initiators (JES- or WLM-managed) are available on each system.

At a programming level, JES3's job scheduling logic is also competitive. When a job moves to the GMS select queue, the GMS Supervisors for each potentially eligible system are simultaneously posted (made available for dispatching by JES3's Multi-Function Monitor). The key difference is that the dispatching of the GMS Supervisor tasks is rotated by the MFM, within internal priorities established by a load-balancing algorithm, and each GMS Supervisor task selects only one job before giving up control to the MFM, thus allowing other GMS Supervisor tasks to be dispatched. In addition, the internal dispatching priority of a GMS Supervisor task is reset according to the load-balancing algorithm whenever a job is selected or an initiator is stopped or started on that system. (When only WLM-managed initiators exist, all priorities remain

equal so dispatching is strictly rotated.) This technique ensures that job initiations are properly balanced throughout the JESplex at all times.

In an attempt to address JES2's workload balancing issues, IBM made some changes to z/OS V1R4 WLM. JES2's propensity to over-initiate a single system can be throttled by reducing the number of initiators available to that system. z/OS V1R4 WLM is more aggressive in reducing the number of WLM-managed initiators on overworked systems. After this change, the same problems with JES2 exist, but their impact is reduced.

JES3 continues to provide better workload balancing both before and after z/OS V1R4.

Output Management

There is a great deal of overlap in the capabilities of the JESes in this area, though neither is a complete subset of the other. This is to be expected since output management is one of the most fundamental responsibilities assigned to JES and customer expectations with respect to output handling are similar in all environments.

Nevertheless, there are some significant differences. For example, JES3 provides the ability to change the number of copies that will be printed for a data set waiting on a queue, whereas JES2 does not. On the other hand, the Scheduler Facility Services SSI call (SSI 70), which allows system management utilities to modify the contents of SWB-maintained information such as Name and Address, is supported by JES2 and not by JES3 (though requirement MR0411002334 exists to address this issue for JES3).

The two most-important differences between the JESes, in terms of output management, are described in the following topics.

Output Grouping

JES2 manages output at a group level. An output group is a named resource, which may be referred to in operator commands, and may consist of any number of data sets with similar output characteristics. Changes made to existing output attributes are performed at the output group level. JES2 does not support changes to output characteristics at the individual data set level.

JES3 also manages output at a group level. However, JES3's groupings are not named resources. They are simply a convenient assemblage of data sets with like output characteristics. Because JES3's groups are not named resources, they are not referenced in operator commands. Instead, operator commands refer to output by data set name and/or the values of various output characteristics, such as destination, forms, or output class. A single operator command may change a single data set, multiple data sets within a job, or multiple data sets in a mixture of jobs.

After changing output characteristics for one or more data sets in a job, JES3 regroups the data sets as necessary. During this regrouping process, new groups may be dynamically created and existing groups may enlarged, reduced, or removed completely.

Output Attribute Inheritance

In JES3, many attributes for controlling output data set handling may be associated with a sysout class. Output created in a class automatically inherits those attributes. This capability can be useful for standardizing and automating output handling because the attributes are assigned at an installation level. This simplifies production JCL and allows changes to be made without user involvement.

For example, an installation could assign a destination of NEWYORK to sysout class N. Output created in class N would be automatically sent to NEWYORK for printing. There is no need for users to explicitly specify DEST=NEWYORK for output created in class N. If conditions changed such that class N printing was to be handled locally or at a node other than NEWYORK, only the definition for sysout class N in the JES3 initialization stream would be modified.

JES2 sysout class definitions do not include output-handling attributes. Only the queue type and disposition options may be associated with a sysout class. All output handling specifications are provided explicitly through JCL, dynamic allocation text units, or user exits.

Dependent Job Control

JES3 provides a means of coordinating the processing of jobs called Dependent job control (DJC). DJC allows jobs to be executed in a specific order, as determined by job dependencies. Job dependencies may occur because of data dependencies or may be defined to achieve better device utilization or to manage job streams. (DJC has an MVS counterpart for conditional execution of job steps within a single job.)

DJC requires no advance preparation with JES3 initialization statements. Application programmers specify all relationships between jobs on JES3 //*NET control statements they submit with the affected jobs. These relationships define the predecessor jobs that must complete before a job is scheduled for execution. Contingencies in case of predecessor job failure are defined as well.

A collection of dependent jobs is called a DJC network. A DJC network is a named entity within the JESplex and may be queried, modified, or canceled.

In a DJC network, some jobs may execute (serially or in parallel) while others remain unscheduled, waiting for predecessor job(s) to finish. A DJC job becomes eligible for scheduling when its hold count reaches

zero. As each job completes, it decrements the hold count for one or more dependent jobs in the DJC network. A job may also decrement the hold count for jobs in other DJC networks. Dependencies between DJC networks may be established in much the same way as job dependencies within a single DJC network. In addition, there is no requirement that all of the jobs comprising a DJC network be submitted together. The jobs may be submitted by different users, in any order, and at various times throughout the life of the DJC network.

Although small and medium installations can use DJC to effectively coordinate their production batch workloads, large installations typically do not. Rather, large installations normally use one of several well-known job scheduling software products for managing complex production batch activity. However, this does not mean DJC remains unused in those environments.

Generally, job scheduling software products are used for production batch jobs only. These products often require a significant amount of administration to define the jobs and their dependencies, an effort not usually warranted for non-production jobs. In addition, administration of these products is usually limited to production coordinators. For this reason, system programmers and end users often turn to DJC for their non-production batch job coordination needs.

JES2 provides no job coordination facilities. The most common solution for coordination of non-production batch jobs in a JES2 environment is to have the last step of one job submit another job by copying its JCL to an internal reader. In this way, jobs may be executed one-by-one using job-step-level condition code checking to determine if the next job should be submitted.

Deadline Scheduling

Deadline scheduling is a facility that allows users to schedule a job to execute at (or by) a certain time (wall clock time or number of minutes or hours relative to job submission time) on a specific date or periodically at a given time every week, month, or year.

The installation defines up to 36 deadline-scheduling algorithms (A-Z and 0-9) and communicates the behavior of these algorithms to its users. Deadline-scheduling algorithms specify such things as the initial priority change, the lead-time for the job (i.e., the amount of time prior to the specified deadline that the initial priority change will be made), and parameters such as priority increment and frequency of incrementation.

The user's JCL specifies the deadline time and date (or cycle relative to week, month or year) and indicates which deadline-scheduling algorithm is to be used.

When the lead-time prior to the deadline passes, the job's priority is set to the initial priority value. After that, the priority is incremented according to the aging parameters in the deadline-scheduling algorithm.

Typically, a deadline job is submitted into a held priority (e.g., priority zero). As the deadline approaches, the job's priority is raised to the initial value and then priority aged if not immediately selected for execution. Deadline scheduling works equally well for jobs intended for JES-managed or WLM-managed initiators.

Like DJC (discussed previously), deadline scheduling for production jobs can be provided through expensive commercial job scheduling software products. But the same limitations apply. Such products often require significant administration work and the capability of doing so is usually limited to production job coordinators. JES3 deadline scheduling is the obvious choice for system programmers and end users requiring scheduling of non-production batch jobs at a specific time.

JES2 provides no deadline scheduling facility.

Health and Performance Monitoring

Traditional monitors, such as IBM's Resource Measurement Facility, provide only limited value for tuning subsystems like JES2 and JES3 because they are unaware of specialized internal JES task and queue structures. In order to properly identify and correct bottlenecks in JES processing, the system programmer must perform monitoring using JES-provided facilities.

JES3 provides three monitoring facilities to assist the system programmer in identifying problem situations:

The first is the JES3 Loop and Wait Monitor. This monitor is always present and ensures the two JES3 main tasks are not looping or in an unintended MVS WAIT. If a main task loop or wait is detected, JES3 issues a WTOR allowing the operator to a) wait for a specific interval before taking further action or b) terminate the thread (drive its JESTAE routine) and give control back to the MFM.

The second is the JES3 Monitor DSP. This monitor can be set to examine specific resources and queues at specific intervals. For example, you could specify that you want to monitor the C/I FSS queue every minute for jobs that have been waiting for two minutes or more. JES3 starts the Monitor DSP and monitors various queues and resources automatically. If any of the conditions you (or JES) specify are detected, operator messages are issued. Unlike the Loop and Wait Monitor, the Monitor DSP does not directly provide the operator with a repair action for the reported anomalies.

The third is the JES3 Monitoring Facility (JMF). JMF samples JES3 resources and queues, gathers and stores raw information, statistically

analyzes the information, and creates printed reports, SMF records, or both at specific intervals. The reports created by JMF can greatly assist the JES3 system programmer to monitor, diagnose, and tune a JES3 system or JESplex. Resources monitored by JMF include spool data management, CPU and storage, internal JES3 functions, device scheduling, and job throughput.

Monitoring in JES2 is performed by a single facility known as the JES2 Health Monitor. Unlike JES3 monitors, which were designed to run as part of the JES3 address space, the JES2 Health Monitor runs as a separate address space and all sampling occurs using cross memory services.

When the JES2 Health Monitor detects a problem, such as a suspected loop or wait in the main task, the operator is alerted. However, no repair facility is provided. JES2's Health Monitor is strictly an information tool.

The JES2 Health Monitor continually gathers statistics similar to those gathered by JES3's JMF. Operator commands exist to display the current hour or full 72-hour history of resource utilization and CPU sampling. Unfortunately, the reports are directed only to consoles and/or the system log. No printed reports are created. No SMF records are produced.

Systems Management

Users in a JES2 environment generally use IBM's SDSF to view and control jobs and other system resources. Likewise, users in a JES3 environment generally use PSI's (E)JES to perform similar functions in their environments.

Many of the differences between these products can be attributed to differences in the underlying capabilities of the two JESes. Other differences can be attributed strictly to the products themselves.

Some of their differences are described in the topics that follow.

Scope of View and Control

Both SDSF and (E)JES provide JESplex-wide scope for the information they display, including buffered sysout data from jobs running on other systems. (E)JES utilizes sysplex services (XCF and, in a parallel sysplex, XES) to provide this functionality. SDSF's implementation requires IBM's MQSeries for z/OS to connect SDSF servers within the JESplex. This implementation has several drawbacks:

- XCF is an integral part of the z/OS operating system. There are no additional costs associated with its use. By contrast, MQSeries is a relatively expensive product. Inspection of IBM software pricing at the time of this writing reveals prices for MQSeries to be between six and eight times those for SDSF, depending on processor capacity. It is

unlikely that customers, with no enterprise-wide need for MQSeries, could justify licensing it based solely on SDSF's requirements.

- XCF signaling performance is better than MQSeries.
- The use of MQSeries involves significant additional configuration work for the system programmer. MQSeries must be configured to allow SDSF to create and use the necessary queues. The security product must be updated to protect the queues and grant end users (SDSF clients) proper access to them. Worst of all, SDSF requires that you configure *server groups*, a list of system and associated server names in your ISFPRMxx member, so that all of the necessary pipes can be created. The system programmer must essentially redefine the JESplex all over again for SDSF, and keep this information synchronized with JES2's definitions when the configuration changes. A well-designed XCF implementation should require no additional configuration work.

Usability

There are numerous usability differences between the two products. A discussion of these differences is beyond the scope of this document.

Product Extensions

The (E)JES Extract Post-Processor package provides product extensions such as those for transmitting copies of sysout to other users via NJE, ftp, or e-mail, and provides a framework for installations to write their own product extensions. SDSF has no equivalent. The only way to extend its capabilities is through direct modification of the code and/or through user exits.

Spool Data Sets

Partitioning

The JES2 spool is one large multi-volume repository. The installation has no control over where sysin and sysout data sets will be allocated, with the exception that spool fencing may be used to limit the total number of volumes on which a single job's spool records may be allocated.

The JES3 spool may be organized into partitions. Spool partitions may be assigned by z/OS image, job class, sysout class, or user exit decision. There is always a default partition. Named spool partitions may optionally designate another named spool partition (or the default partition) as an overflow partition, to be used in the event the original partition becomes full. Effective use of spool partitions can ensure a guaranteed amount of spool space is available to your most important work.

Architecture

Data Set Organization/Placement

To define spool volumes and data sets to JES2, a four-character DASD volume ID prefix is specified in the initialization stream. During initialization, JES2 searches all online DASD volumes in the system and compares their volume IDs with the specified four-character prefix. Any volumes matching the specification are then searched for a fixed data set name (normally SYS1.HASPACE).

This arcane approach is confusing and restricts the names of the volumes where spool data sets may reside. Since every spool data sets has the same name, only one may be allocated on any particular volume and only one of them may be cataloged. Familiar data set utilities (such as ISPF 3.4) cannot be used to easily list and process JES2 spool data sets.

In JES3, each spool data set is given a unique name and may be cataloged. As with any other z/OS product, the data set names – not the volume names – are used to locate the data sets. Spool data sets may appear on volumes of any name and more than one spool data set may be allocated on a volume. Familiar, data set utilities (such as ISPF 3.4) may be used to easily list and process JES3 spool data sets.

Addressing

JES2 uses a 4-byte value to identify a unique spool block within the configuration. The format of this value depends on the type of data set:

- For basic format data sets that use absolute addressing (the only kind prior to z/OS V1R2), the format is MTTR, where M is the 8-bit extent number, TT is the 16-bit track number relative to the start of the volume, and R is the 8-bit physical record (block) number within a track. These data sets must be contained within the first 65,535 tracks of the volume.
- For basic format data sets that use relative addressing (introduced with z/OS V1R2), the format is MTTR, where M is the 8-bit extent number, TT is the 16-bit track number relative to the start of the data set, and R is the 8-bit physical record (block) number within a track. These data sets may reside anywhere on the volume, but must not exceed 65,535 tracks in size.
- For large format data sets (introduced with z/OS V1R7), the format is MTTtr, where M is the 8-bit extent number, TTt is the 20-bit track number relative to the start of the data set, and r is the 4-bit physical record (block) number within a track. These data sets may reside anywhere on the volume and may be up to 1,048,574 tracks in size.

In all cases, the eight-bit **M** field limits the maximum number of JES2 spool data sets to 253 (the checkpoint data sets use the other two extents).

It's clear that JES2 is operating under an architectural constraint with respect to this spool address format. To support data sets larger than 65535 tracks, they were forced to reassign four bits from the record number field to the track field. The approach worked because the largest 3390 volumes in use today support only 984,040 tracks. Unfortunately, such fundamental changes will undoubtedly result in some trauma as they necessitate changes in IBM code, ISV code, and user-written code.

JES3 uses a 6-byte value to identify a unique spool block within the configuration. The format of each value is **MMRRRR**, where **MM** is the 16-bit extent number and **RRRR** is the 32-bit physical record (or block) number relative to the start of the volume. A spool data set may occupy space through block 2^{32}-1, relative to the start of the volume. This addressing scheme supports approximately 356 million tracks per DASD volume, assuming a 4K spool block size and a track size equal to today's 3390 devices. Such a device would have over 7,100 times the capacity of a 3390-3 (approximately 18.4 TB)!

The 16-bit **MM** field architecturally limits the maximum number of JES3 spool data sets 65,535. JES3 currently enforces an arbitrary limit of 1,024.

Job Queue Capacities and Limitations

z/OS V1R2 JES2 and JES3 were enhanced to support an increased job number range up to 999,999. The details of this support have served to highlight some of the differences between JES2 and JES3 in terms of job queueing capacity and architecture.

JES2 works with fixed limits established by the installation. The maximum allowable values are imposed largely by its checkpoint architecture. In z/OS V1R2 JES2, the absolute maximum number of jobs in the job queue is 200,000 and the maximum number of Job Output Elements (JOEs) is 500,000. The good news for JES2 installations is that these limits are considerably higher than those imposed by prior releases of JES2.

With z/OS V1R2 JES3, the maximum number of jobs in the job queue is 999,999, the same as the upper limit of the job number range and nearly five times JES2's upper limit. Other resources, such as Output Scheduler Elements (OSEs), are limited only by the amount of spooling space and virtual storage available to describe them. An installation need not concern itself with establishing and monitoring limits on these resources in a JES3 environment.

JESplex Member Names

JES3 always uses the MVS system name as the name of the member in a JESplex. This is the same name used by sysplex services, MCS console commands, and so forth. A system name may be 1-8 characters in length. JES2 is restricted to four-character JESplex member names. JES2 member names may be derived by applying a *substring* function against the true system names.

Network Job Entry

Network Job Entry (NJE) is supported by both JES3 and JES2. For many years, both have provided support for NJE over BSC and CTC lines as well as between SNA LUs. JES2's SNA/NJE support is self-contained whereas JES3 requires the MVS Bulk Data Transfer SNA/NJE feature to provide the transport mechanism for its SNA/NJE networking.

z/OS V1R7 JES2 provides support for NJE over TCP/IP. The same support is provided for JES3 in z/OS V1R8. This support allows both JESes to now interoperate with other spooling systems (e.g., VM RSCS and VSE/POWER) that have supported NJE over TCP/IP for some time now.

Network Topology

In NJE parlance, a directly connected or adjacent node is one that has a line connecting it to the current node. An indirectly connected or non-adjacent node is one that uses a directly connected node as an intermediate transmission path. Indirectly connected nodes are popular in BSC/NJE networks because the expense of providing direct hardware connections between every node in a large network can be prohibitive. Indirectly connected nodes tend to be used less frequently in SNA/NJE and TCP/IP networks since the connections between nodes are not dedicated hardware devices.

A JES3 node need not understand the topology of the NJE network. It simply needs to know the names and paths of each node that may exchange jobs and sysout with the current node. A path is the name of a directly connected node where jobs and sysout for an indirectly connected node should be sent. JES3's algorithm is path-based. This method of routing NJE work is similar to how routing works in an IP network; routing decisions are made on each host without any knowledge of the overall network topology.

Unlike JES3, JES2 networking uses a topology-based algorithm. A JES2 node must be made aware of the physical and logical connections that exist between other nodes in the network. This information allows the JES2 node to route jobs and sysout to the proper directly connected node. Topological definitions can be voluminous. To simplify them, JES2 provides a powerful component known as the Path Manager.

Unfortunately, the Path Manager may not be used to communicate with non-JES2 nodes such as VM RSCS, JES3, or VSE/POWER.

NJE Performance

JES2 NJE performance can suffer in a multi-image environment. This is because when one JES2 image queues an NJE-related message to another JES2 member in the same JESplex, there is no interrupt mechanism to inform the receiving member that a message exists. The receiving member periodically reads the shared job queue record and examines queue information for new messages.

In JES3, all NJE processing is fully event-driven.

Store-And-Forward

Store-and-forward describes what happens in most NJE networks when data is sent to an indirectly connected node. As NJE data is received at a node, it is stored in that node's spool space until the entire job is received. Once successful receipt is verified, the job is forwarded to the next directly connected node in the path. JES3 uses the traditional store-and-forward technique.

The JES2 Path Manager allows data sent between two indirectly connected JES2 nodes to bypass normal NJE store-and-forward processing (if desired) when all intermediate nodes are also JES2. Instead of storing the entire job at each intermediate node, the NJE packets themselves are routed through each intermediate node until they reach the destination node. The NJE data is stored only in the spool space of the sending node and destination node. This feature is useful when many indirectly connected nodes exist. In the author's opinion, this is one of JES2's most desirable features.

Commands from Remote NJE Nodes

JES2 can be told to accept virtually any command from a remote network node. This is helpful for coordinating startup, shutdown, and other activities from a central location.

JES3 accepts only a limited subset of commands from remote NJE nodes (though SHARE requirement SSJES300351 exists to address this issue for JES3).

Shutdown

To shutdown JES2 normally, all activities must be quiesced. This means all address spaces started under the JES2 subsystem (including Unix

System Services forked procedures and APPC transactions), readers, printers, punches, RJE and NJE lines, etc. must be terminated before JES2 will successfully shut down. It is sometimes difficult for operators to determine what activities are preventing a JES2 normal shutdown.

The biggest challenge presented by JES2's shutdown is in the area of automation. An experienced human operator will usually determine, by issuing several commands, what actions to take in order to facilitate a normal JES2 shutdown. Programming this decision-making process into automation can be tedious. As new scenarios arise, automation scripts to shutdown JES2 become increasingly complex.

JES2 provides alternative termination mechanisms that offer faster shutdowns, but with various caveats and risks. For example, JES2 can be abended or forced (though forcing JES2 is not recommended unless JES2's main TCB is not responding). Another alternative allows JES2 to be immediately terminated, but requires an IPL before restarting JES2.

For JES3, a normal shutdown always occurs immediately, and all JES3 services are suspended. Should the operator choose to restart JES3 without an IPL using any type of hot or local start, JES3 dynamically reconnects all suspended requesters. Automating a normal JES3 shutdown is trivial.

Customization

Both JES subsystems allow installations to customize job flow and other processing. Most customization is performed using IBM-provided user exits. For more complex customization needs, JES3 has an advantage because it allows installations to write their own DSPs. JES2 users wishing to provide additional functionality beyond that allowed by exits must generally modify the JES2 source code itself, though updates to table-pairs can often affect a change without any substantial coding effort.

Ensuring the correctness of user exits and other modifications presents a challenge, even for the most experienced system programmer. Program flow tracing and interactive debugging capabilities are essential. JES3 provides a DSP called "Dump Core" (DC) that provides these capabilities. Using DC, a system programmer can locate modules, set and reset breakpoints (traps) in JES3 or user-written code, inspect and alter main storage, and format storage- or spool-resident control blocks. There is no equivalent functionality in JES2.

JES3 to JES2 Application Availability Comparison

Planned Outage

The conditions that bring about a planned outage differ considerably between the JESes, making it difficult to predict the number of planned outages that will occur due to an installation's JES strategy choice.

For example, JES2 requires an outage to add or delete internal readers, initiators, converter tasks, and NJE nodes, whereas JES3 can add or delete these resources dynamically. On the other hand, JES2 can change SWA control block residency dynamically, whereas JES3 requires a hot start with refresh to perform this task.

JES3 initialization parameters are always JESplex-wide in scope and the vast majority of them can be updated over a hot start with refresh. This process entails a restart of the JES3 address space on the global processor only. Job scheduling and other global functions are suspended during the restart. Once JES3 is reinitialized, all functions resume normally and any updates are automatically propagated to the other systems in the JESplex. For those situations in which a JES3 warm start is required, a JESplex-wide IPL is required.

For JES2 initialization parameters that can be refreshed over a hot start, JES2 must be stopped and restarted on each system in the JESplex where the change is needed. Job scheduling and other JES functions are suspended during the restart, but only on the affected system(s). Once JES2 is reinitialized, all functions resume normally. To refresh JES2 initialization parameters requiring a single-member warm start, an orderly shutdown must first be performed. This requires that all jobs, started tasks, TSO users, NJE connections, printers, etc. be terminated in an orderly fashion (see "Shutdown"). Though technically unnecessary, many installations choose to IPL over a JES2 warm start in order to simplify operational procedures. If an all-member warm start is required, this process must be performed for all systems in the JESplex simultaneously.

Unplanned Outage

The recovery considerations after an unplanned system outage in a multi-image configuration are similar for JES2 members and JES3 local processors.

If a JES2 member fails while serializing the checkpoint, intervention may be required to allow the other systems in the JESplex to continue processing. This is not a consideration for JES3.

When an outage occurs on the JES3 global processor, job scheduling and other important JESplex-wide functions are inhibited. In a sense, this can be considered JES3's *Achilles' heel*. A Dynamic System Interchange

(DSI) is required to assign the role of JES3 global processor to another image in the JESplex, particularly if the outage is expected to be lengthy. This is not a consideration for JES2.

Detecting Errors in the Initialization Stream

JES3 provides a utility for preemptively checking the initialization stream. This utility detects syntax errors and many logical errors, and can be run as a batch job or in the foreground under TSO/E. Many installations use an ISPF edit macro to check the initialization stream directly from within an ISPF edit session.

JES2 provides no equivalent utility for checking the initialization stream. Most JES2 installations start a second copy of JES2 from the console to check for errors in the initialization stream. JES2 performs basic syntax checking of the initialization parameters prior to terminating.

Production Job Failure Recovery/Restart

When a production job fails half way through, the restart costs can be high – in terms of both time and effort. (It's not always possible to simply resubmit the job!) By checking for JCL errors and ensuring all needed data sets are available *before* the job is queued for selection, JES3 helps minimize the need for expensive 0'dark-thirty job restart analysis.

JES3 to JES2 Performance Comparisons

IBM does not provide any information comparing performance under JES2 and JES3. We had no choice but to perform our own analyses.

Two benchmarks were run in different years, using different operating system releases, and on different hardware. Each was run under controlled conditions (idle system time). No attempt was made to simulate real customer production batch workloads or JES configurations.

The first benchmark was run in 1999, in a parallel sysplex consisting of two OS/390 V2R7 systems running in equally weighted LPARs, with an ICMF LPAR on a 9672-R22 processor with a 9345 DASD Subsystem. The second benchmark was run in 2000, in a parallel sysplex consisting of two OS/390 V2R9 systems running in equally weighted LPARs, with an ICMF LPAR on a 7060-H30 processor with the S/390 Internal Disk Subsystem emulating 3390-3 devices.

In each case, both JESes were defined with virtually identical configurations and initialization specifications. The only exception was that JES3 setup processing was not disabled.

For JES2, we tested using both DASD and CF primary checkpoints and used HOLD=50,DORMANCY=50 for the JES2 checkpoint sharing parameters (JES3 has no equivalent as it is event driven).

OS/390 V2R7 Performance Comparison

Fixed Overhead Comparison

The first experiment was to observe the common storage, CPU, and I/O overhead associated with each subsystem when there was no activity. Each system was observed for 15 minutes and the data was extrapolated to hourly utilization.

The results are shown graphically in the following figures.

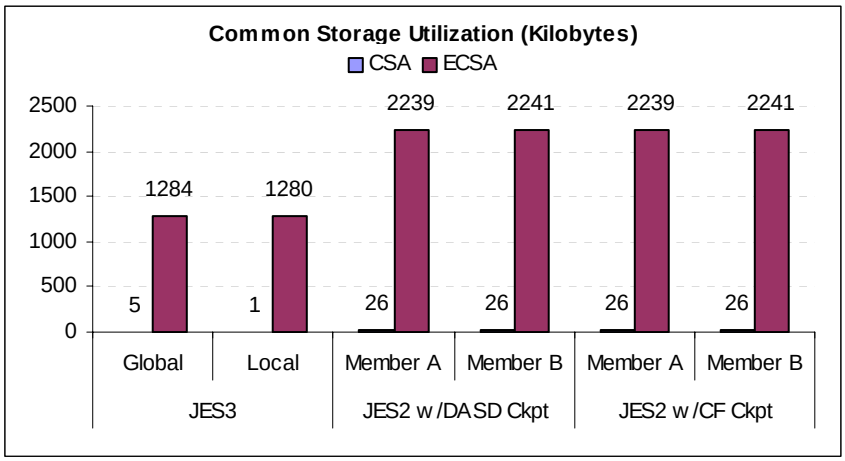

Figure 2 – OS/390 V2R7 Fixed Common Storage Overhead

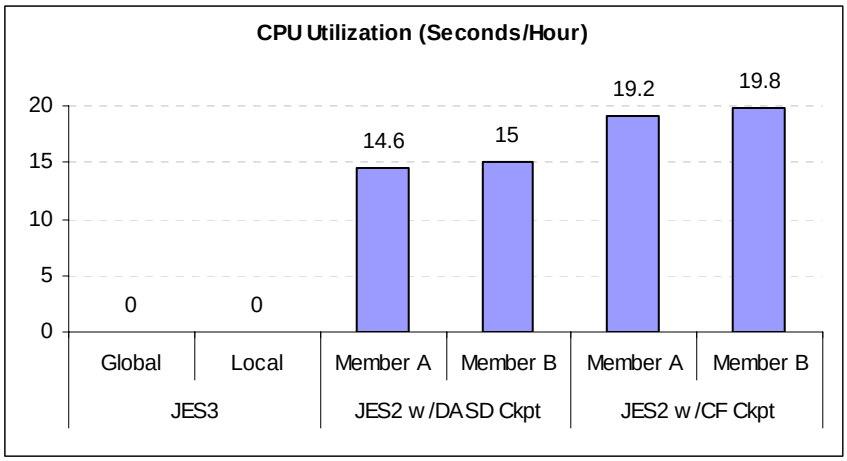

Figure 3 – OS/390 V2R7 Fixed CPU Overhead

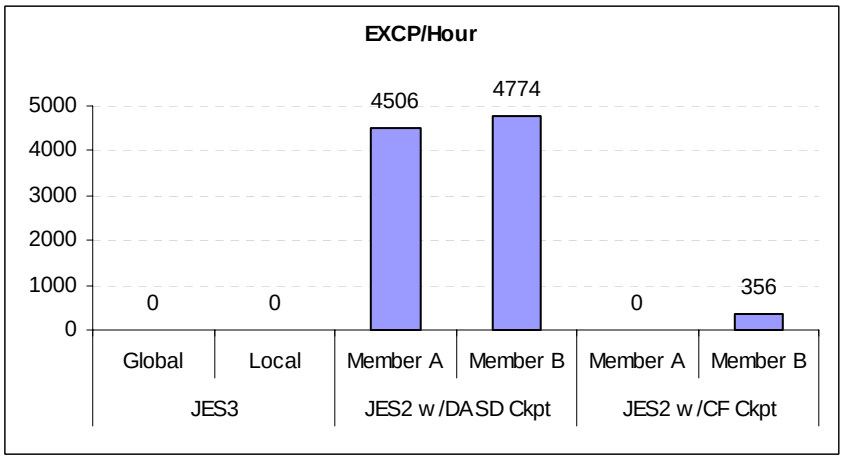

Figure 4 – OS/390 V2R7 Fixed EXCP Overhead

Job Processing Comparison

The second experiment measured throughput and resource utilization required for 100 trivial batch jobs submitted from a TSO/E session. The jobs wrote three thousand 132-byte records to sysout using IEBDG. Both JESes had ten pre-started initiators and two converter tasks (C/I DSPs for JES3) on each system.

When using WLM-managed initiators, we forced WLM to start exactly 20 initiators. This created a pool of initiators approximating the conditions of the pre-started JES initiators.

The results are shown graphically in the following figures.

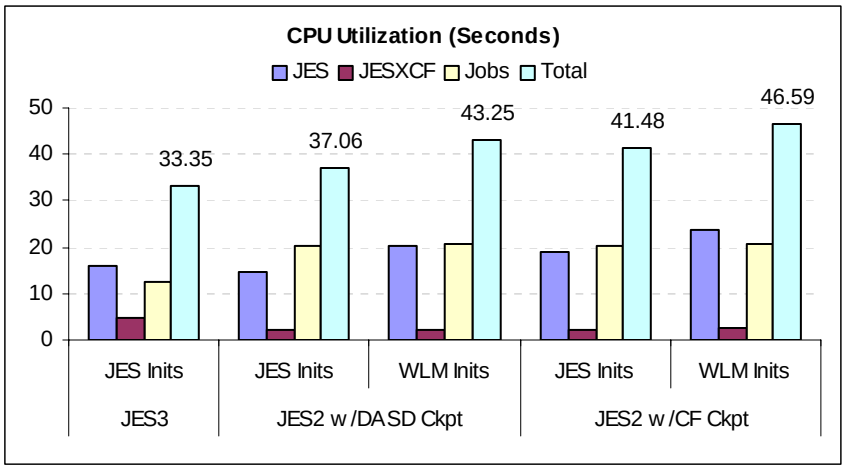

Figure 5 – OS/390 V2R7 CPU Utilization Running the Benchmark

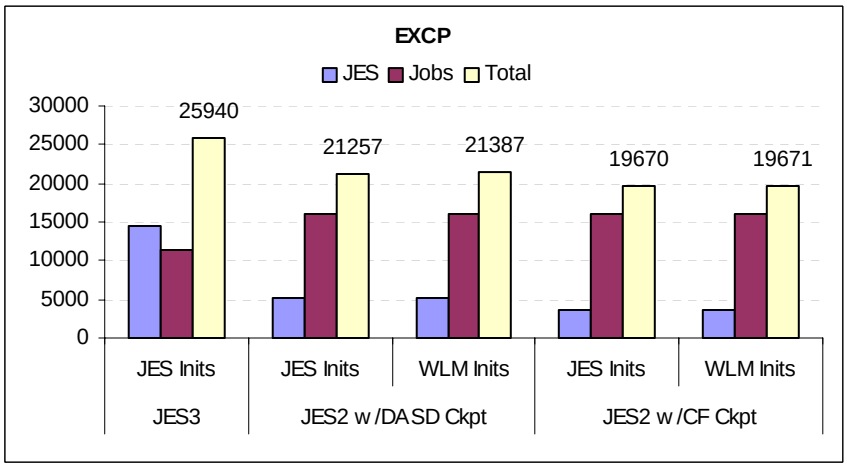

Figure 6 – OS/390 V2R7 EXCP Totals Running the Benchmark

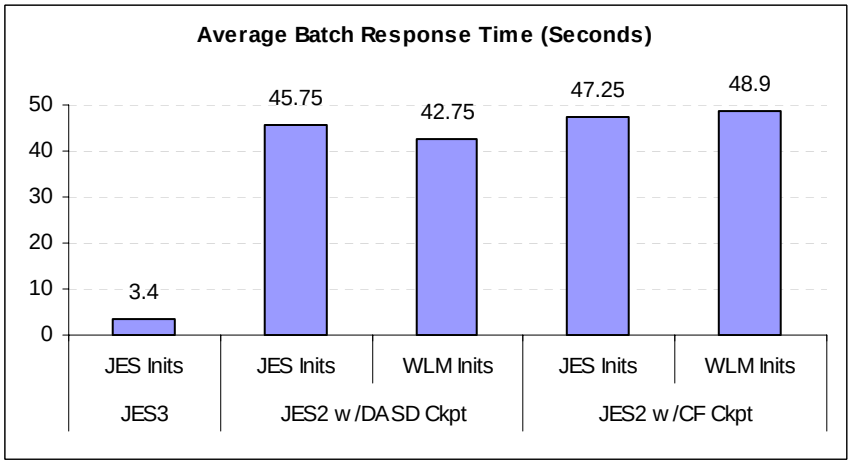

Figure 7 – OS/390 V2R7 Batch Response Times Running the Benchmark

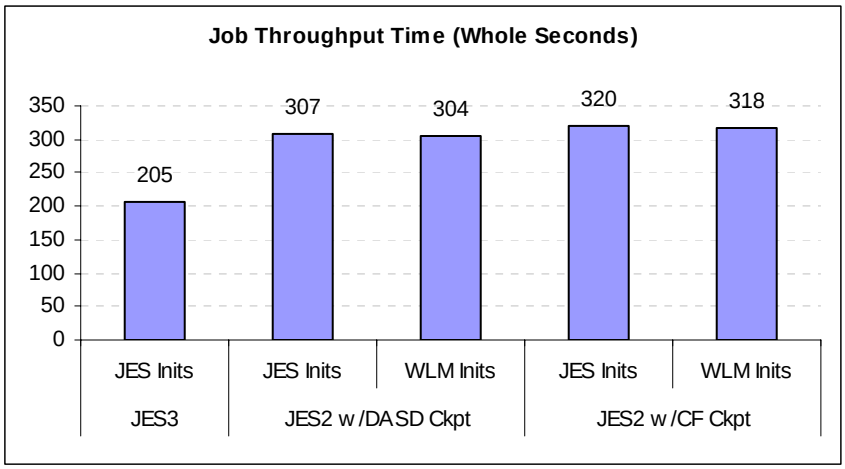

Figure 8 - OS/390 V2R7 Job Throughput Times Running the Benchmark

OS/390 V2R9 Performance Comparison

Fixed Overhead Comparison

The first experiment was to observe the common storage, CPU, and I/O overhead associated with each subsystem when there was no activity. Each system was observed for 15 minutes and the data was extrapolated to hourly utilization.

The results are shown graphically in the following figures.

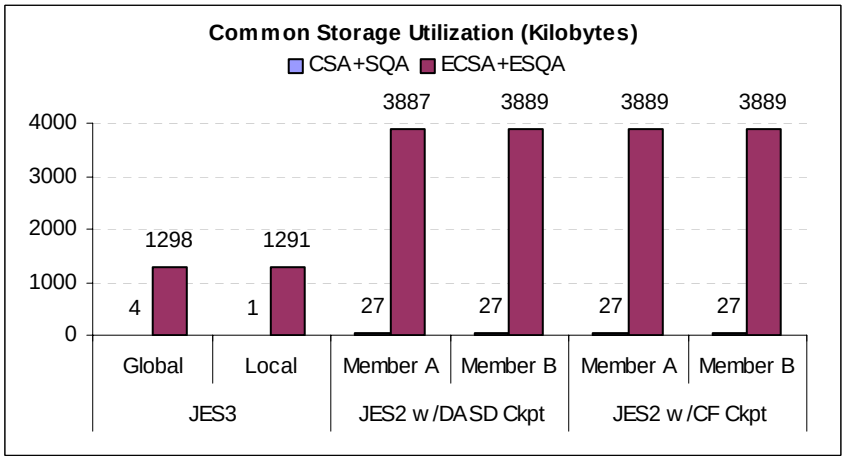

Figure 9 – OS/390 V2R9 Fixed Common Storage Overhead

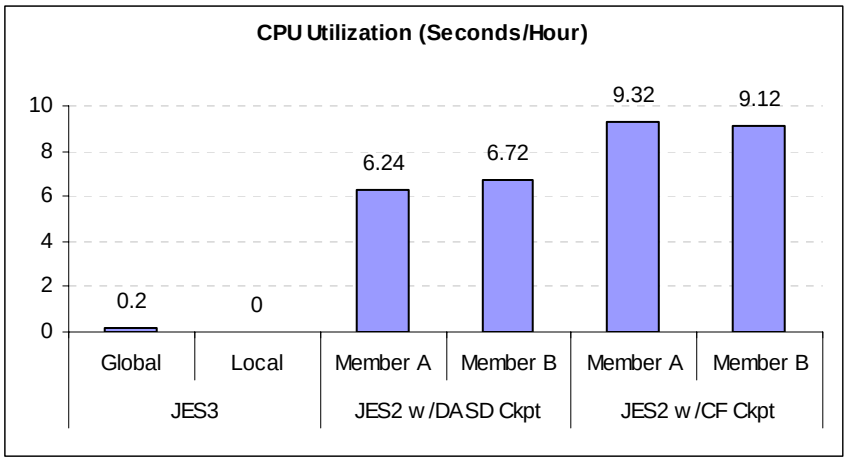

Figure 10 – OS/390 V2R9 Fixed CPU Overhead

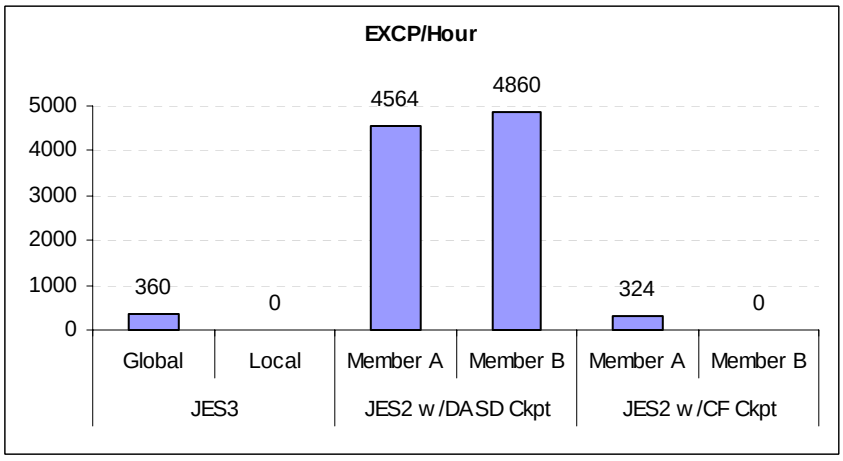

Figure 11 – OS/390 V2R9 Fixed EXCP Overhead

Job Processing Comparison

The second experiment measured throughput and resource utilization required for 100 trivial batch jobs submitted from a TSO/E session. The jobs wrote ten thousand 132-byte records to sysout using IEBDG. (The number of records written to spool was increased for this benchmark in order to obtain measurable response times for JES3 on the faster hardware). Both JESes had ten pre-started initiators and two converter tasks (C/I DSPs for JES3) on each system.

When using WLM-managed initiators, we forced WLM to start exactly 20 initiators. This created a pool of initiators approximating the conditions of the pre-started JES initiators.

The results are shown graphically in the following figures.

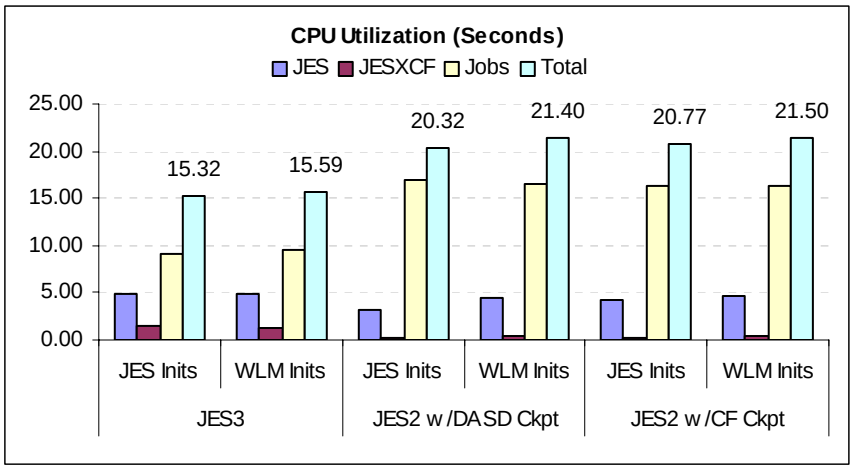

Figure 12 – OS/390 V2R9 CPU Utilization Running the Benchmark

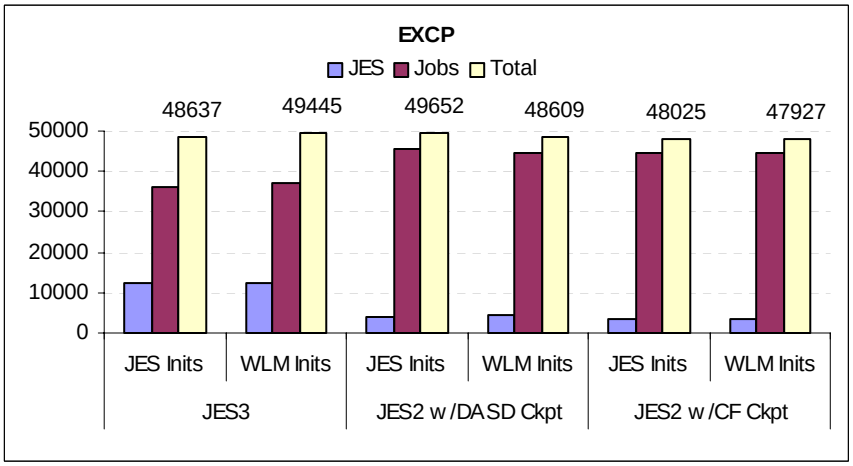

Figure 13 – OS/390 V2R9 EXCP Totals Running the Benchmark

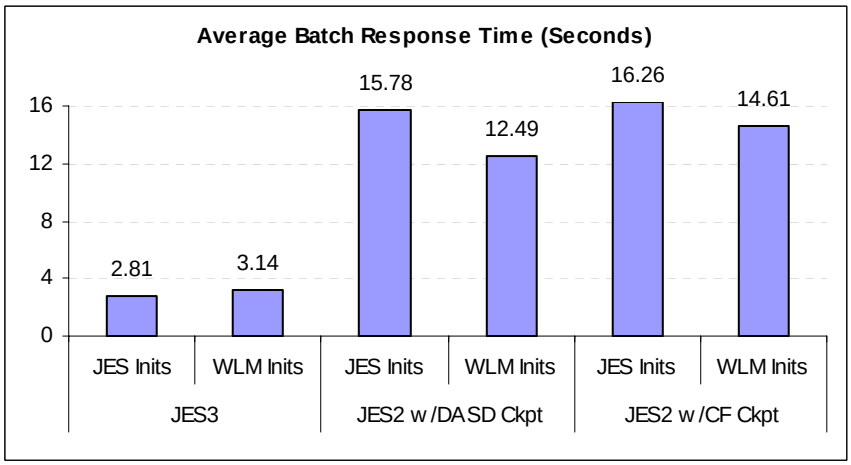

Figure 14 – OS/390 V2R9 Batch Response Times Running the Benchmark

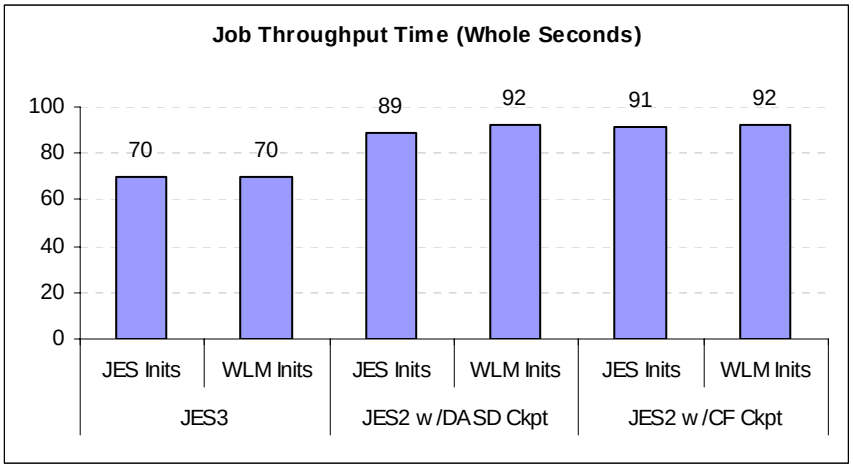

Figure 15 – OS/390 V2R9 Job Throughput Times Running the Benchmark

Performance Analysis

Fixed Overhead Analysis

When idling, JES2 consumes more of all of the resources we observed (common storage, CPU and I/O). The CPU and I/O consumption is mainly due to JES2's requirement for timed checkpoint access.

OS/390 V2R7 JES3 did not consume any CPU or I/O resources during the idle measurement. However, in OS/390 V2R9, the JES3 global processor performed a small amount of timed processing. This is probably due to the requirement for WLM sampling introduced with OS/390 V2R8 JES3.

Benchmark Analysis

In the first benchmark, total execute channel program (EXCP) operations for JES3 were 21 to 22 percent higher than JES2 w/DASD checkpoint, yet the work completed 33 percent faster using 10 to 23 percent fewer CPU resources. Total EXCPs for JES3 were 32 percent higher than JES2 w/CF checkpoint, yet the work completed 36 percent faster using 20 to 28 percent less CPU. Average batch response time under JES3 was 3.4 seconds, 92 to 93 percent faster than JES2.

In the second benchmark, total EXCPs for JES3 were almost identical to JES2 w/DASD checkpoint (ranging from 2 percent lower to 1.7 percent higher), and the work completed 21 to 24 percent faster using 23 to 28 percent fewer CPU resources. Total EXCPs for JES3 were 1.3 to 3 percent higher than JES2 w/CF checkpoint, and the work completed 23 to 24

percent faster using 25 to 29 percent less CPU. Average batch response time under JES3 ranged from 2.8 seconds (JES initiators) to 3.1 seconds (WLM initiators), 75 to 83 percent faster than JES2.

An analysis determined I/O contention was a primary contributor to the slower batch response times under JES2. JES2's spool access algorithm is not cooperative like JES3's and is highly susceptible to performance degradation due to I/O contention. The effects of this shortcoming can be seen clearly in our benchmark even though only 20 jobs were executing.

Though ESCON-attached via four paths and cached, the 9345 DASD used in the first benchmark did not have fast write capability. Earlier drafts of this document predicted that running the same tests on modern, fast-write-capable hardware would reduce the impact of JES3's superior I/O management algorithm. Now that a second benchmark has been run using the S/390 Internal Disk Subsystem (SSA, 40MB/second, full-duplex, 32MB fast write cache, two-level 64MB/128MB write-through cache), we see this prediction was indeed accurate. On the new hardware, JES3's total job throughput time advantage was reduced from 32-36 percent in the first benchmark to 21-24 percent in the second benchmark. Likewise, the batch response time advantage was reduced from 92-93 percent in the first benchmark to 75-83 percent in the second benchmark.

Adding additional spool volumes will reduce contention and allow more jobs to be executed in a given amount of time. Both JESes should benefit equally from the increased parallelism provided by additional spool volumes.

Throughput Analysis

Our benchmarks measured several different JES performance characteristics under controlled circumstances. Clearly, the most interesting data to be obtained from these measurements are the internal throughput rates. That is, the amount of work that can be "pumped" through the system over a fixed time interval.

We took the throughput results from the most recent benchmark and calculated the throughput rate in number of jobs per hour for each of the six different configurations. These throughput rates are expressed in relative terms (the highest rate being 1.000) in Figure 16.

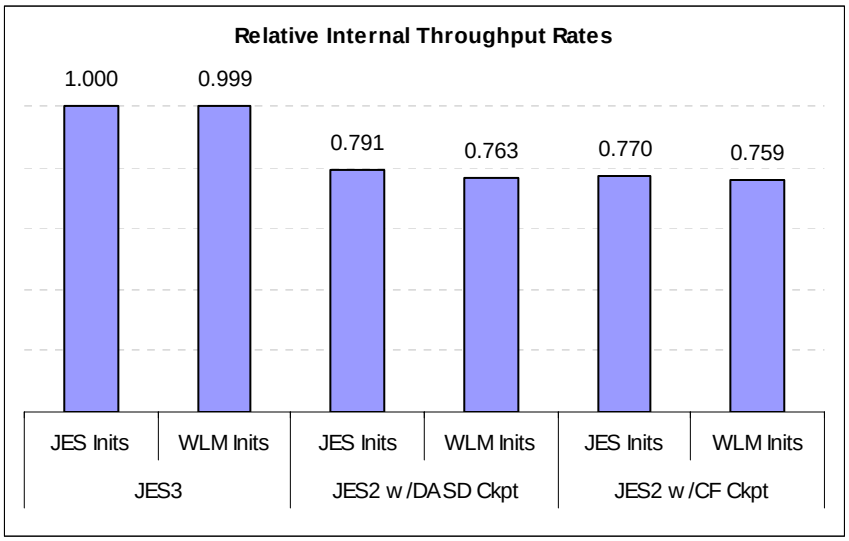

Figure 16 - OS/390 V2R9 Relative Internal Throughput Rates

There is no reason to assume that these JES performance differences should be limited only to batch jobs that produce sysout. It seems reasonable that the differences can, to some extent at least, be extrapolated to any JES spool I/O-related activities, such as printing on high-speed printers, interactively browsing spool-resident reports, or processing spool data via external writers or other facilities. Empirically measuring performance differences in these areas is left as an exercise for the reader.

JES3 to JES2 Comparison of Software Licensing Costs

JES3 is an optional, priced feature of z/OS. This license charge is approximately 4 percent of the z/OS base license charge. Multi-JESplex installations typically pay an additional 2½ percent of the base charge to license the optional SNA/NJE feature. For large installations, these charges typically amount to less than ½ of 1 percent of total mainframe software expenditures.

Most JES3 installations find JES3's superior system management, productivity, and performance characteristics more than justify the difference in licensing costs. Unfortunately, direct cost savings due to these factors are difficult to measure, and differ from one installation to another depending on configuration, usage patterns, and charge-back algorithms. For this reason, no attempt is made to quantify these "soft dollar" savings in this document. Nevertheless, these factors should be considered a major part of any JES3 strategy justification.

As JES2 installations become larger and more sophisticated, the lack of JES3 functionality in their environments becomes more problematic. This lack of function creates opportunities for some independent software vendors to provide JES3 functions to JES2 environments. For example,

JES2 installations wishing to extend their environments with pre-execution device and data set allocation awareness often license a product from MVS Solutions called *ThruPut Manager*. Licensing software products that bring JES3-like functionality to JES2 environments quickly erases any "hard dollar" differences between the JESes. In fact, nearly all of the installations we surveyed found that after carefully analyzing all of the costs involved, a JES2 strategy was actually more expensive than JES3.

The above-mentioned survey was performed in 1999. Once SDSF standardized on MQSeries for its JESplex-wide functions, the costs associated with a JES2 strategy increased substantially. At the same time, z/OS V1R8 provides "NJE over TCP/IP" functionality to JES3 at no extra cost, which reduces JES3 operating costs by eliminating the need for the (fairly expensive) SNA/NJE feature.

Summary

JES3 continues to provide robust functionality, much of which remains unavailable to the JES2 community.

JES3 provides end users with features that increase their productivity, even in a single-system environment.

JES3 balances batch workloads better than JES2 and helps lower costs associated with production job restarts.

JES3's architecture supports more jobs, more output, and "super-large" DASD devices that haven't even been developed yet.

JES3 is more resource efficient than JES2 and runs batch jobs and other spool I/O intensive activities faster.

JES3's costs are going down even as JES2's costs are increasing.

When considering only that JES3 is an optional, priced feature of z/OS, it appears a JES3 strategy may cost more than JES2. However, when all factors are considered, JES3 is often found to provide the better value.