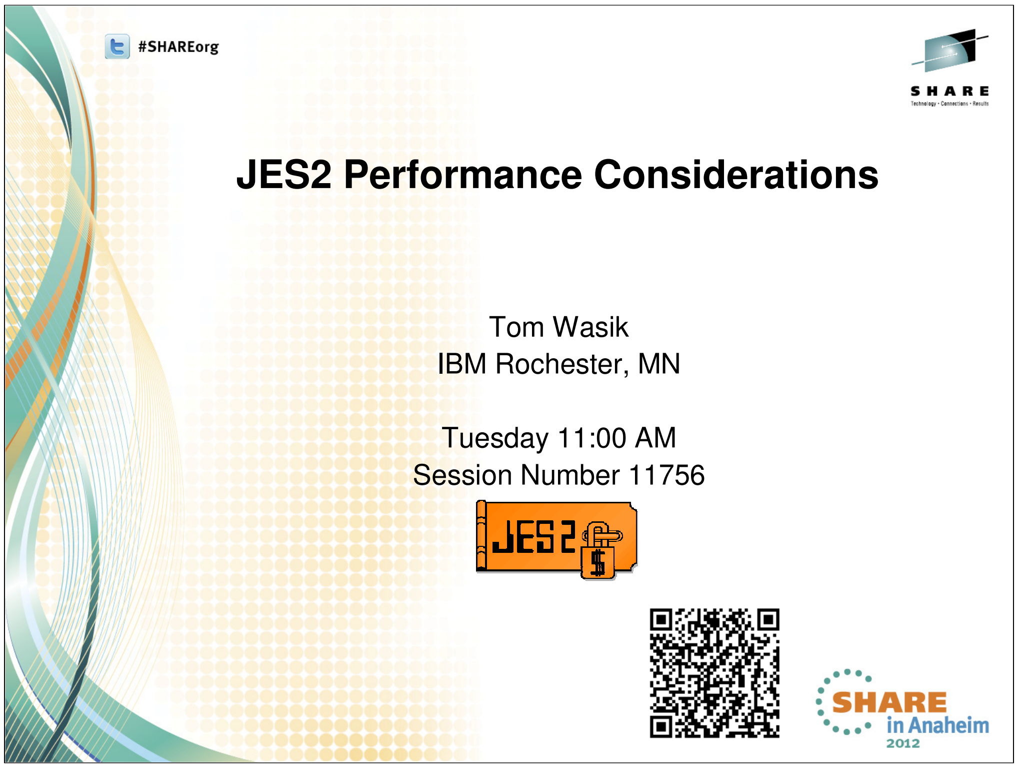


#SHAREorg

SHARE

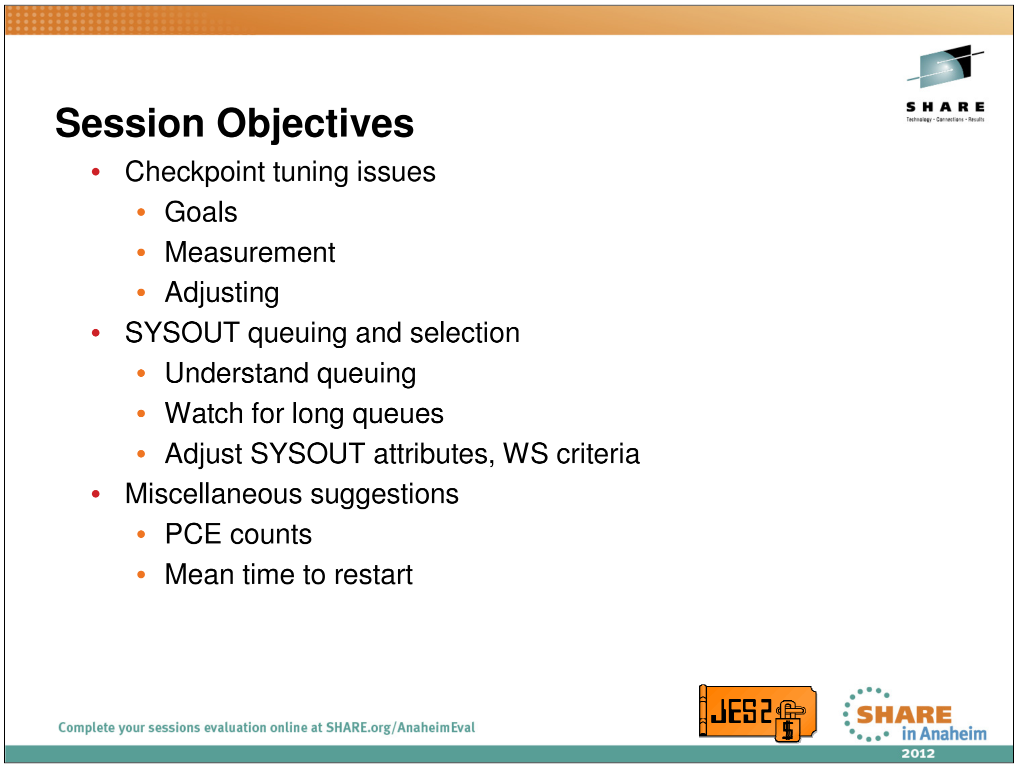
JES2 Performance Considerations

Tom Wasik
IBM Rochester, MN

Tuesday 11:00 AM
Session Number 11756

SHARE in Anaheim 2012

This presentation will cover selected topics in JES2 tuning.



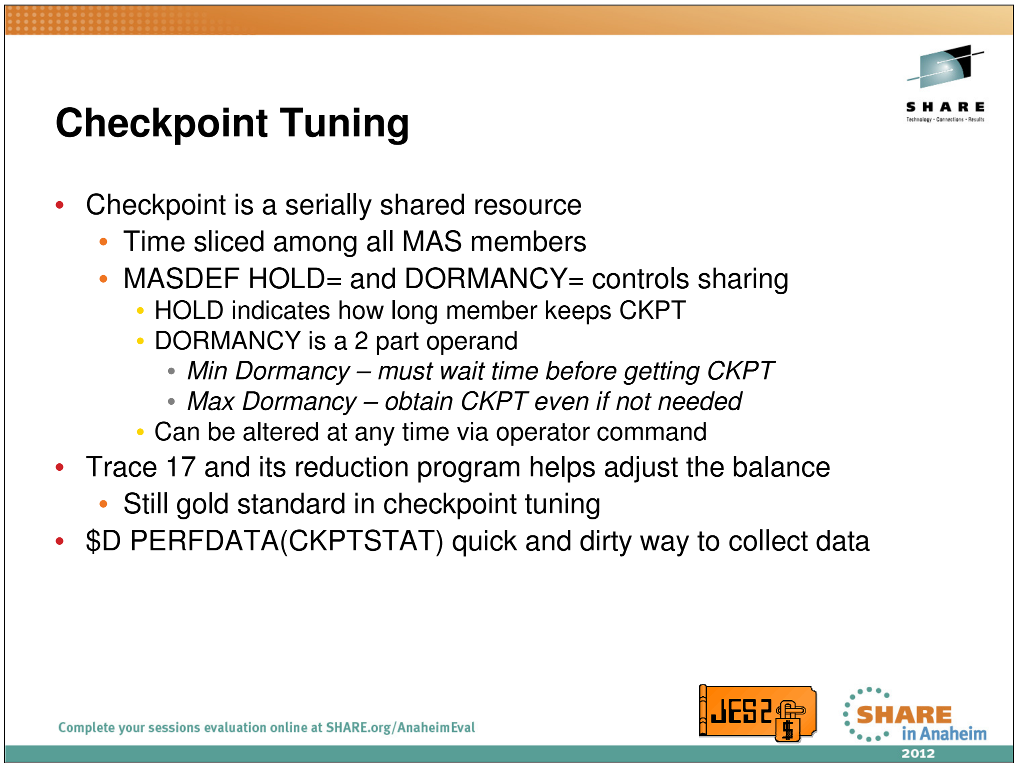
Session Objectives

- Checkpoint tuning issues
  - Goals
  - Measurement
  - Adjusting
- SYSOUT queuing and selection
  - Understand queuing
  - Watch for long queues
  - Adjust SYSOUT attributes, WS criteria
- Miscellaneous suggestions
  - PCE counts
  - Mean time to restart

Complete your sessions evaluation online at SHARE.org/AnaheimEval

This talk will concentrate on the checkpoint data set and tuning of that data set and SYSOUT queuing and work selection. It will cover consideration on how to tune these item for best performance.

Also a number of miscellaneous item will be discussed including PCE counts and mean time to restart (the other MTTR).



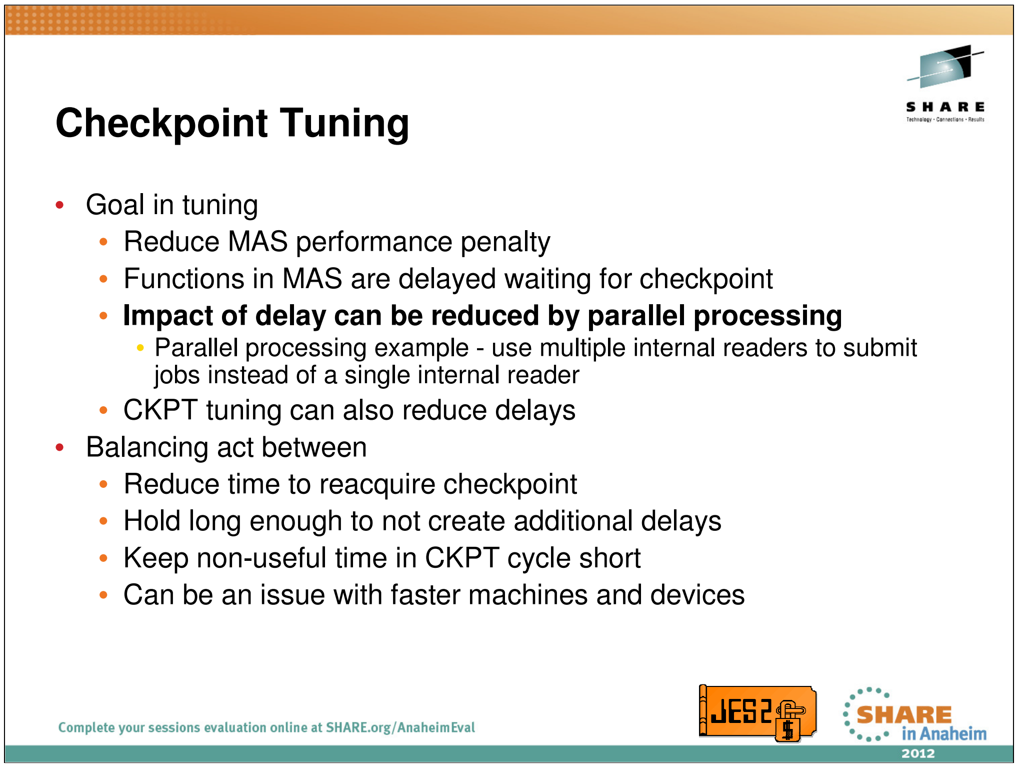
Checkpoint Tuning

- Checkpoint is a serially shared resource
  - Time sliced among all MAS members
  - MASDEF HOLD= and DORMANCY= controls sharing
    - HOLD indicates how long member keeps CKPT
    - DORMANCY is a 2 part operand
      - *Min Dormancy – must wait time before getting CKPT*
      - *Max Dormancy – obtain CKPT even if not needed*
    - Can be altered at any time via operator command
- Trace 17 and its reduction program helps adjust the balance
  - Still gold standard in checkpoint tuning
- $D PERFDATA(CKPTSTAT) quick and dirty way to collect data

Complete your sessions evaluation online at SHARE.org/AnaheimEval

The JES2 checkpoint is a serially shared resource. Essentially it is time sliced among all the active MAS members. The sharing is controlled by the MASDEF HOLD and DORMANCY values. HOLD controls how long a member will hold the checkpoint once it has read all the information in it. Once the hold time expires and checkpoint activity has subsided, the checkpoint is released. Dormancy controls when a member will try to get the checkpoint back. There are 2 values for dormancy, Min and Max dormancy. Min dormancy is the minimum amount of time a member must wait before attempting to re-acquire the checkpoint. No attempt to obtain the checkpoint will be made until that interval expires. Once the Min dormancy expires, a member will attempt to access the checkpoint IF it believes that access is needed (there is a process in JES2 that has requested checkpoint access… done a $QSUSE). If there are no processes that require the checkpoint, the member will not attempt to re-acquire the checkpoint until max dormancy has expired. Looking at actual hold and dormancy values is a good indication of whether a member is properly tuned or not.

The gold standard for looking a checkpoint performance has been trace 17 and its related reduction program. Together they provide hold and dormancy values and low level details about what data was written to the checkpoint. But this is often too much data. It is also not the easiest data to gather. The $D PERFDATA(CKPTSTAT) gives similar (though less) data and is easier to collect.



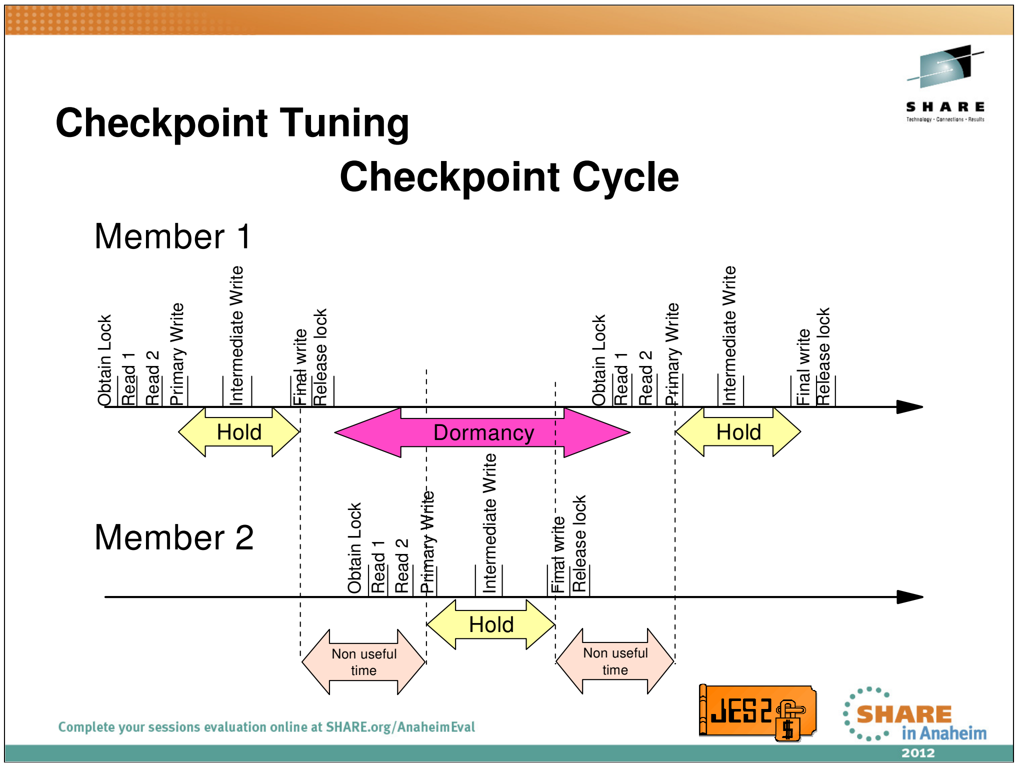
Checkpoint Tuning

- Goal in tuning
  - Reduce MAS performance penalty
  - Functions in MAS are delayed waiting for checkpoint
  - **Impact of delay can be reduced by parallel processing**
    - Parallel processing example - use multiple internal readers to submit jobs instead of a single internal reader
  - CKPT tuning can also reduce delays
- Balancing act between
  - Reduce time to reacquire checkpoint
  - Hold long enough to not create additional delays
  - Keep non-useful time in CKPT cycle short
  - Can be an issue with faster machines and devices

Complete your sessions evaluation online at SHARE.org/AnaheimEval

The goal in checkpoint tuning is to reduce the MAS penalty. Because processes need to access the checkpoint to perform certain functions, there is a latency that is introduced that is proportional to the checkpoint cycle time (how long it take a member to acquire access to the checkpoint, hold the checkpoint, release access, and then re-acquire access). There are 2 primary ways to reduce the MAS penalty, reduce cycle time or increase parallelism. By increasing parallelism, you increase the amount of work done once the checkpoint is obtained, lessening the impact of the checkpoint latency.

But there is a cost to pushing cycle time too low. There is an overhead in acquiring and releasing the checkpoint. This costs in both CPU and other resources, and in an increase in the ratio of non-useful time to productive time in the checkpoint process. As machines and devices get faster, there are issues where an old hold and dormancy value can throw things out of balance.

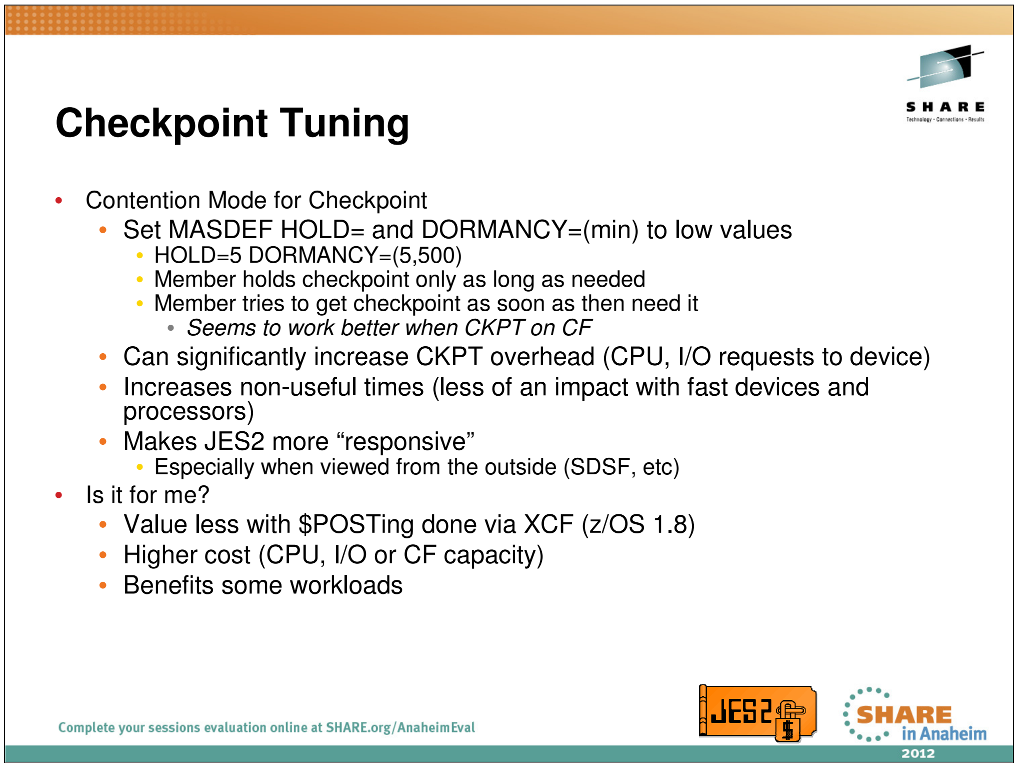


Checkpoint Tuning

Checkpoint Cycle

Complete your sessions evaluation online at SHARE.org/AnaheimEval

This is the basic checkpoint cycle with 2 members. This shows the I/Os performed and the terms used to describe the parts of the cycle. The ratio of non-useful time to hold time is one of the consideration in checkpoint tuning. The non-useful time starts when the final write starts on one member and the primary write starts on the next member. The write/read performance and lock passing time make up the non-useful time.



Checkpoint Tuning

- Contention Mode for Checkpoint
  - Set MASDEF HOLD= and DORMANCY=(min) to low values
    - HOLD=5 DORMANCY=(5,500)
    - Member holds checkpoint only as long as needed
    - Member tries to get checkpoint as soon as then need it
      - *Seems to work better when CKPT on CF*
  - Can significantly increase CKPT overhead (CPU, I/O requests to device)
  - Increases non-useful times (less of an impact with fast devices and processors)
  - Makes JES2 more "responsive"
    - Especially when viewed from the outside (SDSF, etc)
- Is it for me?
  - Value less with $POSTing done via XCF (z/OS 1.8)
  - Higher cost (CPU, I/O or CF capacity)
  - Benefits some workloads

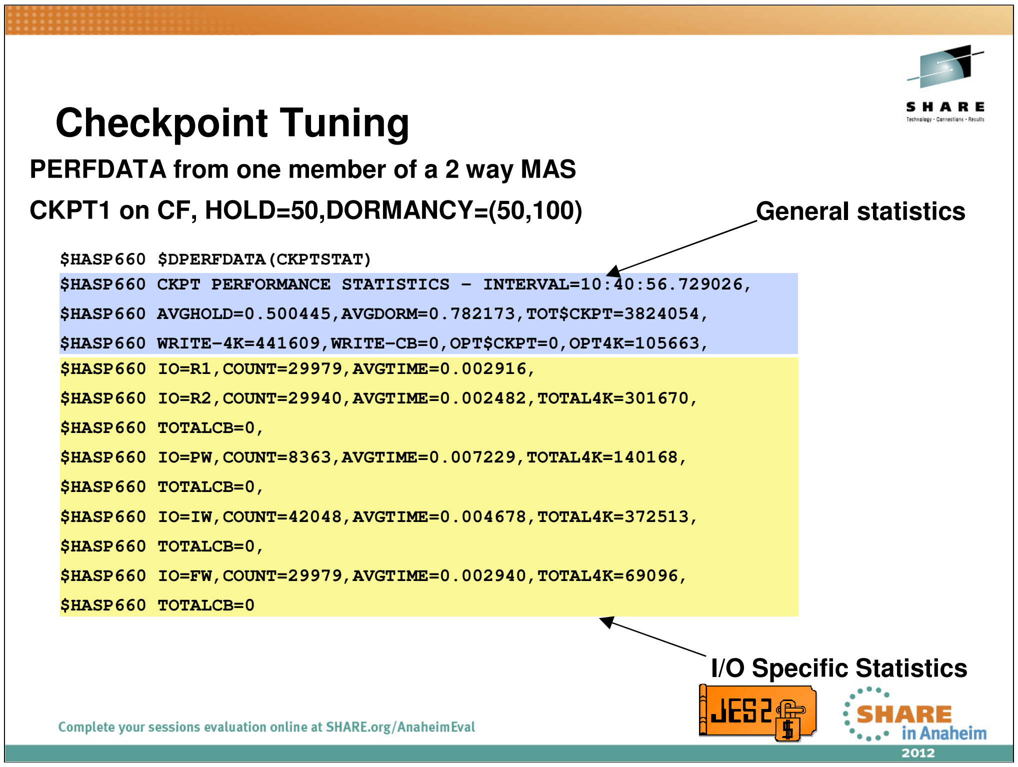
Complete your sessions evaluation online at SHARE.org/AnaheimEval

Arguing contention mode is like talking politics or religion with the potential inlaws. Unless you agree with them totally, no-one will ever win. This is an age old queuing problem that goes beyond computers. Keeping the toilet seat up saves me time if I am then next one to use the toilet and only takes my wife a second to deal with, but my wife (and I dare say most women) just don't not see it that way.

Contention mode make a member go through the overhead of releasing the checkpoint as soon as they no longer need it. But it also gives them the freedom to get it as soon as they need it. The idea is why hold the checkpoint any longer than is absolutely needed. And then go grab it as soon as you need it. This does seem to fit well with the CF FIFO lock processing. The benefit is JES2 is much more responsive. The cost is a significant increase in checkpoint I/O activity and CPU overhead in the JES2 address space. It can also increase the non-useful time (or more correctly the ratio of non-useful to useful time). It is not clear that more responsive implies things get done sooner.

Is it for your shop? Well the value of this goes down with some of the $POSTing changes made in z/OS 1.8. There is a higher CPU cost, greater load on the device(s) that contain the checkpoint. BUT there are some very large shops with highly experienced and skilled system programmers that swear there systems run better this way.

My recommendations, do not try this unless you have the time to monitor the system closely (learn more about trace 17 and the reduction program… which is beyond what this presentation will get into).



Checkpoint Tuning

PERFDATA from one member of a 2 way MAS

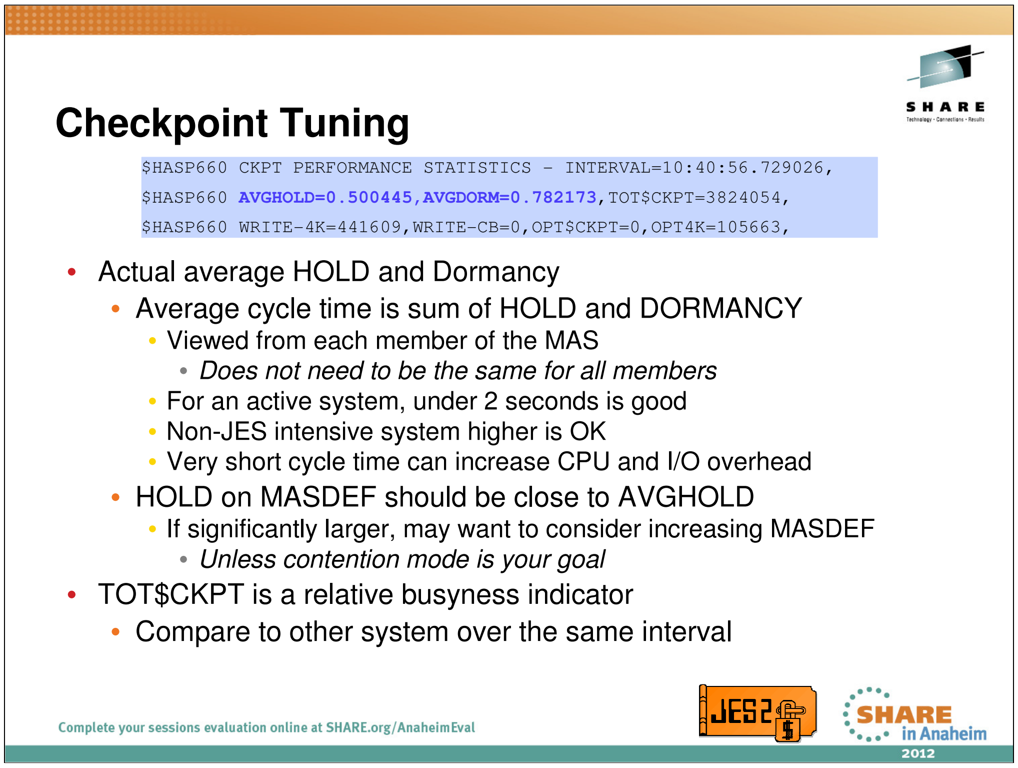
CKPT1 on CF, HOLD=50,DORMANCY=(50,100)

```
$HASP660 $DPERFDATA(CKPTSTAT)
$HASP660 CKPT PERFORMANCE STATISTICS - INTERVAL=10:40:56.729026,
$HASP660 AVGHOLD=0.500445,AVGDORM=0.782173,TOT$CKPT=3824054,
$HASP660 WRITE-4K=441609,WRITE-CB=0,OPT$CKPT=0,OPT4K=105663,
$HASP660 IO=R1,COUNT=29979,AVGTIME=0.002916,
$HASP660 IO=R2,COUNT=29940,AVGTIME=0.002482,TOTAL4K=301670,
$HASP660 TOTALCB=0,
$HASP660 IO=PW,COUNT=8363,AVGTIME=0.007229,TOTAL4K=140168,
$HASP660 TOTALCB=0,
$HASP660 IO=IW,COUNT=42048,AVGTIME=0.004678,TOTAL4K=372513,
$HASP660 TOTALCB=0,
$HASP660 IO=FW,COUNT=29979,AVGTIME=0.002940,TOTAL4K=69096,
$HASP660 TOTALCB=0
```

General statistics
I/O Specific Statistics
Complete your sessions evaluation online at SHARE.org/AnaheimEval
SHARE in Anaheim 2012

This is the output of a $D PERFDATA(CKPTSTAT) for one member of a 2 member MAS. In this MAS, CKPT1 is in a CF structure and CKPT2 is on DASD. HOLD on this member is set to 50 and dormancy is min 50 and max 100.

PERFDATA displays status information from the time PERFDATA was reset until the current time. By default, it displays data from the time the JES2 address space was started to the time of the command. To reset PERFDATA statistics, issued the command $TPERFDATA(CKPTSTAT),RESET. This will reset the statistics for CKPTSTAT and leave the other statistics alone. INTERVAL in the display indicates how long the current sampling window is (in this example, 10 hours and 40 minutes).



Checkpoint Tuning

```
$HASP660 CKPT PERFORMANCE STATISTICS - INTERVAL=10:40:56.729026,
$HASP660 AVGHOLD=0.500445,AVGDORM=0.782173,TOT$CKPT=3824054,
$HASP660 WRITE-4K=441609,WRITE-CB=0,OPT$CKPT=0,OPT4K=105663,
```

- Actual average HOLD and Dormancy
  - Average cycle time is sum of HOLD and DORMANCY
    - Viewed from each member of the MAS
      - *Does not need to be the same for all members*
    - For an active system, under 2 seconds is good
    - Non-JES intensive system higher is OK
    - Very short cycle time can increase CPU and I/O overhead
  - HOLD on MASDEF should be close to AVGHOLD
    - If significantly larger, may want to consider increasing MASDEF
      - *Unless contention mode is your goal*
- TOT$CKPT is a relative busyness indicator
  - Compare to other system over the same interval

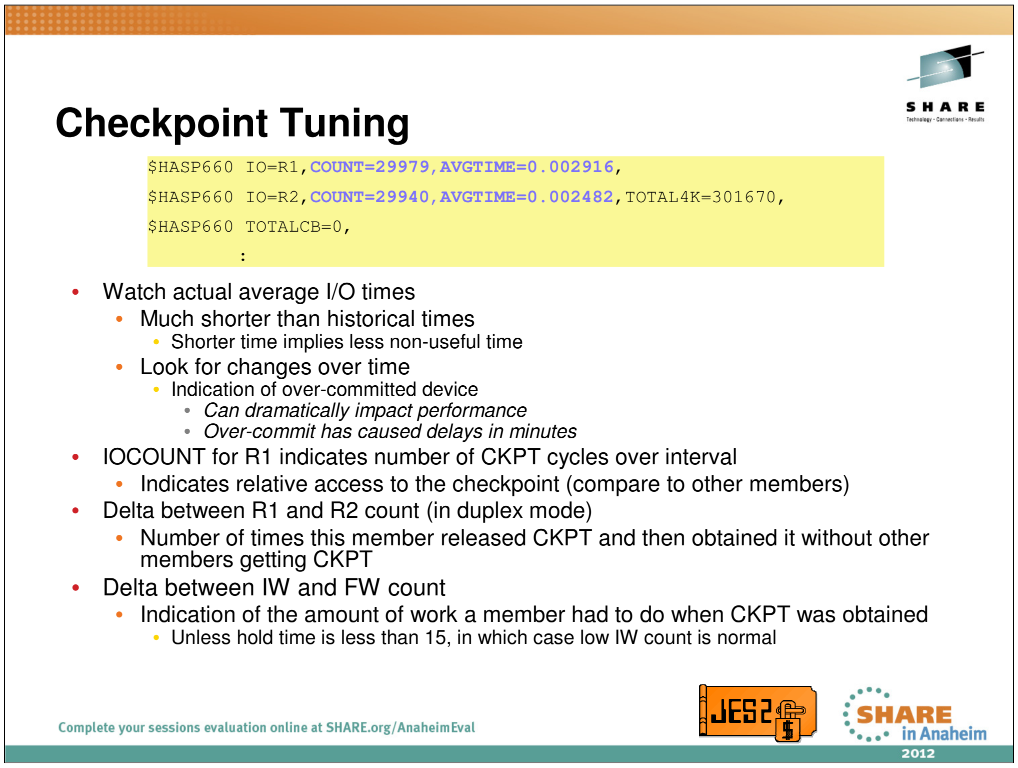
Complete your sessions evaluation online at SHARE.org/AnaheimEval

One of the first things to look at is the AVGHOLD (average hold time) and AVGDORM (average dormancy time). Combine these to get average cycle time. The average cycle time should be close to 2 seconds. System that have very small JES2 workloads can have longer cycle times (closer to 4 seconds).

Cycle times that are too short can cause increased CKPT overhead without significantly improving performance (cycle times less than 1 second) and can actually make processing take longer. This happens when work that could have been done in one hold interval of 0.5 seconds instead take 2 shorter hold intervals. If the cycle time is 1 second, then this process takes 1 second to complete instead of 0.5 seconds. If you want to see this effect you need to use trace 17 and look at the average time PCE $WAITing for the CKPT and compare this to average dormancy. Or you can just keep cycle times closer to 2 seconds by increasing HOLD and not worry about it.

If AVGHOLD is much larger than HOLD, then this system may have a HOLD value that is too short. If hold is greater than 15, also look at the IW and FW counts. If the IW count is much smaller than the FW count, then this system is so busy, it cannot even get an IW started in the allotted hold time. This system could benefit from a longer hold time (closer to the AVGHOLD time it is actually running with). Note that this is an easy way to see a system that is not getting enough hold time. But there is no easy way to determine a hold time is too long.

The TOT$CKPT can be used as a relative CKPT activity indicator when compared to other members over the same interval. It can be thrown off by command activity (automatic commands that are issued on only one member) and other asymmetric processing. But depending on the importance of that activity, it may warrant favoring systems that have a relatively high TOT$CKPT value.



Checkpoint Tuning

```
$HASP660 IO=R1,COUNT=29979,AVGTIME=0.002916,
$HASP660 IO=R2,COUNT=29940,AVGTIME=0.002482,TOTAL4K=301670,
$HASP660 TOTALCB=0,
            :
```

- Watch actual average I/O times
  - Much shorter than historical times
    - Shorter time implies less non-useful time
  - Look for changes over time
    - Indication of over-committed device
      - *Can dramatically impact performance*
      - *Over-commit has caused delays in minutes*
- IOCOUNT for R1 indicates number of CKPT cycles over interval
  - Indicates relative access to the checkpoint (compare to other members)
- Delta between R1 and R2 count (in duplex mode)
  - Number of times this member released CKPT and then obtained it without other members getting CKPT
- Delta between IW and FW count
  - Indication of the amount of work a member had to do when CKPT was obtained
    - Unless hold time is less than 15, in which case low IW count is normal

Complete your sessions evaluation online at SHARE.org/AnaheimEval

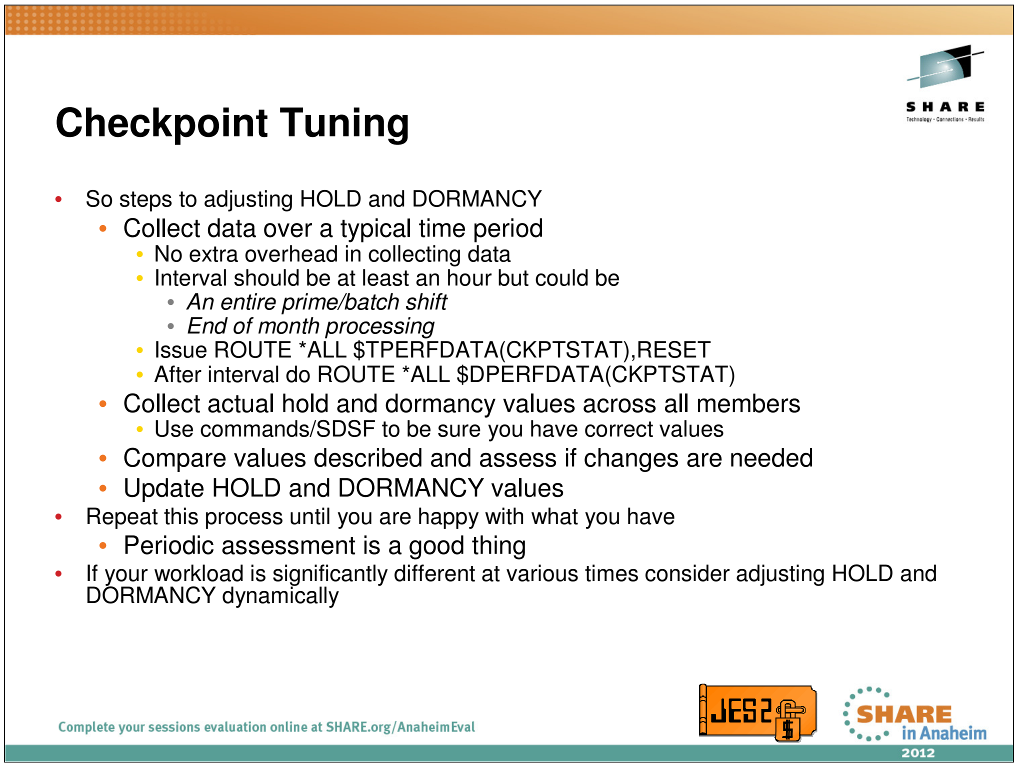
When looking at the records for each I/O, concentrate on the average I/O time and the count field. You should become familiar with what the I/O times for the various writes are on your system when it is running well. Then use these values as a basis for later examinations of the times looking for unexplained growth. The times on modern devices with modern processors are much shorter that historical times (from 10 years ago). What this shorter time implies is a shorter non-useful time and that the I/O times really do not need to be considered when calculating hold and dormancy times.

One thing you are looking for is the impact of over-committing the device. Too much other activity on the device can lead to significant delays in the JES2 I/O times. This is especially true of checkpoint on CF. JES2 writes blocks to the CF in bursts. If the CF is busy with other exploiters, these bursts can sometimes overwhelm the CF and cause significant delays to both JES2 and other exploiters. This will start as a small, but noticeable increase in the I/Os to the CF but can escalate to significant delays.

The COUNT= field tells how many of a particular type of I/O has completed in the interval. Based on the timing of the last reset and the current display, then can be inconsistent by one or 2. Looking at these counts can tell how well the checkpoint is performing. The R1 (read 1) and FW (Final Write) count tell you how checkpoint cycles this member has completed in the interval. This is useful to compare to other members to understand relative access to the checkpoint. Because of the way dormancy works, relative access can also be thought of as relative need.

The delta between the R1 (read 1) and R2 (read 2) counts (in duplex mode) reflects the number of times this member released the checkpoint and then re-acquired it without other members accessing the checkpoint.

The delta between the IW (intermediate write) and the FW (final write) indicates the relative work a member has to do once it gets the checkpoint. As long as hold is at least 15, JES2 will schedule one intermediate write per checkpoint cycle (if hold is less than 15 then none are scheduled). If there are less intermediate writes than expected, this indicates a system so busy that it could not spend time doing a checkpoint write, This system could benefit from a longer hold time. If there are the expected number this system may have too long a hold time (in an active system, this number should be above the FW count). The amount above the expected count, it is one indicator of the relative checkpoint use of this system as compared to other members.



Checkpoint Tuning

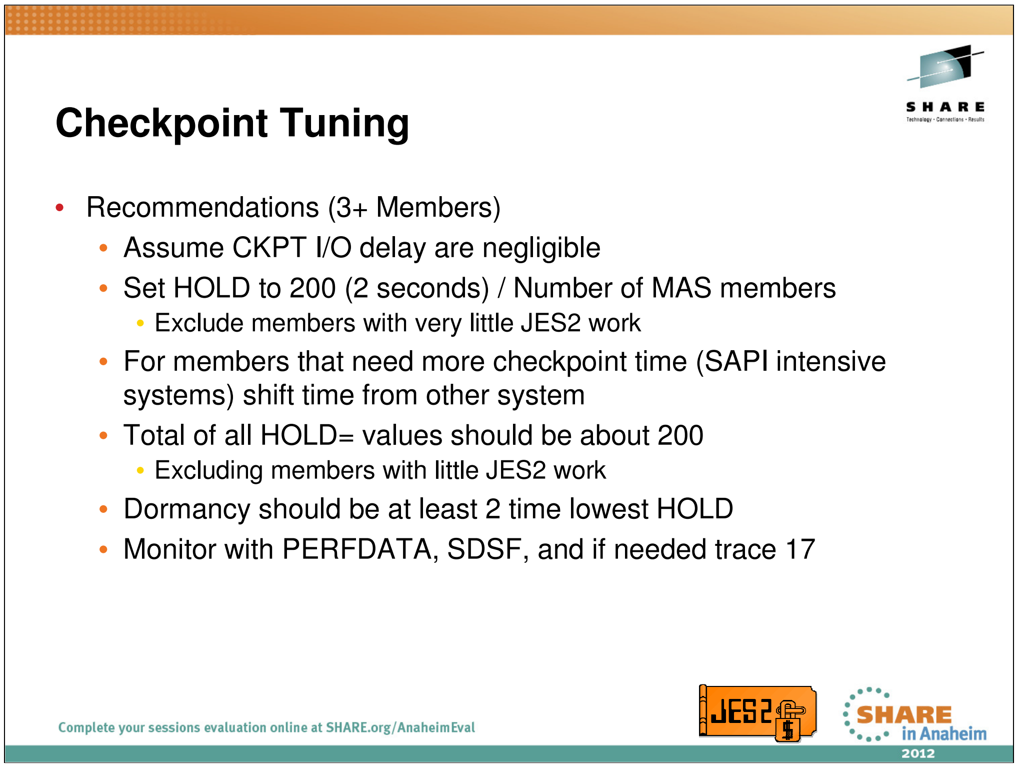
- So steps to adjusting HOLD and DORMANCY
  - Collect data over a typical time period
    - No extra overhead in collecting data
    - Interval should be at least an hour but could be
      - *An entire prime/batch shift*
      - *End of month processing*
    - Issue ROUTE *ALL $TPERFDATA(CKPTSTAT),RESET
    - After interval do ROUTE *ALL $DPERFDATA(CKPTSTAT)
  - Collect actual hold and dormancy values across all members
    - Use commands/SDSF to be sure you have correct values
  - Compare values described and assess if changes are needed
  - Update HOLD and DORMANCY values
- Repeat this process until you are happy with what you have
  - Periodic assessment is a good thing
- If your workload is significantly different at various times consider adjusting HOLD and DORMANCY dynamically

Complete your sessions evaluation online at SHARE.org/AnaheimEval

First thing to do when adjusting HOLD and DORMANCY is to select a time interval. Pick a time when the system is running a typical workload. It can be the middle of the day, during your batch window, etc. You should collect at least and hours worth of data. You can collect data for a longer period of time (such as the entire batch window, of an entire prime shift). You could do an entire 24 hour period if that is what you want to average your tuning to. Use the ROUTE *ALL command to send a RESET and Display PERFDATA command to all members. This ensures you have a consistent interval. Be sure to collect the actual HOLD and DORMANCY values via command (or SDSF). There are many systems that set these values via command at startup instead of using just the initialization deck In fact, you should probably collect then at the start and end of the interval to ensure that they did not actually change over the interval (that can really confuse your). Also, once you collect data, ensure the intervals are about the same. If a system is unexpectedly restarted, then the samples are not very useful.

Use the information to compare the systems and asses if changes are needed (depending on how much of a cowboy you are, you may want to collect data a couple of times before changing anything). Update the values and start another interval to see the effect of what you just did. Repeat this process until you are satisfied with the numbers you are using. Be sure to check with others to see if they are also happy with the performance (after all, it is not about making these number look good, you want the applications that run on the system to perform well). You should do this assessment periodically to ensure your HOLD and Dormancy values are good.

You may also want to consider adjusting these values dynamically based on the characteristics of the work running. This is not normally needed. But if a particular workload is causing problems, it may be what is needed. Also be sure you are aware if this is happening when you are gathering sampling data. Changing HOLD and DORMANCY during the interval can make looking at the data very confusing.



Checkpoint Tuning

- Recommendations (3+ Members)
  - Assume CKPT I/O delay are negligible
  - Set HOLD to 200 (2 seconds) / Number of MAS members
    - Exclude members with very little JES2 work
  - For members that need more checkpoint time (SAPI intensive systems) shift time from other system
  - Total of all HOLD= values should be about 200
    - Excluding members with little JES2 work
  - Dormancy should be at least 2 time lowest HOLD
  - Monitor with PERFDATA, SDSF, and if needed trace 17

Complete your sessions evaluation online at SHARE.org/AnaheimEval

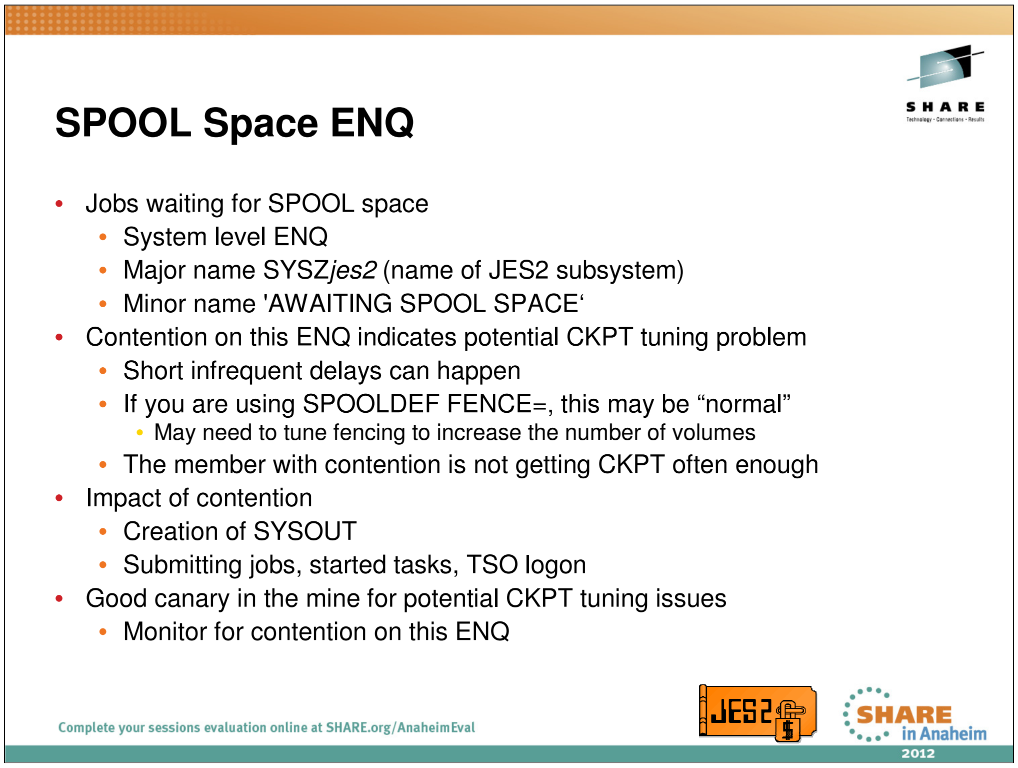
Here are some basic recommendations for setting HOLD and DORMANCY. Note I am assuming a 3 way and larger MAS.

You can assume that I/O delays re negligible. In the past, the read 1, read 2 and final write times were used to calculate adjustments to hold and dormancy. With modern processors and devices, these times are no longer significant.

Start out with a HOLD value the is 200 divided by the number of members in your MAS. Adjust the hold values to favor systems that need more checkpoint time but the total of all hold times for all members of the MAS should be 200. Also avoid dropping a member below 15 unless it is a very large MAS or there is a member that uses little checkpoint services. In the case of a member that does use very few JES2 services, set there hold to 15 and do not consider their system at all in calculating the other members hold values.

The minimum dormancy should be set to two times the lowest hold value you came up with (unless this is a 2 way MAS, then set it to a little less than the hold value on the other member).

Monitor the values with PERFDATA(CKPTSTAT) and SDSF as well as looking at application performance. Trace 17 and the reduction program give lots more details on this but can take more time to analyze.



SPOOL Space ENQ

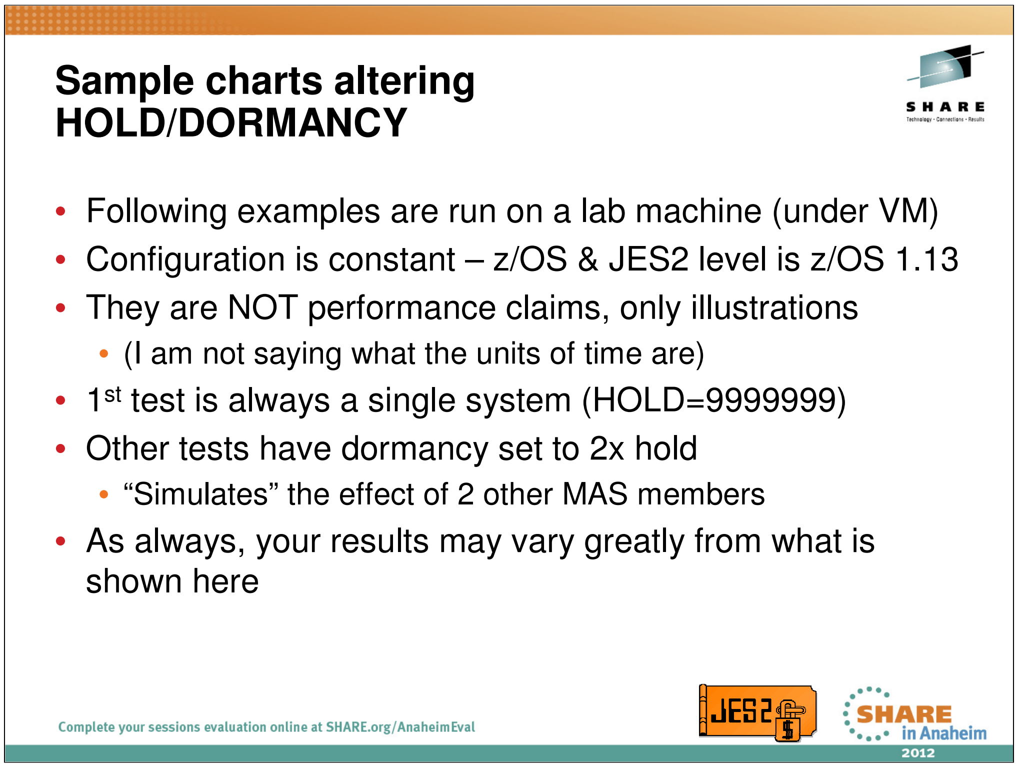
- Jobs waiting for SPOOL space
  - System level ENQ
  - Major name SYSZ*jes2* (name of JES2 subsystem)
  - Minor name 'AWAITING SPOOL SPACE‘
- Contention on this ENQ indicates potential CKPT tuning problem
  - Short infrequent delays can happen
  - If you are using SPOOLDEF FENCE=, this may be “normal”
    - May need to tune fencing to increase the number of volumes
  - The member with contention is not getting CKPT often enough
- Impact of contention
  - Creation of SYSOUT
  - Submitting jobs, started tasks, TSO logon
- Good canary in the mine for potential CKPT tuning issues
  - Monitor for contention on this ENQ

Complete your sessions evaluation online at SHARE.org/AnaheimEval

Now that you know how to tune checkpoint, how do you know you have to? One indication that has been showing up lately is the JES2 SPOOL space ENQ. JES2 maintains a cache of SPOOL space for use by address spaces when JES2 does not own the checkpoint. This cache is a fixed size. When the cache is empty (or if SPOOL fencing is on, if there are no appropriate volumes in the cache) address spaces start to queue up on the ‘AWAITING SPOOL SPACE’ system ENQ. Actually, only the second and later requestor queue up so contention indicates 3 address spaces cannot get the SPOOL space they need (1st requestor, holder of the ENQ, and the requestors that come after that).

If you are seeing significant contention on this ENQ, then the member with the contention is not getting the checkpoint often enough. Short, infrequent delays may be normal, but a consistent list of waiters is not. The problem in this case is cycle time. You may need to lower the cycle time, you may need to lower the minimum dormancy on the member with the problem, or simply lower the hold time on all the members.

Think of this as your canary in the mine. This ENQ contention will often show up long before there are any other problems with checkpoint. The fact that there is contention may NOT prove there is a problem with the MASDEF values, but it does indicate that there is some problem and you need to start looking. It could be CF (we have seen problems with the CF or XCF couple data set cause all kinds of problems), or DASD, or some other problem keeping JES2 from passing the checkpoint around. But then again, if you just changed MASDEF (or have NOT changed it in 20 years) it could be a checkpoint tuning problem.



Sample charts altering HOLD/DORMANCY

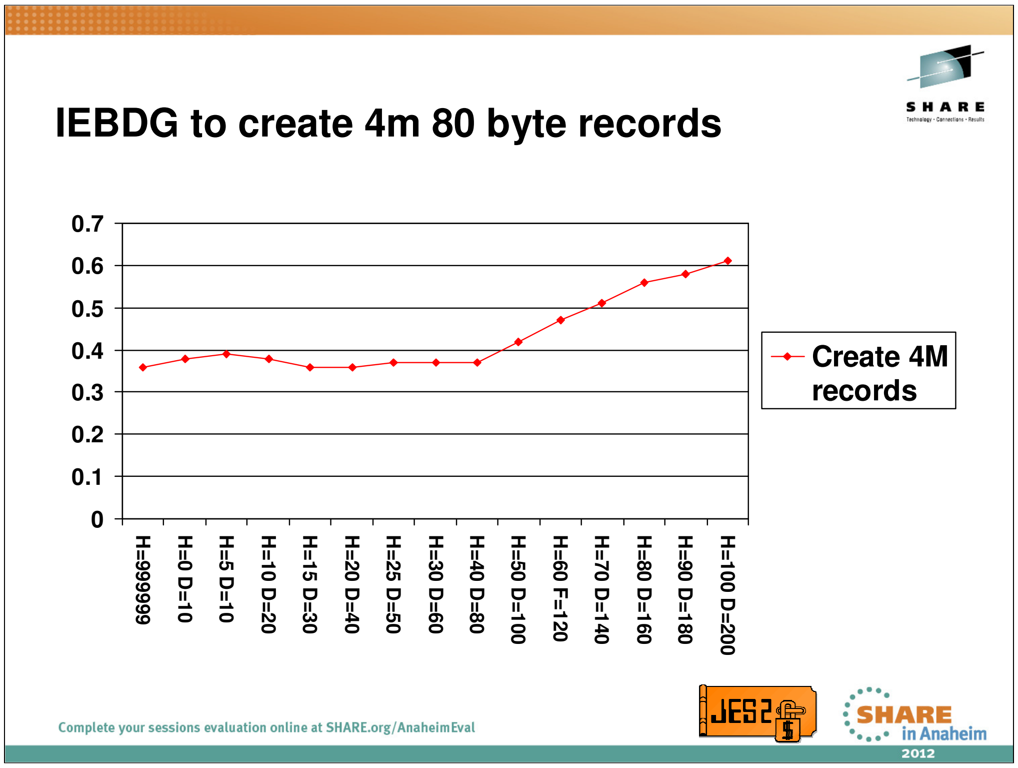
- Following examples are run on a lab machine (under VM)
- Configuration is constant – z/OS & JES2 level is z/OS 1.13
- They are NOT performance claims, only illustrations
  - (I am not saying what the units of time are)
- 1st test is always a single system (HOLD=9999999)
- Other tests have dormancy set to 2x hold
  - “Simulates” the effect of 2 other MAS members
- As always, your results may vary greatly from what is shown here

Complete your sessions evaluation online at SHARE.org/AnaheimEval

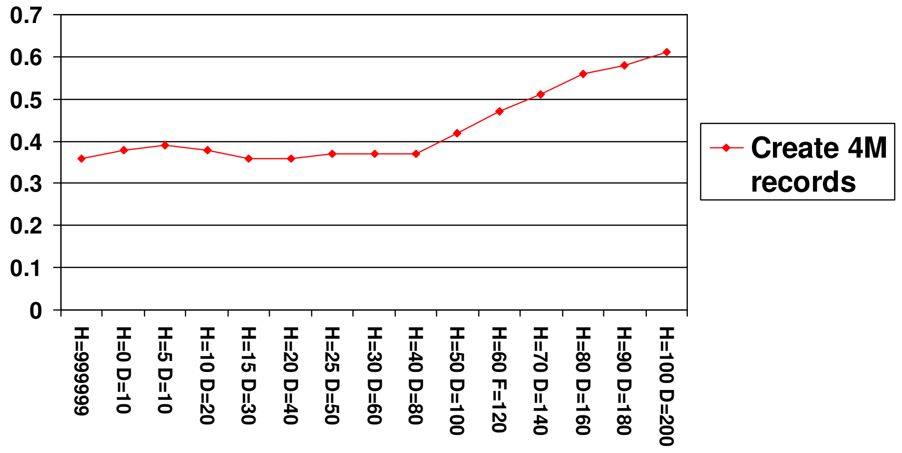
To illustrate the effect of HOLD and DORMANCY on a workload, a few workloads were run with various values for hold and dormancy. The configuration in all the tests was kept constant. Though the tests were run on a second level VM guest machine, they were run at a time when the workload on the VM system was light (middle of the night). These are NOT performance claims nor are they meant as suggested hold and dormancy values to run with. They are instead examples to help understand the effects of these parameters on execution time.

The first test is always done with HOLD=9999999 to simulate a single system. Other tests have DORMANCY set to 2 times hold to simulate the effect of a 3 way MAS.

Your results may greatly vary from what is presented here.

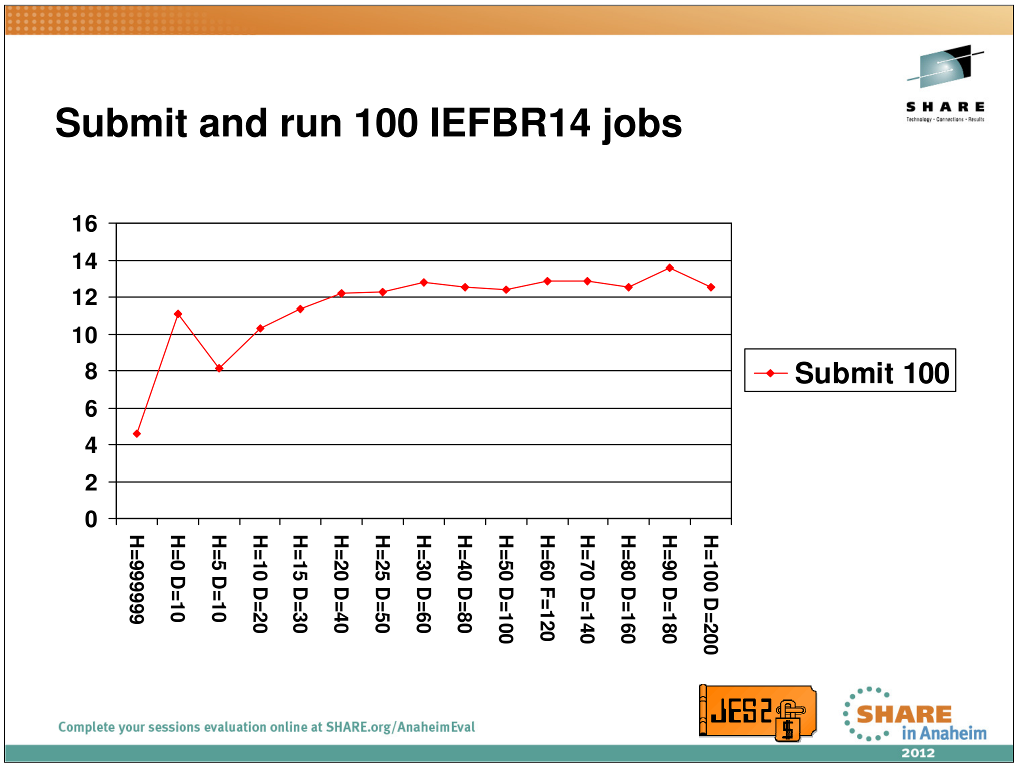


IEBDG to create 4m 80 byte records

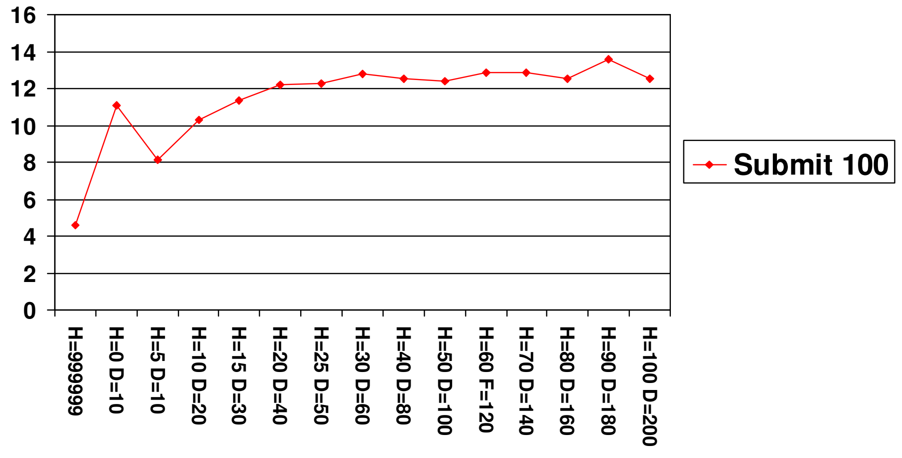


Complete your sessions evaluation online at SHARE.org/AnaheimEval

This is a simple IEBDG job that generates 4,000,000, 80 byte SYSOUT records. Notice how the single system value, though faster than any other sample, is very close to the other values. The main limiting factor here is SPOOL space in the BLOB. The BLOB is updated on every intermediate and final write. If you recall, there is no intermediate write done if hold is less than 15. So notice the dip in the curve at HOLD=15. This is the effect of the extra intermediate write. This extra write helps until dormancy starts kicking in at about HOLD=50,DORMANCY=100.

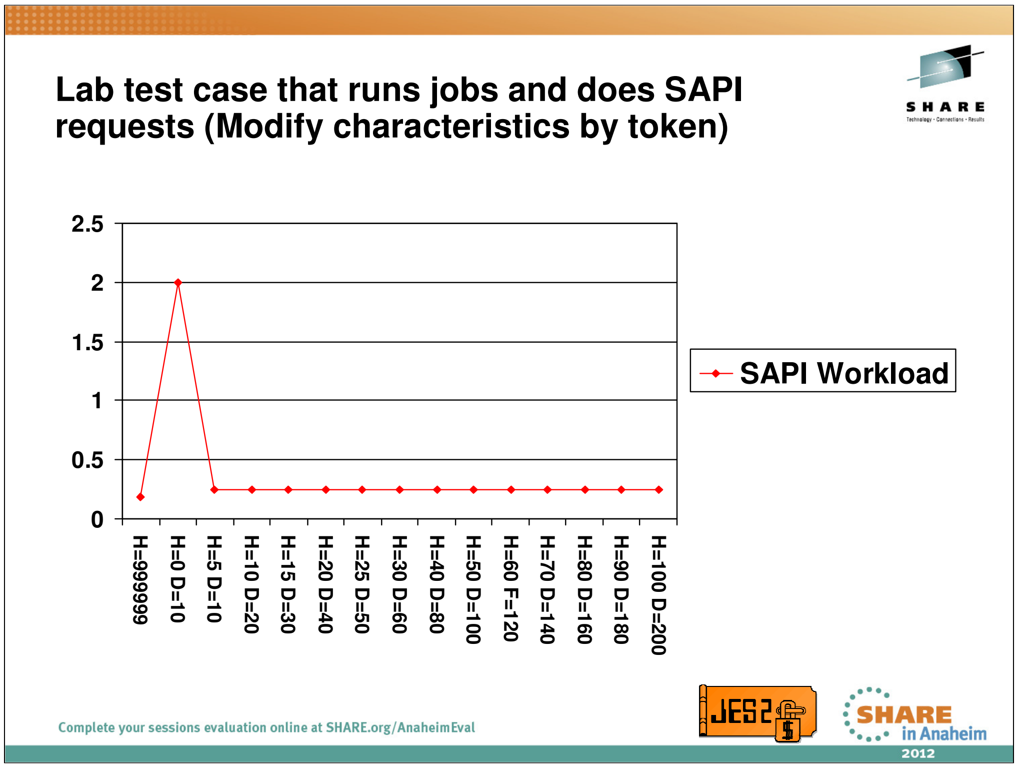


Submit and run 100 IEFBR14 jobs

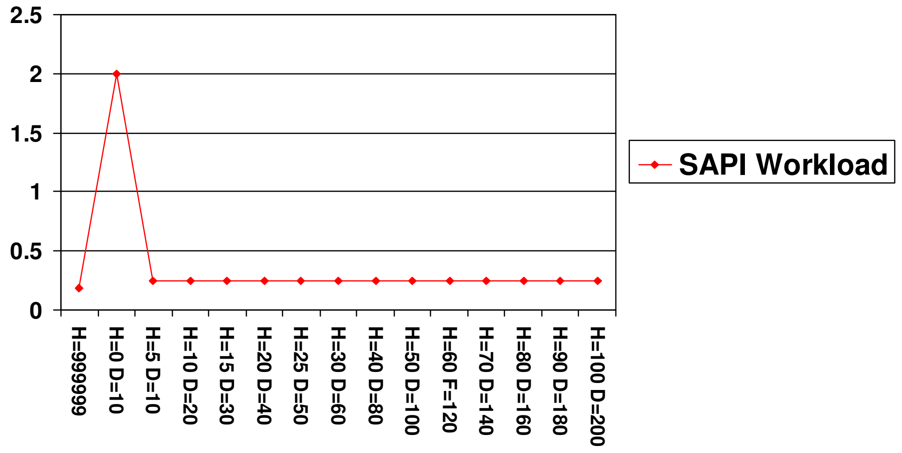


Complete your sessions evaluation online at SHARE.org/AnaheimEval

This is a single TSO submit of a job stream with 100 2 card IEFBR14 jobs. Each of the 100 jobs has a unique job name. What is happening here is that every job submitted requires a CKPT access followed by a SPOOL access and then another CKPT access. This is what causes most of the delays you see. But it is especially noticeable when HOLD is pushed down to 0. This aggravates the situation since every SPOOL access results in the loss of the CKPT and a corresponding DORMANCY delay. Also notice that the times do fluctuate as hold gets longer. This is typical based on how long a SPOOL write takes and how many can be done in a CKPT HOLD interval. Sometimes a little more time allows one more SPOOL I/O within the HOLD period, allowing a bit faster job completion.

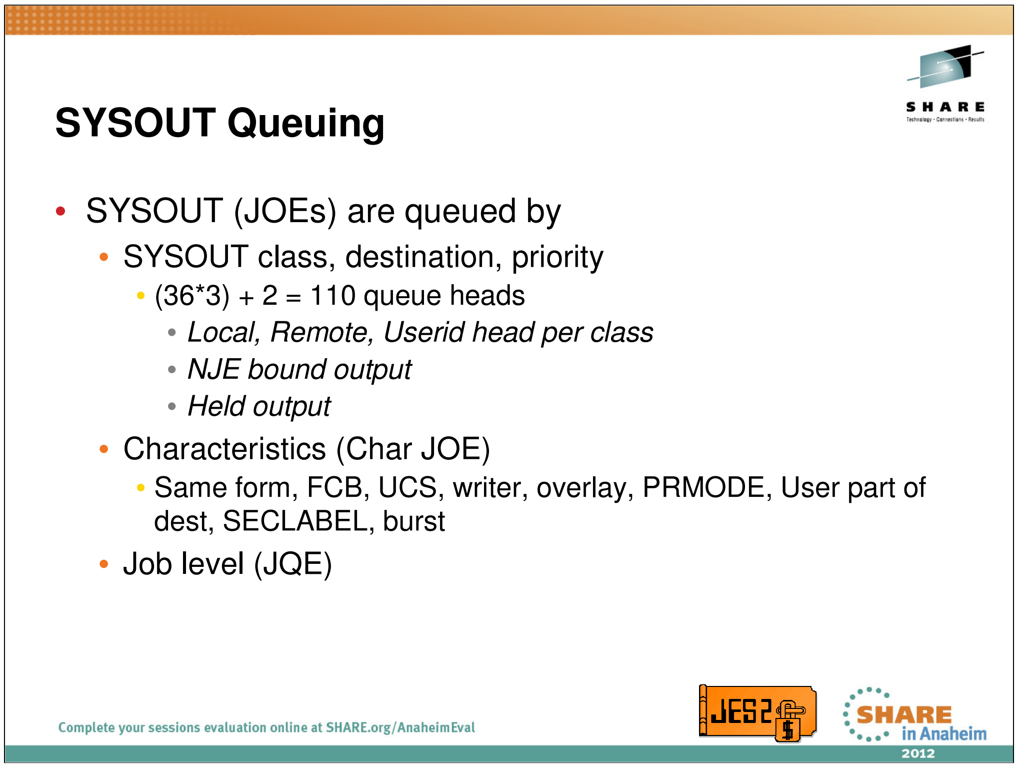


Lab test case that runs jobs and does SAPI requests (Modify characteristics by token)



Complete your sessions evaluation online at SHARE.org/AnaheimEval

This is one we discovered when doing some SAPI testing. This is a test case that does a number of SAPI requests by token and then alters the characteristics of a single data set. The test case submits about 10 jobs, makes the SAPI requests, and then purges the jobs. This is a recreateable result where HOLD=0 takes 8 times longer to complete than any other HOLD value. It is all based on the number of SPOOL accesses followed by CKPT access that can be done in one HOLD period. This is the most dramatic example of HOLD impacting external performance I am aware of.



SYSOUT Queuing

- SYSOUT (JOEs) are queued by
  - SYSOUT class, destination, priority
    - (36*3) + 2 = 110 queue heads
      - *Local, Remote, Userid head per class*
      - *NJE bound output*
      - *Held output*
  - Characteristics (Char JOE)
    - Same form, FCB, UCS, writer, overlay, PRMODE, User part of dest, SECLABEL, burst
  - Job level (JQE)

Complete your sessions evaluation online at SHARE.org/AnaheimEval

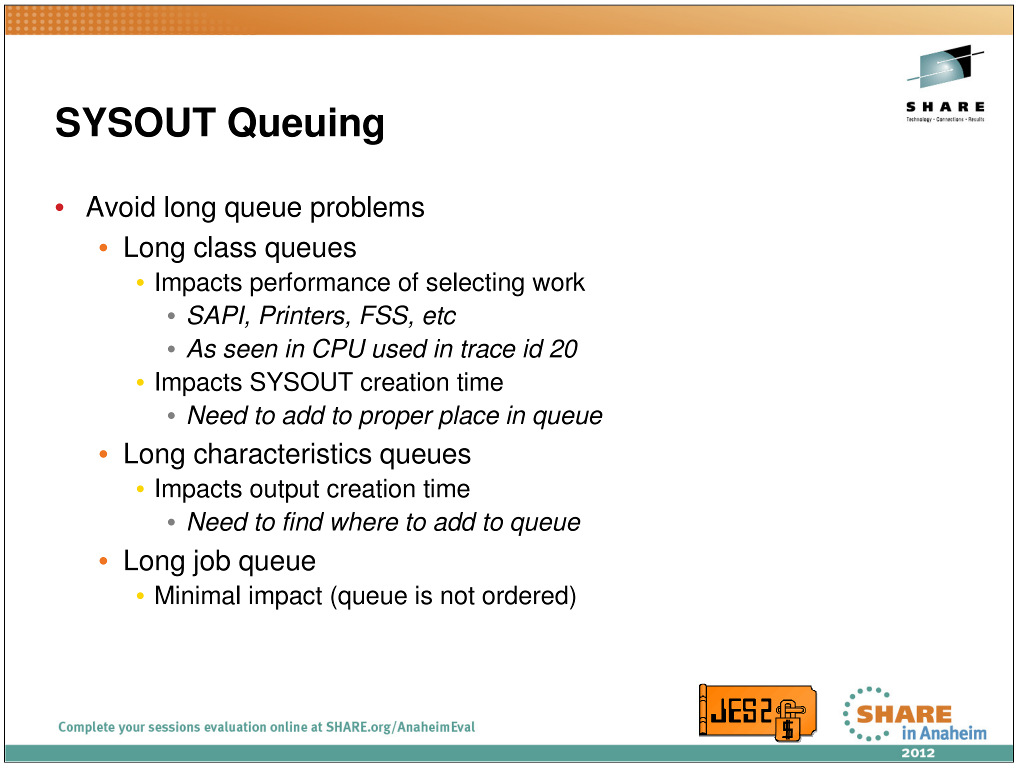
There are 3 major queues running through the output queue (JOEs). The class queue, the characteristics queue, and the job level queue. For each SYSOUT class there are 3 sub queues, a local, remote, and userid queue. So this is 36*3 = 108 queue heads.

The userid queue is a misnomer. It refers to any destination that cannot be resolved to a binary JES2 route code (ie local, U1234, or R1234). If the destination has a DESTID statement that equates it to a binary route code, then this is not a userid destination. This is important to understand later.

There are also 2 separate queues, one for ALL NJE output and one for ALL held output.

A second queue that runs through the output queue is the characteristics queue (the Char JOE queue). All JOEs that have the same value for form, FCB, UCS, writer, overlay, PRMODE, User part of destination, SECLABEL, and burst are queued to the same Char JOE.

Finally, there is a job level queue. All output for the same job is queued off the JQE.



SYSOUT Queuing

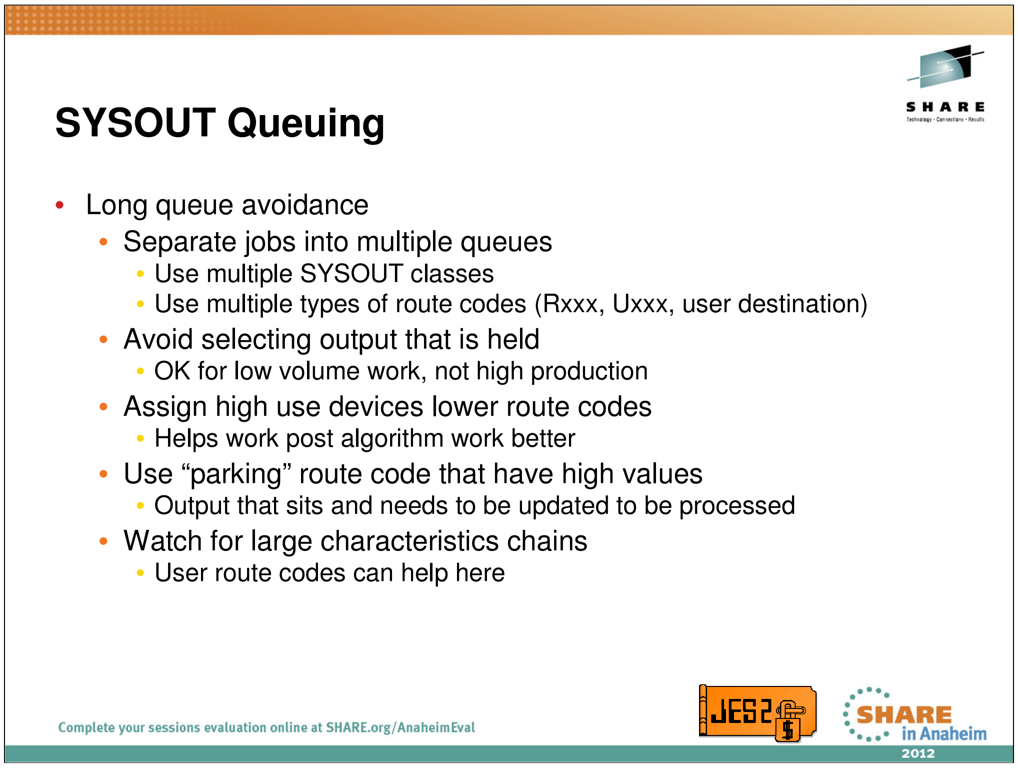
- Avoid long queue problems
  - Long class queues
    - Impacts performance of selecting work
      - *SAPI, Printers, FSS, etc*
      - *As seen in CPU used in trace id 20*
    - Impacts SYSOUT creation time
      - *Need to add to proper place in queue*
  - Long characteristics queues
    - Impacts output creation time
      - *Need to find where to add to queue*
  - Long job queue
    - Minimal impact (queue is not ordered)

Complete your sessions evaluation online at SHARE.org/AnaheimEval

One way to avoid performance problems is to ensure that SYSOUT queues do not get too long. If too many elements are on one queue, then this can impact processing of elements on the queue. In general, removing an element from a queue is not a problem. JES2 maintains a backwards pointer to simplify deleting an element. But large queues can impact SYSOUT creation and selection.

The 108 class queues and the NJE queue are maintained in order by binary route code and priority within the route code. The held queue is simple LIFO queues. Adding to the class or NJE queue requires looping through the elements looking for the correct place to add an element. Higher route codes implies a larger search. Similarly, selecting from a class queue (especially if route code is part of the selection) requires looping to the correct place in the queue to start processing. The held queue is not ordered in any meaningful way (except new stuff is at the beginning) so selecting from that queue can be slow.

The characteristic queue is ordered by SYSOUT class and then priority. Again, adding to that queue involves scanning to the correct place to add the element. Since we do not select using the characteristic queue, a large queue only impacts creating JOEs.



SYSOUT Queuing

- Long queue avoidance
  - Separate jobs into multiple queues
    - Use multiple SYSOUT classes
    - Use multiple types of route codes (Rxxx, Uxxx, user destination)
  - Avoid selecting output that is held
    - OK for low volume work, not high production
  - Assign high use devices lower route codes
    - Helps work post algorithm work better
  - Use "parking" route code that have high values
    - Output that sits and needs to be updated to be processed
  - Watch for large characteristics chains
    - User route codes can help here

Complete your sessions evaluation online at SHARE.org/AnaheimEval

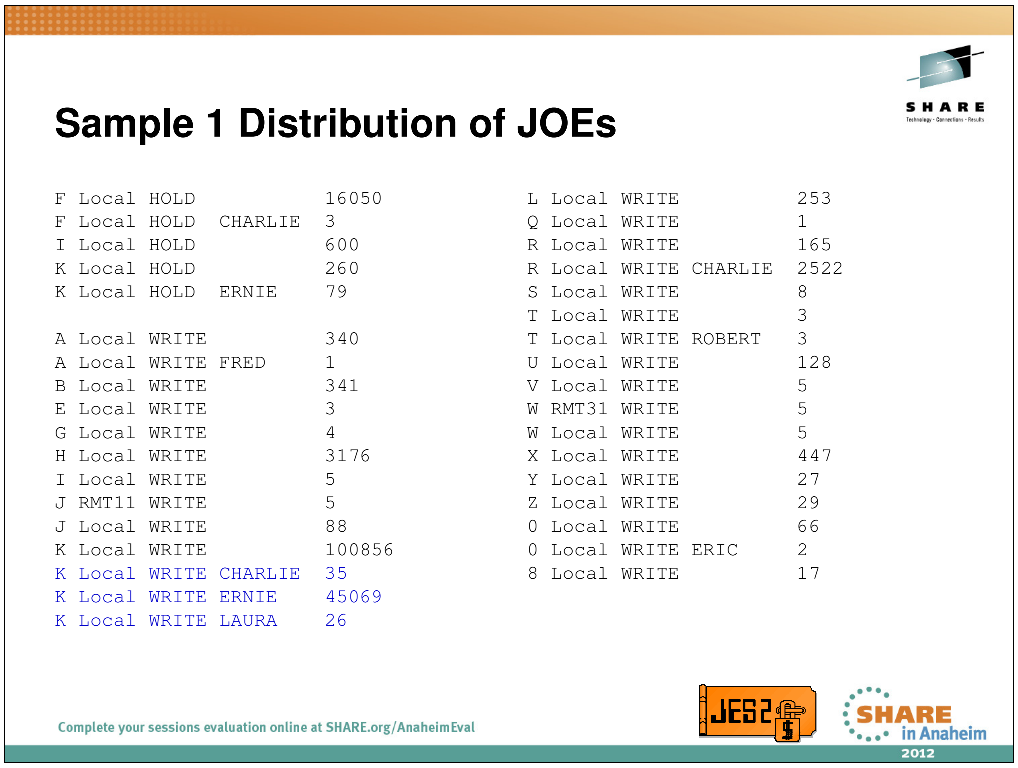
Here are some techniques to avoid long SYSOUT queues. The most basic is to ensure that the 108 SYSOUT queues are all well utilized. Having 90% of your print output is class A with one of 1000 different Uxxxx route codes will not give the best performance. Separate them either by altering DESTIDs to route some output to Rxxxx and other to user route code OR get them to use different SYSOUT classes (as appropriate).

Under all circumstances, try to avoid major processing that select held output. This is probably going to give the worst performance of any SYSOUT selection. Held was intended for TSO users to be able to look at output and either delete it or print it (by making it not held). It was not intended for 1000 convenience printers to process data set via SAPI. The TSO or web application that looks for some data set is OK. If you need to process held output for a specific job on the held queue, it may be faster to find the SYSOUT using extended status and then use the SYSOUT token to select it using SAPI.

JES2 work selection is optimized for a large number of relatively idle printers. Printers that generally only have one piece of output queued to them perform better than those that build up a print queue. Those low volume printers should be assigned higher route codes than high use printers that do build a queue.

If there are printers that no longer exist (but output may be queued to them) or if there is a need to "park" SYSOUT on a class queue (route it to a dummy device for some later processing) assign those devices the higher route codes. This keeps them out of the normal selection scans.

Large characteristics queues can creep up on a system. One way to avoid this is the use of user route codes. Though this can cause problems on the class queues, it can reduce problems on the characteristics queue. Use the $D F command to display the characteristics JOEs and add the counts for all the queues displayed. This indicates how large each characteristics queue is.

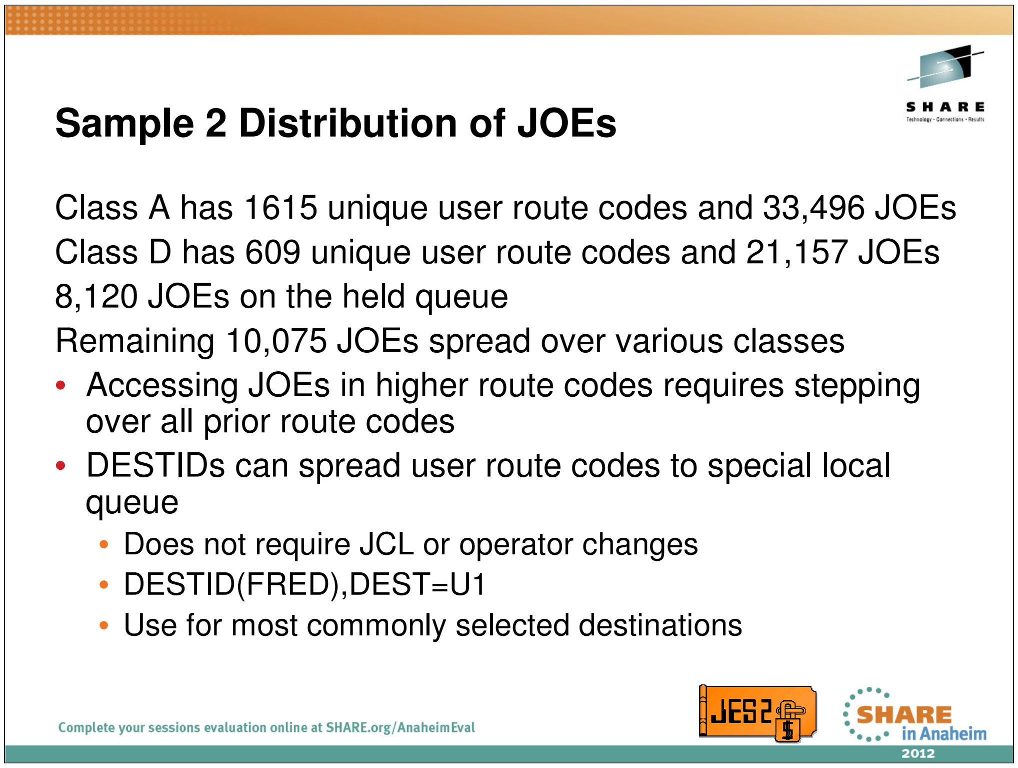


Sample 1 Distribution of JOEs

```
F Local HOLD            16050           L Local WRITE         253
F Local HOLD  CHARLIE   3               Q Local WRITE         1
I Local HOLD            600             R Local WRITE         165
K Local HOLD            260             R Local WRITE CHARLIE 2522
K Local HOLD  ERNIE     79              S Local WRITE         8
                                        T Local WRITE         3
A Local WRITE           340             T Local WRITE ROBERT  3
A Local WRITE FRED      1               U Local WRITE         128
B Local WRITE           341             V Local WRITE         5
E Local WRITE           3               W RMT31 WRITE         5
G Local WRITE           4               W Local WRITE         5
H Local WRITE           3176            X Local WRITE         447
I Local WRITE           5               Y Local WRITE         27
J RMT11 WRITE           5               Z Local WRITE         29
J Local WRITE           88              0 Local WRITE         66
K Local WRITE           100856          0 Local WRITE ERIC    2
K Local WRITE CHARLIE   35              8 Local WRITE         17
K Local WRITE ERNIE     45069
K Local WRITE LAURA     26
```

Complete your sessions evaluation online at SHARE.org/AnaheimEval

We have had to look at distributions of JOEs on the various JOE chains over the years. Here is a distribution from a customer that shows where their JOEs are queued. For the most part, the non-held JOEs are all on unique queue heads and depending on how they are selecting output to process, there should be little queuing impact for this customer. Only the class K user destination queue has a bit of an overload, but only if there is lots of LAURA routed output being selected. That would have to step over the 45K ERNIE output to be selected. My only recommendation for this customer would be to consider assigning ERNIE a DESTID with a remote route code thus moving the 45K JOEs to the remote class K queue.



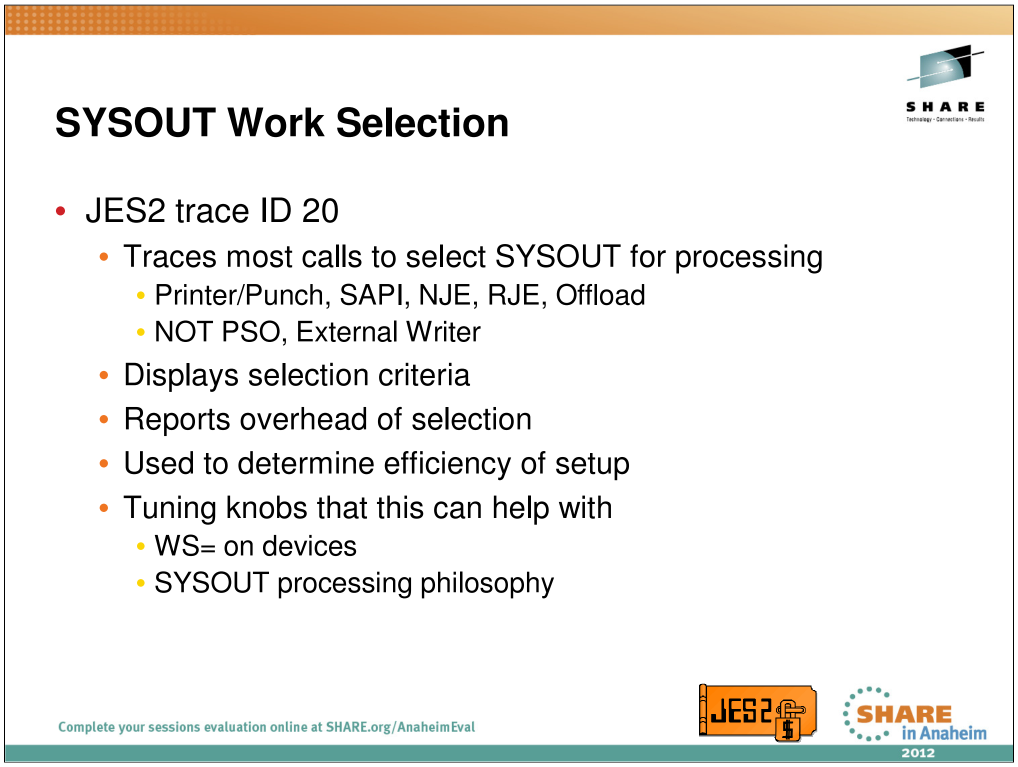
Sample 2 Distribution of JOEs

Class A has 1615 unique user route codes and 33,496 JOEs
Class D has 609 unique user route codes and 21,157 JOEs
8,120 JOEs on the held queue
Remaining 10,075 JOEs spread over various classes

- Accessing JOEs in higher route codes requires stepping over all prior route codes
- DESTIDs can spread user route codes to special local queue
  - Does not require JCL or operator changes
  - DESTID(FRED),DEST=U1
  - Use for most commonly selected destinations

Complete your sessions evaluation online at SHARE.org/AnaheimEval

Now this customer has a mess on their hands. They have 33K JOEs queued to 1,615 user route codes all in class A. The full list would have been an eye chart. Some route codes has thousands of JOEs. This customer would benefit from distributing their JOEs to other queues using either DESTIDs to assign remote or special local route codes to some of the destinations, or by spreading the SYSOUT across more classes. Since this is a point in time capture of the data, it is possible that some or all of this work may be waiting to be printed. But there are still improvements that can be made.

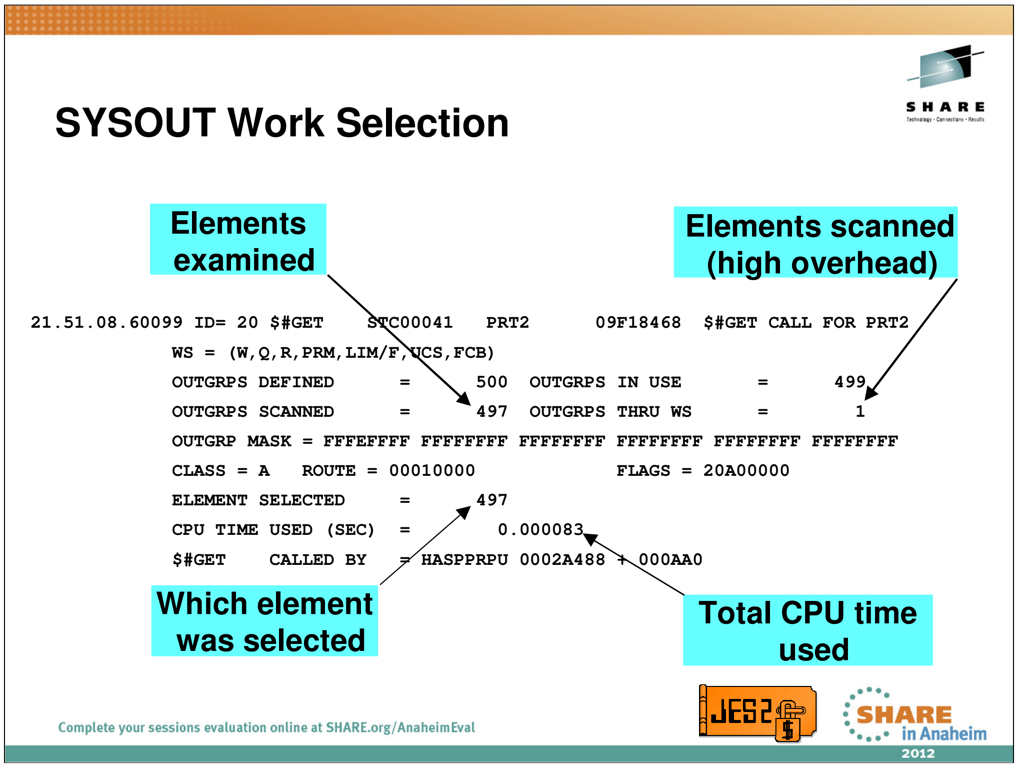


SYSOUT Work Selection

- JES2 trace ID 20
  - Traces most calls to select SYSOUT for processing
    - Printer/Punch, SAPI, NJE, RJE, Offload
    - NOT PSO, External Writer
  - Displays selection criteria
  - Reports overhead of selection
  - Used to determine efficiency of setup
  - Tuning knobs that this can help with
    - WS= on devices
    - SYSOUT processing philosophy

Complete your sessions evaluation online at SHARE.org/AnaheimEval

Trace ID 20 can be used to get a sense of the overhead associated with SYSOUT work selection. It can be used to understand the impact of large class queues or to optimize work selection criteria. The reports on real devices, SAPI, offload and NJE. It cannot be used for PSO requests (including external writers). The trace include the device name, the work selection list, and the effort used to select an item (including the CPU associated with the request).



SYSOUT Work Selection

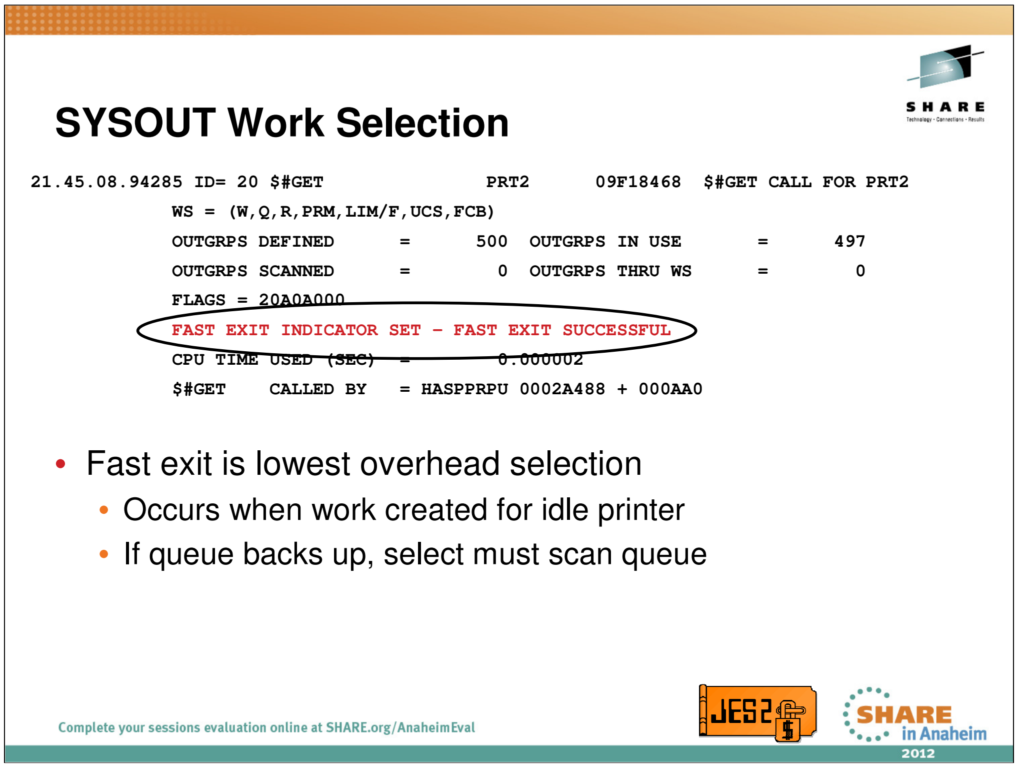
```
21.51.08.60099 ID= 20 $#GET    STC00041   PRT2      09F18468   $#GET CALL FOR PRT2
             WS = (W,Q,R,PRM,LIM/F,UCS,FCB)
             OUTGRPS DEFINED      =       500  OUTGRPS IN USE       =       499
             OUTGRPS SCANNED      =       497  OUTGRPS THRU WS      =         1
             OUTGRP MASK = FFFEFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
             CLASS = A   ROUTE = 00010000            FLAGS = 20A00000
             ELEMENT SELECTED     =       497
             CPU TIME USED (SEC)  =         0.000083
             $#GET    CALLED BY   = HASPPRPU 0002A488 + 000AA0
```

Elements examined
Elements scanned (high overhead)
Which element was selected
Total CPU time used
Complete your sessions evaluation online at SHARE.org/AnaheimEval
SHARE in Anaheim 2012

The 4 most interesting elements of trace 20 are shown here. The first thing to look at is the elements examined vs the elements scanned. Scanning is a high CPU overhead process. Keeping this number down will greatly improve performance. In many cases, you can keep this number down to 1. This number is mostly impacted by the WS list specified. The elements examined is how many elements were looked at and quickly dismissed. Keeping this number down is a factor of how the output is queued. This is an indication of queue length.

The element selected indicates if we kept scanning after we found the element we finally selected. If this is not the same (or nearly the same) as the elements examined, then the selection criteria is causing extra overhead.

The CPU time used is a raw value that indicates the cost of this selection. Ultimately, this is the number that needs to be kept low.



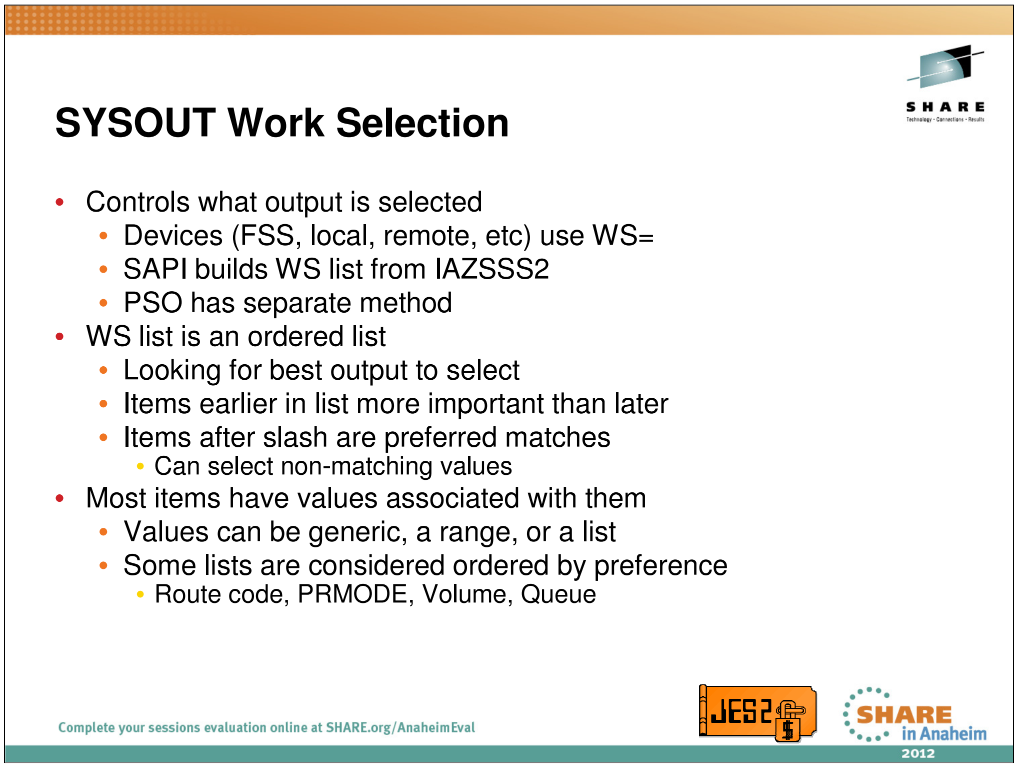
SYSOUT Work Selection

```
21.45.08.94285 ID= 20 $#GET               PRT2      09F18468  $#GET CALL FOR PRT2
             WS = (W,Q,R,PRM,LIM/F,UCS,FCB)
             OUTGRPS DEFINED     =      500  OUTGRPS IN USE       =      497
             OUTGRPS SCANNED     =        0  OUTGRPS THRU WS      =        0
             FLAGS = 20A0A000
             FAST EXIT INDICATOR SET - FAST EXIT SUCCESSFUL
             CPU TIME USED (SEC)  =         0.000002
             $#GET    CALLED BY   = HASPPRPU 0002A488 + 000AA0
```

- Fast exit is lowest overhead selection
  - Occurs when work created for idle printer
  - If queue backs up, select must scan queue

Complete your sessions evaluation online at SHARE.org/AnaheimEval

Print selection tries to assign new output directly to an output device. If the device is idle, it can perform the assignment and when the device wakes up, it simply processes the assigned output group. This is the fastest way a device can select output. However, this processing does not work if there is a queue of output.



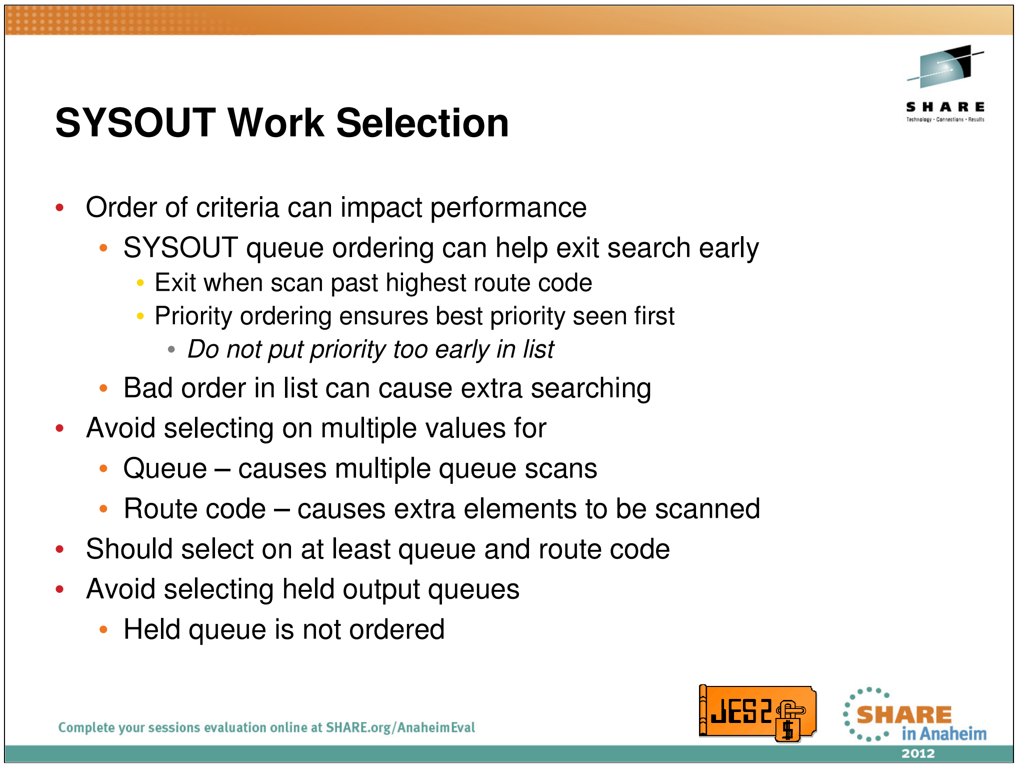
SYSOUT Work Selection

- Controls what output is selected
  - Devices (FSS, local, remote, etc) use WS=
  - SAPI builds WS list from IAZSSS2
  - PSO has separate method
- WS list is an ordered list
  - Looking for best output to select
  - Items earlier in list more important than later
  - Items after slash are preferred matches
    - Can select non-matching values
- Most items have values associated with them
  - Values can be generic, a range, or a list
  - Some lists are considered ordered by preference
    - Route code, PRMODE, Volume, Queue

Complete your sessions evaluation online at SHARE.org/AnaheimEval

SYSOUT work selection controls what output is selected for processing. Selection is controlled by an list of criteria called the work selection list (specified as WS= on most devices). SAPI constructs this list using parameters passed on the IAZSSS2 macro. The PSO interface (including the external writer) uses a separate method to select work to process.

The order of items in the work selection list controls what attribute is most important when selecting output. The earlier in the list an item is, the more important a match on that item is.



SYSOUT Work Selection

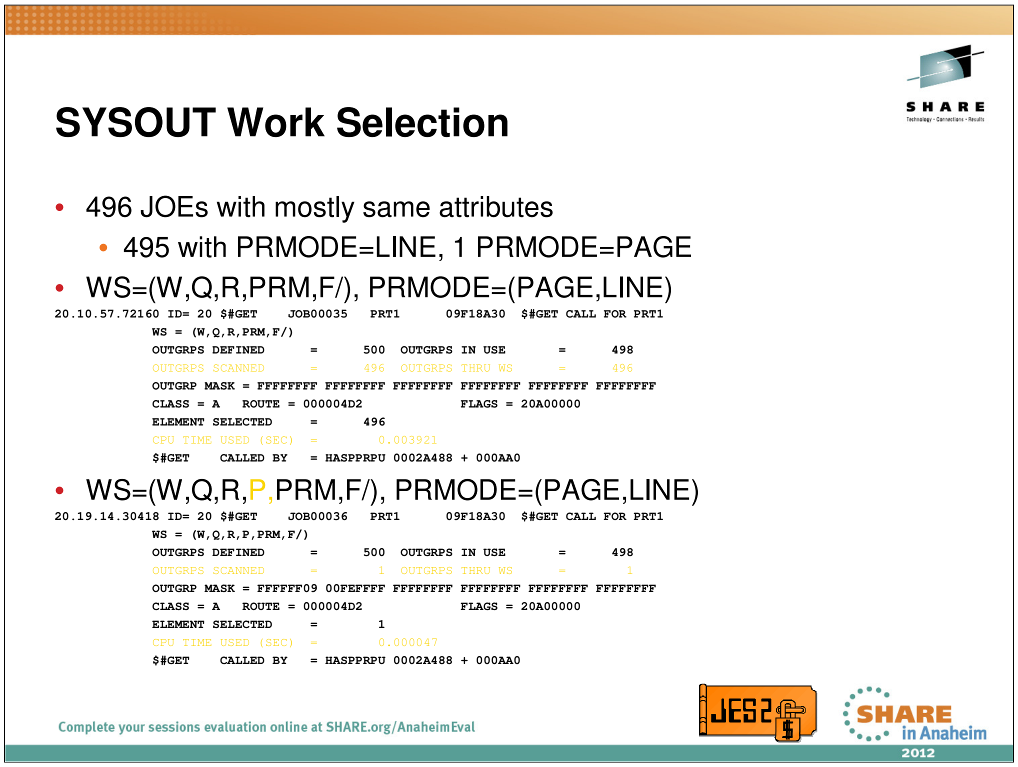
- Order of criteria can impact performance
  - SYSOUT queue ordering can help exit search early
    - Exit when scan past highest route code
    - Priority ordering ensures best priority seen first
      - *Do not put priority too early in list*
  - Bad order in list can cause extra searching
- Avoid selecting on multiple values for
  - Queue – causes multiple queue scans
  - Route code – causes extra elements to be scanned
- Should select on at least queue and route code
- Avoid selecting held output queues
  - Held queue is not ordered

Complete your sessions evaluation online at SHARE.org/AnaheimEval

SYSOUT work selection is always looking for the best output group to assign to a device. Because of this, it is possible to find a JOE to process but then continue searching for a better JOE. This may be needed if that is the requirement of the device. But often, it does not matter which JOE is selected as long as it matches the criteria. One way to control the processing is to take advantage of how the SYSOUT is queued. The code will exit the scan once it is clear that a better JOE will not be found.

One way to ensure that a better JOE cannot be found is to not code multiple values for Queue and Route code. Multiple values just forces the code to scan multiple queues looking for a match. Also, devices should select on at least queue and route code. If they do not, then extra queues must be scanned to find the best output.

Avoid selecting from the held queue since it is not ordered and often the entire queue must be scanned looking for a proper match.



SYSOUT Work Selection

- 496 JOEs with mostly same attributes
  - 495 with PRMODE=LINE, 1 PRMODE=PAGE
- WS=(W,Q,R,PRM,F/), PRMODE=(PAGE,LINE)

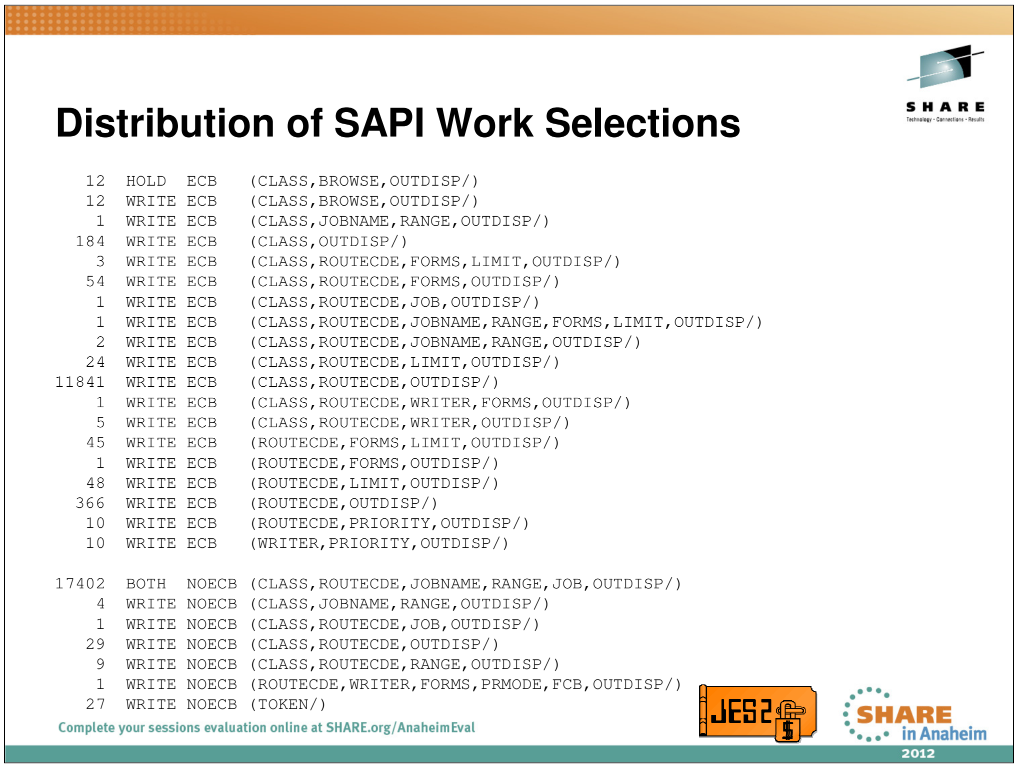
```
20.10.57.72160 ID= 20 $#GET    JOB00035   PRT1      09F18A30  $#GET CALL FOR PRT1
             WS = (W,Q,R,PRM,F/)
             OUTGRPS DEFINED      =      500  OUTGRPS IN USE       =      498
             OUTGRPS SCANNED      =      496  OUTGRPS THRU WS      =      496
             OUTGRP MASK = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
             CLASS = A   ROUTE = 000004D2             FLAGS = 20A00000
             ELEMENT SELECTED     =      496
             CPU TIME USED (SEC)  =        0.003921
             $#GET    CALLED BY   = HASPPRPU 0002A488 + 000AA0
```

- WS=(W,Q,R,P,PRM,F/), PRMODE=(PAGE,LINE)

```
20.19.14.30418 ID= 20 $#GET    JOB00036   PRT1      09F18A30  $#GET CALL FOR PRT1
             WS = (W,Q,R,P,PRM,F/)
             OUTGRPS DEFINED      =      500  OUTGRPS IN USE       =      498
             OUTGRPS SCANNED      =        1  OUTGRPS THRU WS      =        1
             OUTGRP MASK = FFFFFF09 00FEFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
             CLASS = A   ROUTE = 000004D2             FLAGS = 20A00000
             ELEMENT SELECTED     =        1
             CPU TIME USED (SEC)  =        0.000047
             $#GET    CALLED BY   = HASPPRPU 0002A488 + 000AA0
```

Complete your sessions evaluation online at SHARE.org/AnaheimEval

A test was set up with 496 identical SYSOUT JOEs with the exception that the last JOE has PRMODE=PAGE (all others have PRMODE=LINE). Work selection was set up with a fairly standard list that included PRMODE. The PRMODE for the device specified both PAGE and LINE mode (in that order). Since PRMODE is an order dependent list, PAGE mode is preferred over LINE mode output. As a result, when the printer was started, the PAGE mode JOE was selected AFTER scanning all 496 JOEs in the queue. However, if the intent was not to prefer PAGE mode over LINE mode, you can greatly improve performance by adding Priority to the WS list AFTER Queue and Routcode. Priority has the effect of negating the order preference of any criteria after it. However, do NOT move priority above Routcode if there are multiple Routcodes in the list. Since class queues are ordered by routcode and then priority, putting priority ahead of routcode in the WS list will cause excessive scanning of output looking for the best priority.



Distribution of SAPI Work Selections

```
   12  HOLD  ECB   (CLASS,BROWSE,OUTDISP/)
   12  WRITE ECB   (CLASS,BROWSE,OUTDISP/)
    1  WRITE ECB   (CLASS,JOBNAME,RANGE,OUTDISP/)
  184  WRITE ECB   (CLASS,OUTDISP/)
    3  WRITE ECB   (CLASS,ROUTECDE,FORMS,LIMIT,OUTDISP/)
   54  WRITE ECB   (CLASS,ROUTECDE,FORMS,OUTDISP/)
    1  WRITE ECB   (CLASS,ROUTECDE,JOB,OUTDISP/)
    1  WRITE ECB   (CLASS,ROUTECDE,JOBNAME,RANGE,FORMS,LIMIT,OUTDISP/)
    2  WRITE ECB   (CLASS,ROUTECDE,JOBNAME,RANGE,OUTDISP/)
   24  WRITE ECB   (CLASS,ROUTECDE,LIMIT,OUTDISP/)
11841  WRITE ECB   (CLASS,ROUTECDE,OUTDISP/)
    1  WRITE ECB   (CLASS,ROUTECDE,WRITER,FORMS,OUTDISP/)
    5  WRITE ECB   (CLASS,ROUTECDE,WRITER,OUTDISP/)
   45  WRITE ECB   (ROUTECDE,FORMS,LIMIT,OUTDISP/)
    1  WRITE ECB   (ROUTECDE,FORMS,OUTDISP/)
   48  WRITE ECB   (ROUTECDE,LIMIT,OUTDISP/)
  366  WRITE ECB   (ROUTECDE,OUTDISP/)
   10  WRITE ECB   (ROUTECDE,PRIORITY,OUTDISP/)
   10  WRITE ECB   (WRITER,PRIORITY,OUTDISP/)

17402  BOTH  NOECB (CLASS,ROUTECDE,JOBNAME,RANGE,JOB,OUTDISP/)
    4  WRITE NOECB (CLASS,JOBNAME,RANGE,OUTDISP/)
    1  WRITE NOECB (CLASS,ROUTECDE,JOB,OUTDISP/)
   29  WRITE NOECB (CLASS,ROUTECDE,OUTDISP/)
    9  WRITE NOECB (CLASS,ROUTECDE,RANGE,OUTDISP/)
    1  WRITE NOECB (ROUTECDE,WRITER,FORMS,PRMODE,FCB,OUTDISP/)
   27  WRITE NOECB (TOKEN/)
```

Complete your sessions evaluation online at SHARE.org/AnaheimEval

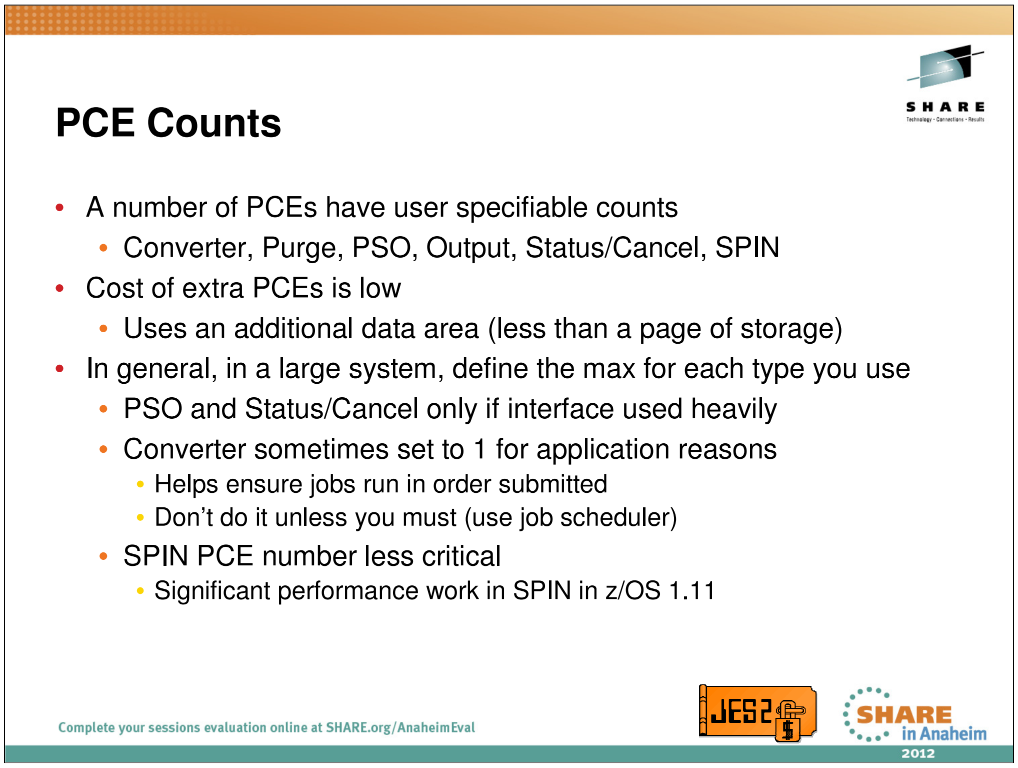
This is an interesting chart. This data is accumulated from multiple dumps from multiple customers. This is a total of all SAPI request in all the dumps. The top ones use ECBs and are POSTed when new work arrive. The bottom ones are not posted and just select work until there is no more to select. A couple of interesting selection choices:

HOLD ECB (CLASS,BROWSE,OUTDISP/) – this is selecting from the HELD queue by class. Since the held queue is not organized by class, this is an expensive request.

(ROUTECDE,….) – these do NOT select on class and thus must search all non-held class queues for matches on the passed route code. If we think back to some of the earlier examples, this can be a very expensive scan that looks at many queues for a specific route code.

(WRITER,…) – this is also expensive since there is nothing other than writer that we can match on. WRITER is not a criteria in any of the sorting, so this request will examine every non-held JOE using work selection looking for a match. This is the most expensive of the searches listed.

17402 BOTH NOECB (CLASS,ROUTECDE,JOBNAME,RANGE,JOB,OUTDISP/) – this is interesting. This number happens to be from a single dump where they exhausted one of the SAPI related data spaces. Turns out to be an application error that caused a SAPI request to not be cleaned up. Over time, it filled a data space and caused an IPL.



PCE Counts

- A number of PCEs have user specifiable counts
  - Converter, Purge, PSO, Output, Status/Cancel, SPIN
- Cost of extra PCEs is low
  - Uses an additional data area (less than a page of storage)
- In general, in a large system, define the max for each type you use
  - PSO and Status/Cancel only if interface used heavily
  - Converter sometimes set to 1 for application reasons
    - Helps ensure jobs run in order submitted
    - Don't do it unless you must (use job scheduler)
  - SPIN PCE number less critical
    - Significant performance work in SPIN in z/OS 1.11

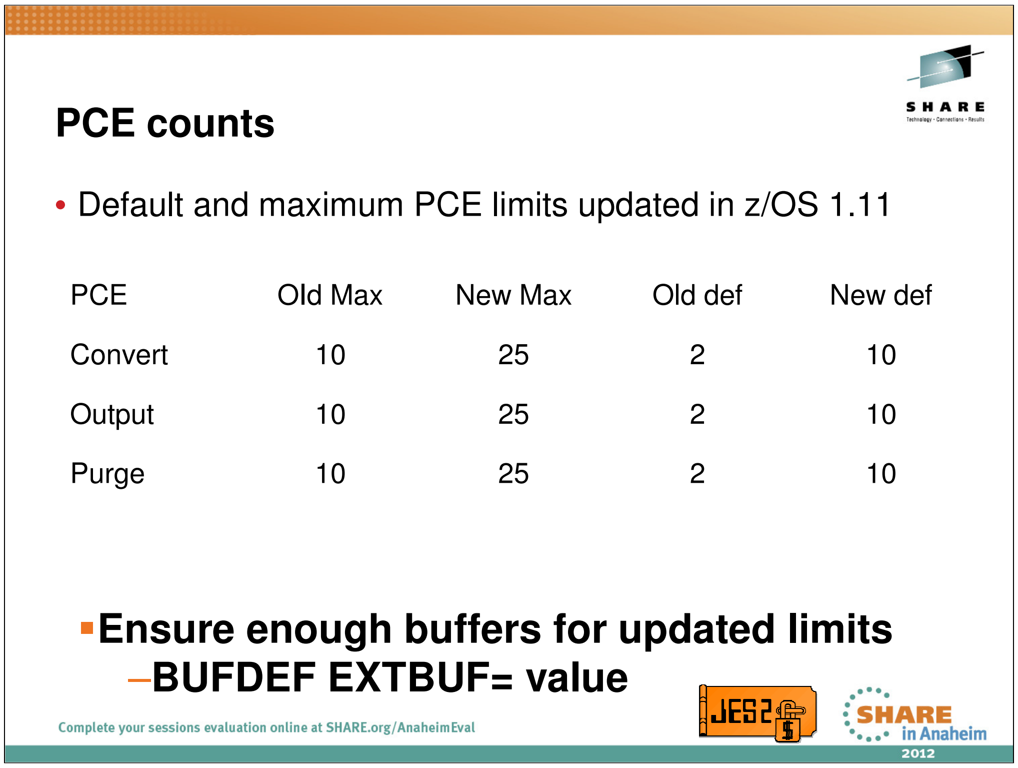
Complete your sessions evaluation online at SHARE.org/AnaheimEval

The number of certain PCEs can be specified via an initialization option. The PCEDEF initialization statement control the number JES2 defines. It can only be specified at initialization (no commands to adjust the number). In general, a large system should set these values to the maximum. There is little cost (performance or resources) but lots of benefits. However, there are some PCEs that you may not want to increase the number of:

•The PSO and Status/Cancel (STAC) PCE process requests for the PSO interface (including external writer, TSO RECEIVE and OUTPUT commands) and the TSO STATUS and CANCEL commands. If these interfaces are not heavily used, then increasing these PCEs would not do much good.

•The converter PCE count in some installation is set to 1. This is because application sometimes require that jobs are executed in the order they are submitted. With more than 1 converter PCE, then jobs can run in different order because conversion time is related to the complexity of the JCL. This could cause simple jobs to convert before complex jobs and thus the jobs run out of order. This is generally only an issue for installation that explicitly set the converter PCE count to 1. IBM does not recommend setting converter PCE counts to 1. You should use job schedulers to ensure jobs run in the correct order.

•The SPIN PCE count is somewhat less critical. The count is mainly for redundancy rather than performance. Also the overall SPIN process performance has been greatly enhanced with changes made in z/OS 1.11. This performance enhancement does not require additional PCEs.



PCE counts

- Default and maximum PCE limits updated in z/OS 1.11

PCE	Old Max	New Max	Old def	New def
Convert	10	25	2	10
Output	10	25	2	10
Purge	10	25	2	10

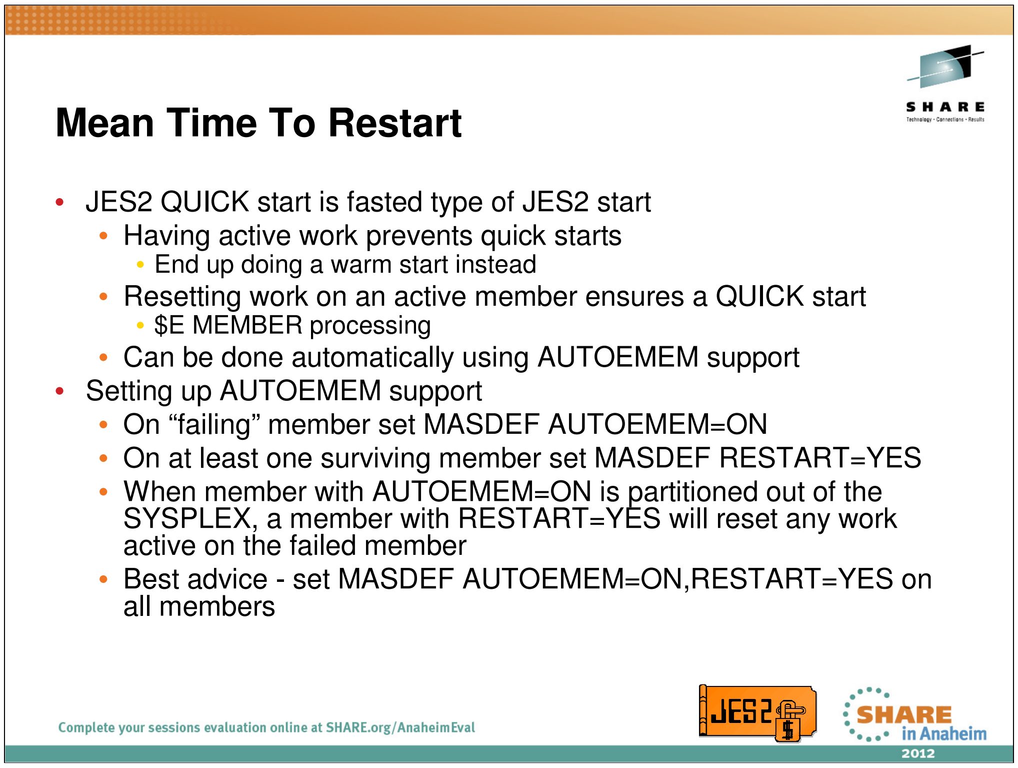
- **Ensure enough buffers for updated limits**
  - **BUFDEF EXTBUF= value**

Complete your sessions evaluation online at SHARE.org/AnaheimEval

The default and maximum number of PCEs has been increased in z/OS 1.11. Increasing the number of PCEs will improve parallelism and improve performance.

The converter PCEs does most of it's work in a subtask, keeping the main task free to do other work. Increasing the number of converter PCEs improves job submission time.

The OUTPUT and PURGE PCEs are both I/O intensive processes. Increasing the number of these PCEs improves throughput when jobs complete and are purged. However, these PCEs both need buffers to perform their I/O. If you have specify these values in your initialization deck, you may need to increase the value when you increase the number of purge or output PCEs. A general rule of thumb is to define 3 buffers per PCE you define.



Mean Time To Restart

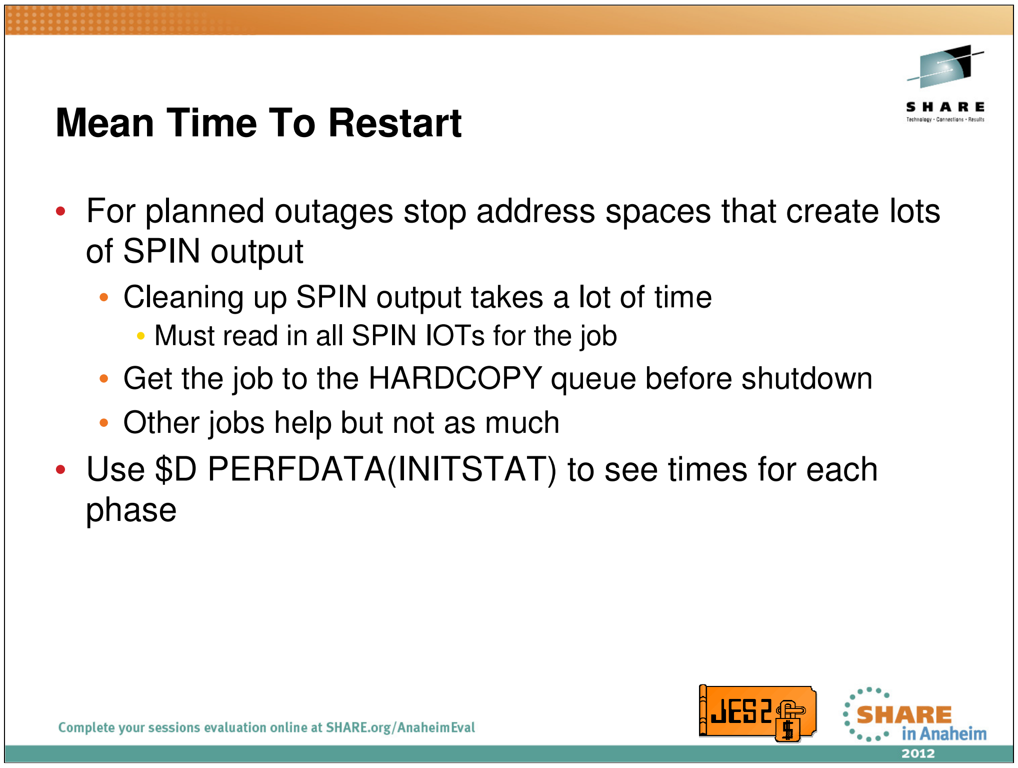
- JES2 QUICK start is fasted type of JES2 start
  - Having active work prevents quick starts
    - End up doing a warm start instead
  - Resetting work on an active member ensures a QUICK start
    - $E MEMBER processing
  - Can be done automatically using AUTOEMEM support
- Setting up AUTOEMEM support
  - On “failing” member set MASDEF AUTOEMEM=ON
  - On at least one surviving member set MASDEF RESTART=YES
  - When member with AUTOEMEM=ON is partitioned out of the SYSPLEX, a member with RESTART=YES will reset any work active on the failed member
  - Best advice - set MASDEF AUTOEMEM=ON,RESTART=YES on all members

Complete your sessions evaluation online at SHARE.org/AnaheimEval

This is the “other MTTR”, mean time to restart. The goal is to reduce the time it takes to get a member back up after a system crash. One thing you can do to speed up JES2 restart Is to ensure JES2 that the failed member does a quick start. If there is work (jobs and SYSOUT) that is still marked active when a system is restarted, warm start processing must find that work and reset it so that it can complete processing. This work can be done when the member is restarted (as part of startup processing) or it can be done by a surviving member. By having it done by a surviving member, it ensures the failed member does a quick start. The process of resetting with from a failed member is called E-Member processing.

E-Member processing can be done by an operator command or automatically by JES2. There are 2 options on MASDEF that control this processing. The first is the AUTOEMEM operand, Setting this to ON indicates that the member should be restarted automatically if it fails. This should be set on the failing member. In fact, all members should probably have this set to ON. The second operand is RESTART. This controls what member can do restart processing for failed members. This should be set to YES on members that have the capacity to restart the failed work (in general, any member can do this). Note, JES2 will temporarily update the MASDEF HOLD= value for any member performing the reset processing while the processing is active.

In general, it is good practice to set MASDEF AUTOEMEM=ON, RESTART=YES on all MAS members.



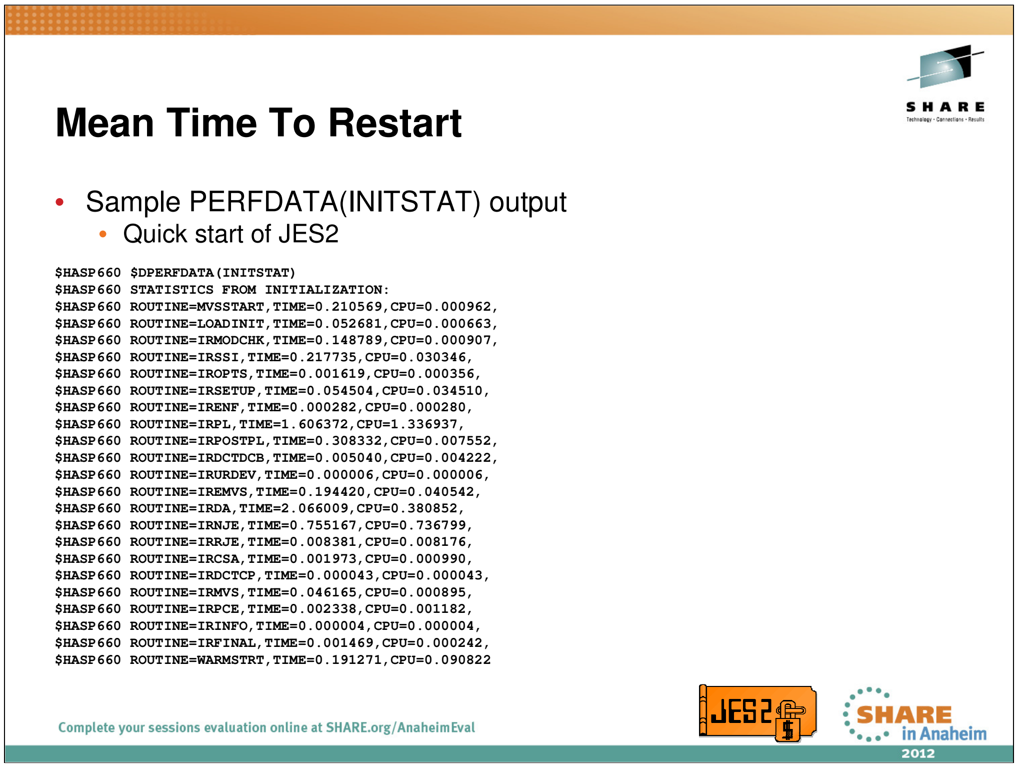
Mean Time To Restart

- For planned outages stop address spaces that create lots of SPIN output
  - Cleaning up SPIN output takes a lot of time
    - Must read in all SPIN IOTs for the job
  - Get the job to the HARDCOPY queue before shutdown
  - Other jobs help but not as much
- Use $D PERFDATA(INITSTAT) to see times for each phase

Complete your sessions evaluation online at SHARE.org/AnaheimEval

If this is a planned outage, one other thing that can speed up restart processing is to get jobs that create lots of SPIN output our of execution before shutdown. If a job was executing and it ever produced SPIN SYSOUT, then restart processing must read the control blocks for ALL the SPIN output the job has ever created (include stuff that was purged) in an attempt to locate output that was active and had not spun. This can involve a significant amount of I/O. If you can get these jobs to the hardcopy queue, then restart processing does not need to do anything for these jobs. Any job that you can get to the hardcopy queue will improve restart time, but not as much as ones that create SPIN SYSOUT.

There is a command in JES2 that breaks down where JES2 is spending its time during initialization. The $DPERFDATA(INITSTAT) can be used at any time after JES2 has started to display the time for each phase of JES2 warm start,



Mean Time To Restart

- Sample PERFDATA(INITSTAT) output
  - Quick start of JES2

```
$HASP660 $DPERFDATA(INITSTAT)
$HASP660 STATISTICS FROM INITIALIZATION:
$HASP660 ROUTINE=MVSSTART,TIME=0.210569,CPU=0.000962,
$HASP660 ROUTINE=LOADINIT,TIME=0.052681,CPU=0.000663,
$HASP660 ROUTINE=IRMODCHK,TIME=0.148789,CPU=0.000907,
$HASP660 ROUTINE=IRSSI,TIME=0.217735,CPU=0.030346,
$HASP660 ROUTINE=IROPTS,TIME=0.001619,CPU=0.000356,
$HASP660 ROUTINE=IRSETUP,TIME=0.054504,CPU=0.034510,
$HASP660 ROUTINE=IRENF,TIME=0.000282,CPU=0.000280,
$HASP660 ROUTINE=IRPL,TIME=1.606372,CPU=1.336937,
$HASP660 ROUTINE=IRPOSTPL,TIME=0.308332,CPU=0.007552,
$HASP660 ROUTINE=IRDCTDCB,TIME=0.005040,CPU=0.004222,
$HASP660 ROUTINE=IRURDEV,TIME=0.000006,CPU=0.000006,
$HASP660 ROUTINE=IREMVS,TIME=0.194420,CPU=0.040542,
$HASP660 ROUTINE=IRDA,TIME=2.066009,CPU=0.380852,
$HASP660 ROUTINE=IRNJE,TIME=0.755167,CPU=0.736799,
$HASP660 ROUTINE=IRRJE,TIME=0.008381,CPU=0.008176,
$HASP660 ROUTINE=IRCSA,TIME=0.001973,CPU=0.000990,
$HASP660 ROUTINE=IRDCTCP,TIME=0.000043,CPU=0.000043,
$HASP660 ROUTINE=IRMVS,TIME=0.046165,CPU=0.000895,
$HASP660 ROUTINE=IRPCE,TIME=0.002338,CPU=0.001182,
$HASP660 ROUTINE=IRINFO,TIME=0.000004,CPU=0.000004,
$HASP660 ROUTINE=IRFINAL,TIME=0.001469,CPU=0.000242,
$HASP660 ROUTINE=WARMSTRT,TIME=0.191271,CPU=0.090822
```

Complete your sessions evaluation online at SHARE.org/AnaheimEval

Some of the note worthy phases that are returned:

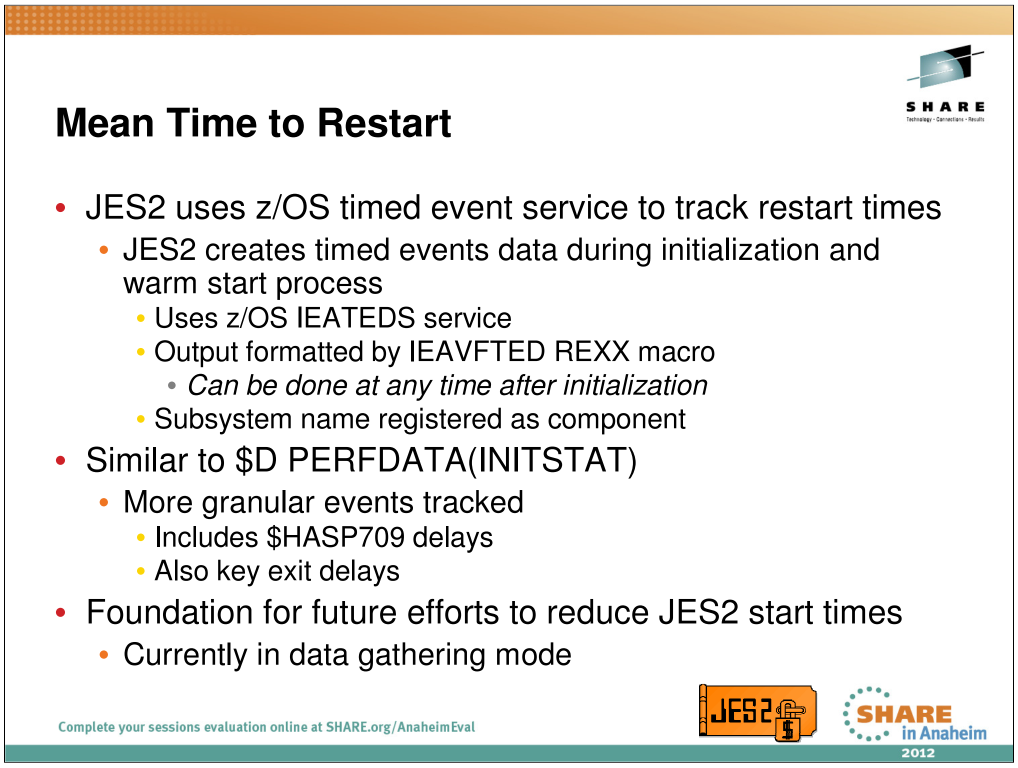
•MVSSTART – Time from the start JES2 command until JES2 first gets control (PROC processing, loading HASJES20, etc)

•LOADINIT – Time to load the HASPINIT load module

•IRPL – Initialization deck processing

•IRDA – Checkpoint and SPOOL initialization. Job and output queue verification

•WARMSTRT – Warm start processing (reset work busy on this system).

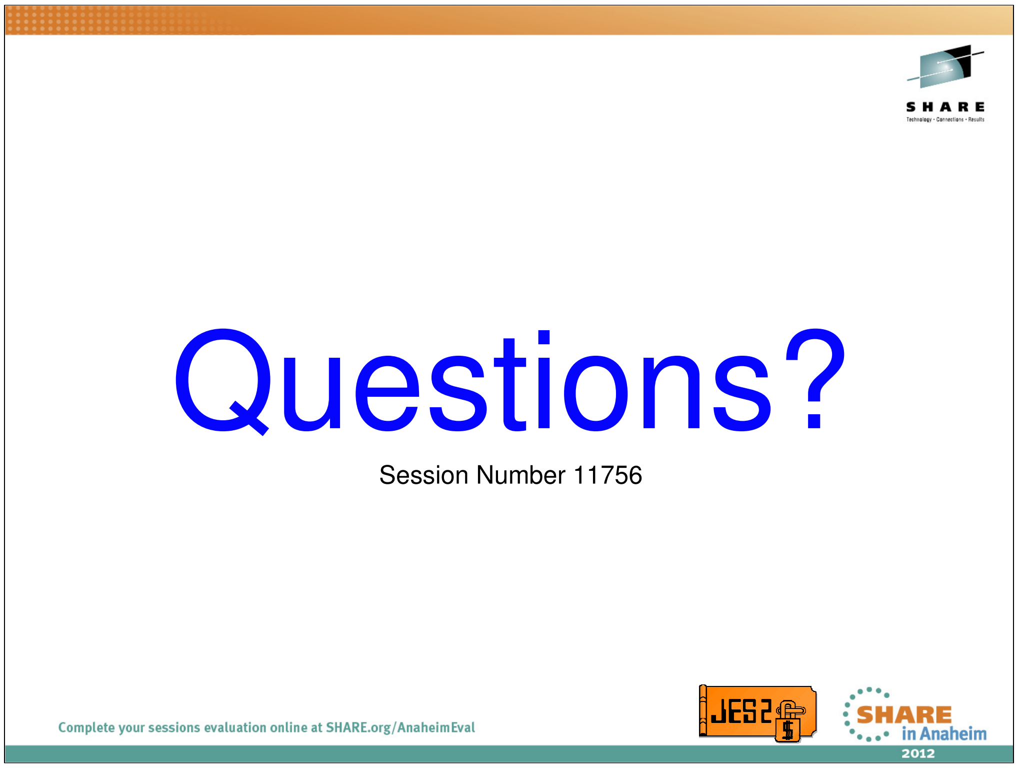


Mean Time to Restart

- JES2 uses z/OS timed event service to track restart times
  - JES2 creates timed events data during initialization and warm start process
    - Uses z/OS IEATEDS service
    - Output formatted by IEAVFTED REXX macro
      - *Can be done at any time after initialization*
    - Subsystem name registered as component
- Similar to $D PERFDATA(INITSTAT)
  - More granular events tracked
    - Includes $HASP709 delays
    - Also key exit delays
- Foundation for future efforts to reduce JES2 start times
  - Currently in data gathering mode

Complete your sessions evaluation online at SHARE.org/AnaheimEval

JES2 does use the new z/OS timed event service to record our startup events. This allows them to be examined in the context of other address spaces trying to start. The events recorded are a superset of the data in $DPERFDATA(INITSTAT). They include things that cause HASP709 delay messages and delays caused by exits. This is a foundation for future studied to understand how we can improve overall system restart times. If you want to see these records, use the IEAVFTED REXX service to format them any time after an IPL. For more information on using this service see http://publib.boulder.ibm.com/infocenter/zos/v1r12/topic/com.ibm.zos.r12.ieaa200/iea2a2b0774.htm



Questions?

Session Number 11756

Complete your sessions evaluation online at SHARE.org/AnaheimEval