Tuning PSB Pools Containing Large PSBs for Scheduling Performance

Bruce Naylor, IMS Development
Cedric Chen, IMS Performance

IBM Silicon Valley Lab
June 18, 2002

SUMMARY

To resolve a recent IMS customer performance problem involving severe scheduling delays after migrating to Shared Queues, a special study was launched in 2Q02 at SVL. The performance problem included degradation in transaction rate and in response time. IMS statistics showed a large increase in latch contentions, hold time, and wait time for the IMS scheduling-related latches. The conclusion of the study is that the problem was caused by the CSA PSB pool being too small, combined with a workload that included several very large PSBs with many PCBs and segments. The study shows that with a constrained PSB pool (DLI or CSA) and many large PSBs, the "cast out" processing done in the PSB pool manager can become a severe IMS performance bottleneck.

As a result of this study, APAR PQ61259 was developed to provide improved statistics and diagnostics in the following areas:

- **Latch Statistics:** Corrects problems which could (infrequently) cause incorrect values to be collected for certain time-related latch statistics.
- **Latch Diagnostics:** Changes the format of the IMS latch traces to include a full 8-byte STCK time stamp.
- **PSB Pool Statistics:** Adds new statistics to the IMS PSB pool manager to help give indications of performance problems with this pool.
- **IMS Log Record Analysis:** Changes the IMS log print formatter (DFSERA30) to decode the STCK values present in all IMS log records into human-readable time values.

There are five pools associated with scheduling:

- PSB pools (PSBP and DPSB)
- PSB work pool
- DMB pool
- DMB work pool
- EPCB pool

Section 9.9 of *IMS V7 Performance Monitoring and Tuning Update* (SG24-6404) has guidelines of tuning these 5 scheduling pools.

This paper focuses on tuning the PSB pools using the new enhanced performance data.

THE PERFORMANCE PROBLEM

After migrating to a shared queues environment on IMS Version 6, the customer reported severe performance degradation -- both in transaction rate and in response time. IMS Performance Analyzer reports showed large latch wait counts and times, and large latch hold times for the scheduling-related latches (SCHD, TCTB, APSB, PDRB, PSBP, DMBP, DMBB, and SUBQ). The CSA part of the PSB pool (pool type = DLMP) was set to 4 meg, and was 100% allocated. The DLI part of the PSB pool (pool type = DPSB) was set to 40 meg, and was 60% allocated. Note that having one or the other of the PSB pools at or near 100% allocation is not that unusual.

Another characteristic of this situation was that many of the transactions (in one dump, 62% of the active transaction codes (those with non-zero enqueue counts)) were in the same scheduling class. Having many transactions in a single class has been observed to cause performance degradation at other accounts and has also been observed to be of greater impact when migrating to shared queues.

One final characteristic at the customer's installation was the presence of certain transaction programs which did not follow the IMS convention of continuing to issue a GU call to the I/O PCB for the next message until they received a "QC" status code. These transactions instead received control, issued a GU for the first message, processed the message, and then terminated. In a shared queues environment, this can cause additional scheduling, because, when the program goes through termination, IMS has not done the final CQSREAD call to determine whether the message queue for the transaction is empty. This will cause IMS to try to schedule another dependent region to process the possible next message on the queue. Then, the terminating region will go look for more work to do, and will likely schedule for the same transaction again (unless limited, say by MAXRGN). We were concerned that this was also contributing to the increased latch contention.

PERFORMANCE STUDY

We attempted to duplicate the customer's symptoms, using the performance lab at SVL in San Jose, CA. Initial attempts to recreate this performance degradation at SVL were not successful. While we did not have the same workload as the customer, we attempted to recreate the same tuning conditions using our performance DSW workload by:

- Placing all transaction codes in the same scheduling class.
- Changing some (20%) of the transaction programs to terminate after the first message without a final GU call to the I/O PCB.
- Constraining the CSA PSB pool

None of these changes resulted in more than a small (5 to 10%) degradation in transaction rate. Latch contention remained low.

Further analysis of customer dumps revealed what we came to conclude as the problem. There were a small number of PSBs that were significantly larger than most (CSA PSBs around 50,000 bytes vs. an average of 4600 bytes; DLI PSBs around 500,000 bytes vs. an average of 27,000 bytes). Many of these PSBs had many PCBs for the same databases. As an example, one PSB contained 301 DB PCBs, and a total of 1879 segments. In some cases, the PCBs for the same databases all had the same SENSEGs with the same PROCOPTs; in others, the SENSEGs were different.

These large PSBs, coupled with a constrained PSB pool (in this case, CSA) can cause the process of finding space in the PSB pool to take a very long time (estimates for the customer environment are in the hundreds of milliseconds). With this information, we changed our DSW PSBs to include several with many repeated PCBs. When we ran with these PSBs and with a constrained CSA PSB pool, we were able to duplicate the performance problem. In fact, the performance degradation we observed was significantly greater than that seen at the customer. This was due primarily to our having created much larger SDB (segment descriptor block) chains by a factor of about 30.

Similar effects to the customer’s problem were seen: a dramatic drop in transaction rate, and very high latch contention, wait, and hold times for the scheduling-related latches. We added instrumentation into the PSB pool storage manager (DFSDLMP0), and observed that the process of obtaining PSB pool space was taking in the range of two to nine seconds. This effect was present whether or not the programs were present that did not loop back to do a second GU call to the I/O PCB.

ANALYSIS

The PSB pool is managed differently from many other IMS pools. Space is allocated for a PSB when needed during scheduling, and the PSB is built. When the program terminates, the PSB is marked as no longer in use, but the space is not released back to the pool. Instead, the PSB is left in the pool and is available for reuse if the same program is rescheduled later. This reduces the impact of creating the PSB for a newly-scheduled instance of a program that has already run.

Unless the PSB pools (both CSA and DLI) are large enough to contain all possible PSBs that were used during an IMS run, at some point one or the other pool will fill up. When this occurs, a request for space for a new PSB can fail. The PSB pool manager then goes through a process to “cast out” existing but not currently scheduled PSBs from the pool in order to make room for the new PSB.

During this cast out process, all of the DB PCBs in the PSB are scanned. Each PCB represents a database that the program can access. For each PCB, the SDBs associated with it are scanned. Each SDB represents a segment with the database that the program is sensitive to (SENSEG). Each SDB is chained off of a PSDB block associated with the DMB for the database, and must be removed from this chain. Thus, the chain of SDBs off of the SDB’s PSDB must be scanned to locate each SDB associated with the PSB being cast out.

The PSB cast out algorithm is:

```
do for each PCB in the PSB
  do for each SDB in the PCB
    locate the PSDB for the SDB
    scan the SDBs on the PSDB chain until SDB is found
    remove SDB from PSDB chain
  end
end
```

The number of instructions to perform the above SDB unhook processing can be calculated by the following formula:

$$130 + 9p + 114s + 8q$$

Where:

p = number of PCBs in a PSB
s = total number of SDBs pointed to by all PCBs in the PSB
q = total number of SDBs scanned on the PSDB queues to find the SDBs to be unhooked

This formula assumes latch trace is off, and there are no latch contentions.

For example, for one PSB in a customer dump, these values were: p = 301, s = 1879, q = 1255639. Thus, the above formula indicates:

130 + 9*301 + 114*1879 + 8*1255639 = 10,262,157 instructions

On a 100 MIP CP, this would take 102 milliseconds, assuming no latch contentions.

The cost to cast out a PSB increases as the number of PCBs and the number of SDBs increases. Further, every copy of a PSB (parallel scheduling) results in more SDBs being added to the PSDB chains for each segment type. As the number of PSBs increase in the pool, the PSDB chains get longer, and the time to cast out a PSB when this becomes necessary increases.

The cast out process occurs while holding several of the scheduling-related latches. Several of these latches are held exclusive, which means no other process that needs the latch can get it until the cast out process is completed. One of the latches that is held is the TCTB latch. There is one TCTB latch per transaction class - so dependent regions that are in the same class as the one doing the cast out are prevented from scheduling until the cast out process is complete.

While we do not know why these applications had such large PSBs with seemingly redundant PCBs, it may have been an attempt to minimize application changes as new functions were added to those applications over the years.

The above performance problem could happen in both non-shared queues and shared queues environments. However, with shared queues, the demand for PSBs in the pool can be greater than with non shared queues. This is due to the different scheduling algorithms that are used with shared queues. A constrained PSB pool in either CSA or DLI private could become much more acute. If cast out processing was occurring in small amounts before moving to shared queues, it might not be noticeable, even for large PSBs. Moving to shared queues could increase the cast out rate, increasing scheduling time and latch hold time, and thus increasing contention.

EXTRA SCHEDULING UNDER SHARED QUEUES

IMS Shared Queues manages the IMS message queues as a “pull” system. That is, messages are placed on the shared queue and all IMS systems are notified that a message is available. Any system which can process the message and has the capacity (an available region) will go “pull” the message off of the queue. This is in contrast to a “push” system where messages are directed to a specific system to be processed.

When new transactions arrive on empty transaction ready queues, potentially a number of IMSs and their dependent regions (subject to MAXRGN value specified for this transaction) could be alerted to process the new work. But if there is only one transaction message, only one IMS and one of its regions will successfully acquire and process this transaction. It is likely other regions invoked might end up in unsuccessful scheduling -- false schedulings (07 log records with 0 in DLRMCNT field). This “pulling” design starts work initiation early to provide the maximum processing capability during high volume processing. Shared queues is designed to run best when there are a large number of transactions arriving, and continuously being “pulled” and processed without work request re-invocation.

Compared to a non-SQ system, SQ does increase the number of false scheduling incidents, and thus generates the extra scheduling and the extra scheduling pool storage requirements.

ENHANCED LATCH AND SCHEDULING STATISTICS (PQ61259)

As the result of the above performance study, IMS V6 APAR PQ61259 was opened to fix and enhance certain IMS statistics and diagnostic characteristics.

Latch Manager Statistics:

During the analysis of the performance problem, a few logic errors were discovered in the way the current IMS latch manager statistics are collected. Certain time-related statistics (for example, latch hold time and latch wait time) were not being computed correctly in all cases. Most of the time, these numbers are reliable and correct. However, there are timing conditions that can occur within the latch manager that could cause these times to be inaccurate by a significant amount (for instance, a time value might be incremented by over 4000 seconds when it should have been sub-second). The occurrences of these errors would be rare, but the error size should they occur would be significant, and could result in incorrect conclusions when doing performance analysis. PQ61259 fixes these logic errors. Note that we do not believe that the latch statistics errors were involved in the latch numbers used during this study.

Additionally, a change was made as to *when* the latch manager collects statistics. The latch manager macro (DFSCLM) attempts to get and release a requested latch inline when possible. This provides the best possible performance, since getting a latch inline avoids the overhead of a call to a module to obtain the latch. A out-of-line call is made to DFSCLM00 (get) or DFSCLM10 (release) only when certain conditions occur:

- The latch is not immediately available (GET)
- Errors occur which require more complicated processing than available with the inline macro expansion (GET, REL)
- The latch trace is turned on (GET, REL)

Latch statistics are only collected by the latch processing modules; they are not collected for latches that are granted and released in the inline DFSCLM macro expansion. Additionally, prior to PQ61259, certain latch statistics were currently only collected when the DC monitor was active. This led to the following situation:

Latch Trace	DC Monitor	# Requests / Hold Time Stats
Off	Off	Not collected
Off	On	Collected, but inaccurate. Stats are only collected when the calls happen to go out of line. Calls for latches obtained inline are not counted.
On	Off	Not collected
On	On	Collected

Thus, the only time that the number of latch requests and the latch hold time statistics were valid was when both the latch trace and the DC monitor were turned on. Worse, when the latch trace was off and the monitor was on, statistics were collected, but they were not accurate (since they were only collected when the DFSCLM call was to be processed out of line).

APAR PQ61259 changes the latch manager statistics processing to no longer be dependent on the state of the DC monitor. It also changes the code to skip the collection of the “# of latch requests” statistic and the “latch hold time” statistic if the latch trace is not on. This prevents the condition noted in the second line of the previous table, where statistics are collected only for out of line calls. Note that latch wait counts and wait time statistics are always collected, regardless of the state of the latch trace.

PSB Pool Statistics:

Prior to APAR PQ61259, the statistics available about the PSB pools provided only summary data - information about pool size, current amount used, and the pool’s high water mark. No statistics were collected about the “extra” processing that occurs with the PSB pool -- particularly relating to PSB cast out processing.

PQ61259 adds several new statistics fields to the scheduling statistic log record (type X’4506’). These fields include counts and time statistics about various types of PSB pool processing. The following table summarizes the new fields in the 4506 log record pertaining to the “get PSB pool space” processing:

Field Name	Description
st4506_psbgRqsts	The number of requests to get PSB pool space (the number of calls to DFSDLMP0)
st4506_psbgNSpcCSA	The number of times there was not sufficient space in the CSA PSB pool to satisfy a request for PSB space (meaning one or more PSBs had to be cast out to make room).
st4506_psbgCoCSA	The number of PSBs that had to be cast out to make room for a new PSB due to insufficient space in the CSA PSB pool.
st4506_psbgFailCSA	The number of times a call to get PSB pool space failed to find any space at all in the CSA PSB pool - even after performing cast out processing.
st4506_psbgNSpcDLI	The number of times there was not sufficient space in the DLI PSB pool to satisfy a request for PSB space.
st4506_psbgCoDLI	The number of PSBs that had to be cast out to make room for a new PSB due to insufficient space in the DLI PSB pool.
st4506_psbgFailDLI	The number of times a call to get PSB pool space failed to find any space at all in the DLI PSB pool - even after performing cast out processing.
st4506_psbgNSpcBoth	The number of times there was not sufficient space in both the CSA and DLI pools to satisfy a request for PSB space (meaning cast out processing occurred for both pools).
st4506_psbgTimeNCO	The cumulative time spent getting PSB pool space for those calls that did not require cast out processing.
st4506_psbgTimeCO	The cumulative time spent getting PSB pool space for those calls that required cast out processing to find room.

Note that the following applies to all of the above statistics:

- All values are cumulative since the start of the IMS address space. To determine the value of one of these statistics for an interval between two checkpoints, you must subtract the value from the earlier checkpoint from the value from the later checkpoint.
- Count fields are 32-bit unsigned numbers (so they can increment from X’7FFFFFFF’ to X’80000000’).

- Time fields are 64-bit unsigned numbers in STCK units. This means that bit 51 is defined as being incremented once every microsecond. You can shift the values in the time fields right by 12 bits to obtain the time value in microseconds.
- The statistics fields are updated without any special serialization (i.e., no compare-and-swap). This means that if two dispatchable units try to update the same statistic field at exactly the same time, one of the updates might be lost. This is done to minimize the performance cost of collecting the statistics. These statistics should be viewed as general aggregate indicators, not as 100% accurate values.

Also note that PQ61259 added other similar fields to the x'4506' log record containing statistics about the "get PSB work pool" processing and the "release PSB pool" processing. Since the focus of this paper is the PSB pool get processing, these other statistics will not be discussed further here.

With the above statistics, it is now possible to analyze what is occurring within the PSB pool manager in terms of cast out processing and processing time. Prior to PQ61259, it was difficult to tell when there was a performance problem caused by PSB pool processing. Here are some interesting ways to combine the above statistics to generate some numbers useful for monitoring your PSB pool. Assume for all of the following examples that the values for the fields mentioned are really the delta between two successive checkpoints (since all fields are cumulative).

The number of calls to get PSB pool space that required one or more cast outs of any kind (CSA or DLI) is given by:

```
tNSpc = st4506_psbgNSpcCSA + st4506_psbgNSpcDLI - st4506_psbgNSpcBoth
```

Note the subtraction of the number of times both CSA and DLI had no space. This is required because in such a case *both* the CSA counter and the DLI counter would have been incremented - so the call would be double counted if the "both" statistic were not taken into account.

The number of calls to get PSB pool space that did *not* require a cast out is then:

`tSpc = st4506_psbgRqsts - tNspc` (from equation above)

The ratio of the # of calls that required a cast out and the total number of calls will tell you the percentage of calls to the PSB pool manager that require a cast out:

```
pctNspc = tNspc / st4506_psgbRqsts
```

For instance, if you had 200 calls to get PSB space, and 50 of them required one or more cast outs, then 25% of your calls to get a PSB forced another PSB out of the pool. However, that number by itself may not indicate a problem. You should also look at the rate of calls. For example, if those 200 calls occurred over a period of 20 minutes (1200 seconds), then you are only making a call to get a PSB once every six seconds, and you are doing cast out processing once every 24 seconds, on average.

The average time it takes to process a get PSB request when no cast out processing occurs is:

```
avgTimeNCO = (st4506_psbgTimeNCO / 4096) / tSpc
```

This gives the average time per get request in microseconds (dividing by 4096 can be done in assembler via a SRDL Rx,12 instruction). Similarly, the average time to get a PSB when cast out processing occurs is:

```
avgTimeCO = (st4506_psbgTimeCO / 4096) / tNSpc
```

If the average time to get a PSB with cast out processing becomes large, you may have a performance problem. But again, do not forget to consider the rate at which these events occur. If it takes 500 milliseconds to do a cast out, but it occurs one time in a minute, it’s not a problem. If it is occurring two times a second - it is probably having a detrimental effect!

The average number of PSBs that have to be cast out when cast out processing is necessary is given by:

```
avgPsbCoCSA = st4506_psbgCoCSA / st4506_psbgNspcCSA
```

and

```
avgPsbCoDLI = st4506_psbgCoDLI / st4506_psbgNspcDLI
```

For example, if there were 42 cast outs done because of no CSA space, and if there were 10 calls to get a PSB that had to do cast out processing for CSA, then on average, each of the 10 calls caused 4.2 PSBs to be cast out.

A sample report showing the new statistics is included at the end of this paper.

Latch Manager Trace Enhancements:

The latch manager trace is a diagnostic tool within IMS that keeps a record of the requests to get and release latches. Each trace entry is a 32 byte record, and contains information about the latch being gotten or freed, the latch mode (exclusive or shared), the type of latch, the latch caller, etc. One piece of information that was not present in the latch trace prior to PQ61259 is the time that the latch call was made. With PQ61259, a full 8-byte STCK value is now placed in the latch trace. This information could be useful when performing detailed analysis of performance behavior of IMS, because it now allows detailed time intervals to be determined for recent latch activity.

IMS Log Print Utility Enhancements:

As of IMS V6, all IMS log records have a suffix which contains a full 8 byte hardware STCK value of the time the log record was written. Very often, the IMS log is important in the analysis of performance problems. IMS writes log records at various points in the processing of a transaction. The interval between those records can indicate where in the processing delays are occurring.

Since each record has a hardware STCK in it, it is possible to know exactly when a record was written. PQ61259 enhances the IMS log print utility exit DFSERA30 to decode the STCK into human-readable time, and to print that time on the record ID line. The time is printed in UTC (GMT). Sample output from DFSERA30 looks like the following:

```
36 RECORD - 2002-04-04 00:21:05.851895 UTC
 00000000 000000   00A00000 36054504  04000000 00068248   C4E2E6C1 D3E3C140  08000018 00000000
 00000020 000020   C9D4E2F2 40404040  B76D1DC7 D574BD81   C9D4E2F2 40404040  B76D1DCA 489FFFC8
 00000040 000040   40000000 2002094F  00210585 1888032D   00000002 00000001  09B32E0D E3F3F2F7
 00000060 000060   F0C64040 C4E2E6C1  D3E3C140 40404040   40404040 00000007  00066AE8 03066A90
 00000080 000080   00000002 00000001  00000000 00000000   B76D1DCA 4DDF7E08  00000000 008D47A3

33 RECORD - 2002-04-04 00:21:05.852060 UTC
 00000000 000000   00440000 33010401  00068248 C9D4E2F2   40404040 B76D1DC7  D574BD81 C9D4E2F2
 00000020 000020   40404040 B76D1DCA  489FFFC8 40000000   08000018 B76D1DCA  4DE9CDC8 00000000
 00000040 000040   008D47A4

37 RECORD - 2002-04-04 00:21:05.854182 UTC
 00000000 000000   00740000 37010001  00710000 1F48C114   C9D4E2F2 40404040  00035ABA 00000000
 00000020 000020   B76D1DCA 4E6E2848  FFFF94B6 C9D4E2F2   40404040 B76D1DC7  D574BD81 C9D4E2F2
```

```
   00000040 000040   40404040 B76D1DC7  D574BD81 40000000   E3F3F2F7 F0C64040  08000000 00000003
   00000060 000060   00000000 B76D1DCA  4E6E6B88 00000000   008D47A5

  33 RECORD - 2002-04-04 00:21:05.854472 UTC
   00000000 000000   00480000 33419E01  1F48C114 C9D4E2F2   40404040 B76D1DC7  D574BD81 C9D4E2F2
   00000020 000020   40404040 B76D1DCA  488EC808 80000000   08000012 00800000  B76D1DCA 4E808B09
   00000040 000040   00000000 008D47A6
```

TUNING WITH LARGE PSBS

Ideally, all PSBs with a large number (say perhaps over one hundred) of TP and DB PCBs should be identified and reviewed for unused and duplicated PCBs. For overall IMS system performance, these PCBs should be removed from these large PSBs to minimize the impact of cast out processing as well as reduce the PSB pool storage requirements.

We recognize that removing PCBs from PSBs can require application changes, and that this may not be an acceptable short-term solution. One alternative is to leave the unused DB PCBs in the PSBs, but to remove all but one SENSEG statement for these PCBs. Doing this preserves the order of the PCBs in the PCB list, but still reduces the number of SDBs that need to be unchained during PSB cast out processing. Further, if a PSB has several unused PCBs for the same database, then each PCB should specify a *different* segment on its SENSEG. This helps reduce the physical SDB chain length off of the PSDBs, which thus reduces search time during the cast out process.

There may be cases where it is impossible to reduce the size of large PSBs further. When this is the case, the following suggestions can be used to reduce the effect of long cast out processing:

- Isolate the transactions using programs with large PSBs into their own scheduling class. This can help reduce the impact of long cast out processing on transactions in other classes.

- Consider using wait for input (WFI) or pseudo wait for input (PWFI) for the transactions using the large PSBs, to prevent or at least minimize these PSBs from being selected for cast out processing.

- Limit the number of parallel scheduled regions for the large PSB transactions to the minimum that you need to effectively process your transaction volumes, either through the MAXRGN parameter or by restricting the number of regions that can process the transactions' classes.

Another approach to improving performance with large PSBs is to increase your CSA and DLI PSB pool sizes to a point where all PSBs needed can fit into the pool with little or no cast out processing. Determining the size of the pools requires some analysis of your system to determine how much storage all of your PSBs together would require. One approach is as follows:

- Determine the working set of transactions for your system. By "working set", we mean the set of transactions that are actually used in your IMS system (you may have many more transactions defined than are routinely used - those that are unused or are very rarely used can be excluded from your working set).

- For each transaction, compute MAXRGN times PSB size for both the CSA and DLI portions of the PSB. You can obtain the PSB sizes from the output of the ACBGEN.

- Add up all of the (MAXRGN * CSA PSB size) calculations for all of your transactions. Do the same for all of the (MAXRGN * DLI PSB size) calculations. The two sums should be the theoretical maximum size needed for the PSB pools to contain all of your working set PSBs. You can add an extra 5 to 10% as a buffer to allow for changes in workload and/or rarely-used transactions.

- Run with the calculated sizes and monitor the PSB pool usage over a period of time. You may well find that the PSB pools are bigger than needed, and that neither the CSA nor the DLI pool ever reaches 100% used. Keep track of the high water marks for these pools. You could then reduce the pool sizes once you are comfortable with the stability of your PSB pool high usage numbers or just leave them larger than necessary to allow for future changes in workload.

As always, workload changes over time can change the demand for resources such as pool space. Thus, it is prudent to continue monitoring the PSB pool usage going forward, and to make adjustments as your system's workload changes. You should also monitor the new PSB pool statistics added with APAR PQ61259 to watch for excessive cast out processing.

The following tuning suggestions related to PSBs stated in the *IMS V7 Performance Monitoring and Tuning Update* may also be helpful:

- Make intent lists resident. Specify RES=Y in DFSPBxxx (this is the default).
- Make highly executed PSBs resident. Also make resident the PSBs for WFI transactions and for long running or frequently executed BMPs. Also consider making resident unusually large PSBs.

SAMPLE SCHEDULING STATISTICS REPORTS

At the time of the writing of this paper, none of the new PSB pool scheduling statistics are reported by any of the IMS performance monitoring tools, such as IMS Performance Analyzer. The reports shown below were produced by an internal tool written to help unit test and validate the new statistics. The intent of showing these statistics are to illustrate how they help externalize the PSB pool cast out situation.

The first report shows the new PSB pool statistics during an interval where no cast outs occurred in the PSB pool. This is the desired “normal” operating mode. Note that all times shown are in seconds.

```
IMS Scheduling Statistics Record Report

Start time (UTC):  2002-06-07 16:08:42.586772
End time (UTC):    2002-06-07 16:16:07.863234

Interval (secs):       445.276

# Active BMPs:               0            0
# Active MPPs:               0            0

                                  Count   Per Second  Per Schedule
                                  -----   ----------  ------------
Number of Schedules:               8369       18.795
Number PSBs Examined:              8369       18.795         1.000

Not Sched: Pgm Conflicts:             0        0.000         0.000
Not Sched: Intent Conflicts           0        0.000         0.000
Not Sched: Misc Reasons:              0        0.000         0.000

                                  Count   Per Second      Per Rqst  Per Not Avail
                                  -----   ----------      --------  -------------
PSB Pool Get Requests:               41        0.092

CSA Not Avail:                        0        0.000         0.000
CSA Cast Outs:                        0        0.000         0.000
CSA Failures:                         0        0.000         0.000

DLI Not Avail:                        0        0.000         0.000
DLI Cast Outs:                        0        0.000         0.000
DLI Failures:                         0        0.000         0.000

Both Not Avail:                       0        0.000         0.000

PSB Pool Release Requests:         8369       18.795

PSBW Pool Get Requests:            8369       18.795
PSBW Failures:                        0        0.000         0.000

                                  Total  Avg per Req   Avg Per C/O
                                  -----  -----------   -----------
PSB Get Time (no C/O):            0.002     0.000051
PSB Get Time (cast outs):         0.000

PSB Release Time:                 0.029     0.000003

PSBW Get Time:                    0.019     0.000002
```

The next report shows the new PSB pool statistics during an interval where severe cast out processing delays were occurring. In this instance, there were 47 calls to get PSB pool space where the space was not immediately available. However, these 47 calls required a total of 2014 PSBs to be cast out in order to find room for the new PSBs - or and average of 42.8 cast outs for each request that required one or more cast outs. Each cast out took 92 ms on average. Each request that required cast out processing took 3.9 seconds on average to complete.

```
IMS Scheduling Statistics Record Report

Start time (UTC):  2002-06-07 16:16:18.863858
End time (UTC):    2002-06-07 16:24:39.666294

Interval (secs):       500.802

# Active BMPs:               0            0
# Active MPPs:               1          127

                                  Count   Per Second  Per Schedule
                                  -----   ----------  ------------
Number of Schedules:              28924       57.755
Number PSBs Examined:             28925       57.757       1.000

Not Sched: Pgm Conflicts:             0        0.000       0.000
Not Sched: Intent Conflicts           0        0.000       0.000
Not Sched: Misc Reasons:              0        0.000       0.000

                                  Count   Per Second      Per Rqst  Per Not Avail
                                  -----   ----------      --------  -------------
PSB Pool Get Requests:             1655        3.304

CSA Not Avail:                       47        0.093       0.028
CSA Cast Outs:                     2014        4.021       1.216         42.851
CSA Failures:                         0        0.000       0.000          0.000

DLI Not Avail:                        0        0.000       0.000
DLI Cast Outs:                        0        0.000       0.000
DLI Failures:                         0        0.000       0.000

Both Not Avail:                       0        0.000       0.000

PSB Pool Release Requests:        28836       57.579

PSBW Pool Get Requests:           28924       57.755
PSBW Failures:                        0        0.000       0.000

                                  Total  Avg per Req   Avg Per C/O
                                  -----  -----------   -----------
PSB Get Time (no C/O):            0.196     0.000122
PSB Get Time (cast outs):       186.009     3.957644     0.092358

PSB Release Time:                 5.781     0.000200


PSBW Get Time:                    0.179     0.000006
```

ACKNOWLEDGEMENTS

The authors wish to thank the following members of the performance study team for their contribution to the results documented in this report:

- Gianmario Bescape', IBM ITS Italy
- Luigi Cetorelli, IBM ITS Italy
- Dario D’Angelo, IMS Service and Support
- Hiram Neal, IMS Performance
- Dave Viguers, IMS Performance