Cheryl Watson's Sept./October 1992

TUNING Letter

A Practical Journal of MVS Tuning and Measurement Advice

DOUBLE ISSUE!

This is a special reprint of our September/October, 1992 issue. In 1993 we'll be publishing a double-sized issue every other month, so this particular issue is more representative of the longer, fact-filled issues you'll be seeing then.

Over thirteen hundred firms, in over fifty countries, now take the ***TUNING Letter.*** The reader comments on the back page (and the 90% renewal rate) reflect their enthusiasm. They use it to increase performance, to reduce costs, and to provide valuable internal education and documentation.

All of our back issues are still available, and we highly recommend buying the ones that are of interest to your installation. Becasue we don't want to reprint the basics of a subject that may have taken many pages to explain, we instead make reference to the past month and page when that past material would be helpful. The back issues are always kept current by the UPDATE sheets we send from time to time. They contain corrections or additional material and are to be stored with the original.

Please read and use this issue. Make as many copies as your location needs (our policy with all issues). We hope you'll agree that this is a valuable and very practical newsletter which should pay for itself many times over.

This issue is a continuation of the August 92 issue covering Application Tuning from the viewpoint of reducing I/O. If you have it available, please read the section on application tuning in the August issue, especially "Getting Started" and "MVS Measurements: Batch Jobs." The issue included techniques for tuning batch applications, design considerations, measurements of batch applications, use of sort exits, and methods to reduce I/O (including blocksizes, buffers, tape mounts, random versus sequential, fixed vs variable, improving VSAM, and use of VIO).

Most of my recommendations apply to CICS applications as well as batch applications. Next month's issue will be devoted to tuning CICS from an MVS point of

IN THIS ISSUE

view and will of course point out which of the techniques mentioned in the August issue and this issue also apply to CICS.

If you're interested in reducing I/O by using data-in-memory (DIM) applications in MVS/ESA, you'll probably be interested in IBM manual "GG24-3698 - MVS/ESA and Data in Memory Performance Studies". It's a 300+ page "redbook" on benchmarks and tests run by IBM using a variety of DIM applications.

FOCUS: APPLICATION TUNING

INTRODUCTION

If you haven't read the August 92 issue, you should go back and read the GETTING STARTED section on page 2 and the MVS MEASUREMENTS: BATCH JOBS on page 16. They provide a method for approaching tuning of applications and techniques for measuring them.

The measurement section should be reviewed because it's the basis for where you want to start tuning. If a job is primarily CPU-bound, then tune the CPU; if it's I/O-bound, then tune the I/O.

Howard Glastetter from the State of Washington reminded me that I should have mentioned Top Twenty programs, as well as Top Twenty jobs. In last month's issue I suggested that you summarize data by jobs, then sort in descending resource usage. You should also perform the same step, but summarize on program instead of job. That is, find all the step termination records (type 30, subtype 4), sort by program name, then accumulate resources (CPU time, EXCPs, etc.) by program name. Now sort in descending sequence by resource (CPU or EXCPs) and print the top twenty (or forty).

You may find frequently used programs, such as sorts or IEBGENERs, that don't show up individually, but do show up as large resource users when you combine multiple executions. These are certainly programs that you want to spend some time tuning. Often, you'll find these are vendor programs and you can contact the vendor for techniques in tuning their product.

REDUCING CPU

If CPU usage is a major component of the elapsed time (or large enough to warrant the effort), then you can consider some of the following recommendations to reduce the CPU. In almost all cases, reduction of the CPU time will reduce the charge back for a job and the elapsed time of a job.

To find the jobs and programs consuming the most CPU time, accumulate all seven CPU measures from the SMF type 30 records for jobs or programs: TCB time, SRB time, initiator TCB, initiator SRB, hiperspace time, region control task time, and I/O interrupt time. Then produce a "top twenty" list of the largest CPU users and concentrate on them to reduce total CPU time.

REDUCE THE I/O

Last month's issue and several sections of this issue deal with reducing the number of I/Os issued by a job. Reducing physical I/Os will almost always reduce the CPU time of a job. The only exception would be the replacement of I/Os by use of some of the ESA features, such as hiperbatch or Batch LSR (or any other facility that uses hiperspaces instead of traditional I/Os).

USE A PROGRAM ANALYZER

A program analyzer is the easiest and best way to determine where the inefficiencies of a program are. They can show where the majority of the CPU time is spent. Often it's concentrated in just a few lines of code. The analyzers typically show you what address locations are responsible for the most CPU time and where the program is being delayed most (e.g. a dataset read). You can then work to reduce that area of delay.

While you might consider using these products after an application is in production, I think their most valuable use is during initial coding and testing. If you can find some poor-performing areas, this is the ideal time to modify the program! You can find inefficient data storage (constant conversion between packed and binary fields), poor table handling techniques (old index techniques), inefficient loops, etc. through these analyzers. Clean up the code before it goes into production and it will save you a lot of grief!

During production turnover reviews, you could require that application developers describe the sections of code that account for most of the CPU time and what steps were taken to reduce that CPU usage. If this re-

quirement is generally enforced then developers will learn to use application tuning tools on a regular basis.

A commonly used product is STROBE from Programart. See the Product Highlight later in this issue. Several online MVS monitors, such as TMON/MVS from Landmark Corporation and Omegamon from Candle Corporation, also provide a similar analysis. Whichever product you choose, make sure the application programmers find it easy to use. Their use is the key to a successful implementation. See the PRODUCT HIGHLIGHT on STROBE for samples of reports available from this type of monitor.

I've been able to reduce CPU time by 30% or more by identifying poorly coded routines and replacing them with efficiently coded routines.

USE EFFICIENT CODING TECHNIQUES

The following coding techniques apply specifically to VS COBOL II, but many of the recommendations apply to earlier versions of COBOL and other languages as well. Be sure to check the Application Programming Guide when you go to a new release (VS COBOL II manual for Releases 3.0 to 3.2 is SC26-4045). This manual contains several recommendations for improving the performance of your COBOL programs.

☐ REDUCE CONVERSION OF DATA AREAS

Minimize the number of times fields are converted from one format to another and minimize the comparing of fields of two different formats. During the design stage decide how fields will be stored and keep their handling consistent. In general, calculations are more efficient when done in binary fields but printing numeric fields is best when the field is packed. If a field will be used in calculations and also reported, define it as a packed field. If it will only be used in calculations and never printed, define it as a binary field. When defining binary fields in COBOL, define them with a sign and an odd number of digits (e.g. PIC S9(5) COMP-4). Keep binary fields to less than 15 digits or a routine will need to be called to manipulate them. Packed fields (COMP-3) are also more efficient if less than 15 digits.

☐ AVOID DESTRUCTIVE MOVES

Destructive moves are those where a field is initialized by initializing the first byte to a blank or zero and then moving the field to itself. In COBOL, this might be:

```
01 FULL-FIELD.
   02 CLEAR-IT  PIC X.
   02 REST-OF-FIELD PIC X(99).
...
    MOVE 'Z' TO CLEAR-IT.
    MOVE FULL-FIELD TO REST-OF-FIELD.
```

This type of move is very, very inefficient (and yet it can be found in many programs). A better method is to define a constant field the same size as the original field and move the entire field at once. If a field will never be modified, initialize it with a VALUE statement and you can avoid clearing it.

☐ REDUCE INDEXING REFERENCES

Referencing indexed or subscripted fields takes more CPU than referencing non-indexed fields. Reduce the indexed references whenever possible. Look at the following example:

```
01 STATE-TABLE.
  02 STATE-ENTRY OCCURS 50 TIMES.
    04 STATE-CODE PIC XX.
    04 STATE-NAME PIC X(25).
    04 STATE-POPULATION PIC 9(5)
        ZONED-DECIMAL.
    04 STATE-FLOWER PIC X(20).
    04 STATE-BIRD   PIC X(20).
    04 STATE-GOVERNOR  PIC X(30).
01 ST-HOLD.
  02 ST-CODE PIC XX.
  02 ST-NAME PIC X(25).
  02 etc.

PERFORM SEARCHIT VARYING ST-NDX
    FROM 1 TO 50.
IF FOUNDIT
    MOVE STATE-NAME (ST-NDX) TO
        ST-NAME.
    MOVE STATE-POPULATION (ST-NDX) TO ..
    ... .
```

Better:

```
PERFORM SEARCHIT VARYING ST-NDX
    FROM 1 TO 50.
IF FOUNDIT
    MOVE STATE-ENTRY (ST-NDX) TO
        ST-HOLD.
    MOVE ST-NAME TO ...
    MOVE ST-POPULATION TO ...
```

While this is a simplification of the code, I hope you can see the intent. The second example only used the subscript once to move the entry out of the table,

and subsequent references then didn't require a subscript. If you're going to make multiple references using indexes or subscripts, it may take less CPU time to move the entire entry out of the indexed table area first.

☐ USE EFFICIENT INDEXING TECHNIQUES

Table lookups often account for a large portion of CPU time in a program. Even if you don't have a program analyzer, you should try to evaluate the processing for each table in a program. You'll often find very inefficient techniques that can be improved.

You should always use indexes instead of subscripts when possible. Indexes are associated with a table and therefore are relative to the table location. Indexes take less CPU time than subscripts. A table could be specified in one of the following two ways:

```
01 TABLE1.
   05 TABLE-SEC OCCURS 100 TIMES
        INDEXED BY ID1.
      10 FLD1   PIC X.
      10 FLD2   PIC X(10).
      10 etc.
```

Worse:

```
01 TABLE1.
   05 ID1   PIC S9(3) COMP-4.
   05 TABLE-SEC OCCURS 100 TIMES.
      10 FLD1   PIC X.
      10 FLD2   PIC X(10).
      10 etc.

PERFORM SEARCH-IT VARYING ID1 FROM 1
    BY 1 UNTIL END-IT.
```

In these examples, the first table would produce more efficient code because ID1 is defined as an index rather than a subscript. Indexing also allows a more efficient use of the SEARCH verb.

When you design a system, consider which fields will be used for subscripts into tables. The best format for subscripts is binary, signed fields. A binary field with 9 digits is good, but less than 4 is even better. When using a table with OCCURS DEPENDING ON logic, ensure that the object of the OCCURS DEPENDING ON is a binary, signed, odd-digit field (e.g. PIC S999 COMP-4). Continual reference to variable-length fields is expensive, so move them to a fixed length area as soon as possible in the processing.

Whenever possible, use direct indexing instead of table lookups. For example, in a payroll system that must do a lot of state table lookups, assign a number to each state, keep the state number in binary and use it to go directly to the corresponding state entry instead of keeping the two-character state code and continually doing state-lookups.

☐ USE EFFICIENT FILE TECHNIQUES

When accessing variable-length blocked sequential files (QSAM or SAM) for output, use the APPLY WRITE-ONLY clause in the File Definition. When this phrase is specified, the buffer is truncated and written only when the space available in the buffer is smaller than the size of the next record. If not specified, the buffer is truncated when the space left is less than the maximum record size (which causes shorter blocks, more I/Os and more disk or tape space). Since use of the clause reduces calls to data management services, it also reduces CPU time. See the AWO compile option later.

Create alternate VSAM indexes with IDCAMS instead of using the AIXBLD run-time option.

CONSIDER OPTIONS

Often, there are multiple methods to accomplish the same result. Try both techniques and evaluate which techniques produces the most efficient code. As an example, some COBOL programs use the DISPLAY verb to produce a summary report while others use the WRITE verb. The WRITE is more efficient because you can use QSAM buffering to improve the performance of the file (DISPLAY uses unbuffered QSAM).

CONSIDER DYNAMIC CALLS vs RESIDENT

Dynamic linkage to a program (loading it in during execution) provides a great deal of flexibility for testing and production. These routines can be easily changed, recompiled, and then automatically used the next time they're referenced. Dynamically linked programs are loaded the first time they're called, however, and take CPU time during the load. If you link to a dynamic routine multiple times during the execution, the logic should be changed to include it in a composite linkedit. The "load" is then done once during linkedit time instead of daily during each execution. Moving the load to linkedit time rather than execution is a very minor CPU reduction if the call occurs once in the program, but can be significant if there are many calls and the program is reloaded each time.

Dynamic calls improve ease of maintenance but increase CPU time during execution. One exception to this is a routine that is seldom used. For example, you might produce a report only when errors occur (perhaps one day out of twenty). In this case, it might make more sense to keep the error report program dynamically called. If it were linked together with the other routines, it would increase the size of the load module of the main program while providing no benefit (except once every twenty times).

Does this mean you should ALWAYS avoid dynamic calls. No way! Use them to improve maintainability, but if you need to reduce CPU time during execution, it's one of the simple methods to do it.

ONLY DO A JOB ONCE

During the design phase of an application, determine where certain steps, such as data validation, will occur and then only perform the function once. In all other programs, assume that the function has been completed before. I once looked at a payroll system that had six different programs validating the state code by doing a table lookup. This should have been done in only the first program, not in all succeeding programs!

The same type of logic applies for validation of numeric data. I've seen programs that might be 10th or 11th into an application, executing code that looks like: "IF FLDA NUMERIC THEN ADD FLDS ...". This code is often introduced not in the design phase, but during the testing phase when the programmer encounters some invalid data. In order to avoid abending, the programmer adds these validation steps during testing (and never removes them!).

Place validation processing at the first time a field enters the application and then assume the field is correct from that point on.

CONSIDER TEST vs PRODUCTION TECHNIQUES

In all languages, it's best to remove testing logic or facilities prior to production. In most every case, these testing techniques take more CPU time during execution and take up more space in the program. For example, the COBOL "READY TRACE", "EXHIBIT", and "DISPLAY" verbs and use of the TEST compile option occupy space in the program (thus increasing the working set size) and take more CPU time in order to bypass them during a production run. It's much better to remove them completely before going into production.

Optimization should always be used before going into production, but should seldom be used in test. Optimizing a program takes more CPU time during the compilation, but produces more efficient code. For example, in COBOL II, you should use a parameter of 'TEST' during the testing phase (this allows use of the TEST facility), but you should use 'OPTIMIZE' during the final stages of testing and production. OPTIMIZE normally produces more efficient code (such as putting PERFORMed routines inline).

Remove test files instead of dummying them before going into production. For example, several programmers may create test files or reports during the testing phase, then turn them off by "dummying" the files before production (changing the DD statement to "DD DUMMY"). This just wastes CPU time. Remove all code that references the files before going into production.

There's a fine line in determining when you should change to production mode. Most sites don't want to recompile just before production without testing, so you want to change the parameters just before the "final" test. Knowing when the "final" test is, is a little tricky! You should consider a pre-production run where performance and service level testing of the code and files can be performed before production acceptance.

DESIGN FOR SORTING

During the design phase of an application, consider the possible sort uses. For example, a contiguous sort key is the most efficient. If the record layout provides the most common key fields in a contiguous set of fields, the sorts will be more efficient and take less CPU time. Also, design the fields so a minimal size field can be used for a sort key. In our state example, for instance, if you carry both the state name and the state code, sort on the state code since it's shorter.

Also, avoid sorting on similar fields. I've seen several programs where multiple sort keys are used where one will do. For example, in a payroll application, sort on social-security number or name, but not both. (You're probably thinking that I make these things up, but I'm not. The worst case I saw had a file sorted on seven fields (six of which were imbedded in the seventh!) It's really not unusual to find these instances, so look for them and eliminate them.

CHECK WITH VENDOR

If you don't have the source of the code, ask the vendor for ways to reduce CPU time in the product. They often have recommendations for sites that need to reduce the CPU time.

USE BEST COMPILE OPTIONS

Consider the following options for compilation time. You might put some in the standard COBOL proc or simply encourage their use. Compile options can be added in the JCL:

```
//STEP1  EXEC PGM=IGYCRCTL,
//           PARM='LIST,OBJECT,otherparms'
```

Compile Options:

AWO This forces APPLY WRITE-ONLY for all applicable files. See the prior discussion on APPLY WRITE-ONLY. This should always be included since the default is NOAWO.

DYNAM/NODYNAM See the previous discussion on dynamic calls. DYNAM takes less time during compilation, is easier for maintenance, and eliminates multiple composite linkedits. It also takes slightly more CPU time during program execution. For the majority of jobs, use DYNAM, but if you need to reduce the CPU time on a job that calls many routines, then use NODYNAM. Default is NODYNAM.

FASTSRT Indicates that COBOL should allow the sort utility to perform all I/Os. NOFASTSRT (the default) will cause the sort to call COBOL for every record.

NUMPROC(PFD) This option bypasses checking of a valid sign for COMP-3 and DISPLAY signed numeric data and will therefore take less CPU time. If the data is coming from an external source, you may want to use NUMPROC(NOPFD) or NUMPROC(MIG) which will add extra code to validate the sign. NUMPROC(NOPFD) is the default.

OPTIMIZE This will produce more efficient code for execution, but will take more CPU time and elapsed time for the compilation. Use OPTIMIZE before going to production, but NOOPTIMIZE for initial testing. See the earlier discussion on OPTIMIZE. NOOPTIMIZE is the default.

RENT This option creates a reentrant program that can be loaded above the 16Mb line. This takes slightly more CPU during execution to ensure that the program is reentrant, but saves virtual storage space below the line. That space can then be used for other features, such as buffers that can reduce CPU time. RENT forces RESIDENT (see below). NORENT is the default.

RESIDENT This option will actually take more CPU time during execution since the COBOL subroutines need to be loaded at during the job's execution. Use of NORESIDENT, however, eliminates the possibility of running on CICS, running above the 16Mb line, dynamic calls, using the TEST compiler option, and sharing the library. NORESIDENT also requires all programs to be linkedited if there are any updates to COBOL subroutines. You may consider use of NORESIDENT for a CPU-bound job that must be reduced. Just be aware of the restrictions. NORESIDENT is the default.

NOTEST TEST allows use of the debug tool, but generates a lot of additional code. Therefore, use NOTEST unless the programmer expects to use the debug tool. NOTEST is the default.

TRUNC(OPT) This option eliminates the addition of extra code to truncate receiving fields of arithmetic operations. TRUNC(BIN) and TRUNC(STD) generate code to truncate fields to correspond to the PICTURE clause. Refer to the Application Programming Guide for a fuller discussion of this option.

In summary, you might consider using the following options during initial testing:

```
LIST,OBJECT,AWO,DYNAM,FASTSRT,RESIDENT,TEST
```

Then use the following for production:

```
LIST,OBJECT,AWO,FASTSRT,OPTIMIZE,RENT
```

If you REALLY need to reduce CPU, then override with the following parameters (the NODYNAM may require different linkedit procedures and NUMPROC & TRUNC need to be thoroughly retested!):

```
NODYNAM,NUMPROC(PFD),NORESIDENT,TRUNC(OPT)
```

CHARGE BACK ISSUES

Any reduction of CPU time will likely change your charges for the execution. Simply be aware of this fact. If these are real dollars, your tuning could reduce your revenue.

REDUCING VIRTUAL STORAGE

If you need to reduce the amount of virtual storage used by a program (either because the region size is a limiting factor or to reduce the working set size), consider the following options.

USE DYNAMIC ROUTINES

Dynamically called routines are only brought in when a program needs them. Infrequently used routines are good candidates for dynamically called routines. They will, however, take more CPU time for load during execution. But if several routines are used at different times during processing, you can reduce the composite module size and overlap them by loading one, deleting it, and loading the next one. Freeing up region size could also allow you to provide more buffers (as discussed in the August issue) and improve the I/O processing.

USE LPA

If any of the programs are reentrant, you can consider the use of the link pack area, LPA. LPA modules are loaded into virtual storage at IPL and reside there for the duration of the IPL. Any pages that aren't referenced reside on the PLPA page data set. You should only put high used modules here and they should be linked to reside above the 16Mb line if at all possible.

The benefits are outstanding! Not only do you save the I/O and CPU time to load the module into storage, but you reduce the virtual storage of the job. The job simply references the LPA version of the program and doesn't need to load the module into the job's address space. So you end up with a smaller virtual storage requirement which produces a smaller working set and probably less paging.

This is applicable to many modules that are used by several jobs, such as SAS modules, any TSO ISPF modules, TSO application modules, reentrant sort routines, and many others.

REDUCE THE I/O STORAGE

Several of the techniques mentioned in the August issue to improve I/O processing take more virtual storage. These include use of larger blocksizes and buffers. If you need to reduce the virtual storage requirements, consider reducing the blocksize and buffers. Only do this for small, low-activity files, however or you'll end up increasing the CPU time and elapsed time in order to free up some virtual storage.

REMOVE TESTING FACILITIES

Most of the testing facilities mentioned in the REDUCING CPU section take virtual storage. If you eliminate them, you'll also reduce the working set size of the job.

BENEFITS & COSTS

Reduction of the working set size itself may have no benefit on a job other than the reduction of elapsed time. This occurs because the program might use fewer paging operations to perform the same amount of work. If virtual storage is reduced by eliminating unused features, such as test facilities, you'll probably also see a reduction in CPU time. If it's reduced by eliminating I/O enhancements such as large blocksizes or buffers or by loading programs dynamically, the CPU time may increase and the elapsed time might increase. You'll have to track all of the measures to determine the effect.

The primary benefit of reduced elapsed time is a higher probability of meeting service level objectives for batch jobs. It may be worth increased CPU and I/O costs in order to meet service level objectives.

CHARGE BACK ISSUES

As indicated in the prior paragraphs, reducing the virtual storage may result in a change of CPU time, EXCPs, and elapsed time. It's not always predictable. Look at the sections for reducing CPU and I/O for the charge back considerations. Since charge back may be based on meeting service level objectives, you stand the chance of hitting the objectives more frequently when you've reduced elapsed time.

From Dave Barry, my favorite columnist:

"BUG - a cute little humorous term used to explain why the computer had your shipping department send 150 highly sophisticated jet-fighter servo motors, worth over $26,000 apiece, to fishermen in the Ryuku Islands, who are using them as anchors."

REDUCING ELAPSED TIME

In order to understand how to reduce the elapsed time of a job, you need to know where most of the time is spent. Look at the August 92 issue on page 20 for a technique to determine a breakout of elapsed time from SMF data. You can also use an online monitor to determine where the majority of the time is spent. Then spend your effort working on the major reason for elapsed time. For example, if only 5% of the elapsed time is execution and 70% is I/O, spend your time on reducing the I/O component of elapsed time.

REDUCE I/O, CPU, & PAGING

Any reduction in the number of I/Os, the length of time to perform an I/O, the amount of CPU time consumed, or the amount of paging, will reduce the elapsed time. While this seems obvious, many people forget to work on the basics before stepping into more complex areas. Simply determine where the majority of time is spent and then concentrate on that area.

REDUCE SWAPPING

There are two types of swaps - those caused by the program and those caused by SRM attempting to maintain the balance of the resources (unilateral or exchange swaps). If a job has been swapped out, you can determine the type of swap by analyzing the SMF data (see the August 92 issue on page 20). "Swapped out & ready" refers to swaps controlled by SRM. To reduce these swaps, you must analyze the SRM parameters. This is really a system programmer's function and not an application tuning function (so I won't address it here).

If you want to see what type of swap or how many swaps a job is experiencing, you can use an online monitor. For example, Figure 1 shows an RMF Monitor II ASD report. If you display swapped out jobs, such as using ASD A,A,A in Monitor II, then you can see the reason for the last swap, R LS (DW for detected waits and LW for long waits), and the number of swaps, TX SC (the transaction swap count). If there are many swaps, then you know that the job has been delayed at least that many times.

The swaps caused by a program usually fall into either long waits or detected waits. Long waits occur when the program issues a wait with TYPE=LONG or issues a STIMER with a wait of over 1/2 second. Long waits due to TYPE=LONG are most often issued by programs that expect to wait for some response, either from an operator or from another program. There are usually very few application waits for this reason. STIMERs with waits over 1/2 second may occur in a vendor program that is collecting measurement data over a long period of time and collects or samples the data very frequently, such as every second or twice a second. Any monitor or collector that obtains data this frequently should be marked as non-swappable in the PPT.

A detected wait occurs when a program goes into a wait and is still waiting after 2 seconds (it's actually calculated as 106 SRM seconds divided by the number of SRM seconds per second and comparing to the minimum of 2 seconds). Detected waits are often produced by enqueue conflicts, shared DASD RESERVEs, or programs that forgot to issue TYPE=LONG waits. It may take some work to determine the reason for the detected waits, but you can reduce the swaps if you can find the reason. The enqueue conflicts and shared DASD problems can normally be identified by looking at enqueue delays and device activity reports. See the MVS MEASUREMENT section on interpreting enqueue reports.

Another indication of enqueue delay can be found in the RMF Monitor I Swap Placement Activity report when ENQUEUE EXCHANGE swaps occur. An enqueue exchange swap occurs if a swapped out and ready user is

```
F                              MIG=319 CPU= 97 UIC= 38 PFR=  63          ASD      T

09:03:41          P P C  R  DP   CS  ESF   CS   TAR   X   PIN   ES   TX  SWAP  WSM
JOBNAME  DMN      G P L  LS PR    F       TAR   WSS   M    RT   RT   SC   RV   RV

*MASTER*   0      0 1 NS     FF   50   23          0        0.0  0.0   0    0
PCAUTH    10     10 1 NS     7A   34    0    0     0   X    0.0   15   0  150
RASP      10     10 1 NS     7A   34    0    0     0   X    0.0   15   0    0
TRACE     10     10 1 NS     7A  125    0    0     0   X    0.0   15   0    0
GRS        0      0 1 NS     FF 1381   14 16.7K 32.8K   X    0.0   15   0  150
DUMPSRV    0      0 1 NS     7A  125    0    0     0        0.0   15   0  150
CONSOLE   10     10 1 NS     FF   47    2          0   X    0.0   15   1  150
ALLOCAS   10     10 1 NS     7A  109   91          0   X    0.0   15   1    0
LLA       10     10 1 NS     7A  211  125    0     0   X    0.0   20   0    0
BATJOB1    3     14 1 WL LW  52   58   34                    0.0    5 125    0
```

Figure 1 - RMF Monitor II ASD Display (ESA)

holding a resource that a swapped in user wants. When that condition occurs, SRM will swap in the out and ready user and will swap out another address space (not necessarily the one asking for the resource). Enqueue delays can still occur without seeing any of these swaps, because the swaps will only occur in a constrained system where SRM has some jobs swapped out and ready.

In either the case of long waits or detected waits, if you cannot reduce the number of swaps and they are excessive or causing a delay in the application, you can reduce them by making the address space non-swappable. This is done by specifying the program name in the SCHEDxx parmlib member and indicating "NOSWAP". The minimal storage used for this should easily be paid for in reduced CPU time. If storage is a concern, you can also storage isolate or storage restrict it.

REDUCE ENQUEUES

Enqueue delays don't necessarily cause a detected swap or an enqueue exchange swap. In fact, if the enqueue is resolved within two seconds, SRM won't swap it as a detected wait, and there will be no indication that a conflict existed. There's a possibility that an enqueue conflict will identify the jobs that hold the resource and are waiting on the resource by looking at the RMF Monitor I Enqueue Report. See the later section on MVS MEASUREMENTS for analysis of enqueues.

REDUCE TAPE MOUNTS

This was covered in last month's issue, but is a major factor in elapsed time and so should be included in this discussion. Tape mount delay can often amount to over 90% of a job's elapsed time. Watch this carefully. As mentioned last month, you can reduce tape mount delay by:

Moving the dataset to DASD
Avoid use of UNIT=AFF=ddname
Use UNIT=(TAPE,2) if multi-volume
Increasing the blocksize to reduce the volumes.

REDUCE OPERATOR REPLIES

Programs that require operator replies may see very long elapsed times if the operators are busy with other priorities. Avoid these at all costs. They are especially damaging to sites that hope to go to a "lights out" operation. Find some way to automate the process without requiring operator intervention. This process must be accomplished before automated operations can be fully implemented - so you might as well start now!

BENEFITS

The primary benefit of reduced elapsed time is a higher probability of meeting service level objectives for batch jobs. A secondary benefit in reducing the elapsed time is the possibility to run more jobs during the same period of time (increasing the throughput rate). Reduction of swaps will reduce total I/O and CPU time on the system, normally seen in uncaptured time or master scheduler CPU times and not the actual jobs.

COSTS

There are no identifiable costs in the items mentioned above.

CHARGE BACK ISSUES

Since most of the swap time is not reported in the address space records, there is little effect on charge back. However, since charge back may be based on meeting service level objectives, you stand the chance of hitting the objectives more frequently when you've reduced elapsed time.

If you're running MVS/ESA and do charge back on RCT (region control task) time and SRB time, you may see a reduction in costs because there is some swap time accounted for in both of these fields.

REFERENCES

See the May 91 issue (page 3 for swapping in general and page 19 for unnecessary swaps). Review the October, November & December 91 issues on SRM swapping. See the MVS MEASUREMENT section in this issue for enqueue analysis.

USING BATCH LSR

Parts of the following section were first included in the September 91 issue on page 11, but are included here with updates and additional detail. I'm including them again because batch LSR undoubtedly provides some of the most impressive savings of any of the ESA facilities. Users of non-IBM products, such as VSAM I/O PLUS from SoftWorks (see the PRODUCT HIGHLIGHT), have been enjoying these types of benefits for the last several years.

BLSR (Batch Local Shared Resources) is one of the best things to come along in quite a while. It provides buffer lookaside facility for batch jobs accessing VSAM files (KSDS or RRDS) and is available in ESA SP 3 with APAR OY24097 and is standard in ESA SP 4. Prior to

IBM's implementation, there were several products from other vendors that provided the same facility. These are still very good choices. Some of these were mentioned in the Product Bibliography in the July 91 issue.

BLSR allows using LSR with batch jobs by only changing the JCL and does not require hiperspaces. The main benefit that it provides is its "lookaside" processing. In a standard NSR (non-shared resources) environment, VSAM will use the minimum number of data buffers and not look to see if data is already in storage. For example, in NSR if you read CI 100, then read in CI 2000, the second CI will overlay the first. Then, if you need to read CI 100 again, it will require another I/O. With batch LSR, if you provide multiple buffers, VSAM will leave the CIs in the buffers until it needs more buffers. So reading CI 100 will place it in the first buffer; reading CI 2000 would place it in the second buffer; and the read of CI 100 again would simply access the in-storage buffer, saving an I/O.

This lookaside processing also applies to indexes. In the standard NSR batch job, only one index record for each level is looked at. In LSR, you could bring all indexes in storage and eliminate all index I/O activity.

A second use of BLSR is to allow a single set of buffers for multiple VSAM components. This allows a reduction in virtual storage needed for accessing many VSAM files simultaneously.

A third use of BLSR is the ability to place VSAM buffers and control blocks above the 16Mb line without having to use hiperspaces.

BLSR is extremely beneficial for high activity KSDS or RRDS files with random processing, especially where CIs are accessed multiple times. It can degrade performance for sequential processing, since it doesn't use chained scheduling or read-ahead logic. Before I explain how batch LSR is implemented, let me give you a couple of examples of its use.

What initially caught my eye about BLSR was a presentation given by Terri Voelker of Consolidated Papers. She had implemented BLSR on two jobs to find the following savings: Reduced elapsed time from 20 minutes to 2 minutes in the first job after implementing BLSR (the second job went from 168 to 69 minutes), reduced CPU seconds from 62 to 47 (and 245 to 113), reduced EXCPs from 57K to 880 (and 10.5 million to 8.1 million). One other major job was reduced from almost 70 million EXCPs to less than 2 million. All significant savings with minimal effort!

Tom Aubrey of Commercial Union Insurance found similar significant savings with the use of BLSR. One job went from over 80 minutes to less than 18 minutes, with a 96% reduction in I/Os (from 181,461 to 7,012) and a 63% reduction in CPU time. Another job went from over 5 hours elapsed to less than 13 minutes with a 75% reduction in CPU and over 99% reduction in I/Os.

Now have I got your attention? What's the negative side of this? It takes more storage of course. Use of BLSR takes more virtual storage, more central storage, and more fixed central storage. If these are batch jobs run during non-prime time, however, it shouldn't be a major problem. Just be careful of its use when your online systems are up. If you have expanded storage, you can also use hiperspaces for the buffers. An additional feature is that BLSR can provide VSAM deferred write, where the CI is not written until the buffer is needed for another CI. This is effective if there are multiple updates to a single CI, but leads to integrity issues if there are failures.

How to implement it? Fairly simple.

- ☐ Order the manuals for BLSR and review them: GC28-1059 (BLSR for SP 3) and GC28-1672 (BLSR for SP 4). Also get GG24-3698, MVS/ESA and Data in Member Performance Studies.

- ☐ Ensure that you have the software to support BLSR. This includes MVS/ESA SP 3.1.0 with APAR OY24097, or MVS/ESA SP 4. You'll also need DFP 2.3 with APAR OY23661 or DFP 3.1 or above with APAR OY20341. Look at the fix for APAR OY33523 which prevents excessive page fixes below 16Mb when using BLSR (applicable to SP 3).

- ☐ Identify jobs that might be good candidates for BLSR. These are easiest to identify by looking at SMF type 64 records (written at VSAM close). You can extract any type 64 records with the following characteristics:

  . VSAM KSDS or RRDS
  . Randomly accessed (SMF64MC1 indicates mode)
  . High number of EXCPs
  . High elapsed times (from open to close)

  Jobs that get the most benefit from BLSR are those that access records in the same section of the file, but no SMF records provide that amount of detail.

  The SMF record contains the jobname, VSAM cluster name, component type (data or index), component name, and time of day. Since BLSR takes a lot of virtual storage and processor storage, you may not

want to implement BLSR for a job that runs when online systems are up and running.

You can also ask your IBM SE for a marketing tool called BLSRAID. This program looks at the type 64 records to identify good candidates using the same criteria mentioned above. This is a SAS program that should first be changed to sort in descending EXCP sequence to reduce the effort of analysis.

- ☐ Install the subsystem as described in the manuals, which is simply updating some parmlib members. This includes adding a line to IEFSSNxx in SYS1.PARMLIB to define BLSR:

  ssnm.CSRBISUB

  where ssnm is the name (recommendation is BLSR). This requires an re-IPL of the system to install it.

- ☐ BLSR allows any user to create hiperspace buffers by using parameters HBUFND and HBUFNI. If you want to restrict this capability, implement RACF control as described in the BLSR manual.

- ☐ Once you have found some applicable jobs, test one at a time by running them with BSLR. You'll simply need to change the JCL. If a job looks like it might get some benefit from BLSR, then you can continue to tune it by adding additional buffers.

  In each job, you'll need to add an additional JCL statement. Here's a small example:

  JCL before BLSR:

  //FILX DD DSN=cluster,DISP=SHR

  JCL after BLSR:

  //FIL DD DSN=cluster,DISP=SHR
  //FILX DD SUBSYS=(BLSR,'DDNAME=FIL')

- ☐ Once you've found a job that can achieve some savings with BLSR, then tune the job with buffers. The following parameters can be added to the SUBSYS statement: BUFND, BUFNI, HBUFND, HBUFNI, RMODE31, STRNO, DEFERW, SHRPOOL, BUFSD, BUFSI, and MSG. Many of these parameters were added to SP 3 with APAR OY33523.

  RMODE31 indicates whether you want the VSAM buffers and control blocks to reside above the 16Mb line and is only available in SP 4. The values of this subparameter are:

  ALL - buffers and control blocks above 16Mb
  BUFF - only buffers above 16Mb
  CB - only control blocks above 16Mb
  NONE - buffers & control blocks below 16Mb

  Unless you have a specific reason to keep these below the line, you should specify RMODE31=ALL. The default is RMODE31=CB.

  BUFND is the number of data buffers. If RMODE31 indicates ALL or BUFF, the default BUFND is the number of buffers that will fit in 5Mb. If buffers are below the line, the default is the number of buffers that will fit in 250K. If virtual storage (and therefore central storage) is available, specifying a large number of buffers will reduce the elapsed time, number of EXCPs and CPU time of a job.

  BUFNI is the number of index buffers. The defaults are the same as BUFND. For random processing, you only need enough buffers to contain the entire index set. If you provide at least that many buffers, you can practically eliminate all EXCPs to the index component.

  HBUFND is the number of data buffers that will be created in a hiperspace. The default is to not use hiperspace buffers and use the address space buffers only. RACF may be used to restrict use of these hiperspaces. In general, unless you have much more expanded storage than central storage, address space buffers provide more savings than hiperspace buffers (less CPU time). In a system with an overabundance of ESTOR, you might be able to get more buffers using hiperspace buffers and therefore provide more savings.

  HBUFNI is the number of index buffers created in a hiperspace. The same logic holds true as for HBUFND.

  STRNO is used to indicate how many concurrent accesses may be made to the VSAM LSR pool. The default of 16 should be sufficient for almost all jobs. Most batch jobs only need one string per VSAM file plus one string for each alternate index, so you can easily determine if more than 16 strings are needed.

  DEFERW=YES or NO indicates whether VSAM deferred write (DFR) is to be used. Non-deferred write (NDF) causes VSAM to write the CI immediately to DASD when a write or put is issued. DFR only writes the CI when the CI is needed for another request. If you have a lot of updates within a single CI, DFR could reduce the number of I/Os. There is

an exposure, however, if you are logging updates and the system comes down. Using DFR, you might think the update has been completed, but it will not have been physically updated on the file. DEFERW can reduce the number of I/Os, but be careful of its use.

SHRPOOL is used to specify one of 16 LSR pools to the dataset. As a default, BLSR will put each dataset in a separate LSR pool. If you want to combine certain datasets in the same pool, specify them with this parameter. Note: manual GG24-3698 indicates that you can't share pools, but manual GC28-1672 indicates that you can. I haven't tried it nor met anyone who has, but if you have, please let me know!

MSG indicates the level of message that should be placed in the user's job log. Values are: E - only error messages, W - warning and error messages, I - informational, warning and error messages. The default is W. I'd recommend using I until you're more experienced with BLSR, then change it back to W or even E.

BUFSD is the size of the data buffer. It defaults to the data CI size. You'll only need to specify this when placing multiple VSAM files with different CI sizes in the same shared pool.

BUFSI is the size of the index buffer. It defaults to the index CI size. See the previous paragraph.

To test the effect of these parameters, you'll need to run the job without BLSR and with BLSR and look at the differences in elapsed time, CPU time, EXCPs, and physical I/Os. You can look at type 64 records to get the number of EXCPs and the type 30 records to get the amount of CPU and elapsed times. A convenient way to get physical I/Os on the data and index components is to run a LISTCAT ALL before and after each run and look at the number of I/Os before and after. The September 91 issue described how to analyze LISTCAT output.

As an example of using these parameters, the JCL described earlier might look like this after testing:

```
//FIL  DD DSN=cluster,DISP=SHR
//FILX DD SUBSYS=(BLSR,'DDNAME=FIL',
//         'BUFND=5000,BUFNI=500,
//         RMODE31=ALL,MSG=I')
```

RESTRICTIONS

The following facilities or conditions are incompatible with BLSR and will be identified by BLSR and a message issued:

MACRF option RESET (reuse)
MACRF option UBF (user buffering)
System Data Set (bit in the ACB)
MACRF option ICI (improved CI processing)
Control blocks in common (bit in the ACB)
MACRF option GRS (global shared resources)
Empty data set

The following facilities or conditions are incompatible with BLSR, but can't be identified by BLSR:

Chained RPLs (NXTRPL option on RPL)
Implied sequential positioning (NSR assumes first GET is to start of file).

BENEFITS

If BLSR is effective in a job, it will reduce the amount of CPU time, elapsed time, and number of EXCPs. If BLSR is not effective in a job, it will take more virtual storage and not get any reduction in CPU, elapsed or EXCPs.

COSTS

Batch LSR requires more virtual storage and so will typically require more central and expanded storage. If the job is swapped out, the working set will be larger and the swap will take longer. The additional storage could impact other workloads on the system since the system paging rate could increase.

The primary cost of BLSR comes from the research to determine applicable jobs that can use BLSR. This will take someone's time, either the application programmer's or the system programmer's. It will also take additional jobs to analyze SMF records and run test jobs to see if they benefit from BLSR. Then there will be additional tuning time to determine the optimum number of buffers.

CHARGE BACK ISSUES

Boy, will this affect the charges for a job! You can't help it when you reduce CPU time by 75% and I/Os by 99%. Charges for BLSR jobs should be significantly less than not using BLSR. The question, really, is who will be charged for the research time.

USING HIPERBATCH

Hiperbatch is an ESA facility that has few applications, but when it does apply, it's astounding!

Hiperbatch works with DLF (Data Lookaside Facility) in using a hiperspace for accessing VSAM or QSAM datasets. It has the potential of eliminating thousands of I/Os and reducing the CPU time and elapsed time significantly. It can also increase all of these resources if used with programs that can't benefit from the facility.

DLF is implemented with a combination of RACF and MVS facilities. A parmlib member, COFDLFxx, defines the ESTOR limits: MAXEXPB (the maximum ESTOR megabytes that hiperbatch will use) and PCTRETB (the maximum percent of MAXEXPB that will be used for RETAINed datasets which are defined later). DLF runs as a started task and is normally started at IPL with:

S DLF,SUB=MSTR,NN=xx

where xx is the suffix of COFDLFxx.

There are two types of DLF datasets: RETAINed and non-RETAINed. A RETAINed dataset starts to use the hiperspace when the file is created or when the first WRITE to the dataset occurs. RETAINed frames can only be deleted by specifically running a job to delete them. If you forget to run the job, the frames will continue to occupy ESTOR. A non-RETAINed dataset uses a hiperspace on the first READ and deletes the frames when the last user closes the file. Essentially, DLF manages a hiperspace that will hold pages as they're referenced for a hiperbatch dataset. Savings are seen when a subsequent access from the original address space or any other address space needs to access the same data.

Hiperbatch is very good in the following three instances: 1) a dataset set is simultaneously accessed by multiple address spaces, 2) a subset of a dataset's records are repeatedly accessed by the same or multiple address spaces, and 3) a dataset (small enough to reside in ESTOR) is created in one step and accessed by one or more address spaces or steps directly after creation.

Because hiperbatch uses the Move Page facility (which takes half of the time of traditional moves between CSTOR and ESTOR and thus is responsible for much of the savings), it can only be used in environments that support Move Page. Move Page (MVPG) is supported on ES/3090 model J machines, ES/3090S machines with EC 228862), and ES/9000 machines. Check with your Amdahl or Hitachi rep for availability on those machines.

The DLF address space must be active before hiperbatch can be used. It's this address space that owns the hiperspace(s) required by the dataset. RACF 1.9 supports DLF by requiring a hiperbatch/DLF Eligibility Exit, COFXDLF1. Examples are in SYS1.SAMPLIB via DEV APARs OY28154 or OY30210 for SP 3. To define a dataset that will use hiperbatch, the dataset name must be specified in this exit. That means that you can't easily indicate use of hiperbatch via the JCL!

There are several restrictions to keep in mind when evaluating datasets that might use hiperbatch:

General restrictions:
- Hiperbatch won't run on 4381 9XE models
- Checkpoint/restart is not supported
- DB2 tables are not supported
- Not supported on machines without Move Page facility
- Not supported for shared DASD systems
- Not supported for PDSes
- Not supported for IEBGENER (use ICEGENER or SYNCGENR instead)
- Initially read or written to sequentially

VSAM datasets:
- CISIZE must be multiple of 4K
- ESDS, RRDS, KSDS allowed, but not LDS
- DFSORT can use
- No shared resources (LSR or GSR)
- No catalogs
- Not accessed with shareoptions 3 or 4

QSAM datasets:
- DASD only
- DFSORT can only use via E15/E35 exits.

IBM has a marketing tool called HBAID that may help find some good candidates for hiperbatch. It uses SMF to try to determine multiple accesses that can benefit from hiperbatch. Or you can write a program to look at dataset access. Look for QSAM datasets in the SMF type 14 & 15 records. Collect data for high activity QSAM datasets accessed by multiple users during the same time period.

To get the maximum benefit from hiperbatch, you may have to change scheduling to execute those jobs at the same time. Look for VSAM datasets in the SMF type 64 records. Collect data for high activity VSAM datasets where the file corresponds to the restrictions listed above. Then you'll simply have to test out each of the datasets. Since hiperbatch can take resources away from other users, you should restrict your testing to high priority batch or TSO users with very high activity files (found at the top of the top twenty list, for example).

Yes, this takes lots & lots of work! Many sites have tried for a month or more to find appropriate datasets with no success. Others have found one or two files that can really benefit from hiperbatch and have results that look like: reduction of 85% of EXCPs with a correspond-

ing 10% increase in CPU and 75% reduction in elapsed time; reduction of EXCPs from 5 million to 200,000 (with an increase in CPU time and a tremendous decrease in elapsed time)! IBM provides a Hiperbatch monitor in SYS1.SAMPLIB that can be used to monitor hiperbatch using a TSO command, COFDMON. (I'm not sure if this is available in only SP 4 or not).

I'm not going to take the space for a step-by-step implementation of hiperbatch, but simply suggest it as a possibility to reduce I/O. Before you start on a project to determine the benefits of hiperbatch, obtain IBM manual GC28-1673, MVS/ESA SP 4 Application Development Guide: Hiperbatch, and read it from cover to cover before starting your project! The SP 3.1.3 manual is GC28-1200. Another excellent manual was mentioned in the BLSR section and is GG24-3698, MVS/ESA and Data in Memory Performance Studies.

Hiperbatch can greatly increase the CPU busy. In one of the examples from GG24-1698, CPU busy went from 50-70% to 75-95% because the CPU that was used in 24 minutes had to be used in 17 minutes elapsed time. This is true even if you ignore the increase in overhead due to use of ESTORE and hiperbatch.

I personally think that other options, such as batch LSR and passing VIO datasets, are much more effective and infinitely easier to use than hiperbatch. You might consider, however, teaching the application designers about the option of hiperbatch. If they knew a facility such as hiperbatch exists, they might be able to design systems that can take advantage of the facility. The real benefit of hiperbatch will be the uses made of it by subsystems, such as DB2.

If you choose to use hiperbatch, you might consider using a little trick with non-retained datasets. Assume you have several jobs that read the same dataset and they all run at night (but maybe not every one of them at the same time). Since they are only read, they're implemented as non-retained datasets. But if one job reads the dataset (which is then placed in the hiperspace) and completes before the next job starts, the hiperspace would be deleted. In order to keep the dataset in the hiperspace until all jobs have accessed it, some sites will start up a job that opens the file for input, goes into a wait, but doesn't have any accesses (this will cause the dataset to be managed by DLF). Then all jobs that need the dataset are run. When the last job is completed, it can force a cancellation of the original job. This is referred to as a sleeper job and an example of this can be found in the DIM manual I previously mentioned, GG24-3698.

MISCELLANEOUS

MORE ON REDUCING I/O

Be sure to look twice at last month's recommendation on using VIO. I was talking to Tom Fleming from Mellon Bank last week and he mentioned that they had some remarkable success with using VIO. He cited a 10-hour job that was reduced to six hours by implementing VIO. They restrict the VIO files to no more than 100 cylinders and have found very good applications for them.

One more way to reduce I/O is to reduce the I/O of searching PDS directories and loading programs into storage. In an MVS/ESA environment, you can consider the use of LLA and VLF to reduce I/O delays. See the sections on LLA and VLF later in this issue.

TECHNICAL SUPPORT ARTICLE

There was a good article in the August 92 issue of Technical Support titled "Applications Tuning: The Final Frontier?" by Steven K. Thornbrugh. The majority of the article discusses a strategy and methodology for applications tuning. He ends the article with a list of recommendations for tuning consideration. I'll simply list the items that haven't already been covered in last month's issue (August 92) and this one. This list identifies some of the things that can be done, but not how to implement then. Perhaps in a later issue!

Miscellaneous Considerations:

- Eliminate image copies with DBRC.
- Eliminate redundant data base backup jobs.
- Reduce job run frequencies.
- Reduce data base reorganizations if done frequently.
- Convert IMS BMPs to DLI batch.
- Use current levels of vendor's application software.
- Reschedule work from prime shift to offshift.
- Use the lowest job priority necessary to get work done.
- Use software modeling tools.

IMS Considerations:

- Use IMS call analysis to eliminate redundant I/O.
- Watch data base structures (e.g. insert/delete rules, physical logical relationships and long twin chains).
- Redesign data bases where appropriate.
- Establish daily IMS log file for critical data bases.

Use pointer checker to locate long twin chains.
Evaluate program logic for "hot spots".
Rewrite programs as needed.
Split large programs.
Reduce IMS calls.
Optimize search fields.
Specify read limits.
Use physical views of logical segments for some replace operations.
Utilize path calls.

Throughput Considerations:

Cache selected data bases.
Eliminate exclusive use of data bases if possible.
Use parallel processing to overlap jobs.
Utilize remote printer facilities.
Split selected batch jobstreams.
Utilize a job scheduler.

MVS MEASUREMENTS

ENQUEUE ANALYSIS

Enqueue contention can be collected and reported through RMF Monitor I. In order to understand the report, let me give an overview of enqueue first. Programs can use the enqueue mechanism to serialize their work, typically done to prevent simultaneous update. The most common form of enqueue is the enqueue on a data set.

Enqueues are initiated by issuing an ENQ macro which specifies a major name (called qname), a minor name (called rname), and a type of allocation. The type of allocation can be shared or exclusive. Exclusive indicates that only one user may access the resource and shared indicates that multiple users may access the resource at once. Each application uses a standard set of names. Dataset allocation, for example, uses a major name of SYSDSN, a minor name of the dataset, and uses DISP=SHR for shared enqueues and DISP=OLD for exclusive enqueues.

The ENQ macro converts into a call to GRS, Global Resource Serialization, who in turn checks its table of currently enqueued resources. If the resource is not allocated to a conflicting type of allocation (shared if exclusive is requested or allocated exclusive), the user can continue. Otherwise, the user must wait for the resource to be released with a DEQ macro. If the wait is excessive, the waiting address space could be swapped out on a DETECTED WAIT. If you are experiencing many detected waits, you could look to see if there are many enqueue delays.

Another indication of enqueue delay is the presence of ENQUEUE EXCHANGE swaps. These only occur if SRM has swapped out some ready users and a swapped out user is holding a resource that a swapped in user needs. In this situation, SRM will swap in the user holding the resource and swap out another user (not necessarily the one requesting the resource). The swapped in job will have a high priority to stay swapped in until it has accumulated the amount of TCB service units specified on the ERV in IEAOPTxx (the default is 500).

RMF Monitor I has two reports that may be of use. The level of detail is determined by parameters defined in ERBRMFxx when RMF Monitor I is started. The parameter ENQ(DETAIL) collects job level information for resources and ENQ(SUMMARY) simply collects total statistics for resources. Many sites don't want to collect the detail level information because it's too much data. However, summary data won't tell you what jobs are experiencing the delays.

Here's my recommendation if you're trying to reduce the large volume of data. Use ENQ(SUMMARY) for a

E N Q U E U E A C T I V I T Y PAGE 1

MVS/ESA SYSTEM ID SYSA DATE 07/18/92 INTERVAL 15.00.988
SP3.1.3 RPT VERSION 4.1.2 TIME 09.45.02 CYCLE 1.000 SECONDS

-NAME- MAJOR / MINOR		CONTENTION TIME MIN	MAX	TOT	AVG	%QLEN DIST 1	2	3	4+	AVG Q LNGTH	REQUEST TYPE -EXCL- MIN	MAX	-SHARE- MIN	MAX	TOTAL EVENT
SYSDSN															
PAYR.MON.FILE1	(SYSTEMS)	13.173	44.613	201.09	28.727	100	0.0	0.0	0.0	1.00	1	1	0	0	7
SYSIGGV2															
CATALOG.PAYR	(SYSTEMS)	0.003	4.544	4.637	0.927	80.0	20.0	0.0	0.0	1.20	0	1	0	1	5

Figure 2 - RMF Monitor I Enqueue Summary Report

```
                                   E N Q U E U E   A C T I V I T Y                                    PAGE    1
            MVS/ESA                      SYSTEM ID SYSA            DATE 07/18/92          INTERVAL 15.00.988
            SP3.1.3                      RPT VERSION 4.1.2         TIME 09.45.02          CYCLE 1.000 SECONDS

-NAME-  ----- CONTENTION TIME -----  -- JOBS AT MAXIMUM CONTENTION--  -%QLEN DISTRIBUTION-  AVG Q  -REQUEST TYPE -  TOTAL
MAJOR     MIN     MAX     TOT     AVG  ----- OWN -----  ----- WAIT ----    1    2    3    4+  LNGTH  -EXCL-   -SHARE-  EVENT
 MINOR                                TOT      NAME     TOT      NAME                             MIN MAX MIN MAX
                                             SYSNAME           SYSNAME

SYSDSN
 PAYR.MON.FILE1      (SYSTEMS)

        13.173 44.613 201.09 28.727    1 PAYJOB1(E)      2 G0708001(E)   100  0.0  0.0  0.0   1.00   1   1   0   0      7
                                         SYSA              SYSA

SYSIGGV2
 CATALOG.PAYR        (SYSTEMS)

         0.003  4.544  4.637  0.927    1 CATALOG (S)     2 CATALOG (E)  80.0 20.0  0.0  0.0   1.20   0   1   0   1      5
                                         SYSA              SYSA
                                                          CATALOG (S)
                                                           SYSA
```

Figure 3 - RMF Monitor I Enqueue Detail Report

few days or weeks until you've identified the typical long enqueues (described later). Then specify ENQ(SUMMARY) once and ENQ(DETAIL,majorname) for each majorname that has long enqueues. To decrease the volume even further, you can qualify it with: ENQ(DETAIL,majorname,minorname). This will give you the details with minimal overhead.

Resource names are often not documented if they're from non-IBM vendors, but most can be identified by their prefix (e.g. UCC...). Most MVS names are documented in the MVS/ESA System Reference Manual (LY28-1011 for SP 3, LY28-1820 for SP 4) or the MVS/XA Debugging Guide, Volume 1 (LC28-1164). A few of the most common major and minor names are:

SYSDSN,dsname	Data set enqueues
SYSIGGV1,MCATOPEN	Master catalog
SYSIGGV2,catalog	Catalog
SYSVSAM,dsname	VSAM
SYSVTOC,volser	VTOC activity

Now let's look at an analysis of the reports. Figure 2 shows a sample of a summary report and Figure 3 shows a sample detail report. The detail report has simply added the name of the job(s) holding the resource and those waiting. Let's look at the summary report first in Figure 2. This shows two resources that had enqueue contention: a dataset, PAYR.MON.FILE1 and a catalog, CATALOG.PAYR. The columns called CONTENTION TIME show the minimum, maximum, total, and average time that contention existed on the resource. These units are in seconds. The far right column is the total number of enqueue occurrences that had contention.

In Figure 2, the dataset had seven enqueue conflicts. The shortest one lasted 13.173 seconds, the longest was 44.613 seconds, the total of all seven was 201.09 seconds for an average of 28.727. Were these all the enqueues to this dataset? No, just the ones that were delayed. If batch jobs are accessing this dataset, the delays aren't enormous, but they'd be devastating for any online access. Notice that the summary report doesn't give the job names. Additional fields in the report include the AVG Q LNGTH which is the average number of users waiting for the resource. For the dataset, this was an average of 1.00.

The %QLEN DISTRIBUTION shows the percent of time a queue existed for each number in the queue. So in the catalog line, we see that 80% of the time there was one job in the queue and 20% of the time there were 2 jobs waiting for the resource. The REQUEST TYPE shows the minimum and maximum number of users waiting by the type of request. Use this report to find resources that are causing long delays (perhaps more than a 5 second average) and track them using the detail report.

Figure 3 shows the corresponding detail report. Now we can see the jobs involved in the enqueue. For the dataset, PAYR.MON.FILE1, we see that the job, JOBPAY1, owned it exclusively while job G0708001 wanted it exclusively. The catalog was owned by the CATALOG address space and also required by CATALOG. The original requests were for jobs that aren't identified. This simply shows that a conflict existed on the catalog. Catalog delays, by the way, can be reduced by dividing the catalog into smaller user catalogs or by using VLF to store catalog records (only in MVS/ESA). See the section later in this issue on TUNING CATALOG & VLF.

A max of two jobs are shown (either using the resource or waiting for the resource), so other jobs may have been involved in enqueue delays. The system sim-

ply picks the jobs during the period of highest contention. For the dataset entry, that means (probably!) that PAYJOB1 was holding the dataset for 44.613 seconds while G0708001 wanted it. The primary things to look for are high priority jobs that are waiting for resources from lower priority jobs. Try to find a way to eliminate these conflicts. This is often accomplished through changes in schedules.

MVS OVERVIEW

VLF OVERVIEW

VLF, Virtual Lookaside Facility, is an ESA facility that can be thought of as a general purpose data space manager. Some people have referred to it as a data space "access method". It is a facility that can create and manage data spaces for other applications. The initial IBM users of this facility include LLA (Library Lookaside Facility), TSO, catalog, and RACF. VLF is a standard feature of SP4 and is available on SP 3.1.3 with APAR OY24097.

VLF runs as a non-swappable address space which is normally started at IPL time. Parameters in SYS1.PARMLIB define classes that can be managed by VLF. VLF will create two data spaces for each class, one for objects and one containing control info about the objects. Objects can be anything, such as modules, CLISTs, EXECs, catalog records, or RACF group records.

VLF is normally started after IPL with:

```
S VLF,SUB=MSTR,NN=xx
```

where xx is the suffix of a parmlib member, COFVLFxx. The sample, distributed PROC is:

```
//VLF PROC NN=00
//VLF EXEC PGM=COFMINIT,PARM='NN=&NN'
//IEFPARM DD DSN=SYS1.PARMLIB,DISP=SHR
```

IEFPARM defines the location of the parmlib member and can be something other than SYS1.PARMLIB.

COFVLFxx

This member defines the maximum number of virtual space (in 4K frames) that can be assigned to the data spaces for each of the classes. Internally VLF manages classes; each class having one or more major names associated with it; and each major name having one or more minor names associated with it. For example, the TSO class name is IKJEXEC, a major name would be a CLIST library, such as SYS3.CLIB, and the minor names (or objects) would be CLIST names, such as #ISMF.

The COFVLFxx member defines classes, major names, and virtual storage limits. There are two formats. If the objects are members of partitioned data sets, the parameter EDSN defines the data set. Use of EDSN indicates that VLF will automatically be notified if a member of the library is changed on the same system with a STOW. If the objects are managed by the application, EMAJ is used. Following are samples for the four MVS facilities:

```
CLASS NAME(CSVLLA)        /* LLA */
   EMAJ(LLA)
   MAXVIRT(4096)
CLASS NAME(IKJEXEC)       /* TSO */
   EDSN(SYS3.ESA.CLIB)
   EDSN(SYS1.ISRCLIB)
   MAXVIRT(256)
CLASS NAME(IGGCAS)        /* CATALOG */
   EMAJ(SYS1.CATALOG.TSO)
   EMAJ(SYS1.USERCAT1)
   MAXVIRT(256)
CLASS NAME(IRRGTS)        /* RACF */
   EMAJ(GTS)
   MAXVIRT(256)
```

The LLA class must be defined with EMAJ(LLA). The MAXVIRT(4096), which is the default, indicates that 4096 4k-frames (or 16Mb) are to be allowed for data space use for LLA modules. The TSO class defines the CLIST and REXX EXEC libraries to be managed by VLF and 256 pages (or 1Mb) for TSO CLISTs. The CATALOG class defines all catalogs that are to be managed by VLF. And the RACF class defines a group tree in storage used for group authority processing. Recommendations for managing the first three of these classes is given in later sections. RACF can only be used in a shared environment when all users of the RACF database are at RACF 1.9 or later.

User applications can also use VLF for managing objects within a data space. The applications must be authorized programs in supervisor state or system key and run in task mode. There is a set of VLF macros that allow an application to define a class (COFDEFIN), define programs that can use VLF (COFIDENT), create objects (COFCREAT), retrieve objects (COFRETRI), indicate a change in an object or invalidate an object (COFNOTIF), disallow use (COFREMOV), and delete a

class (COFPURGE). More information can be found in the MVS/ESA SPL: Application Development Macro Reference (GC28-1857 - SP3, GC28-1647 - SP 4).

STATISTICS

Every fifteen minutes, VLF will write an SMF type 41, subtype 3 record containing information for every active class. This record is automatically provided in SP 4, but requires two APARs for SP 3 sites (OY28799 and OY28800). The following information is available for each class:

SMF41CLS - Class name
SMF41MVT - MAXVIRT from COFVLFxx
SMF41USD - VSTOR being used
SMF41SRC - # of searches of data space
SMF41FND - # objects found in data space
SMF41ADD - # objects added to data space
SMF41DEL - # objects deleted
SMF41TRM - # objects trimmed from data space because another object needed to be added
SMF41LRG - Largest object attempted to put in cache

From this information, you can determine the hit ratio by dividing SMF41FND by SMF41SRC. In general, if you get less than an 80% hit ratio, VLF isn't being used efficiently. This statement doesn't apply to LLA which always shows close to 100% hit ratio. To improve the hit ratio, you can either increase the number of objects in cache by increasing MAXVIRT, or you can decrease the total number of objects that are eligible to go into the data space. For example, you might reduce the number of data sets managed by TSO or LLA. Figure 4 shows a sample report based on type 41 data for the TSO/E class. Specific suggestions for reducing the number of objects are given in the corresponding section for LLA, TSO, and Catalog.

In Figure 4, you can see that a MAXVIRT of 8Mb has been assigned to TSO/E (IKJEXEC), but VLF is using less than 30% of it (MEM USED). The hit ratio is between 34 and 63%, which is not very efficient. In this example, the activity is quite low (308 max requests in 15 minutes).

IBM Marketing Tools also has a tool called VLFAID. VLFAID uses a GTF trace to determine which libraries are good candidates for VLF and LLA management. There is overhead associated with running GTF trace for all load activity, so caution is advised in a severely CPU-constrained environment.

STORAGE ISOLATION

If you keep a great deal of data in data spaces managed by VLF, VLF's large data spaces may cause storage constraint (either for VLF or for other users). You may find that you'll need to storage isolate VLF. I've seen some working sets that are over 80Mb of CSTOR and ESTOR just for VLF. Since the storage isolation value will be unique for VLF, you'll need to assign VLF to its own performance group in the ICS, such as:

```
SUBSYS=STC
    TRXNAME=VLF,PGN=24
```

You can use storage isolation to restrict VLF's impact on other users by coding a limit on the working set size, such as:

```
PGN=24,(DMN=n,PWSS=(0,10000))
```

This is called reverse storage isolation and limits the region to 10,000 central and expanded frames (40Mb). If you want to protect VLF from high paging itself, then use a minimum working set size and allow the real page rate (page-ins per clock second) to determine the number of frames to be protected:

```
PGN=24,(DMN=n,PWSS=(2000,*),PPGRTR=(1,2)...)
```

You can't use the PPGRT parameter for VLF since PPGRT uses the page-ins per execution time instead of residency time and VLF has little execution time. VLF is primarily called using cross-memory services where the CPU time is charged to the caller and not VLF.

DATE	TIME	CLASS	MEM ALLOC	MEM USED	SEARCHES	HITS	HIT PCT	ADDS	DELS	TRIMS	MAX SIZE
92/141	07:40:48	IKJEXEC	8Mb	2352K	308	174	56.5%	5	0	0	51K
92/141	07:55:48	IKJEXEC	8Mb	2384K	239	97	40.6%	11	0	0	51K
92/141	08:10:48	IKJEXEC	8Mb	2408K	195	102	52.3%	7	0	0	51K
92/141	08:25:48	IKJEXEC	8Mb	2448K	237	88	38.1%	8	0	0	51K
92/141	08:40:48	IKJEXEC	8Mb	2460K	195	88	44.2%	4	0	0	51K
92/141	08:55:48	IKJEXEC	8Mb	2484K	205	102	48.8%	9	0	0	51K
92/141	09:10:48	IKJEXEC	8Mb	2504K	186	119	63.3%	6	0	0	51K
92/141	09:25:48	IKJEXEC	8Mb	2524K	138	45	34.6%	5	0	0	51K

Figure 4 - VLF Statistics for TSO/E

DISPATCH PRIORITY

There are a variety of opinions regarding setting of the dispatch priority of VLF. The answer is that "IT DEPENDS". Most accesses to VLF are done using cross-memory program calls and therefore do not depend on the VLF dispatch priority. Because of that, several people recommend that you use a low dispatch priority for VLF. I can't agree. VLF uses its dispatch priority during initialization, when responding to operator commands, and performing refreshes or updates.

Some sources, including the IBM Init & Tuning manual, recommend putting VLF below your IMS control region and CICS terminal owning region. I have a hard time agreeing with this view also. Often, IMS and CICS use VLF and will be delayed if VLF is delayed. Additionally, higher priority tasks than CICS and IMS may need VLF services. For that reason, I recommend placing VLF dispatch priority above your online systems and other high priority work. If you later find that your systems are delayed by VLF (by looking at an online monitor), then lower it, but I doubt that you'll need to. VLF normally takes so little CPU time that it really doesn't pose a problem. LLA is a different matter and is discussed in the next section.

LLA OVERVIEW

LLA, Library Lookaside, is a facility that has been around for several years; it's just been greatly updated in ESA. It still performs the original function as LLA, Linklist Lookaside, which was to keep the directory entries for SYS1.LINKLIB and other linklist libraries (as defined in member LNKLSTxx) in storage. This reduces or eliminates the number of I/Os to the physical directory.

In ESA, LLA has been expanded to perform two more functions. LLA can manage the directories for many load (and non-load) libraries in addition to the linklist libraries. It allows you to specify a number of PDSes whose directories are to be stored in LLA's address space virtual storage and searched in virtual storage. A second function that ESA LLA can perform is the management of load modules by VLF. Active modules can be placed in a VLF-managed data space and simply be moved directly from the VLF data space to a user's address space rather than being brought into the user's address space by physical I/Os.

Both of these facilities are meant to reduce the number of physical I/Os. To better understand how this works, let's look at how LLA handles directories. There are two types of directories - those specified with FREEZE and those with NOFREEZE. FREEZE tells LLA that the directory will not be updated without notifying LLA, so LLA can therefore read the directory into storage and do all in-storage searches. FREEZE is the default for the linklist libraries and eliminates directory I/Os. NOFREEZE tells LLA that the directory can be updated at any time and therefore LLA cannot depend on the in-storage directory. There is really no reduction of I/O to the directory of NOFREEZE libraries since LLA must go to the DASD directory each time. However, if you let LLA use VLF you can reduce I/Os during the load of modules if they're managed by the VLF data space.

If the LLA class is defined to VLF in the COFVLFxx parmlib member, then VLF can manage modules for LLA. LLA keeps statistics on the number of fetches from a library and the length of each fetch. After either 2000 module fetches from a library or after the 10th fetch of a single module, LLA will initiate the module staging function. Staging refers to copying modules from DASD to the VLF data space.

When staging begins, each module in the directory list is given a staging value based on four components: response time, contention, storage, and installation. VLF will calculate the staging value based on the value for each component (from -10,000 to +10,000) times a weighting factor (from 0 to 100) for each component. The response time is based on a calculation derived from savings that a fetch from VLF would provide and has a weight factor of 75.

The contention is based on the average difference between the minimum fetch LLA or DASD and the duration of the fetch and has a weight factor of 50. It provides an indication of the contention caused by shared DASD. Storage is based on the number of processor bytes used for keeping a module in VLF and has a weight factor of 25. Installation is a user-defined cost provided by an exit and has a default value of zero.

Modules which have high staging values will be staged into the VLF data space (or remain there if they had been previously staged). Modules with low staging values that are already in the VLF data space will be overlaid. As an example, let's begin with an IPL and assume that LLA will simply collect information for awhile (until 2000 requests have been made or a module has been requested 10 times). During that period of time, LLA will collect statistics for every module loaded. When staging initiates, LLA will calculate a staging value for all the modules in the directory, based on the length of time to load the module, whether it's on shared DASD, the size of the module, and the weighting factors. It will then determine which of the modules can reside in

VLF's data space and notify VLF of the modules to be managed. The modules that will be managed by VLF are all loaded into storage at once. This staging will then occur periodically during the day, after another 2000 requests or 10 requests to a single non-VLF-managed module.

There are two exits that can be used to change the standard staging, CSVLLIX1 and CSVLLIX2. These are described later. Modules that are staged in the VLF data space will take less I/Os to obtain the module, but more CPU time. The elapsed time will almost always be less.

As mentioned before, the benefit of NOFREEZE libraries is that they can benefit from VLF management. If VLF is not used, then NOFREEZE libraries do not provide any benefit because each call to a module requires that the module be read from DASD. This is because the NOFREEZE modules are updated only on DASD and the LLA directories may not point at the most recent modules.

STARTING LLA

If LLA is not going to use VLF to hold modules, it can be started directly after IPL, such as:

```
S LLA,LLA=xx,SUB=MSTR
```

where "xx" is the suffix to member CSVLLAxx. If LLA is going to use VLF, it must be started after VLF has been started, such as:

```
S VLF,NN=xx,SUB=MSTR
S LLA,LLA=xx
```

CSVLLAxx

There can be one or more parmlib members that define the LLA libraries. The first member is pointed to by the start command, but that member can then point to other members of parmlib or to other datasets. The parameters are as follows:

LIBRARIES(dsn1,dsn2,...)
: where -LNKLST- is a special ddname to indicate SYS1.LINKLIB and all LNKLSTxx libraries. This defines libraries to be added to LLA control.

REMOVE(dsn1,dsn2,...)
: This defines libraries to be removed from LLA control.

LIBRARIES(dsn1,dsn2,...)
MEMBERS(mem1,mem2,...)
: Used during a refresh to identify specific modules and libraries to be refreshed or updated.

LNKMEMBERS(mem1,mem2,...)
: Used to identify linklist modules to be refreshed.

FREEZE/NOFREEZE(dsn1,dsn2,...)
: Defines the FREEZE characteristics of each of the datasets defined in the LIBRARIES statement. -LNKLST- defaults to FREEZE, all others defaults to NOFREEZE.

PDS(dsn1) SUFFIX(xx)
: Defines an indirect pointer. If this is specified, LLA will select the CSVLLAxx member from dsn1. This is often used to allow other applications to control their own library designation.

In order to understand how these are used, let's look at a simple example. Assume the following parmlib members exist:

```
COFVLF00:
  CLASS NAME(LLA)
    EMAJ(LLA)
    MAXVIRT(4096)

CSVLLA00:
  LIBRARIES(-LNKLST-,SYS2.LOAD,APPL.LOAD)
  FREEZE(SYS2.LOAD)
  PDS(SYS1.PARMLIB) SUFFIX(CI)

CSVLLACI:
  LIBRARIES(CICS1.LOAD)

CSVLLAUP:
  LIBRARIES(SYS2.LOAD)
  MEMBERS(MEMBRTA)
```

At start up, you might specify:

```
S VLF,NN=00,SUB=MSTR
S LLA,LLA=00
```

The start of VLF indicates that COFVLF00 is to be used and this member shows that LLA can use VLF. The start of LLA indicates that CSALLA00 is to be used. and CSVLLA00 points to CSVLLACI in SYS1.PARMLIB that contains additional parameters.

This set of members would cause all the linklist directories to be read into storage, along with the directories for SYS2.LOAD, APPL.LOAD, and CICS1.LOAD. CICS1.LOAD would have been obtained through the indirect reference of PDS(SYS1.PARMLIB) SUFFIX(CI). Directory searches for linklist and SYS2.LOAD will be resolved without any I/Os because of the FREEZE option (specified for SYS2.LOAD and default for -LNKLST-); modules for linklist,

SYS2.LOAD, APPL.LOAD, and CICS1.LOAD will be managed by VLF, since VLF was started before LLA and COFVLF00 specified that LLA could use VLF.

Now, assume that you've updated module MEMBRTA in SYS2.LOAD. You can notify LLA and VLF with the following command:

F LLA,UPDATE=**UP**

This will look at CSVLLA**UP** and identify that member MEMBRTA in SYS2.LOAD needs to be refreshed. Actually, MEMBRTA will simply be deleted from the VLF data space, the directory entry will be updated, and the use count reset to zero.

EXITS

LLA provides two user exits to allow you to obtain statistics and control modules. The exits are CSVLLIX1 and CSVLLIX2. Statistics for each module are kept by LLA and passed to the exits. These statistics include: elapsed time to fetch the module, variations in elapsed time with VLF or IEWFETCH, EPA (entry point address), number of fetches per EPA, UIC, and migration age.

CSVLLIX1 is invoked during EVERY program fetch. This exit is used to monitor and collect statistics, modify the 2000 fetch default, force staging for a module, and/or modify statistics to influence the trigger of staging. Since there could be considerable overhead by invoking this exit, it's highly recommended that you try to use CSVLLIX2 instead. If you use CSVLLIX1 ensure that the code is as efficient as possible!

CSVLLIX2 is called for each module when LLA triggers its staging function. This is done far less frequently than module loads and so takes less CPU time than invoking CSVLLIX1 each program fetch. CSVLLIX2 is used to analyze fetch statistics, change the weight factors, provide an installation cost, and force a module to be staged or to be bypassed for staging.

There is an old Washington Systems Center Flash #8908 that is still a very useful document in helping to understand LLA. While it was written for SP 3, most of it is applicable to SP 4.

TUNING CATALOG & VLF

The catalog address space (CAS) can use VLF to store catalog records. Highly referenced catalog entries can then be accessed without any I/Os and catalog searches. I need to warn you, however, that there have been several APARs reported about this facility and you should be as current as possible with maintenance. Otherwise you could destroy your catalogs. No matter how current the maintenance, be sure to back up your catalogs prior to trying this facility. Don't let me discourage you too much - several sites are seeing a major benefit by using VLF for catalog. Just be careful!

You can implement CAS and VLF simply by defining the CAS class in COFVLFxx, as shown in the VLF overview (CLASS IGGCAS). Any catalog defined in COFVLFxx will be controlled by VLF. The first reference to a catalog record will place the record in VLF's data space. Subsequent references on a non-shared device will be able to use the record from the VLF data space rather than doing a physical I/O. VLF will add records until the data space is full (based on MAXVIRT) and then start overlaying the least-recently-used (LRU) entries. Therefore, highly referenced records will tend to stay in the data space.

A shared catalog and shared device present another problem. You would certainly want to know if an update occurred on another system, so CAS must access the disk every time if the volume is defined as shared. When CAS is using VLF, however, it has a more efficient technique to determine if a record has been updated. CAS keeps a record of updates in the VVR within the BCS. This area can accommodate 90 updates before it fills up. CAS will look in the update record to see if a record in VLF has been updated. If not, CAS uses the VLF in-storage version of the catalog record. If it has been updated, CAS performs the standard catalog search (which takes more I/Os than the single I/O to retrieve the update record), gets the new record and passes it to VLF to overlay the previous version.

If there are more than 90 updates, CAS won't know which records to trust any more, so all records for that catalog in the data space is refreshed. These refreshes empty the data space and cause CAS and VLF to start over (which isn't an efficient way to go!). You could see an increase in response times when accessing the catalog after a refresh. As a general guideline, shared catalogs with many updates are not good candidates for CAS and VLF.

You can get statistics on CAS and VLF performance in two ways. The best way is to use an operator command to get the catalog statistics by catalog. The command is:

F CATALOG,REPORT,VLF

This will give you the following stats: number of records added, number of requests, number of successful

```
F CATALOG,REPORT,VLF
IEC351I CATALOG ADDRESS SPACE MODIFY COMMAND ACTIVE
IEC161I 084(048,054,IGG0CLFQ)-003,OVNDLOAT,STEP8,VQINDX,,,
IEC161I CATALOG.UCAT6,,CATALOG.UCAT6
IEC359I CATALOG REPORT OUTPUT 871
**CAS**************************************CATALOG DATA SPACE CACHE**
**     HIT%  RECORDS  SEARCHES   HITS   DELETES  SHARING  INVALID   **
*************************************************************************
** CATALOG.UCAT1                                                       **
**     072%  0000008E 000002F4 00000222 00000011 00000000 00000000  **
** CATALOG.UCAT2                                                       **
**     072%  00001A0B 000232A0 0001CF01 00001EFA 00000000 00000000  **
** CATALOG.UCAT3                                                       **
**     094%  00001251 00033C1A 000311CE 000012F6 00000000 00000000  **
** CATALOG.UCAT4                                                       **
**     079%  00028170 000F3AC2 000C1504 00002A0C 00000000 00000000  **
** CATALOG.UCAT5                                                       **
**     027%  0001411C 0002EFF0 0000C267 000011C4 000001A3 0000001B  **
** CATALOG.UCAT6                                                       **
**     083%  00009117 00042D16 000378B1 00001763 00000000 00000000  **
** CATALOG.UCAT7                                                       **
**     092%  00005B1A 0006CE57 00064345 00000BB1 00000000 00000000  **
** CATALOG.MASTCAT                                                     **
**     048%  00000991 00039F2C 0001C309 000000FE 00000000 00000000  **
*************************************************************************
**     HIT%  RECORDS  SEARCHES   HITS   DELETES  SHARING  INVALID   **
**CAS**************************************CATALOG DATA SPACE CACHE**
```

Figure 5 - VLF CATALOG Display Command Output

requests, number of records deleted from VLF, number of VLF updates from shared catalog, number of VLF refreshes due to over 90 updates. Figure 5 is a sample from this command.

(Yes, it certainly would be nice if the values were converted from hex, but you take what you can get!) Notice that CATALOG.UCAT5 had several refreshes (as shown in the INVALID column). That was the reason for the low hit ratio of 27%. This is obviously not a good candidate for VLF due to the large number of updates, and therefore, refreshes. Notice also, that the master catalog doesn't have as high a hit ratio as the others. More about that later.

You can also get statistics from the VLF type 41 records that are written every 15 minutes. Figure 6 is an example of this data. The percent of hits is simply calculated by dividing the number of hits by the number of searches. You should try for a goal of at least 80 to 90%. Notice how the percent decreases whenever there is a refresh (see the column called "OBJECT TRIMS").

How do you improve the percent of hits? You can either provide more virtual storage on the COFVLFxx member in the MAXVIRT parameter or you can put fewer catalog records in the data space. You might have a catalog, for example, that doesn't get hit much during the day. The few references during the day would bring a record into the data space but the record wouldn't get hit often and produce a poor hit ratio. Some sites use a different VLF parmlib member for day and night processing.

There is a lot of documentation that indicates that the master catalog shouldn't be managed by VLF. That recommendation is based on the fact that most sites use a master catalog primarily for pointers to user catalogs. The alias table is maintained in the catalog address space and wouldn't get any benefit from VLF. Most of the system data sets in the master catalog have other pointers and don't really require a catalog search every time. Therefore, the recommendation to avoid putting master catalog in VLF stems from the assumption that you have a "clean" master catalog. If you don't, you might benefit from adding your master catalog to VLF control as well.

Obviously the most efficient way to use catalogs and VLF is to have non-shared catalogs and provide enough MAXVIRT data space for every catalog record. This would eliminate all read I/Os but the first reference to every catalog record. It wouldn't save write I/Os because if a catalog record is updated, the record is written back to DASD, as well as any index records that need updating and the catalog update record (the one that holds 90 updates).

But you probably can't really do all that, so here are some alternatives:

- ☐ Most of the following alternatives require changing the COFVLFxx member. Changes in this member will only become effective when VLF is stopped (P VLF) and restarted (S VLF). Starting and stopping VLF is fairly traumatic to the system and any application using it, such as LLA and TSO. Therefore you should consider how you're going to manage

DATE	TIME	CLASS	MEM ALLOC	MEM USED	PCT USED	SEARCHES	HITS	HIT PCT	ADDS	DELS	TRIMS	MAX SIZE
92/141	07:40:48	CAS	20.8Mb	18.2Mb	87.5%	2682	2613	97.4%	77	73	0	21.0K
92/141	07:55:48	CAS	20.8Mb	18.2Mb	87.5%	1807	1747	96.6%	65	58	0	21.0K
92/141	08:10:48	CAS	20.8Mb	18.2Mb	87.6%	2090	1694	81.0%	144	21	0	21.0K
92/141	08:25:48	CAS	20.8Mb	18.2Mb	87.6%	3341	3267	97.7%	75	74	0	21.0K
92/141	08:40:48	CAS	20.8Mb	18.3Mb	88.4%	9097	7389	81.2%	1515	195	0	21.0K
92/141	08:55:48	CAS	20.8Mb	19.6Mb	94.5%	23493	4368	18.5%	19216	340	7875	21.0K
92/141	09:10:49	CAS	20.8Mb	18.7Mb	90.2%	28677	2045	7.1%	26275	117	35167	21.0K
92/141	09:25:49	CAS	20.8Mb	18.9Mb	91.2%	17651	14724	83.4%	1694	34	0	21.0K
92/141	09:40:49	CAS	20.8Mb	19.2Mb	92.6%	24796	22062	88.9%	2290	16	0	21.0K
92/141	09:55:49	CAS	20.8Mb	18.7Mb	89.9%	14389	2114	14.6%	12228	9	20439	21.0K
92/141	10:10:49	CAS	20.8Mb	18.6Mb	89.8%	1192	1130	94.7%	180	176	0	21.0K
92/141	10:25:49	CAS	20.8Mb	18.6Mb	89.8%	312	305	97.7%	27	27	0	21.0K

Figure 6 - Sample Catalog VLF Statistics

changes to VLF. You might decide, for example, to stop and start VLF every morning at 5am when it will cause the least impact to the system. The only way I can recommend changing it during the day is if the catalog is the only user of VLF and you're trying to determine the correct catalogs to control with VLF.

☐ Start a tracking procedure to collect the number of I/O operations to any catalog volume before you start a tuning process. The object of using VLF for catalogs is to reduce the number of I/Os and the length of I/Os to a DASD device. For example, you can use the RMF Monitor I Device Activity report (Figure 7) to collect SSCHS per second (Device Activity Rate column) and response time (Avg Resp Time in milliseconds) for the catalog volumes. See Figure 7 for an example of this report. The objective of the tuning effort is to reduce either the number of I/Os (on a non-shared device) or the length of an I/O (on a shared device). You'll be using storage and a small amount of CPU to achieve that reduction.

VOLUME SERIAL	LCU	DEVICE ACTIVITY RATE	AVG RESP TIME
DSK427	00D	0.013	11
TSO044	00D	4.200	18
TSO045	00D	2.723	20
DSK256	00D	0.001	8
DSK431	00D	0.566	15

Figure 7 - RMF Monitor I DASD Activity

☐ If you've already implemented VLF and catalogs:

1. Issue the catalog display command and evaluate the catalogs. Catalogs on shared systems that get many refreshes and/or low hit ratios should be removed from control (that is, remove them from the COFVLFxx member).

2. Produce a report from the type 41 records to look at the overall hit ratio of the catalogs. A hit ratio of less than 80% indicates a poor selection of choices.

3. If the hit ratio is too low for catalogs (from the type 41 records) and the system is not storage-constrained, increase MAXVIRT. Do it in stages so you can tell when you've gotten the best benefit. Perhaps add 20% at a time. There will come a point where additional storage will improve the hit ratio by only a fraction. If you do this increase slowly, you'll be able to tell when to stop. As you add storage evenly for example, you might see the hit ratio move from 66% to 70% to 78% to 84% to 88% to 90% to 91% to 92% to 93%, etc. It looks like you should stop at 88% or 90% to obtain the best improvement for the associated storage increase.

If you can't increase the hit ratio, the problem could be due to a single poor catalog. Try removing the catalog with the lowest hit ratio and reevaluate the statistics. If this still doesn't work, start from scratch with the recommendations below.

4. If the hit ratio is high and you want to reduce the storage constraint, then start reducing MAXVIRT in the same manner as just described. Reduce it by the same amount each time and reevaluate the hit ratio after the change. You'll be able to find a break-even point along the way.

☐ If you haven't implemented VLF for catalogs, consider the following steps:

1. Get up to current maintenance. Search all the APARs and ensure that you're current.

2. Backup a catalog that you want to test.

3. Add the catalog name to COFVLFxx with the

minimum MAXVIRT(256). A catalog typically uses only 60 frames, so this should be sufficient.

4. Monitor the statistics with the operator command to show the hit ratio.

5. Remove that catalog from VLF control and add another one. Keep the statistics. Do this for all catalogs that you hope to control via VLF.

6. Now, add all the catalogs that got a good hit ratio (over 80%) with 256 frames to COFVLFxx. Now you can tune MAXVIRT as described above.

The reason for this technique of looking at one catalog at a time is because analyzing multiple catalogs at one time is very difficult. For example, you might have a catalog with a very poor hit ratio but a high activity. That catalog would keep bringing records into the data space, overlaying other records from other catalogs. The hit ratio of the other catalogs would be reduced because of the first catalog. There isn't a good way to find this situation without looking at one catalog at a time.

- ☐ When catalog use of VLF is effective (hit ratios of over 90%), consider the following:

1. Reduce the number of user catalogs. Many people have split a catalog when the activity rate grows on a single catalog and I/O delays are seen. If you've done that in the past and are able to reduce the I/O delays on your catalogs, you might be able to combine them and reduce maintenance effort and backup procedures.

2. When designing new naming structures, consider the impact of catalogs that have a high number of updates versus those that have few updates. For example, one site designed a naming structure where all libraries and common datasets had a unique high-level prefix. They were all placed in a single catalog that had very few updates. You can also define a set of high-level indexes for your work datasets that can be controlled by a non-VLF managed catalog. While this technique isn't applicable to all sites, it's at least worth the consideration if you're in the middle of defining new naming standards (such as for SMS).

TUNING TSO & VLF

TSO/E can use VLF to store CLISTs and REXX EXECs. There are really two savings in this use. If VLF contains the CLIST or EXEC, TSO can eliminate the I/Os to retrieve the member from disk. Additionally, CLISTs must be recompiled before any execution and the compiled version is the one stored in VLF's data space. Thus, you can reduce the CPU time used to compile the unchanged CLIST multiple times.

There are two problems when trying to use VLF for TSO. First and foremost is the update problem. VLF is automatically notified of a change to a CLIST or EXEC only when STOW is used. Unfortunately, STOW isn't used in the following situations:

1. AMASPZAP
2. IEBCOPY
3. IPOUPDTE
4. ISPF 3.3
5. Shared Systems

If any of these techniques are used to update a CLIST or EXEC in a library, the user will have to manually notify VLF that a change has occurred and VLF will use the DASD version instead of the VLF version at that time. VLFNOTE is a TSO command that is used to notify VLF of a change. SP 4.3 has announced that it will support global VLFNOTE in a sysplex environment.

Another problem when using VLF is that many CLISTs are used only once during a day. You actually use more CPU time for CLISTs that are only used once or at least infrequently enough to keep them in the VLF data space.

The best performance improvements using this feature are obtained when frequently referenced, large CLISTs or EXECs are managed by VLF. Here are some suggestions for tuning this environment:

- ☐ Develop procedures and documentation for users to ensure that VLF is notified when CLISTs or EXECs are updated. Show people how to use the VLFNOTE command and set up procedures for batch jobs that update the libraries.

- ☐ Tune VLF using the type 41 records to provide enough MAXVIRT for TSO to produce a high hit ratio (over 80%). If you can't achieve a good hit ratio without using an extremely large amount of storage (1 Mb should be sufficient for most environments), then consider removing some of the libraries from TSO control.

- ☐ Review the concatenation of CLIST and EXEC libraries. Place the VLF-managed libraries at the front of the concatenations (if possible) in order to get the most benefit from VLF.

- ☐ The most efficient way to use VLF for TSO is to

define system-wide CLIST and EXEC libraries that only contain frequently used modules. This library would then be controlled by a single person or group who would ensure that all VLF systems are notified of a change. I've used this technique for one system where the MAXVIRT was reduced from 16M to 1M and the hit ratio improved from 75% to 100%. Not a bad deal!

TUNING LLA

There are a lot of considerations for tuning LLA, especially with its use of VLF. To understand them, you should be aware of how LLA works as described in the LLA OVERVIEW.

Let's first consider the use of FREEZE in the CSVLLAxx member. This feature provides the best performance improvements because LLA can bypass the I/O for all reads of the directory. Remember that the FREEZE parameter has nothing to do with VLF usage. The best libraries to use are those with frequent accesses and few updates. These can include production application libraries, system libraries, TSO panel libraries, subsystem or vendor libraries, or other commonly used libraries (e.g. COBOL subroutines that are loaded due to the RESIDENT option discussed earlier).

LLA's use of VLF is used to reduce the I/Os to load frequently used, seldom-changed, modules. Since LLA and VLF will work to identify these types of modules, there is little reason to keep libraries managed by VLF that have few modules that fit this requirement. Libraries that don't get any benefit from VLF and are specified as NOFREEZE (and so don't get any benefit from a reduction in directory searches) should be removed from LLA control.

If CLASS(LLA) is defined in COFVLFxx, then LLA will periodically stage modules (move them) to VLF's data space and VLF can pass them back for subsequent loads. That implies that the benefits are primarily from frequently referenced modules. Application production libraries, therefore, are seldom good candidates for VLF since most programs in such libraries are only called once.

Keep in mind also that LLA is not notified about program changes. In the NOFREEZE libraries, LLA will always use the directory entry to obtain the latest modules. In the FREEZE libraries, LLA assumes that you will notify it with a refresh or update when a module changes. Therefore, consider the frequency of update when you decide whether to use FREEZE or NOFREEZE

Here are some suggestions for getting the most out of VLF:

- ☐ Determine which libraries have high access and few updates. This is a lot more difficult than it seems. If you have a product that provides this information, then you're ahead of the game. You can use logic to "asssume" which libraries meet the criteria. If you have a DASD monitor that keeps I/Os by dataset name, you can use that information.

- ☐ Once you've found that set of libraries, collect data on the number of I/Os to the libraries or devices (from the RMF Monitor I Device Activity report for example - see Figure 7) and add them to CSVLLAxx as FREEZE libraries. You'll also need to establish procedures for notifying LLA of a change to any of the libraries. Collect statistics on the device statistics and you should see immediate savings in a decrease to the volumes.

- ☐ Review all library concatenations. The LLA-managed libraries should always be first in a concatenation. LLA is called for all libraries, so in a concatenation with a non-LLA library followed by a LLA library, LLA will be called first for the non-LLA (and will return immediately saying that it's not managed), the program will then issue a directory search on DASD, and if the module isn't found will call LLA again for the LLA-managed library.

- ☐ Now review your libraries and try to determine which libraries might be good candidates for VLF. These libraries would have frequently accessed modules that could benefit from residency in virtual storage. If the libraries are infrequently updated, add them to CSVLLAxx as FREEZE libraries, otherwise add them as NOFREEZE. Again, track the device activity as a measure of goodness.

  You can use the VLF statistics (Figure 8) to see how many searches were satisfied without an I/O. Unfortunately, you can't use the hit ratio to determine anything! It will almost always be close to 100%. LLA simply won't call VLF if it knows the module won't be there. After all, it was LLA that staged the modules to VLF so it knows which modules were sent there. The only reason that the hit ratio isn't 100% every time is because VLF may have deleted or trimmed some modules because the data space was full or modules were refreshed.

- ☐ If you want to get better statistics as to which modules from which libraries are making use of VLF, you'll have to implement one of the LLA exits. As mentioned before, I'd highly recommend use of

DATE	TIME	CLASS	MEM ALLOC	MEM USED	SEARCHES	HITS	HIT PCT	ADDS	DELS	TRIMS	MAX SIZE
92/141	07:40:48	CSVLLA	16Mb	13Mb	14462	14462	100.0%	5	0	0	26.8K
92/141	07:55:48	CSVLLA	16Mb	13Mb	7657	7656	100.0%	10	0	0	26.8K
92/141	08:10:48	CSVLLA	16Mb	14Mb	7542	7513	99.6%	10	4	0	26.8K
92/141	08:25:48	CSVLLA	16Mb	14Mb	7704	7704	100.0%	0	0	0	26.8K
92/141	08:40:48	CSVLLA	16Mb	14Mb	6318	6318	100.0%	0	0	0	26.8K
92/141	08:55:48	CSVLLA	16Mb	14Mb	6418	6375	99.3%	0	0	0	26.8K
92/141	09:10:48	CSVLLA	16Mb	14Mb	3242	3242	100.0%	0	0	0	26.8K
92/141	09:25:48	CSVLLA	16Mb	14Mb	4292	4289	100.0%	3	0	0	26.8K

Figure 8 - VLF Statistics for CSVLLA

CSVLLIX2 instead of CSVLLIX1. Once you've collected the statistics, you may find that one or more of the libraries just doesn't provide that much benefit.

One site used the following technique. It was very effective, but I doubt that it will stand up over time as new subsystems are implemented and workloads change. They implemented VLF right after a storage and CPU upgrade, gave VLF lots and lots of MAXVIRT storage, and put ALL libraries under VLF control. They then implemented an exit to collect data on which modules were being staged into the data space. They then created a single library for all of their high use modules. That was the only library, besides linklist of course, that was defined to LLA. They were able to reduce the MAXVIRT (just large enough to hold the entire library) and got some tremendous savings without the overhead of LLA trying to manage and restage modules. An interesting option.

- ☐ Review the LLA dispatch priority. As I mentioned before, the IBM Init & Tuning manual recommends placing LLA below your CICS terminal owning region (TOR) and IMS control region. I recommend keeping it fairly high, even above CICS and IMS if they are using LLA-managed libraries. I'll go into more detail about CICS's use of LLA next month.

  LLA is normally accessed with cross-memory services, so the dispatch priority is the user's priority, and CPU time will be charged to the user. LLA's priority is used when it's refreshing or updating members and libraries and when it's staging modules to VLF. If this occurs infrequently or if you're running on a system with 2 or more CPUs, then you should give LLA a very high priority. After all, if you need to refresh a CICS module, you don't want to be delayed behind CICS! And multiple CPUs ensures that the highest priority online systems could get to another CPU. If this updating occurs frequently enough to disrupt your online systems, then and only then, give it a priority below your highest priority online regions.

- ☐ Since LLA and VLF require knowledge of the libraries and your involvement in determining the best libraries for LLA and VLF control, consider other products that provide a similar function, such as Legent's PMO and Quick-Fetch. See the PRODUCT HIGHLIGHT for more information.

IN BRIEF

RMF APAR

In RMF SP 4.2.1 and 4.2.2 post processor reports, you'll find some invalid dates. There was an error in converting the julian date to character format. The routine assumed that January had only 29 days in 1992. That means that without the APAR, you'll find incorrect dates in your reports. See APAR OY51568 for the applicable fix.

SRM APAR

In the January 92 (page 18) issue, I mentioned a special criteria age for SRM ESCT parameters. I said that the value "32767" was a special value and indicated that the frames in that group (types of frames within a domain) were NEVER to go to expanded storage. This is also documented in the SRM Init & Tuning Reference. One of our subscribers tested this and found that it didn't work as indicated. It turns out that it doesn't work at all. If the migration age is over 32767, the frames will still go to expanded storage. APAR OY57191 indicates that the problem is closed as a documentation problem (they were going to simply change the documentation). Last week, an IBM developer stated that they intended to implement it as documented. Whether that will happen soon is up in the air. It's still very effective to code a high value in the ESCT parameters if you want to discourage movement of frames to expanded.

Case	CP1	CP2 | CP3 | CP4	Notes
A	1 Dedicated CP	3 Dedicated CPs	Standalone 1-way and 3-way
B	1 Dedicated CP	2 Shared CPs --- 2 Shared CPs	Standalone 1-way. Two 2-ways sharing 3 CPs.
C	1 Dedicated CP	2 Shared CPs --- 3 Shared CPs	Standalone 1-way. A 2-way and a 3-way sharing 3 CPs
D	1 Shared CP --- 2 Shared CPs --- 3 Shared CPs		A 1-way, 2-way and 3-way sharing 4 CPs.
E	2 Dedicated CPs	1 Shared CP --- 2 Shared CPs	Standalone 2-way. A 1-way and 2-way sharing 2 CPs. 400S in Single Image Mode

Case	Number of Logical CPs	LPARs Sharing CPs	Number of Non-Dedicated CPs	Dedicated CPs	Dedicated LPARs	Overhead Guide lines
A	0	0	0	4	2	2.00
B	4	2	3	1	1	7.67
C	5	2	3	1	1	9.17
D	6	3	4	0	0	9.75
E	3	2	2	2	1	7.50

Table I - PR/SM Overhead Formula - © 1991, 1992 Alan M. Sherkow

PR/SM OVERHEAD

One of the most frequently asked questions I get concerns the overhead of PR/SM. I've discussed PR/SM overhead in the following issues: February 91 (entire issue), March 91 page 7 & 10 (I/O elongation and new PR/SM APAR), May 91 page 24 (PR/SM capping), December 91 page 12 (dedicated & shared), May 92 page 4 and 6 (CPU variability), and June 92 (issue on benchmarking, see page 6).

There really isn't a good rule of thumb regarding the overhead of PR/SM because it very much depends on your workload, the architecture of the hardware buffers, and the level of the microcode. We can estimate, however, the relative overheads between one configuration and another. Here are two techniques to compare relative overheads.

IBM has published a formula to estimate "relative" overhead of PR/SM based on the number of shared LPs (logical CPUs) and the number of shared CPs (physical CPUs). The relationship can be described as:

(# of shared LPs) / (# of shared CPs)

To show an example, consider the following. A system has 6 physical CPUs that are shared between four LPARs. One of the LPARs has a single dedicated CP. If you assign all 5 of the other LPs to each LPAR, the relative overhead would be (5 * 3)/5 = 3. Assigning only 3 LPs to each LPAR would produce (3 * 3)/5 = 1.8 or about a 60% decrease in overhead (1.8 / 3). Notice that this doesn't indicate the amount of overhead, just the relationship. This means that the least amount of overhead occurs when you assign the minimum number of LPs to each LPAR. I've found this to be very true and had indicated so in the February 91 issue as one of the tuning recommendations.

```
                                P A R T I T I O N   D A T A   R E P O R T

          MVS/ESA                  SYSTEM ID PROD          START 01/29/91          INTERVAL 04.59.988
          SP3.1.3                  RPT VERSION 4.2.1       TIME  10.00.02          CYCLE 1.000 SECONDS

MVS PARTITION NAME                 PROD

NUMBER OF CONFIGURED PARTITIONS      4

NUMBER OF PHYSICAL PROCESSORS        3

WAIT COMPLETION                     NO*

DISPATCH INTERVAL                DYNAMIC*

-------LOGICAL PARTITION DATA--------  -- LOGICAL PARTITION PROCESSOR DATA ---  ---- AVERAGE PROCESSOR UTILIZATION PERCENTAGES ----
                                       NUMBER OF   ----DISPATCH TIME DATA-----  LOGICAL PROCESSORS  ----- PHYSICAL PROCESSORS ----
  NAME     STATUS  WEIGHTS  CAPPING    LOG PRCRS    EFFECTIVE       TOTAL      EFFECTIVE   TOTAL  LPAR MGMT   EFFECTIVE    TOTAL

PROD         A       DED                   1      00.04.59.534   00.04.59.907     99.84    99.97     0.04       33.28      33.32
ESATEST      A         4        YES        2      00.00.24.224   00.00.27.294      4.03     4.54     0.34        2.69       3.03
VMDEVL       A        48        NO         2      00.05.24.812   00.05.25.594     54.13    54.26     0.08       36.09      36.17
TEST         A        48        NO         1      00.04.02.396   00.04.02 788     80.80    80.93     0.04       26.93      26.97
*PHYSICAL*                                        ------------   00.00.03.935                         0.43                  0.43
 TOTAL                                            00.14.50.966   00.14.59.518                         0.93       98.99      99.92
```

Figure 9 - RMF Monitor I Partition Data Report

Alan Sherkow, a well-known independent consultant based in Whitefish Bay, Wisconsin, provides a more detailed estimate based on his observations in client sites. His conclusion is that the number of LPARs is also a factor in PR/SM overhead, as well as the presence of dedicated LPARs. His revised formula (as presented in Enterprise Systems Journal in March 91 and a 92 SHARE presentation) is:

(1.5 * # of shared LPs) +
((# of LPARs sharing CPs) / (# of shared CPs)) +
((# of dedicated LPs) / (# of dedicated LPARs))

and would produce the following relationship from our previous example (when assigning 5 LPs per LPAR):

(1.5 * 15) + (3 / 5) + (1 / 1) = 24.1

assigning only 3 LPs each would produce:

(1.5 * 9) + (3 / 5) + (1 / 1) = 15.1

This represents a 63% decrease (15.1 / 24.1) in overhead. Again, this isn't an actual overhead, just a relationship between two different configurations. This relationship works quite well when comparing configurations on machines with similar architectures or technologies, but like the IBM estimate doesn't provide as good a forecast if you're trying to compare to different technologies. For example, there are enough architectural differences between air-cooled and water-cooled processors that you'll find other differences that will affect the overhead. Differences in the way high-speed buffers are handled also impacts the relationship.

You can use this technique to produce a report comparing different configurations. Table I shows an example of Alan's using this technique. I've used this several times and have found it to be a much closer estimate of overhead than IBM's. Alan also has a paper related to this work in the CMG 91 Proceedings.

Notice that both these techniques represent the total relative overhead on the machine. In my own studies, I've found that when you run benchmarks on the actual jobs, you'll find the overhead is apportioned to each of the LPARs unevenly. The larger LPARs will see a smaller amount of overhead and the smaller LPARs will see a larger amount of overhead, but I haven't found a good way to quantify the difference.

While we're on the subject of PR/SM overhead, I've got to make my case regarding the RMF report that provides PR/SM overhead values. This was presented in the March 91 issue on page 10. A sample report is shown in Figure 9. Several people seem to feel that this report shows all PR/SM overhead. In fact, some of the documentation on the report states that. This is wrong! The report shows the amount of time that **PR/SM could capture** as time spent doing PR/SM dispatching and LPAR management. In Figure 9, this shows a total of 0.93% PR/SM overhead. This is time that is seen as uncaptured when viewed from a single LPAR, but can actually be captured by PR/SM itself.

This reported amount in the RMF partition report is only a small part of the overhead (unless the low utilization effect (LUE) is a factor). The majority of PR/SM overhead can be found in the TCB, SRB, and uncaptured

times of each LPAR. The overhead is more due to multiprocessing than PR/SM, but it still exists and can be significant. The cause is overhead due to misses in the high speed buffer and the elongation of address spaces due to I/O elongation. The only way you can identify the scope of this overhead is to run benchmarks that quantify the change. You'll typically find different overheads depending on the type of workload. As an example, one system that I looked at showed a PR/SM overhead of 2.3% from the RMF report, 10.4% overhead in CPU time in their CICS regions and 13.6% overhead in CPU time in their batch work. The CPU overheads were determined with benchmarks. This was an environment with two LPARs, each taking about 50% of the machine.

MVS/ESA SP 4.3 ENHANCEMENTS

NEW FEATURES

MVS/ESA SP 4.3 is scheduled for March 31, 1993 and provides several new facilities that people have been asking for. If you look at the announcements, you'll see a heavy emphasis on improvements in APPC/MVS and HCD (Hardware Configuration Definition). But there are several improvements in the SMF and RMF area that are especially helpful and haven't really been stressed in the announcements. This section simply provides a preview of some of the features that will be coming.

SMF ENHANCEMENTS

This information was presented at SHARE by Bill Richardson from the SMF Development group in Poughkeepsie. Thanks to Bill (and others) for the wonderful new features! The major SMF enhancement is a new synchronization feature. Several sites (and SHARE and GUIDE requirements) had requested the ability to synchronize SMF interval recording with RMF intervals. This would allow you to compare similar periods of time. SP 4.3 provides that facility. It's moved the primary interval definition and synchronization to the SMF component so it will be available to all applications. New parameters in the SMFPRMxx member include:

INTVAL(30) indicates the default global interval value in minutes for all applications. This can be overridden in each application.

SYNCVAL(0) indicates the default global synchronization in minutes for all applications. Use of synchronization is determined by the application.

SMF interval recording can then use these values in the following way. In the SYS or SUBSYS parameter, SMF defines interval recording with the following options:

SYS(...INTERVAL(hhmmss)..)
SYS(...INTERVAL(SMF,SYNC)..)
SYS(...INTERVAL(SMF,NOSYNC)..)

Use of hhmmss provides the same facility as before SP 4.3. SMF will write interval records out at an interval that starts with the start of the step. For example, an INTERVAL(006000) would write interval records out 60 minutes after the start of the step, again at 120 minutes after the start, etc. INTERVAL(SMF) says that interval records will be written out at intervals specified by the SMF INTVAL interval and SYNC indicates the intervals should be synchronized based on the SYNCVAL instead of the step start time (NOSYNC). Using this feature and the following feature for RMF (described in RMF ENHANCEMENTS), you can now get all records out for the same time period.

This suggests that the number of records being written at one time will increase and the performance of SMF could be more of an issue. Because of this, I would highly recommend that you take the effort to increase the SMF blocksize as suggested in the March 91 issue (page 12) and increase the buffers for SMF as described in the same article prior to implementing synchronized intervals in SP 4.3.

A major benefit is the ability for any application to interrogate SMF and obtain the current interval and sync value, so all applications can be using the same periods of time. It will really simplify the job for performance analysts who need to match data from multiple subsystems. (I had originally written the prior statement to say "really simplify the performance analysts". One of my editors caught it. He thought that managers all over the country would like to accomplish that feat!)

With the synchronization facility and use of SMF's interval recording, you could easily look at a period of time (e.g. one hour of peak time) and compare the RMF activity for a performance group and associate it with all the SMF type 30 interval records (subtypes 2 and 3) to identify the jobs that compose that performance group.

Another minor change to SMF was a change to in-

```
                                        ---- Percent ----    ------- Amount --------
System Information                      CSA ECSA SQA ESQA      CSA   ECSA   SQA  ESQA
 IPL Definitions                                             2316K 3268K  940K 9312K
 Peak Allocation Values                  53   91 105   88   1228K 2960K  987K 8188K
 Average CSA to SQA Conversion           34    0             784K     0
 Average Use Summary                     53   91 105   88   1228K 2960K  986K 8188K
 Available at End of Range               47    9  78   12   1088K  308K  738K 1124K

Job Information              ELAP    -- Percent Used -    ----- Amount Used -----
Name     Act C DMN   PG ASID Time    CSA ECSA SQA ESQA      CSA   ECSA   SQA  ESQA
%MVS                                   0    0   8    5        0   4800 73760  471K
%REMAIN                                3    2   0    9    62240 72376  1024  812K
RSQA         S   5   1   24  15.1M     0    0  83    0        0    416  781K  248
*MASTER*     S   0   0    1   1.2M     8   32  11   18     197K 1052K  102K 1632K
VTMLCL       S   5   1   19   1.2M     0   26   0    0    21064  851K   384  1960
...
```

Figure 10 - RMF Monitor III STORC Display (SP 4.3)

clude field SMF30DSV in all interval records instead of only the step termination record. This is the virtual storage high water mark for the amount of space allocated to data spaces and hiperspaces for an address space. The field is still somewhat disappointing since it doesn't include authorized program use, such as VLF, but right now it's the best information we've got.

RMF ENHANCEMENTS

There are several RMF enhancements coming with SP 4.3. The major additions include the ability to use SMF synchronization, a common storage monitor, and tape mount delay statistics. This material was provided in a presentation given by Dieter König from the RMF Development lab in Böblingen, Germany. Many thanks, Dieter, for the information.

The RMF synchronization is indicated with one of the following parameters in ERBRMFxx.

SYNC(SMF)
SYNC(RMF,mmM)
NOSYNC

SYNC(SMF) is the default, which corresponds to the current RMF default of 30 minutes. The current parameter of SYNC(mmM) will be interpreted as SYNC(RMF,mmM).

I think the major enhancement to RMF is the new common storage monitor. It's a feature of RMF Monitor III that allows you to track CSA and SQA usage by address space. You may already have this ability in other online monitors, such as Boole & Babbage's CMF, Candle's Omegamon/MVS and Landmark's TMON/MVS. But this is the first time it's available for all blue sites!

There are two new screens in Monitor III as seen in Figures 10 and 11. The first screen is a STORC, Figure 10, that displays who is currently using CSA and SQA. It's sorted in descending sequence of total virtual storage frames allocated. From this screen you can see the largest users of the common areas. The second screen is a STORCR display, Figure 11, that shows how much space was used and deallocated. An "N" in the ACTIVE column indicates that the job completed and left some storage. Unfortunately, RMF doesn't allow you to force it to be freed as with some of the other monitors, but at least it tells you who did it!

Another enhancement is the addition of tape mount delay times. The information will be available in the RMF Monitor I Device Activity report, Figure 12. The Monitor I data now includes three new fields for tape

```
                                                  Amount of Common Storage
                                Job Ended         Not Released at End of Job
Jobname   JES-ID      Date        Time             CSA   ECSA   SQA   ESQA

%REMAIN                                          62240 72376 13312  820K
RSQA      STC00031  03/24/92  10.42.31             0     0     0  820K
CATALOG             03/24/92  10.13.15         62240 11024   912  1848
IRRDPTAB  STC00004  03/24/92  10.14.42             0 30120     0     0
...
```

Figure 11 - RMF Monitor III STORCR Display (SP 4.3)

devices:

NUMBER OF MOUNTS
AVG MOUNT TIME (hh:mm:ss)
TIME DEVICE ALLOC (hh:mm:ss)

From this information you'll be able to determine the average mount times for devices for any interval during the day. The number of mounts can be used to determine when the highest volumes occurred. The total time the device was allocated during the interval can be used by capacity planners to determine when, and if, new drives need to be added (including the time of day they're needed). The asterisk at the start of the NUMBER OF MOUNTS field occurs whenever there is a mount pending condition at the start of the interval. An asterisk at the end of the field indicates a mount pending condition existed at the end of the interval.

There are several other RMF enhancements with SP 4.3 including:

new exceptions (such as SLIP trap alert, common storage usage, transaction response time, device resource utilization, and report performance group data);

the ability to define new groups of jobs or performance groups as an installation needs;

a GROUP Response Time report;

several report enhancements (like the ability to define which columns are to be displayed and the addition of TCB and SRB times and device I/O activity and response time);

a data compression and decompression facility that can save up to 60% on VSAM space for Monitor III data. The compression facility was needed because the common storage and group response data doubled the size of the Monitor III data.

There will be more on these new facilities as SP 4.3 becomes available. This is a pretty exciting release of RMF and I'm looking forward to using it!

SPREADSHEET - a kind of program that lets you sit at your desk and ask all kinds of neat "what if?" questions and generate thousands of numbers instead of actually working.

Dave Barry

Q & A

Q. I've seen some terrific benefits on jobs by using additional buffers for QSAM files as you mentioned in the July 91 issue. Is there a limit to the number of buffers I can use? Also, there are some jobs that don't show any significant difference. Can you explain why?

A. There is a limit to the number of usable buffers. First of all, if you don't specify buffers, SAM (Sequential Access Method) will use a default of 5 buffers for QSAM and 2 buffers for VSAM. If you specify buffers, SAM has a DASD limit of 31 buffers or 240K (whichever is smaller) for a physical I/O. That means a file with a blocksize of 24K could only use 10 buffers as a maximum, but a blocksize of 4K could use 60 buffers. This is the limit for a single channel program or SSCH.

If you want to overlap processing time and I/O time, specify double the maximum number of buffers. In the previous examples, I'd use 20 24K buffers or 120 4K buffers to avoid the "galloping" effect of having two few buffers. Remember that additional buffers affect the amount of the job's virtual storage and possibly central storage. Also, the buffers for a single I/O (e.g. 240K) are fixed below the line for the duration of the I/O. If you're storage constrained due to too many fixed frames below the line, this may be a problem. See the later Q & A on fixed frames below the 16Mb line.

As to the second question, I may know what the problem is. If files are allocated in track allocation rather than cylinder and are not accessed with ECKD CCWs, the SSCH (channel program) will be limited to a single track I/O. See the following Q & A for more information on this topic.

Q. Is there a performance impact between using track and cylinder allocation?

A. It depends! If the data set is accessed with ECKD CCWs and is behind a controller that supports ECKD CCWs, there is no difference in performance. These controllers (such as 3880/13 and 3990s) provide a facility called DEFINE EXTENT. This means that the extent is known and the end of extent doesn't have to be checked at the end of every track.

If the dataset is not behind this type of controller, logic is needed to check the end of extent at the end of every track if the dataset was allocated in tracks and at the end of every cylinder if the dataset was allocated in

DEV NUM	DEVICE TYPE	VOLUME SERIAL	LCU	DEVICE ACTIVITY RATE	AVG RESP TIME	AVG IOSQ TIME	AVG DPB DLY	AVG CUB DLY	AVG DB DLY	AVG PEND TIME	AVG DISC TIME	AVG CONN TIME	% DEV CONN	% DEV UTIL	% DEV RESV	NUMBER OF MOUNTS	AVG MOUNT TIME	TIME DEVICE ALLOC
180	3480		009	0.000	0	0		0.0	0.0	0.0	0.0	0.0	0.00	0.00	0.00	* 2*	12	36
180	3480		009	0.000	0	0		0.0	0.0	0.0	0.0	0.0	0.00	0.00	0.00	1	1:24	1:48
		LCU	009	0.000	0	0		0.0	0.0	0.0	0.0	0.0	0.00	0.00	0.00	3	36	1:12

Figure 12 - RMF Monitor I Tape Device Activity Report (SP 4.3)

cylinders. Thus, each I/O can be no more than one track if the file was allocated in tracks, regardless of the number of buffers defined. It's because of this that cylinder allocation provides better performance than track allocation. By the way, you should ignore the entry on IBMLINK that says this is an old "folk tale". The good news is that current software releases and newer controllers support ECKD so the problem is disappearing and track allocation doesn't cause as many problems as it did.

Q. I keep getting a message that says I've exceeded my limit of fixed frames below the 16Mb line. What's causing it, what can I do about it?

A. This is similar to the Q & A from the August 92 issue, but I've been seeing several occurrences of this. Let me start with an explanation of why it occurs.

There are several control blocks and data areas that must reside below the 16Mb line because they are accessed using 24-bit addressing. Associated with this are CCW type 0 channel programs. These older channel programs require the channel program and (sometimes) the buffers be located below the line. Unfortunately, QSAM is still one of the access methods that uses CCWs of type 0, although the buffers can live above the line. Buffers are especially noticeable below the line because they are fixed in storage for the duration of the I/O.

Because the central storage below 16Mb is limited (in comparison to the storage above 16Mb), SRM monitors its use. If you run out of pageable frames below 16Mb, you'll probably crash the system! The IEAOPTxx parmlib member has two parameters that relate to the use of fixed storage frames below the 16Mb line. The first parameter, MCCFXEPR=92 (92 is the default), indicates the percent of storage that is fixed below the 16Mb line. If over 92% of storage is fixed, SRM will notify the operator. A similar parameter, RCCFXET=(82,88), is used to control MPL adjustment. If the percent of fixed frames goes above 88%, SRM will lower a domain's MPL (and possibly cause a unilateral swap out). SRM can only increase a domain's MPL if all other resources are underutilized AND the percent of fixed frames below 16Mb is less than 82%. Notice the relationship between the two parameters. This high value of RCCFXET should be less than the MCCFXEPR value to allow SRM to prevent the actual shortage. That is, SRM should swap out a user prior to hitting the actual limit.

Of course, one way to decrease the number of SRM interruptions is to increase these values, but I wouldn't. They're pretty high as it is. The real solution is to identify the major fixed storage users and limit their impact. A monitor, such as the RMF Monitor II ARD screen shown in Figure 13, can identify these fixed storage users. The column, FX BLW, shows the number of frames that are fixed below the 16Mb boundary. Look for the larger users and determine some way to restrict them. If they are all the same type of program, such as IEBCOPYs or sorts, then find a way to place them in a separate performance group and domain and limit them with a domain constraint.

DFSORT

There are some standard users that fix many frames and you should be aware of them. The sort programs, for example, can use a large amount of fixed storage for a long period of time. DFSORT, for example, has two defaults that will cause a problem. One of the parameters, EXCPVR, indicates how channel programs are to be defined. EXCPVR=ALL indicates that all I/O buffers for input, output and sort work files should be defined as REAL storage frames, not virtual, and fixed in storage for the duration of the sort. Use of EXCPVR=ALL reduces the CPU time of the sort, but requires more fixed frames for the duration of the sort (many of those below the line). EXCPVR=NOWRK indicates that sort should use these fixed buffers for the input and output files, but not the sort work files. This provides some of the CPU improvement without fixing as many buffers. EXCPVR=NONE states that the buffers will be fixed by the operating system just for the duration of the I/O, but will take more CPU time. Certainly if the number of fixed frames is not causing any restraint, the default of EXCPVR=ALL is the most efficient option to use.

Since many of these buffers are below the 16Mb line (as defined by parameter, MAXLIM, and type of sort), use of EXCPVR=ALL may increase the problem of fixed frames below 16Mb. The parameter, MAXLIM, with a default of 1Mb indicates the amount of storage below the 16Mb line that can be used for buffers. MAXLIM applies to the following uses of DFSORT that exclusively use buffers below the 16Mb line:

. COPY
. MERGE
. E15 and E35 exits
. BSAM

If each sort takes 1Mb of fixed frames below the line, you'd be hard-pressed to get more than a few sorts using these facilities running concurrently. The solution is to either reduce the amount of buffer space or use virtual pages instead of real (or fixed pages). I'd recommend that you first consider reducing the amount of buffer space, since the EXCPVR=ALL provides significant CPU time reductions. As an option, you could use the sort exit, ICEIXIT, to change the the option to EXCPVR=NONE when using any of the facilities that require buffers below the line. When using MAXLIM, remember that the REGION parameter must be large enough to accommodate the storage requested.

Another parameter, TMAXLIM, defines the total virtual storage both above and below the 16Mb line. 4M is the default and includes MAXLIM. Using the normal defaults, DFSORT would use 1Mb below the line and 3 Mb above the line. If fixed frames below 16Mb are a problem, you could decrease MAXLIM to 500K and leave TMAXLIM as it is. Therefore, 3.5Mb of pages would be fixed above the line if you specify EXCPVR=ALL or EXCPVR=NOWRK. This is a lot of frames! There's a recommendation in the DFSORT Tuning Guide, GG24-3294, that says you get additional performance benefits for DFSORT by increasing TMAXLIM to 8Mb. If you specify EXCPVR=ALL, that's similar to varying 8Mb of central storage offline! Wow! Unless the machine is empty or you have lots and lots and lots of storage, I can't support this recommendation.

In a non-storage constrained environment, you can leave the defaults, but beware of running too many sorts at one time. If you experience any fixed frame shortages, you should consider reducing MAXLIM (if constraint is below the line), reducing TMAXLIM (if constraint is below and above the line), changing EXCPVR from ALL to NOWRK (to reduce the fixing slightly) or NONE (to eliminate the long-term fixing), or limiting the number of concurrent sorts.

IEBCOPY

Another offender of using too many fixed frames below the line is IEBCOPY. There's a common recommendation to use "WORK=4" on the parameter for IEBCOPY. This indicates that IEBCOPY should use 4Mb for buffers. Running too many IEBCOPYs at one time, with WORK=4, will deplete the number of frames below 16Mb. Either restrict the number of IEBCOPY jobs or reduce the WORK=4 to WORK=2 or WORK=1 (the default).

BUFFERS

You might also see a problem with too many fixed frames below the line if you're using a lot of buffers in multiple jobs. In general, the buffers are only fixed for a very small period of time (20 ms for example), and are then changed to pageable. The problem comes, however, because the page is marked as needing to be fixed below 16Mb. This means that the RSM will attempt to page it in below 16Mb during future page-ins. Thus, these pages contribute to the storage used below 16Mb. This means that if you're experiencing a shortage of frames below 16Mb, then you should reduce the large number of buffer specifications.

We're seeing many more instances of this problem then ever before. I think it's because the speed of the machines and the multi-processors are simply allowing more concurrent jobs in the system and many more jobs are requiring very large amounts of central storage below the 16Mb line.

```
F                          MIG=319 CPU= 95 UIC= 29 PDT= 108 DPR=116   ARD      T

09:03:18 DEV    FX  LS NLS X SRM   TCB     CPU     PIN  EXCP SWAP LPA CSA NV  VIO
JOBNAME  CONN  BEL  QA  QA M ABS   TIME    TIME    RT   RATE RATE RT  RT  RT  RT

*MASTER*  2377   0  32   0    .00  113.72  450.28  0.8  1.70 0.00 .01 .00 .08 .00
PCAUTH    .0000  2  24   0 X .00    0.02    0.02  0.0  0.00 0.00 .00 .00 .00 .00
TRACE     .0000  0 123   1 X .00    0.01    0.02  0.0  0.00 0.00 .00 .00 .00 .00
GRS       .0000  0  20   5 X .00    0.04    0.07  0.0  0.00 0.00 .00 .00 .00 .00
CONSOLE  16.25   0  15   0 X .00   28.01   32.42  0.0  0.45 0.00 .00 .00 .03 .00
ALLOCAS   .0000  0  17   0 X .00    0.03    0.03  0.3  0.00 0.00 .00 .00 .34 .00
LLA      41.80   0  18   0 X .00    8.25   11.26  0.3 10.06 0.00 .00 .00 .29 .00
SMF      43.35   0  17   0 X .00    0.22    5.56  0.0  0.00 0.00 .00 .00 .02 .00
CATALOG  911.2   0  49   0 X .00  602.01  629.04  0.1  0.11 0.00 .00 .00 .07 .00
JES2     863.4  37  34  39    .00  288.78  364.88  7.0 ----- 0.00 .00 .02 .26 .00
CHERYL   773.0 461  21 461    .00  574.34  630.04  0.1 10.02 0.00 .00 .00 .01 .00
```

Figure 13 - RMF Monitor II ARD Display

```
                                  ** PROGRAM USAGE BY PROCEDURE **

MODULE - WR00                                SECTION - WR160
                                             SOURCE LANGUAGE-ANS COBOL VS                                    VC23X18

  LINE      PROCEDURE              STARTING     INTERVAL    CPU TIME PERCENT    CPU TIME HISTOGRAM  MARGIN OF ERROR:  1.41%
 NUMBER       NAME                 LOCATION     LENGTH      SOLO     TOTAL      .00     6.50     13.00     19.50     26.00

   4759  END-BRANCH-EXIT          00B4CA            6        .00       .00      .
   4760  BEGIN                    00B4D0          106        .00       .00      .
   4769  BEGIN-2                  00B53A          350        .00       .00      .
   4776      MOVE                 00B698          120        .01       .01      .
   4779  READE                    00B710           68        .11       .11      .
   4782      PERFORM              00B754           66      14.41     14.41      ......................
   4784      ADD                  00B796           56        .07       .07      .
   4795  EXTEND-BILL              00B7CE          670        .00       .00      .
   4809  PRINT-TOTAL-RECS         00BA6C          604        .00       .00      .
   4828  CLEAR-PRICE-TABLE        00BCC8           36      24.63     24.63      ....................................
   4830  SAT-AND-LAST-ROUTINE     00BCEC          736        .00       .00      .
                                                                  -----     -----
   SECTION   WR160    TOTALS                                     39.23     39.23      .
```

Figure 14 - STROBE - Sample Program Usage Report

APARS

One last item to check is any outstanding APARs. Here are some APARs (some with related PTFs) that relate to the fixed frame problem. Check them out to see if they're applicable to your system:

OY17933 - RCEBELFX not accurate
OY30733 - RCEBELFX negative, causes spin loops and IRA500E & IRRA400E messages
OY29750 - Fixed frames
OY32514 - Pages getmained with LOC=(ANY,ANY) but fixed below 16Mb
OZ70029 - Very old

PRODUCT HIGHLIGHT

This section provides some information on products that subscribers have found to be especially helpful. This is not a recommendation to buy the products, but simply a suggestion to look at them if you need a specific facility. I have not done a market analysis and reviewed other similar packages - I'm just passing on subscribers recommendations for products that I'm familiar with and have used successfully. If you have the July 91 issue, I had several suggestions on page 14 for anyone evaluating software packages.

PRODUCT HIGHLIGHT: STROBE

Strobe is a program from Programart that analyzes jobs while they're executing and produces reports that help you determine where to tune the program. It shows you delays for I/Os, system services, operator requests, and several other delays. It also shows you where the majority of the CPU time is occurring so you can analyze the program for inefficiencies. Figures 14 and 15 show sample reports from a typical STROBE run.

Figure 14, for example, shows an analysis of the execution job in a procedure. You can see that almost 25% of the CPU time was spent in a 36-byte area called CLEAR-PRICE-TABLE. Maybe someone could figure out a more efficient method of clearing the table! Figure 15 shows the percent of elapsed time by program or files. In this example, it shows that DFHSIP spent 9.14% of the run time using the CPU and the file referenced by ddname, LXZ20SD, spent 18.04% of the run time using the CPU and 14.73% of the run time being serviced by the I/O subsystem.

There are other reports as well showing dataset activity, dataset characteristics, DASD usage by cylinder and many others. The reason that I'm mentioning this product is that several subscribers have called to say that they've found significant savings by using Strobe on their larger applications. Most say the the product pays for itself with just a few applications.

```
                                  ** RESOURCE DEMAND DISTRIBUTION **

                          ---- PERCENT OF RUN TIME -----   ---- PERCENT OF RUN TIME SPENT -----  CUMULATIVE PROCEDURES
TASK OR     RESOURCE      SERVICED   SERVICED   SERVICED    SOLO     SOLO      SOLO     CAUSING      SOLO     CAUSING
 DDNAME                   BY CPU     BY I/O   BY EITHER   IN CPU   IN I/O   IN EITHER  CPU WAIT     TIME     CPU WAIT

DFHSIP      CPU             9.14       .00        9.14     8.73      .00       8.73     52.83        8.73      52.83
LXZ02SD     3380           18.04     14.73       32.75    18.02    13.99      32.03     13.92       40.76      66.75
DOTFILMT    3380            3.14       .27        3.41     2.81      .27       3.08      1.58       43.84      68.33
```

Figure 15 - STROBE - Sample Resource Demand Distribution

Information on Strobe can be obtained from Programart Corporation, (617) 661-3020 or (508) 468-1155. They've also just announced a new product called APMPower, an OS/2-based product to allow you to download the Strobe data to your PC and lead you through a step-by-step analysis of your application.

PRODUCT HIGHLIGHT: VSAM I/O PLUS

Softworks, Inc. has been gaining a lot of loyal customers with their family of VSAM management products. Their VSAM I/O Plus is a product that has been providing LSR processing for batch VSAM files for quite awhile now. This is also available for MVS/SP sites who can't get the benefits of batch LSR. It supports hiperspaces and differs from BLSR in that it doesn't require any JCL changes. The savings from their customers show reductions of 90% in CPU and EXCPs. I've received several calls from subscribers who have been very impressed with their savings. The product starts at about $13,500, so it could be fairly easy to justify based on the savings.

Softworks is in Clinton, Maryland at 800-638-9254 or 301 868-6740 (fax).

PRODUCT HIGHLIGHT: PMO & QUICK-FETCH

Legent provides two products that compete with and complement LLA and VLF. Most people who have compared PMO & Quick-Fetch to VLF & LLA choose Legent's products for their flexibility.

PMO, Program Management Optimizer, manages library directories dynamically and reduces directory searches. One of its primary benefits over LLA is that it automatically identifies updates even across systems. It can either replace LLA or support LLA and selectively refresh LLA when an update occurs. Another advantage of PMO over LLA is that it will dynamically determine the most applicable libraries to manage.

Quick-Fetch is a complementary program that keep high used modules in storage in order to reduce the I/O for loading. It provides a similar function to VLF's support of LLA. It can automatically identify when updates occur and so can always find the most current version. When used with PMO, it can also identify cross-system updates.

Both products provide online reports or batch reports showing module and library activity. Using the reports from PMO, for example, you could identify good candidates for adding to link pack area. You can contact Legent at 800-676-5468 x92.

REQUESTED PRODUCT

The following product has been asked for by subscribers and I've received information from the vendor. I have no experience with the product but wanted to respond to people who had been asking for the capability. If you know of other products that perform similar functions, please let me know.

DYNA-STEP is a product to dynamically allocate TSO steplibs. Since use of steplibs will degrade the performance of any TSO user (see my January 91 issue, which is still free), steplibs should only be used when necessary. This product allows a user to allocate a steplib when it's needed and then free it up again to improve performance. DYNA-STEP starts at $8,200 is marketed by Tone Software in Anaheim, California, (800) 833-8663, (714) 991-9460, or fax (714) 991-1831.

FEEDBACK

FIXED DP FOR IMS

From Paul Gersch, AAA Michigan: "In the January 92 issue, page 20, the feedback from Steve Samson said that using MTTW for IMS MPRs and BMPs would give higher priority to I/O bound transactions. Since most people are running IMS with a DLI region, an I/O bound transaction would still look CPU bound to the SRM because the I/O is executed by the DLI region. Because of this, I recommend using fixed for IMS message regions." I tend to agree with Michael. The problem I've seen is that transactions within an MPR don't all look the same - some are I/O-bound and some are CPU-bound and few are consistent. Therefore, use of MTTW would not be able to predict the next transaction and could produce the opposite result. That's why I always use a fixed dispatch priority for any online system.

SRM APAR DESCRIBES STCS

Ed Kahler from Motorola, Inc. suggested that shops converting from SP 3.1.3 to SP 4.2 look at APAR OY52039. I agree and think that everyone should take a look at it. This APAR provides additional documentation that will be added to the Init & Tuning Guide (GC28-1634) regarding the ability to assign different performance groups, dispatch priority, and storage isolation values to system address spaces. In summary, the documentation update includes the following description of address spaces and performance group assignments for

privileged programs and system address spaces. I've added my own points of view in italic.

PRIVILEGED

SRM automatically assigns all privileged work to performance group zero. Any work in performance group zero is assigned to domain zero. Privileged work is defined by the user in the program properties table (PPT) via the SCHEDxx member of parmlib. Any programs with the keyword of PRIV in SCHEDxx are considered to be privileged and will be non-swappable except for long wait swaps and will run in the privileged dispatch priority. This priority is assigned by the user in the PVLDP= parameter of the IPS (IEAIPSxx in parmlib). Privileged programs cannot be assigned to a non-zero PGN.

SYSTEM PRIVILEGED

Some MVS components have separate programs that are treated as privileged even though they are not in the PPT. These address spaces are assigned to performance group zero and a high dispatch priority, X'FF'. Examples of these address spaces are:

MASTER	Master scheduler
DUMPSRV	Dumping services
CATALOG	Catalog address space
SMF	System Management Facility
GRS	Global Resource Serialization

GRS is an exception in this list because it can be assigned to a non-zero PGN by the user. The users can't override the GRS dispatch priority of X'FF' on the new PGN, but they can override the system default storage isolation. *GRS is given a forced storage isolation of PWSS=(32000,*) which indicates that pages won't be stolen from GRS. The reason behind this was based on the use of GRS in a global environment where other systems could be delayed as well as the host system. In sites that don't use global GRS, you may want to reduce that storage isolation by placing GRS in a separate performance group. A value of PWSS=(0,*) on the PGN will cause GRS to be treated like all other address spaces and I highly recommend this for non-global GRS sites.*

HIGH DPRTY SYSTEM ADDRESS SPACES

Some address spaces are assigned to the standard STC performance group and can be assigned to any PGN in the IPS, but will always have a high dispatch priority OF X'FF'. These address spaces include:

CONSOLE	Communications (operator interface)
RASP	Real Storage Manager (RSM)
XCFAS	Cross-system Coupling Facility (XCF)
IOSAS	Input/output supervisor (IOS)
SMXC	DF/SMS address space for PDSEs

You could assign any of these to their own PGN by assigning TRXNAME to a PGN, such as:

```
SUBSYS=STC,PGN=5
    TRXNAME=CONSOLE,PGN=25
    TRXNAME=RASP,PGN=26
```

Just remember that the DP= parameter on these PGNs will not have any effect.

OTHER SYSTEM ADDRESS SPACES

Other system address spaces will be treated like ANY other workloads in the system. You can specify performance groups, storage isolation, and dispatch priorities for them. They include:

PCAUTH	Cross memory services
TRACE	System trace
ALLOCAS	Allocation address space
LLA	Library Lookaside
VLF	Virtual Lookaside

The APAR mentions that setting a high priority for LLA and VLF will reduce the time to initialize and refresh LLA and VLF, but warns against running them at a higher dispatch priority than the IMS control region or CICS TOR (terminal owning region). *I disagree with this last recommendation. First of all, the only time that this is of concern is on a uni-processor, because LLA or VLF could only occupy one CPU at a time, thus allowing IMS or CICS to execute on another CPU. Primarily I disagree with it because IMS and CICS can use LLA and VLF as a server and you wouldn't want them to wait for a refresh, for example. Therefore, except on a uni-processor, I would run LLA and VLF at a very high dispatch priority. See more comments in the VLF OVERVIEW and TUNING LLA & VLF.*

CHERYL'S RECOMMENDATION

I would recommend that you define a single unique performance group for all MVS system address spaces and assign the TRXNAMEs in the ICS. Give this performance group a very high dispatch priority. It probably won't consume a large amount of service because most service is reported in the caller's address space and not in the system address space. To determine the overhead of MVS, you can add performance group zero and this new PGN. Then assign all other STCs to a lower priority below any online systems. This is very beneficial to those sites that run with 50 to 150 started tasks all day long and don't know what type of jobs run as STCs.

A sample ICS and IPS for SP 4.2 might look like:

```
SUBSYS=STC,PGN=3,RPGN=300
    TRXNAME=ALLOCAS,PGN=31
    TRXNAME=CONSOLE,PGN=31
    TRXNAME=GRS,PGN=31
    TRXNAME=IOSAS,PGN=31
    TRXNAME=LLA,PGN=31
    TRXNAME=PCAUTH,PGN=31
    TRXNAME=RASP,PGN=31
    TRXNAME=SMXC,PGN=31
    TRXNAME=TRACE,PGN=31
    TRXNAME=VLF,PGN=31
    TRXNAME=XCFAS,PGN=31

DMN=3,CNSTR=(10,25),..  /* BATCH */
DMN=5,CNSTR=(999,999)  /* SYSTEM */

PGN=3,(DMN=3,DP=M6,...)
PGN=31,(DMN=5,DP=F93,...)
```

To determine the total resources used by MVS address spaces, add PGN=0 and PGN=31 resources. To see which dispatch priorities and performance groups are are overridden, look at any monitor that shows performance groups and dispatch priorities. Figure 1 shows an RMF Monitor II ASD report showing the performance group and the dispatch priorities (DP PR).

Cheryl Watson's TUNING Letter, September/October, 1992, Vol. 2, Nos. 8, 9

Published monthly by Watson & Walker, Inc.

Publisher: Tom Walker
Editor: Cheryl Watson

PURPOSE: To provide practical MVS tuning knowledge and techniques which achieve savings by reducing costs or improving response times.

SUBSCRIPTION RATES: $295 per year (12 issues). (1993 is 6 issues.) Outside North America, $345 per year. Payment may be by check drawn on a bank located in the U.S., money order, credit card, or wire transfer. Issues are mailed via first class mail. Back issues are available for $25 each ($50 for double issues and 1993 back issues). Send subscriptions, inquiries, address changes, and all correspondence to: Watson & Walker, Inc., 814 Sandringham Lane, Lutz, FL 33549-6801, USA. Tel: (800) 553-4562, (813) 949-3673, Fax: (813) 949-3674

PRAISE FROM OUR READERS:

"Excellent! Has saved us countless dollars!"
"A great tool for training my staff."
"The best periodical I receive...consistently indicates the areas to be monitored after a change."
"The best tuning tool available."
"...should be part of every system programmer's tool box."
"Terrific - it's useful for both novice and experienced performance analysts."
"News you can use."
"Everyone ... looks forward to seeing it."
"A bargain at twice the price!"
"Great, practical, no fluff."
"I especially like the recommendations ..."
"Best thing I have ever come across."
"Solid, down-to-earth advice, presented clearly and logically."

1991 ISSUES

January	Tuning TSO (free issue)
February	PR/SM, Central Storage
March	Central Storage
April	Tuning DASD
May	Tuning Paging & Swapping
June	MVS/ESA Measurements
July	Reducing I/O
August	Reducing CPU
September	Tuning VSAM
October	Designing a new IPS
November	More on IPS Design
December	IPS Design

1992 ISSUES

January	Expanded Storage
February	Service Levels (Online, TSO)
April	Service Levels (Batch, Other, Negotiations)
May	Variability
June	Benchmarking
July	Miscellaneous Tuning Tips
August	Batch Application Tuning I
Sep/Oct	Batch Application Tuning II, VLF, LLA, Batch LSR, Misc. Tuning Tips

SUBSCRIBE!

TO SUBSCRIBE TO CHERYL WATSON'S TUNING LETTER: Call! Or mail or fax this form to Watson & Walker, 814 Sandringham Lane, Lutz, FL 33549-6801, USA. Tel: 800-553-4562, 813-949-3673, Fax: 813-949-3674
NO RISK: You may cancel your subscription at any time and receive a pro-rated refund.

☐ **12 monthly issues - $295 (Outside North America: $345). Checks must be drawn on a U.S. bank. Contact us if you need wire instructions.**

☐ **Check enclosed** ☐ **Bill me** ☐ **P. O.** _____ **Use my** ☐ **VISA** ☐ **MasterCard**
Card Number_____ **Exp. Date** _____ **Signature** _____

BACK ISSUES: $25.00 EACH ($50.00 for 1993 issues) January 1991, our first issue, is free.
Back issues (month and year desired): _____

Be sure to include your FLOOR, ROOM NUMBER, or MAIL STOP where appropriate. PLEASE PRINT.

Name

Company/Mail Stop

Address

City, State, Zip

Country _____ **Telephone** _____ **Fax** _____

This contains updates to the September/October 1992 issue and should be filed at the back of that issue. Please replace any previous update sheets. The date of this update is: **6/30/93.**

MORE ON LLA & VLF (update on 12/92)

On page 17, I described VLF objects as being anything, such as "modules, CLISTs, EXECs, catalog records, ISPF panels, ISPF messages, or RACF group records." This implied that ISPF panels and messages were supported by VLF when in fact they aren't currently supported. They are supported by LLA. That comment generated several questions regarding ISPF directories and LLA and VLF, so here's a short summary.

The ISPF panel and message libraries can definitely be added to LLA control by including them in the CSVLLAxx parameter member. Any directory searches for these libraries will be satisfied by LLA and you will reduce I/O delays for directory searches. They're very good candidates for LLA control due to their high activity. LLA, however, is smart enough to bypass VLF management of these members since VLF is only used by LLA for load modules.

Actually, any library that is accessed via a BLDL access can benefit from LLA, but only load modules will be passed from LLA to VLF control.

LEGENT PHONE NUMBER (update on 12/92)

Here's a better phone number for Legent rather than the one I listed in the Product Highlight of PMO and Quick-Fetch: 800-676-5468 x92.

SUBSCRIBER FEEDBACK (update on 1/93)

Paul Gersch of AAA Michigan (whose name I totally got wrong in the FEEDBACK section - sorry, Paul!) had some additional comments on the September issue. I really appreciate this kind of feedback - please keep it coming!

1) VSAM BLSR - If the VSAM file used to be an ISAM file and was converted to VSAM with no program changes, BLSR won't work (resulted in a loop). It's one of the situations that IBM doesn't support.

2) HBAID - It uses SMF as input, not GTF like I mentioned on page 13. VLFAID, VLBPAA, OLBPAA do use GTF.

3) Quickfetch - Paul compared Quickfetch and VLF in an IMS environment. He found that Quickfetch will produce a higher hit ratio with the same amount of memory or the same ratio if VLF is given more memory (his experience was using 7M for Quickfetch and 12M for VLF). VLF took less CPU time, but Quickfetch produced better reports. He has a paper in the CMG 91 Proceedings on his study.

Bill Fairchild, one of my favorite gurus at Landmark Systems updated me with more info on the Q & A regarding track allocation:

1) On page 31, I said that track allocation makes no difference in performance if the data set is behind a cache controller. As Bill points out, that's not always true. What matters is the type of channel program. Most cache controllers support ECKD CCWs, and use of ECKD CCWs will not effect performance. If a program uses CKD CCWs (as in older, home-grown programs), it will experience a delay due to track allocation. This is seldom a problem since almost all applications now use ECKD CCWs. There's even a front-end to EXCP that converts CKD channel programs to ECKD if the device is on an ECKD-capable controller.

2) On page 32, I also stated that older channel programs (CCW format 0) requires both the channel program and the buffers be located below the line. Bill indicates that the statement isn't totally true. The programs must reside below the 16M line, but the buffers can be above the line if the CCWs use indirect addressing. This feature allows buffers to reside above 16M. It just depends on how it's coded.

MORE ON IRA500E (update on 6/30/93)

Chen-Yu Hu from C. R. Bard, Inc. found another APAR that addresses the fixed frame shortage problem as noted by message IRA500E. This topic was addressed in the August 1992 issue on page 26 and the Sep/Oct 92 issue on pages 32 to 34. They were receiving message IRA500E and RMFMON and TMON/MVS didn't show any storage shortages. From IBMLINK, they found APAR OY57286. After applying PTF UY84041, the problem was solved.