SOURCES OF SAS SYSTEM INFORMATION

IN THE MVS ENVIRONMENT

Michael A. Raithel, Computer Associates International, Inc.

INTRODUCTION

The lexicon of measurement, defined by the twin terms "performance" and "efficiency", surrounds us in everyday life. We are familiar with discussions about athletic performance, automotive performance, stock performance, industry performance, and job performance. We read and hear about fuel efficiency, energy efficiency, and government efficiency. We are comfortable with these terms and understand the comparisons they connote.

The literature of data processing also abounds with the terms of measurement. Industry books, articles, and papers discussing computer performance and efficiency are published every year. Some of these publications provide methodologies and examples of how to write efficient code. Others discuss how programmers can get the best performance from a program or the greatest throughput from a computer. The importance and need for this type of information is obvious and well understood by the data processing community.

Implicit in the discussions of performance and efficiency is the fact that measurements must be taken. Measurement is the only way the central questions of "Better performance than what?" and "More efficient than what?" can be answered. A particular program cannot be proven to be more efficient than another unless some basic form of measurement has been applied to both. Specific metrics must be used to measure and compare the performance and efficiency of programs. But what metrics should be used, and where can they be found in the SAS System?

This paper addresses the sources of SAS System performance information in the MVS environment. The paper begins with an overview of the four part tuning cycle: measurement, comparison, deduction, and modification. It continues with a discussion of the most important performance/efficiency metrics for the SAS System under MVS. Then it examines two sources of SAS program processing statistics. The paper discusses information sources in the CONTENTS procedure and in the SAS systems options. This paper refers exclusively to release 6.06 of the SAS System in the MVS environment.

THE TUNING CYCLE

Some SAS programmers might consider program performance/efficiency issues to be an unwelcome diversion from the job of supplying management with relevant business information. However, in these days of increased fiscal tightness, of corporate downsizing, of data center outsourcing, and of business re-engineering, most programmers recognize the benefits of reducing program overhead. They use good programming practices, effective data set options, and practical systems options to write the most efficient code possible. Then they monitor the resource consumption of their applications and tune them whenever possible.

The SAS programmer tuning an application program engages in a four part cycle composed of measurement, comparison, deduction, and modification. First, the programmer **measures** resource consumption by observing the values of key metrics that contain processing statistics from a program that has executed. Next, the programmer **compares** the processing statistic values against values from earlier runs of the program, or against values from other programs, or against expected values. **Deductions** are then made about the effects program changes have had (or will have) on program performance and efficiency. Then relevant **modifications** are made either to the -

program, to SAS data sets, or to systems options, and the program is rerun. The process is repeated until the programmer is satisfied the program is optimally tuned.

Accepted SAS programming techniques generally yield maximum efficiency for programs under development. Program performance should be measured during the development, unit testing, and parallel systems testing phases of the project lifecycle. Metric values should be compared with those of similar programs and deductions about program efficiency should be formed. Modifications to the program should be completed, if necessary, and the metrics should again be observed. This process ensures that the completed production program not only fulfills its business function, but executes with the lowest amount of overhead.

Existing production programs should have their processing statistics measured and compared to previous runs. When changes that improve program efficiency are deduced they should be tested. If the changes result in lowering the values of important processing metrics, the production program should be modified. This process should be repeated periodically.

The cycle of measurement, comparison, deduction, and modification is central to tuning programs to their greatest efficiency. The tuning cycle should be implemented during the development of new programs and as an audit of existing programs. In both cases, it is important to understand which metrics should be used in measuring the performance of SAS programs.

PERFORMANCE METRICS

There are three important metrics used to measure the efficiency of SAS programs in the MVS environment. They are: CPU time, EXCP count (I/O's), and Memory utilization. These metrics are tied together in a relationship of inverse proportions such that when one is increased the others generally decrease. CPU time, EXCP count, and Memory utilization should be used in the tuning cycle discussed earlier. They provide the framework from which to judge program efficiency issues.

CPU time is the amount of time a central processor spends executing the instructions of a program. Central processors are the "engines" of a computer. Each central processor can only execute the instructions of a single program at any given instant. Since mainframe computers contain from one to six central processors, there is a finite amount of available CPU time. While a particular program is being executed by a central processor, other programs (or started tasks or TSO sessions) are waiting for their turn to use the processor. Thus, the more CPU time a program consumes, the longer other programs have to wait.

EXCP count is the number of Input/Output operations completed by a program. An EXCP either retreives data from or writes data to DASD or tape. EXCP's take milliseconds to complete and during that time the program is put into a wait state. The greater the number of EXCP's a program issues, the longer it takes to execute, and the more elapsed (wall clock) time it takes to complete. EXCP's are the slowest events in the execution of a program. Reducing EXCP count results in a more efficient program that executes faster.

Memory utilization is the amount of "private area" virtual storage used by a program and its data. The private area is the portion of virtual storage set aside for the instructions and data of an executing program. Each batch job or online session has its own private area which is bounded by the system region size. When a program first begins to execute in MVS, it is loaded into the private area of its address space in virtual storage. Additional portions of the private area are used to store the data (SAS data set pages, Program Data Vector, etc.) processed by the program. There is a finite amount of private area memory available to executing programs. Using too little memory can cause a program to increase its EXCP count, because fewer blocks of data can be retained in memory. Using too much memory can cause a program to have an unnecessarily high paging rate as unreferenced data pages are "stolen" by MVS.

CPU time, EXCP count, and Memory utilization must be put into a balance that makes sense in the reader's environment. CPU time and EXCP count can be significantly reduced by using more memory to hold data. (References 1). When less memory is available or less memory is utilized more EXCP's and CPU time will be expended. Programmers must use a knowlege of their own processing environments to decide how to balance these metrics. However, whenever possible, a programmer should try to reduce a program's EXCP count.

Now that the most important performance metrics have been identified, SAS System information sources will be examined. Some sources directly report the values of metrics for programs that have executed. Others provide information that can be used to write more efficient programs.

PROCESSING STATISTICS

When a SAS program is executed (in batch or online) each PROC and Data step is termed a "task." Performance statistics for the processing of each task can be obtained by setting several systems options. The STIMER option ensures that CPU time and job elapsed (wall clock) time are displayed for each task. The MEMRPT option displays memory utilization for each task. The FULLSTATS and FULLSTIMER options direct the SAS System to write CPU time, Elapsed time, EXCP count, Task memory, and Total memory on seperate lines of the SAS log for each task. The STATS option dictates that CPU time and Total memory are to be written to the SAS log. Here is an example of how these systems options can be coded:

```
OPTIONS STIMER MEMRPT FULLSTATS;
```

Figure 1 (at the end of the paper) is an example of task processing statistics generated by a program where the above OPTIONS statement was in effect. The second line in the third **NOTE:** lists CPU time for the task in the form HH:MM:SS.hs, where HH = hours, MM = minutes, SS = seconds, and hs = hundredths of seconds. The task in the example used 29.93 CPU seconds.

Elapsed time, listed after CPU time, is the actual wall clock time the task took to execute. The importance of Elapsed time is often overrated. In the MVS environment, each address space (Batch Job, TSO session, or started task) constantly competes with all others for access to the central processors. When a large number of address spaces are executing there is a greater possibility that any one of them will be swapped out by an address space with a higher system priority. This elongates the elapsed time. When the volume of address spaces is small, individual tasks have less competition for the central processors and have a better chance of not being interrupted by higher priority tasks. This generally lowers elapsed time. So, a whole range of environmental factors such as the mix and volume of address spaces, I/O contention, waits for tape mounts, etc. can affect Elapsed time.

EXCP count is the number of Input/Output operations the task completed. As discussed earlier, EXCP's are the slowest events in the life of a task, so the lower the EXCP count, the faster the task executes. In an online environment executing multiple sessions, the I/O operations for the other sessions will be included in the EXCP count field. Programmers not executing multiple sessions online will get an exact count of the EXCP's expended by the task.

The fifth and sixth lines show Task and Total memory utilization, respectively. Task memory is the amount of memory used by task instructions and data. Total memory is Task memory plus the amount of memory used by the SAS System itself. Both of these fields are divided into memory used for program instructions and memory used for data. These statistics can be used to determine if the memory used by the task is close to the system region size limit for the private area of the address space. If it is well under the limit, more memory can probably be used.

The fourth **NOTE:** in Figure 1 states total SAS session CPU time and Total memory. The CPU time is a cumulative count of all CPU time for a batch job or online session. A small amount of overhead CPU time not entered in any of the

previous task **NOTE:**s is added to this CPU time. So, addition of all task CPU times will fall slightly short of the CPU time in this message. The Total memory listed is the Total memory high-water mark for the entire session.

Figure 2 exhibits two messages commonly found in the MVS Job Log section of batch job sysouts. The **IEF374I** message displays job step CPU time and memory utilization. One of these messages is created for each step executed in a batch job. The **IEF376I** message displays total batch job CPU time for all steps that have executed. A single **IEF376I** message is found at the end of the Job Log. In both messages, "CPU" is the CPU time the SAS program consumed and "SRB" is the CPU time MVS expended managing the execution of the program. Adding CPU and SRB from the **IEF376I** message gives the total CPU time for an entire batch job. These statistics can be useful in tracking the overall performance of a batch job over time; especially if the STIMER and MEMRPT systems options have not been turned on.

THE CONTENTS PROCEDURE

Most SAS programmers are aware of the variety of information available in a PROC CONTENTS listing of a SAS library. The CONTENTS procedure provides information about a library's physical characteristics and about the number and type of data sets within it. Detailed information about each data set's physical attributes and about the variables that compose the observations can also be presented in the listing. Imbedded in the CONTENTS output listing are a number of fields containing performance related information.

Figure 3 is an example of the Directory of a CONTENTS procedure listing of a SAS library. The fields of most importance to performance are listed below.

Engine. This field notes the version of the SAS System that was used to create the library. The SAS System will use this engine for all programs that process the observations in the library's data sets. This field is important because it lets a programmer know which language constructs and performance options should be used to process the data. If the engine is V606, data set performance tools such as indexing and buffering can be utilized. (References 1).

Blocksize. Blocksize is the actual physical block size of the SAS library. This is the basic building block of a SAS data set and has a direct bearing on EXCP count. Large block sizes ensure more observations are moved with each I/O and decrease the EXCP count. A half-track block size is optimal for large SAS libraries residing on DASD. A block size of 32760 is optimal for SAS libraries residing on tape. Programmers should inspect the Blocksize field to determine if an efficient Blocksize has been set for the library. (Refer to the "BLKSIZE=" and the "BLKSIZE(device) =" systems options in References 3 for a thorough discussion of block size).

Blocks per Track. This is the number of blocks per track for SAS libraries residing on DASD. For IBM 3380 and 3390 devices half-track blocking is generally the most efficient; especially for large SAS data sets. Half-track blocks waste the least amount of track space lost to inter-block gaps and increases the number of observations per data set page. In the example, the number 2 means half-track blocking is in effect. (If the number had been 3, then the library would have been blocked at one-third-track). Greater numbers of Blocks per Track force more track bytes to be lost to inter-block gaps and inflate the overall size of the SAS library. Blocks per Track is directly affected by the "BLKSIZE=" and "BLKSIZE(device)=" options mentioned above.

Total Blocks Used. This field represents the total amount of library space occupied by data blocks, index blocks and directory blocks. This field can be used with Blocks per Track to calculate the total amount of tracks the contents of a library occupy on DASD. Simply divide Total Blocks Used by Blocks per Track to get the total amount of tracks needed to store the active blocks in the library. In Figure 3 the calculation would be 28/2 = 14 tracks occupied by data, indexes and directory blocks. This field

is very useful for determining the primary allocation of a SAS library that is to be copied.

Highest Block Used. This is the highest physical block of a SAS library that contains, or has contained, data, index records, or directory information. In an active library, blocks at the "end" of the library are used to store new or modified data sets. The addition of these blocks to the "end" of the library raises the value of the Highest Block Used field. When data sets are deleted, the blocks they inhabited become free space blocks. If the data set occupying the "end" of a library is deleted, the Highest Block Used value does not change. This field records the highest block used to date, not the highest block currently used. The difference between the Highest Block Used and Total Blocks Used is the amount of embedded free space blocks within the library. A large difference is a good indication that a significant number of data sets have been modified or deleted.

Figure 4 is an example of the data set portion of a CONTENTS procedure listing. The fields containing the most important performance information are listed below.

Observations. This is an obvious metric almost not worth mentioning. However, common sense dictates that larger numbers of observations will require greater system overhead to process. Programmers should check this field to determine if their programs are going to access a few hundred, a few thousand, or a few million observations.

Indexes. This field lists the number of indexes created for the data set. If the value is non-zero, the programmer should examine the "Alphabetic List of Indexes and Attributes" for the index names and variables. In the example, the value of Indexes is 2. The Alphabetic List of Indexes and Attributes lists both indexes. The first is a Simple index based on variable DATE. The second is a Composite index named JOBSTEP, composed of variables JOBNAME and STEP. Indexes are very powerful tools for reducing program processing overhead. (References 1). Significant reductions in EXCP count and CPU time can be realized when an index is used with a WHERE clause. Programmers should seek out and exploit indexes whenever possible.

Compressed. This field indicates whether the data within the data set is in compressed format. Compression reduces the overall number of blocks a data set occpupies by converting repeating strings of characters to two and three byte representations. Benchmarks have illustrated that compression can reduce large data set DASD allocations by one-third or more. (References 1). However, the resulting CPU overhead from processing compressed data sets is very high. (References 1 and 4). If the value of the Compressed field is "YES", a programmer can expect optimally low EXCP counts and very high CPU times for programs processing the data set.

Data Set Page Size. This is the number of bytes of data per data set page. The SAS System organizes data sets into pages, onto which observations are stored. Pages are composed of blocks, so they are an integer multiple (1 or more) of the Blocksize discussed earlier. The larger the Data Set Page Size, the greater is the number of observations stored per page, and the fewer are the pages needed for the data set. Large page sizes generally result in reduced I/O and CPU time, especially for large data sets. (References 1).

Subextents. Subextents is the number of discontiguous blocks into which the data set is divided. The SAS System attempts to use free space within libraries when data sets are created. If a data set is too large to fit into imbedded free space it is divided into a piece that fits into the embedded free space and a piece (or pieces) that are placed elsewhere. Subextents is a measure of the degree of data set fragmentation within a SAS library. When this number grows for several large, active data sets within a library, the library should be reorganized (PROC COPY) to put the data sets into contiguous allocations.

Total Blocks Used. This is a key metric for determining the exact size of a SAS data set. Divide Total Blocks Used by Blocks Per Track (from the directory) to obtain the total DASD tracks the data set occupies. A data set may

not currently occupy this exact number of tracks because it may not be contiguously allocated within the library. It could be fragmented and share tracks with other data sets. Special care must be taken with this number because it does not include blocks used by indexes. In the examples, Total Blocks Used is 7 (Figure 4) and Blocks Per Track is 2 (Figure 3). This means the data set occupies 3 1/2 tracks worth of blocks in the SAS data library. The total amount of DASD occupied by data and indexes can be obtained by adding Number of Index File Pages, 5 (Figure 4), to Total Blocks Used, 7, and dividing by Blocks Per Track, 2. The result is that data and indexes occupy 6 tracks worth of blocks within the library.

THE OPTIONS PROCEDURE

The OPTIONS procedure provides important information about an organization's SAS System processing defaults. Though there are over seventy systems options, the five listed below are the most important to program efficiency. A listing of current systems options can be produced with the following statements:

```
PROC OPTIONS;
RUN;
```

BLKSIZE. As discussed earlier, SAS libraries are composed of basic elements called blocks. A block is the minimum unit of physical bytes transferred during an EXCP. BLKSIZE is the system default number of bytes that a SAS library block contains. Low BLKSIZE values for large data sets generally result in higher EXCP counts and higher CPU time. If BLKSIZE is low, it can be overridden by placing "OPTIONS BLKSIZE=n" (where n = an optimal blocksize) at the beginning of a program.

BLKSIZE(device)=. This option provides the flexibility of allowing different blocksizes for different devices. For instance, "BLKSIZE(3380) = HALF" and "BLKSIZE(3390) = THIRD" declares all SAS libraries created on IBM 3380 DASD will have half-track block sizes; those on IBM 3390 DASD will have one-third track block sizes. Programmers can code this on an OPTIONS statement at the beginning of a program to override system defaults. The author recommends setting this value to "HALF" for IBM 3380 and 3390 DASD and at "MAX" for tape data sets.

BUFNO. BUFNO declares the number of data set page buffers that are allocated for each SAS data set a program processes. This number has a great effect on EXCP count, CPU time, and Memory utilization. (References 1). Data set page buffers are allocated in memory, so the larger the BUFNO value, the more memory will be utilized by a program processing SAS data sets. Generally, larger BUFNO values use more memory but cut down the EXCP count and CPU time. A good general number for BUFNO is 5.

BUFSIZE. This option sets the page size of newly created SAS data sets. BUFSIZE should always be an integer multiple of BLKSIZE. Larger BUFSIZE values allow more observations to reside on each data set page and results in fewer EXCP's needed to move the data when it is processed. (References 1). The trade-off for using larger BUFSIZE values is that more memory is needed to hold them. A good general value is to set BUFSIZE equal to two times the BLKSIZE.

CONCLUSIONS

The SAS System operating in the MVS environment contains a number of sources of information about SAS libraries and the resources expended to process the data within them. Programmers who know where to find the information can examine performance metrics and judge the efficiency of their programs. They can create new programs that are more efficient and tune existing programs to be better performers. This effort will reduce overall processing overhead and be a great benefit to their organizations.

ACKNOWLEDGEMENTS

SAS is a registered trademark of SAS Institute, Inc., Cary, NC, USA.

IBM is a registered trademark of International Business Machines Corporation, Armonk, NY, USA.

The author would like to thank U.S. Customs and Computer Associates International, Inc. for supporting his participation in SUGI 17. Thanks are once again due to Mary Krobath for putting this paper in proper SUGI format.

REFERENCES

1. Raithel, Michael A., "A Tale of Two Releases: Benchmarking the Performance of the SAS System Release 6.06 Against Release 5.18", Proceedings of the Sixteenth SAS Users Group International Conference, Cary, NC:SAS Institute Inc., 1991, pp. 494-503.

2. SAS Institute Inc., SAS Language: Reference, Version 6, First Edition, Cary NC:SAS Institute Inc., 1990. 1042pp.

3. SAS Institute Inc., SAS Companion for the MVS Environment, Version 6, First Edition, Cary, NC:SAS Institute Inc., 1990. 589pp.

4. Hardy, Jean, "Efficient SAS Software Programming: A Version 6 Update", Proceedings of the Fourth Annual NorthEast SAS Users Group Conference, Greenwich, CT:NorthEast SAS Users Group, 1991, pp. 247-251.

CONTACT INFORMATION

Michael A. Raithel
PO Box 406
Garrett Park, Md. 20896

Telephone: (703) 440-6467

SEE FOLLOWING PAGES FOR FIGURES 1-4.

```
NOTE: The initialization phase used 0.16 CPU seconds and 2254K.
1          OPTIONS STIMER MEMRPT FULLSTATS;
2
3          DATA EXTRACT;
4          SET  TAPEFILE.WEEKALL;
5               WHERE '01FEB92'D <= DATE <= '15FEB92'D;
6
7          KEEP DATE TAPEDRIV TAPEEXCP TAPEMOUN;
8
9          RUN;

NOTE: The data set WORK.EXTRACT has 99689 observations and 4 variables.
NOTE: The DATA statement used the following resources:
      CPU time      - 00:00:29.93
      Elapsed time - 00:02:16.61
      EXCP count    - 5103
      Task  memory - 1389K (147K data, 1242K program)
      Total memory - 3397K (1540K data, 1857K program)

NOTE: The SAS session used 31.20 CPU seconds and 3397K.
NOTE: SAS Institute Inc., SAS Circle, PO Box 8000, Cary, NC 27512-8000
```

FIGURE 1

```
IEF374I STEP /SAS606  / STOP  92055.1631 CPU  0MIN 31.38SEC  SRB  0MIN 00.78
                                VIRT  940K SYS  228K EXT   2896K SYS    9140K

IEF376I  JOB /SCPMMARY/ STOP  92055.1631 CPU  0MIN 31.38SEC SRB   0MIN 00.78
```

FIGURE 2

CONTENTS PROCEDURE

-----Directory-----

```
Libref:                 OUTONE
Engine:                 V606
Physical Name:          PMR.MAR.TAPEFILE
Unit:                   DISK
Volume:                 STOR27
Disposition:            NEW
Device:                 DISK
Blocksize:              27648
Blocks per Track:       2
Total Blocks Used:      28
Highest Block Used:     28
Members:                5
```

#	Name	Memtype	Indexes
1	WEEKALL	DATA	YES
2	WEEK1	DATA	
3	WEEK2	DATA	
4	WEEK3	DATA	
5	WEEK4	DATA	

FIGURE 3

CONTENTS PROCEDURE

Data Set Name:	OUTONE.WEEKALL	Observations:	2682
Member Type:	DATA	Variables:	7
Engine:	V606	Indexes:	2
Created:	10:12 Friday, February 28, 1992	Observation Length:	56
Last Modified:	10:12 Friday, February 28, 1992	Deleted Observations:	0
Data Set Type:		Compressed:	NO
Label:			

-----Engine/Host Dependent Information-----

Data Set Page Size:	27648
Number of Data Set Pages:	7
First Data Page:	1
Max Obs per Page:	406
Obs in First Data Page:	388
Index File Page Size:	27648
Number of Index File Pages:	5
Physical Name:	PMR.MAR.TAPEFILE
Release Created:	6.06
Release Last Modified:	6.06
Created by:	SCPMMAR$
Last Modified by:	SCPMMAR$
Subextents:	1
Total Blocks Used:	7

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len	Pos	Format	Label
1	DATE	Num	8	0	WORDDATE.	DATE
6	JOBJESNO	Num	8	40		JOB*JES*NUMBER
5	JOBNAME	Char	8	32		JOB*NAME
7	STEP	Char	8	48		JOB*STEP
3	TAPEDRIV	Num	8	16		ALLOCATED*TAPE*DRIVES
2	TAPEEXCP	Num	8	8		TAPE*EXCPS
4	TAPEMOUN	Num	8	24		TAPE*MOUNTS

-----Alphabetic List of Indexes and Attributes-----

#	Index	Var1	Var2
1	DATE		
2	JOBSTEP	JOBNAME	STEP

FIGURE 4