Performance and Capacity Planning Considerations for the SAS® System on Mainframes
Daniel J. Squillace
SAS Institute Inc.

Objectives
The objectives of this tutorial are to acquaint you with performance-sensitive SAS system options in the MVS and VM/SP environments and with other external tuning techniques. Your choice of option settings and tuning decisions can substantially affect the resource requirements required to run SAS programs. We will also go over a basic methodology for estimating resource requirements of SAS applications. Topics we will cover include:

SASLPA in MVS
Other MVS program loading considerations
Saved segments in VM/SP
I/O considerations
Macro processing options
Other SAS system options
Sort considerations
PL/I considerations
Quick and dirty SAS application sizing

SASLPA in MVS
In the MVS environment, the SAS supervisor resides in a set of load modules which are loaded during compilation and execution of SAS DATA and PROC steps. This module loading activity can result in significant I/O and CPU overhead, especially when the SAS job consists of many short DATA and PROC steps. During the installation of the SAS system, you can elect to construct a module, SASLPA, which is a packaging of SAS supervisor components into a single load module. SASLPA can then be executed out of a private step library, a linklist library, or the link pack area (LPA). You can usually expect an 8% reduction in CPU time and I/O counts by doing this. However, the savings can exceed 20% if you execute many short DATA and PROC steps.

In a multi-user environment, you can realize considerable savings in paging and memory requirements by installing SASLPA in LPA. This is especially true in an interactive environment where many users are using the SAS system concurrently. Let’s assume that each of 25 users requires 100K to hold the SASLPA working set. Sharing a copy of SASLPA in LPA in this case would yield a real memory savings of 2.4 megabytes.

On the other hand, in an environment where there is little or no concurrent use of the SAS system, there is little advantage to installing SASLPA in the link pack area. Some degradation might even be observed because loading a private copy of SASLPA is more efficient than page-faulting through an LPA copy.

LPA storage in most MVS systems is a scarce commodity, and often you must make hard decisions when deciding which modules to place there. SASLPA has grown from approximately 250K in release 82.4 to 430K in release 5.08 and will be about 580K in the upcoming maintenance release. Most of the increase in the maintenance release comes from inclusion of the routines through which procedures request supervisor services. However, you can reduce the size of SASLPA by excluding SAS Display Manager, SAS/FSP®, and SAS/GRAPH® modules if appropriate in your environment. In this case, these routines would be loaded from the SAS program library if needed.

Other MVS program loading considerations
Even if SASLPA is used to maximum advantage, you may still find that the SAS program library remains a significant contention point in your system. If this is so, there are several things for you to consider.

Program library blocksize The largest text record the linkage editor will create is 18K (will be 32K in DFP/XA 2.1). Therefore, the SAS load library blocksize should be a value greater than 18K if the device will support it. Blocksize is actually of varying degree of importance depending on your level of MVS, but in no case does it hurt to use the maximum size.

Directory search time reduction Another effective way you can reduce library contention is to reduce the time spent in directory search activity. Directory searches are especially bad because they can consume considerable channel as well as device time. The SAS BLDLTABLE option, Linklist lookaside (LLA) in MVS/XA 2.1 and above, and dynamic BLDL products available from other vendors can all be used to effectively reduce the number of directory searches. At least one dynamic BLDL product allows you to optionally have LPA and linklist be searched before specified JOB, STEP, or TASK libraries thereby eliminating all directory searches for such things as access method modules and PL/I run-time routines.
The SAS BLDTABLE option causes a table to be set up which contains both fixed and dynamic entries. The fixed entries are for frequently used supervisor modules while the dynamic entries are updated on a LRU basis.

Multiple SAS load libraries Contention can also be reduced by splitting the SAS load library into smaller libraries on separate devices.

Cached DASD The SAS load library could be an excellent candidate for cached DASD should you be lucky enough to have such devices. Keep in mind, however, that caching will do little to reduce directory search time.

Program caching At least one vendor offers a product which will do program caching out of a separate program management address space. Essentially, a user-definable amount of virtual storage is set up as a dynamic program pool managed by a least-recently-used algorithm. Programs are transferred to requesting address spaces via cross memory services. This technique can be effective if you have enough real memory to back up the program pool.

Program fetch improvements There are some recent PTFs to DFP program fetch which allow fetch to take advantage of BLD text block counts when loading overlay modules. Since many SAS PROCs are in overlay format, the improvement could be significant.

Saved Segments in VM Saved Segments in VM may be shared, in which case they are analogous to LPA in MVS and yield similar benefits. The SAS shared supervisor segment is about 900K, thus you are likely to see a large real memory savings in a multi-user environment when using the shared supervisor segment.

Furthermore, since saved segments are attached and detached from individual machines as needed, only segments which must be attached at the same time need reside at unique addresses. Thus, VM installations are not restricted to just a few megabytes of shared code and are not as constrained by virtual storage tradeoffs as are MVS installations.

VM installations can also install non-shared code in saved segments. You can elect to install the non-shared portions of the SAS system in saved segments. There are no real memory savings to be had by doing this but there could be significant I/O and CPU time savings. Virtual CPU and I/O savings are clearly there, but it will take further study to identify the net savings after CP time to support the saved segment paging is taken into account. If your data center charges are based in part on virtual machine size, you can reduce your bill by using non-shared segments even if there is not much real resource savings.

In any case, I highly recommend that you at least install the supervisor shared segment. This is definitely the least amount of work and the most gain. Furthermore, in version 5, saved segments are built from the load module code, so maintenance headaches associated with segments in release 82.3 do not exist. You can apply maintenance to the load module libraries, test it there, and then rebuild the saved segments from the load module libraries.

I/O Considerations for MVS

DASD blocksizes It has been shown time and time again that large blocksizes effectively reduce I/O and CPU time in MVS. The same is certainly true with SAS data sets. Use full-track blocking for all DASD devices other than 3380 and use half-track blocking on the 3380. With the maintenance release, blocksize is your only SAS data set I/O control parameter. The SAS BUFNO option no longer has any effect because only one buffer is used. Note that the SAS system always sets RECFM=U and BLKSIZE=8K for the device in the DSCB for the SAS data library. The blocksize actually used is that specified in your SAS options unless the observation size is larger than the blocksize. Since observations do not span blocks, a larger blocksize is used in this case.

Tape blocksizes Use a 32K blocksize for SAS libraries on 3480 cartridges and on 6250 bpi tape. Use no more than a 12K blocksize on 1600 bpi tape because the error correction method used with 1600 bpi tape tends to lose its effectiveness with very large blocksizes.

I/O considerations for VM

Minidisk physical blocks Specify a 4K blocksize for your minidisks which are to contain SAS data sets. This specification will give you the best space utilization on 3380s as well as result in the least amount of physical I/O.

SAS blocksize option Set the SAS blocksize option to 8K. Larger values do not yield any performance improvement because the CMS file system will write at most two consecutive blocks to disk in one I/O. The SAS system will automatically increase the blocksize if the observation length is greater than 8K.

OS Read-only data libraries There is special channel program level code in the CMS SAS system to read SAS data libraries in OS format. Since these libraries can have large physical blocksizes, they can be read more efficiently than CMS SAS data libraries. Therefore, if you have both MVS and VM systems that can
conveniently share DASD, there are both performance and maintenance advantages to using a common OS copy of such things as SAS/AF® menus, CBT libraries, and SAS/GRAPH map data sets.

Virtual I/O You can use the VIOBUF=nK option to specify that up to nK of work data sets will reside in virtual storage rather than on disk. This can be a very effective way of eliminating much work data set I/O provided your working sets can be increased enough to support the virtual I/O. It does not do much good to replace work I/O with paging I/O! The default shipped with release 5.08 was 256K; however it appears that 128K is a better general default and that value will be the default in the CMS maintenance release.

Please note that the VIOBUF storage does not hold a dynamic pool of work data set buffers, but rather contains up to nK of work data sets. To most effectively use virtual I/O, you should scratch work data sets as you finish with them so that the virtual I/O buffers can be allocated to active data sets. If you are using SAS Display Manager, clear your log window periodically because it, too, is a work data set. (The CLEAR command will be available in the maintenance release).

Macro Processing Options

IMPLMAC and MAUTOSOURCE One of the significant enhancements included in Version 5 SAS software is an auto-call facility for macros. It is no longer necessary to explicitly include each macro you wish to use. Macros may instead be pulled in and compiled on demand from a macro library. This facility can save resources in that macros will not be retrieved and compiled unless required. The macro auto-call facility is turned on by specifying MAUTOSOURCE.

More than one macro may be stored in a single library member provided that one macro in the group matches the library member name. This can be a handy way to bring in a group of related macros with one directory search.

There is an interaction of MAUTOSOURCE with another option, IMPLMAC, which you should be aware of. Specification of IMPLMAC allows you to use statement-style macros in your SAS programs. With IMPLMAC in effect, each SAS statement is potentially a macro and the first word (token) in each statement must be checked to see if it is a macro call. When IMPLMAC is in effect without MAUTOSOURCE, no special checking takes place until the first statement-style macro is compiled. When MAUTOSOURCE is on, however, this checking must be done unconditionally. The initial occurrence of a word as the first token of a SAS statement results in a search of the auto-call library. There can be a significant number of directory searches, especially during the compilation of a large data step, in addition to the CPU time necessary to maintain and search the symbol table. The combination of MAUTOSOURCE and IMPLMAC can add 10% or more to the CPU and I/O resources required for a job. As an example, the compilation of a SAS program containing about 420 statements required 95 auto-call library directory searches and an additional 25% CPU time. For best performance, you should set NOIMPLMAC as the installation default.

MSIZE and MLEAVE options When macros are compiled, they are stored in the work data library in 2048-byte execution frames. These frames are brought into memory during macro execution. If insufficient storage has been set aside to hold the execution frames, extra work data set I/O is incurred during macro execution to support frame paging. (MSIZE-MLEAVE/2K) gives the number of memory frames available for macro processing. The default is three. The cost of having too few macro execution frames can be significant while the cost of over allocation is trivial. A good set of installation defaults would be MSIZE=128K and MLEAVE=64K.

Other SAS System Options

Number of history generations The GEN=n option specifies how many generations of history data are to be kept for SAS data sets. When GEN is non-zero, SAS source statements used to create the data set are stored in the directory. This storage of history information can take up considerable library space and add significantly to the number of I/O operations required to process a SAS job. If you specify GEN=0, you can expect to reduce I/O by 10% or more. There are times of course when keeping history information is entirely appropriate, perhaps in studies where it is necessary to keep an audit trail of how the data was processed. Ideally, it would be nice to set GEN=0 as your installation default and have those who need history information explicitly set GEN to a non-zero value. Failing that, you should at least have large production jobs for which the source code is well controlled and documented elsewhere explicitly set GEN=0.

CHKPT/NOCHKPT This option controls whether or not checkpointing is done at the end of every DATA and PROC step. There is no reason to specify CHKPT in batch and little reason in TSO or CMS. In tests I have run, the CHKPT option added 6 to 10 I/Os per step.

The LEAVE option The purpose of this option is to set aside a specified amount of memory for non-SAS memory allocation. Oddly enough, this option does have some performance impact because one extra GETMAIN and two extra FREEMAINS are done by memory management for each storage request when LEAVE is non-zero. Even when using PROCs which issue many memory management requests, the impact of this overhead appears to be negligible in both MVS and CMS. In MVS, SVC traces do confirm the additional GETMAIN/FREEMAIN activity, but
the CPU time impact is considerably less than SMF measurement variability. CMS tests give similar results.

In any case the overhead is there and may show up more significantly in other situations. Therefore you should set LEAVE to 0 unless you have a demonstrated reason to do otherwise. In MVS, by the way, it should never be necessary to set LEAVE to a non-zero value because the IEALIMIT routine provides similar function.

FSEDIT AUTOSAVE. The FSEDIT procedure provides the AUTOSAVE command to allow you to control how often checkpointing is done by closing and reopening the data set. The system default is to do this every 25 observations; hence you will lose no more than 25 observations if the system crashes during an editing session. There is a definite tradeoff between overhead and data loss (or re-entry) potential in setting this option. Setting AUTOSAVE to 1 costs about 5 I/Os per observation with a single variable data set in a small directory.

Display Manager AUTOROLL LOG ON/OFF. In release 5.08, each line written to the log window caused a separate I/O to the terminal. The most noticeable effect of this was heavy data traffic and slow response on remote lines. Local users did not usually notice a response problem due to the many terminal I/Os. However, the CPU time cost was heavy in both cases. The maintenance release will introduce a new option, AUTOROLL LOG ON or OFF. ON will cause the system to behave as it does in release 5.08 while OFF will cause the log window to be updated only when you are prompted for input. At that time of course, you may scroll the log backwards to see what transpired since your last input. Hence the system default will be AUTOROLL LOG OFF. To convince yourself this should be your installation default, run a PROC OPTIONS both ways and compare the results. The CPU time ratio is about 8:1 under MVS and 10:1 under CMS.

SPOOL/NOSPOOL

The SPOOL option is intended to be used when running the SAS system interactively without Display Manager. When SPOOL is in effect, SAS input statements are stored in the WORK library for later retrieval by the %INCLUDE command. You should set NOSPOOL when running under DISPLAY MANAGER because the RECALL command provides similar function.

Sort Considerations

FILSZ. If your system sort utility supports the FILSZ parameter, set FILSZ in your SAS system options. This option can improve sorting efficiency because it will cause the sort to be given more accurate information about the number of records to be sorted.

SORT vs. SORTT. You can use either your own system sort utility or one supplied with the SAS system to sort SAS data sets. Your system sort is used by setting SORTPGM= name and invoking PROC SORT. The SAS system sort may be used by setting SORTPGM=SAS or by invoking PROC SORTT.

SORTT is very efficient when sorting small data sets. Tests run on MVS and CMS against SyncSort™ show the crossover point comes at about 2000 observations. The observation size in these tests was 232 bytes. If you have TSO or CMS users who are sorting a large number of small data sets, it would be worthwhile to set SORTPGM=SAS in the interactive defaults.

DYNAALLOC/NODYNAALLOC. This option indicates whether PROC SORT or your sort utility is to allocate the sort work data sets. Which method should be chosen depends on the characteristics of your system sort. If you specify NODYNAALLOC, the initial invocation of PROC SORT will allocate sort work data sets. If SORTWKNO=x and SORTT=y, then x data sets of at least y cylinders will be allocated. If the sort work space is adequate for subsequent sorts, it will be reused. If not, the work data sets will be replaced with larger ones.

If you specify DYNAALLOC, PROC SORT will ask the system sort utility to allocate SORTWKNO=n work data sets. What happens beyond this point depends on your system sort. The current release of OS SyncSort, for example, will allocate new sortworks if the existing ones are too small whereas previous releases would not. Thus, if DYNAALLOC was specified in jobs where the initial sort was small and a subsequent one was large, the large sort would fail.

A good case for specifying DYNAALLOC exists if you have a system sort which does not allocate sort work data sets at all unless they are needed and then reuses or replaces them on subsequent sorts as required.

SORTWKNO. SORTWKNO=n is used to specify the number of sort work data sets to be allocated by PROC SORT or by your sort utility. This number should be no greater than the actual number of devices available for sort work data set allocation.

EQUALS/NOEQUALS. Specification of EQUALS will cause PROC SORT to pass the EQUALS option to the system sort utility, thereby ensuring that observations with equal keys will retain their order. If nothing is specified, PROC SORT will ensure correct order itself. If NOEQUALS is specified, neither PROC SORT nor the sort program worry about order of duplicate-keyed observations. It seems intuitive that specification of NOEQUALS would save resources.
and in fact some people have reported that to be the case. However, my tests have not shown any measurable difference.

**PL/I Considerations**

**Transient modules in LPA** If you are running MVS, you should place some of the PL/I transient modules in LPA. If you are running release 4, placing the entire transient library in LPA requires about 150K. However, if your only significant user of PL/I is the SAS system, the following fifteen module subset occupying 28K will suffice:

- IBHBESMA
- IBHBESNA
- IBMBOPCA
- IBMBOPCA
- IBMBPITA
- IBMBPITA
- IBMBRVHA
- IBMBRVHA
- IBMBOCAA
- IBMBOCAA
- IBMBOPEA
- IBMBOPEA
- IBMBSOCA
- IBMBSOCA
- IBMBOPAA
- IBMBOPAA
- IBMBOPBA
- IBMBOPBA
- IBHBPIIA
- IBHBPIIA
- IBMBSVIA
- IBMBSVIA

**ISASIZE** When a PL/I program is started, PL/I allocates an Initial Storage Area (ISA) to be used for dynamic storage requests. The size of this area is determined by ISASIZE as coded in the program or as passed by an execution-time parameter. If the ISA is too small, additional operating system storage management overhead is incurred. ISASIZE may be specified using the SAS PARM option as follows: PARM('ISASIZE(nnK)'). In most cases, the ISA size internally specified in SAS PROCs should be sufficient, but there may be some cases where it is not. If you wish to be on the safe side, specify an ISA(64K). A report on ISA usage may also be gathered by coding REPORT in the PARM option and including a PLIDUMP DD statement or FILEDEF. See the appropriate IBM PL/I Optimizing Compiler Programmer's Guide for details.

**Quick and Dirty SAS Application Sizing**

Sometimes we get questions such as "How many CBT course users can I have on my 3084Q?" or "How much CPU time do I need?" The general answer is "It depends!". How much of the 3084Q is available for use? What exactly are you intending to do with SAS/FSP? What follows gives you a way of approximating these answers for yourself.

**Resources required per user** Estimate the CPU, I/O, and working set requirements per user from tests of the proposed applications or from similar existing applications. The necessary data can be obtained from online monitors or from SMF and RMF records. Let's assume for example that you have a CBT course on trial. Monitor a user taking the course and note over the elapsed time how much CPU time he required, how many I/Os he did, and what his working set was. From this data, you can calculate CPU utilization, I/O rate, and memory utilization by the user.

**Total resource requirements** Estimate the total CPU utilization, I/O rate, and memory requirements by multiplying the values for a single user by the number of concurrent users expected. This data can then be used to show whether the new application can be accommodated within the remaining capacity on your system. If your organization has a capacity or facilities planning group, take your estimates of resource requirements to them for more detailed evaluation. They may also be able to supply you with estimates of response time and of end user charges.

**Scale up to large data sets** When making estimates for applications involving sequential processing such as FSBROWSE, be careful to take into account the eventual size of the production data sets. Resource estimates based solely on test data set or pilot production data set runs will not be valid in this case.

**Summary**

The SAS system option settings you choose can substantially affect the performance of your SAS applications. Furthermore, an understanding of those options is necessary for you to make the choices and tradeoffs appropriate for your installation.

SyncSort is a trademark of SYNSORT INC. Englewood Cliffs, NJ, USA