ABSTRACT

Tuning the SAS System can help you make the most of your investment in both hardware and software. Toward this end, the SAS System provides a built-in set of features that will aid you in effectively managing consumption of CPU, memory and I/O resources. This paper will introduce those SAS System features, and will provide information on techniques which make use of them, including discussion of trade-offs that are often necessary. Specific topics covered here include memory management, SAS System image management, SAS I/O considerations and sorting issues, among others. The information in this paper is applicable to your site whether you run the SAS System interactively or in batch.

INTRODUCTION

As with any other sophisticated software system, there can be significant pay-back in tuning the SAS System. That pay-back does not always come easily though. As SAS software increases in flexibility and power, there are fewer absolute "rules of thumb" which may be applied in tuning the system. Rather, much savings from tuning comes from achieving a balance in resource utilization tradeoffs. Furthermore, the optimum points in these tradeoffs depend on the applications for which SAS software is used, and on your installation's policies regarding resource utilization and charge-back. We will cover a few "rules of thumb", but the emphasis in this paper is on describing features provided for tuning and the resource utilization tradeoffs associated with tuning choices.

MEMORY MANAGEMENT

The MEMSIZE Option

Version 6 of the SAS System uses memory (storage) above the 16MB line for most programs and data. This is also sometimes referred to as 31-bit addressing. The few exceptions include a small part of the host supervisor and some data areas which must have 24-bit addressability. Since the MVS JCL parameter, REGION, is ineffective at controlling 31-bit memory usage, a SAS option has been implemented for this purpose. The MEMSIZE option, which must be specified at SAS System initiation, establishes an upper limit on the total amount of virtual memory that the SAS system will use.

By default, MEMSIZE has a value of 0 (zero) which means that the SAS System will use memory up to the maximum amount available. For example, once SAS programs are loaded into memory, they are not deleted until that memory is needed for another purpose. This behavior is designed to allow multiple consecutive procedure invocations without requiring the program to be reloaded each time. However, if there is no upper limit on virtual memory usage, memory is not freed for reuse even when programs stored there are no longer needed. It is therefore recommended that you specify a MEMSIZE limit explicitly. 8MB has proven to be a reasonable value for MEMSIZE in most batch applications, and is the default setting for this option in the configuration files supplied with the SAS System. Interactive applications which use multiple SAS System components such as SAS/ARMS® and SAS/GRAPH® should set MEMSIZE=16M.

Memory Fragmentation Control

There is a set of SAS system options called superblocking options, which serve to control memory fragmentation. These work by setting aside pools or superblocks of memory for different classes of use:

- the SAS portable supervisor
- SAS memory management services
- permanent memory which exists for the life of the SAS session
- temporary memory which exists for the life of an individual task or procedure

For each of these classes there are options for specifying an initial storage allocation amount (ISA) and an overflow storage allocation amount (OSA). Further, a distinction is made between memory used both above and below the 16MB line. These superblocking options are listed here for your reference:

<table>
<thead>
<tr>
<th>OPTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSUPISA</td>
<td>Portable supervisor initial storage allocation</td>
</tr>
<tr>
<td>PSUPOSA</td>
<td>Portable supervisor overflow storage allocation</td>
</tr>
<tr>
<td>VMPAISA</td>
<td>Permanent initial storage allocation above 16M</td>
</tr>
<tr>
<td>VMPAOSA</td>
<td>Permanent overflow storage allocation above 16M</td>
</tr>
<tr>
<td>VMPBISA</td>
<td>Permanent initial storage allocation below 16M</td>
</tr>
<tr>
<td>VMPBOSA</td>
<td>Permanent overflow storage allocation below 16M</td>
</tr>
<tr>
<td>VMTAISA</td>
<td>Temporary initial storage allocation above 16M</td>
</tr>
<tr>
<td>VMTAOSA</td>
<td>Temporary overflow storage allocation above 16M</td>
</tr>
<tr>
<td>VMTBISA</td>
<td>Temporary initial storage allocation below 16M</td>
</tr>
<tr>
<td>VMTBOSA</td>
<td>Temporary overflow storage allocation below 16M</td>
</tr>
<tr>
<td>VMCTLISA</td>
<td>Initial storage allocation for memory management services</td>
</tr>
</tbody>
</table>
In order to eliminate memory fragmentation, the initial storage allocation should be made large enough to satisfy all anticipated memory requirements for your SAS job or session. However, if more memory is required, another allocation of memory is made in the amount specified by the appropriate overflow storage allocation option. It is possible for multiple overflow allocations to take place.

Default values for these options are set in the configuration files that are shipped with the SAS System. In most cases, these values will suffice and should not be altered. However, in the event that you receive a superblock overflow warning message in the SAS log, you may want to increase the appropriate ISA amounts. The ISA amounts are obtained at SAS System initialization and ideally should be set at a level sufficient to satisfy all memory requirements. If this amount is inadequate, then one or more OSA's are obtained and a warning message is issued in the SAS log. If you must modify these option values, keep the following rules in mind:

- ISA allocations are set aside at SAS System initialization and are not released until SAS System termination. Unnecessarily large ISA allocations will therefore waste memory.
- OSA allocations are not made until memory is needed and the request cannot be filled from ISA or prior OSA allocations.
- Too many OSA allocations can increase memory fragmentation.
- In memory shortage situations, OSA units no longer in use will be freed in order to reclaim that memory. ISA units are never freed, not even when there is a shortage.
- Temporary OSA units are freed at the end of the task.


**Out-Of-Memory Conditions**

So, what happens if your job runs out of memory anyway? There are two SAS System options that provide for graceful termination: SYSLEAVE and PROCLEAVE. The SYSLEAVE option reserves memory to ensure that a sufficient amount is available at task or procedure termination to close data sets and perform other necessary clean up. The PROCLEAVE option serves a similar function for procedures. For instance, some procedures are designed to use memory until no more is available, and then continue by opening and using work files. PROCLEAVE ensures there will be enough memory left to open these work files and allocate I/O buffers for them so that the procedure can continue.

**PROGRAM MANAGEMENT**

There are three basic SAS System program configurations:

- unbundled
- bundled (non-LPA version)
- bundled (LPA/ELPA version).

The first of these causes all modules to be loaded individually from the SAS System load library. Running in this manner significantly increases library directory searches and I/O, and is not generally recommended. However the SAS System is shipped with this setting by default since some of the installation tasks must invoke SAS before the installer has had the opportunity to select a bundled version. This configuration is invoked by providing SASHOST as the ENTRY parameter when invoking the SAS System via the JCL cataloged procedure or TSO clist.

The remaining two configurations are bundled versions of the SAS System. Bundles are the result of packaging the many individual modules into one large executable. Invoking a bundled version of the SAS System eliminates dead space between modules and saves the overhead of loading each module individually (Ghosh 1991).

**Non-LPA Bundles**

The non-LPA version is intended for sites that do not want to place SAS modules in the Link Pack Area. Providing an ENTRY of SASXAL at SAS invocation will execute this configuration. In this case, the bundle is loaded into each user's address space. While this will decrease directory library searches and I/O, it has the unfortunate side-effect of increasing individual working set sizes. Therefore this method is not recommended if there will be many users of SAS at your site.

**LPA/ELPA Bundles**

The most effective way to reduce memory requirements in a multi-user SAS environment is to use the LPA/ELPA bundles. Their use will dramatically reduce each user's working set size. There is a large selection of bundles which you may install into LPA/ELPA:

```
<table>
<thead>
<tr>
<th>BUNDLE</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASXAL</td>
<td>202K LPA module</td>
</tr>
<tr>
<td>SASXAL2</td>
<td>604K ELPA module</td>
</tr>
<tr>
<td>SABXSPH</td>
<td>1477K ELPA module</td>
</tr>
<tr>
<td>SABDS</td>
<td>371K ELPA module for data step</td>
</tr>
<tr>
<td>SABZPLH</td>
<td>55K ELPA module (optional) for output formatting</td>
</tr>
<tr>
<td>SABDPOL</td>
<td>453K ELPA module (optional) for SAS/ASSIST®</td>
</tr>
<tr>
<td>SABSCLL</td>
<td>769K ELPA module (optional) for SCL and SAS/ASSIST®</td>
</tr>
<tr>
<td>SABDBGL</td>
<td>127K ELPA module (optional) for the SCL debugger</td>
</tr>
<tr>
<td>SABAFL</td>
<td>803X ELPA module (optional) for SAS/AF</td>
</tr>
<tr>
<td>SABFSPL</td>
<td>351K ELPA module (optional) for SAS/FSP®</td>
</tr>
<tr>
<td>SABXGPH</td>
<td>588K ELPA module (optional) for SAS/GRAPH</td>
</tr>
<tr>
<td>Total</td>
<td>5800K</td>
</tr>
</tbody>
</table>
```
Since only one bundle, SASXAL (202K), is required to run below the line where virtual storage constraint may be a consideration, using the bundled modules is a very straightforward recommendation. Incidentally, all LPA/ELPA bundles can be renamed to simplify maintenance and release transitions. The entry point for invoking this configuration will be the same as the new name for the SASXAL bundle module. Refer to Installation Instructions and System Manager's Guide for the SAS System under MVS, Release 6.08 for details.

SAS DATA LIBRARY CONSIDERATIONS

Version 6 Data Library Characteristics

The MVS SAS Version 6 data library implementation is highly device independent. It is in effect a fixed block architecture (FBA) implementation on count-key-data (CKD) devices. A library is defined as a physical sequential (PS) data set with a fixed-standard (FS) record format and the block size and logical record length set equal to the physical block size to be used for the library. The physical block must be a multiple of 512 bytes. An example of a valid DCB specification is:

DCB= (DSORG=PS , RECFM=FS , BLKSIZE=23040, LRECL=23040)

Observations are stored in pages which are an integral multiple of the physical block size. SAS data sets are stored in an integral number of pages. Therefore, the smallest unit of allocation for a Version 6 SAS data set is a page. Page size is not constrained by physical track size. Therefore observation length is not constrained by physical track size.

Page size may vary at the data set level within the library. Thus space utilization may be improved by using small page sizes (6K or less) for small data sets so that unused page space is minimized. See Clifford (1989) for details on page size considerations and the methods the SAS system uses to calculate the page size for a given data set when not specified.

The following example illustrates the layout of a SAS data library containing two data sets, each with a different page size:

\[
\begin{align*}
\text{Track} 1: & \quad \text{xxxxx xxxx x} \\
\text{Track 2:} & \quad \text{xxx xxxxxx yyyy} \\
\text{Track 3:} & \quad \text{yyyy yyyy yyyy} \\
\text{Track 4:} & \quad \text{b b b b b b b b } \\
\text{Track 5:} & \quad \text{b b b b b b b b } \\
\text{Track 6:} & \quad \text{b b b b b b b b} \\
\text{Track 7:} & \quad \text{b b b b b b b b}
\end{align*}
\]

Data sets X and Y each contain 4 pages

Version 6 I/O Subsystem Tuning

Physical block size tradeoffs

As mentioned earlier, the physical block size of the data library determines the minimum page size and the minimum unit of space allocation for the library. The default is 6K. The primary advantages to the 6K default are that it is relatively efficient across a range of device types and that it leads to lower memory requirements for catalog buffers. The primary disadvantage to the 6K default is that more DASD space is required to hold a given amount of data because of capacity losses due to smaller blocks. An MXG daily PDS, for example, takes 8% more tracks when stored using 6K physical blocks instead of half-track blocks on a 3380. Since the optimal block sizes for SAS catalogs and SAS data sets are not necessarily the same, our recommendation is that you put catalogs and data sets into separate libraries. 6K is a good general physical block size for catalog libraries regardless of device. But, for data sets you should choose a full-track or half-track block size depending on device type for data libraries.

BUFSIZE and BUFNO tradeoffs

Two SAS System options, BUFSIZE and BUFNO, may be used in tuning I/O performance. The unit of I/O transfer when processing a Version 6 library sequentially is BUFSIZE*BUFNO bytes where BUFSIZE is the page size in use for the data set and BUFNO is the number of page buffers to allocate for the data set. BUFSIZE is only applicable for creating output data sets and, once specified, is then a permanent attribute of the data set. The values of BUFSIZE and BUFNO are the primary factors affecting I/O performance for sequential data sets. For random access, BUFNO page buffers form a least-recently-used buffer pool which can significantly reduce physical I/O depending on data access pattern. Additional buffers, of course, require more memory. Buffer storage is above the 16M line.

Squillace (1990) presents a detailed study of BUFNO and BUFSIZE tradeoffs. There is a marked reduction in I/O time and I/O count as BUFNO is increased, although at a higher cost for buffer storage. As a result of this, elapsed times were also significantly reduced. The product of BUFNO and BUFSIZE is the important factor in I/O performance rather than the specific setting of either parameter. For example BUFNO=28, BUFSIZE=6144 yields very similar results to BUFNO=4, BUFSIZE=23040. It is also worth noting that when using 6K blocks, specifying BUFSIZE=24K yields performance results very close to that of BLKSIZE=23040 and BUFSIZE=23040.

So what should you do? Our recommendation is to let BUFSIZE = BLKSIZE when BLKSIZE is set to a full- or half-track value and to set BUFSIZE to a multiple nearly a half-track when BLKSIZE=6K. Set BUFNO to at least 2. However, it is not advisable to let BUFNO*BUFSIZE exceed 135K (three 3380 tracks). In fact, going beyond that point can be detrimental to other users of the system because of the device and channel monopolization which can be caused by long channel programs.
Data Compression Tradeoffs

Version 6 optionally performs data compression. It may be requested either globally or individually at the data set level. Intuitively, we would assume that a data set containing many long character variables would be an excellent candidate for compression. This was confirmed in one instance by compressing an internal problem tracking data set which is comprised mainly of character data values.

In the two examples below, PROC COPY was used to copy data from an uncompressed source data set into uncompressed and compressed result data sets, using COMPRESS=NO and COMPRESS=YES SAS system options respectively. The CPU row shows the amount of time needed to perform the copy on an IBM 3090-400S, and the SPACE values represent resulting storage usage in megabytes. Note that if you wish to compress a data set using PROC COPY, you will need to include the NOCLONE option on your PROC statement. Otherwise, PROC COPY will propagate all the attributes of the source dataset, including its compression status.

### Compression of Character Data

<table>
<thead>
<tr>
<th>Resource</th>
<th>Uncompressed</th>
<th>Compressed</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>13.7 sec</td>
<td>84.3 sec</td>
<td>+70.6 sec</td>
</tr>
<tr>
<td>SPACE</td>
<td>362.0 MB</td>
<td>56.5 MB</td>
<td>-305.5 MB</td>
</tr>
</tbody>
</table>

**CPU Cost/MB:** 0.2 sec

A more interesting situation is posed by data sets containing mostly numeric data, such as a MICS™ performance data base. Even this yielded good results, although not as dramatic as the mostly character data examined in the previous example:

### Compression of Numeric Data

<table>
<thead>
<tr>
<th>Resource</th>
<th>Uncompressed</th>
<th>Compressed</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2.4 sec</td>
<td>25.2 sec</td>
<td>+22.8 sec</td>
</tr>
<tr>
<td>SPACE</td>
<td>44.4 MB</td>
<td>23.5 MB</td>
<td>-20.9 MB</td>
</tr>
</tbody>
</table>

**CPU Cost/MB:** 0.5 sec

The CPU usage and DASD space consumed by compression of data has been isolated in these two examples for illustration. However the overall cost of processing and storage of compressed data can vary considerably and will be unique to the particular mixture of data in your application. Each time the data set is processed by PROC and DATA steps, the data must be compressed and decompressed. Whether or not data compression is worthwhile to you depends on the resource cost allocation policy in your data center. In a situation like the second example above where the benefits are less clear, your decision will be based on which resource is more valuable, DASD space or CPU time.

There are some storage devices such as Storage Technology's iceberg system which perform hardware data compression dynamically. Since this hardware compression is always performed, you may decide not to enable SAS software data compression when using these devices. One reason to choose SAS software compression in addition to hardware compression would be if DASD space charges are a significant portion of your total bill for information services.

**Version 5 Compatibility Engine Considerations**

Special considerations are necessary if you create and access Version 5 format data libraries from Version 6 of the SAS System. In Version 6, the default value for the BLKSIZE system option is 0 (zero). This means that any time a block size is omitted from a LIBNAME or FILENAME statement, the value will be determined by a table of values specific to certain output devices. PROC OPTIONS output displays this table of block size defaults. These values are preset to suffice for most allocations. In the table generated by PROC OPTIONS you will see defaults for both SAS data library allocations (BLKSIZE values), and for non-SAS external file allocations (FILEBLKSIZE values).

The important thing to note here is that the device-specific block size defaults for SAS data libraries pertain only to Version 6 data library allocations. The Version 5 compatibility engine (VS) uses an internal table which retains the old Version 5 full-track and half-track defaults. These will be used by default for new Version 5 library allocations. If you choose to override the defaults using the BLKSIZE system option, you must remember that it will apply to both Version 5 and Version 6 data library allocations and will override the Version 6 values displayed in the table. This is a critical distinction to note because only one block at a time is written to Version 5 data libraries. Specifying a small value for BLKSIZE will have a disastrous effect on I/O performance. If you have applications that will be creating Version 5 data libraries, you should take care not to specify a small value for the BLKSIZE option. Two alternative methods for setting block sizes are:

- Use JCL or TSO ALLOCATE parameters
- Use BLKSIZE(device-type)= options to override the Version 6 default values as indicated in the table.

For a discussion of BLKSIZE, BLKSIZE(device-type) and FILEBLKSIZE(device-type), see "Chapter 12, Host-Specific SAS System Options," SAS Companion for the MVS Environment, Version 6, First Edition.
SORT CONSIDERATIONS

SORTPGM and SORTCUTP

The SAS System includes an internal sort program which is efficient at sorting small volumes of data, often more efficient than invoking a general host sort program for the same data. Prior to Version 6, you were forced to explicitly choose which sort to execute. Version 6 introduces the SORTPGM=BEST option which will invoke the SAS sort program for data smaller than 4M, and the host sort for more than 4M of data. This 4M limit is the default size set via the SORTCUTP option which was added in Release 6.07. You may modify it as needed to optimize sorting for your particular applications. For optimum sorting, we recommend the use of SORTPGM=BEST.

SORTEDBY Data Set Attribute

Beginning with Release 6.07, the header of a data set can contain information describing it and how it is sorted. This information is generated automatically by PROC SORT, or may be specified explicitly via the SORTEDBY= data set option. The former is called "validated" sort order, and the latter is referred to as "asserted" sort order. Using the SORTEDBY= data set option is beneficial in instances where you know that the output data set will be created in sort order. Several procedures take advantage of this information to avoid unnecessary sorts. PROC SORT will not re-sort a data set that is already sorted, and PROC SQL will use this information where possible to avoid internal sorts. You can check the sort order of a data set and its validation status using the CONTENTS procedure.

SAS SYSTEM OPTIONS

STIMER, MEMRPT, STATS and FULLSTATS

In order to tune your application, it is helpful to enable the STIMER, MEMRPT, STATS and FULLSTATS (alias FULLSTIMER) SAS System options. The STIMER option specifies that timing statistics such as CPU and elapsed time should be collected and maintained throughout the SAS session. MEMRPT performs a similar function for memory usage statistics. In order to get these statistics printed in the SAS log, the STATS and optionally the FULLSTATS options must be enabled. With STIMER, MEMRPT and STATS enabled (the default), CPU usage and total memory are reported in the SAS log for each procedure or data step. If FULLSTATS is also enabled, quite a few additional statistics are reported in the log: CPU time, elapsed time, vector affinity time, vector usage time (see VECTOR/NOVECTOR below), RSM hyperspace time, EXCP count and task and total memory usage. As an alternative, you have the option of recording SMF records for each DATA and PROC step. Information on the contents of these records and procedures for collecting them are described in Installation Instructions and System Manager's Guide for the SAS System under MVS, Release 6.08.

MAUTOSOURCE and IMPLMAC

There is an interaction between two SAS System options that you should be aware of. MAUTOSOURCE and IMPLMAC are two options that affect operation of the SAS autocall macro facility. Specifying 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 both IMPLMAC and MAUTOSOURCE are on, however, this checking is done unconditionally. The initial occurrence of a word as the first token of a SAS statement results in a search of the autocall 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. In very limited testing, we have seen the combination of MAUTOSOURCE and IMPLMAC add 20% additional CPU time and 5% additional I/O to a non-trivial job. For best performance, you should leave NOIMPLMAC as the installation default.

SPOOL/NOSPOOL Option

The SPOOL option is intended to be used when running the SAS System interactively without the display manager. When SPOOL is in effect, SAS input statements are stored in a WORK library utility file for later retrieval by %INCLUDE and %LIST commands. Since the SAS System is shipped with SPOOL as the default setting, we recommend resetting it to NOSPOOL. In a batch job with a large number of input lines, we have observed an I/O reduction of as much as 9%.

VECTOR/NOVECTOR Option

The SAS System supports the use of the IBM 3090 Vector Facility for certain SAS procedures, most noticeably PROC GLM. This support is enabled by specifying the SAS system option, VECTOR. Although the internal default specification is NOVECTOR, this is overridden in both the configuration files which are shipped with the SAS System. Your site administrator may have chosen to reset this option to NOVECTOR if there are specific additional resource charges in effect for use of this facility. Before enabling this option, you should check to see if you will be adversely affected by charge-back.
Other Topics

The Display Manager AUTOSCROLL Command

The AUTOSCROLL command, valid in the LOG and OUTPUT windows, controls visual scrolling of output as it is written to these windows. Specifying small scroll increments is very expensive in terms of response time, data traffic, and CPU time. AUTOSCROLL 0 suppresses automatic scrolling and positions the LOG and OUTPUT windows at the bottom of the most recent output when the DATA step or PROC is completed. At that time of course you may peruse the output at will. To see the effect this command can have, set AUTOSCROLL 1 in the LOG window and then run PROC OPTIONS. Then set AUTOSCROLL 0 and run it again. The CPU time ratio is more than 30 to 1. AUTOSCROLL is preset to 0 on MVS.

The EM3179 Device Driver

If you are running Attachmate or any other full-functioned 3270 emulator over a slow connection, specify the SAS option, FSDEVICE=EM3179 at invocation. This will force icon-style menus in applications such as SAS/ASSIST, to be displayed as text-based block menus instead. The block menus require much less network data transfer and will be considerably faster across slower lines.

The Direct Logon Procedure

Beginning with Release 6.07, direct logon support was added to the MVS SAS System. This feature allows sites to isolate SAS users from the TSO environment and eliminate the need for TSO expertise. In addition, direct logon will save a small amount of memory (approximately 50K per user savings in working set size). It is enabled by substituting a special SAS logon procedure for the normal TSO logon procedure. When invoked this way, the SAS System acts as the terminal monitor program. As indicated above, a small amount of memory is saved when using this feature. However an even greater benefit may be realized if you are licensing TSO/E as a measured usage product. When using this feature, a small amount of memory is saved when using this feature. However an even greater benefit may be realized if you are licensing TSO/E as a measured usage product. Whether CPU time for SAS applications will no longer be accumulated as TSO/E usage. For example, a stored program version of the MXG DATA step, which processes SMF data, takes about 19 CPU seconds to compile on a 3090-400S. This time is saved on subsequent executions of the stored program. On an MXG daily PDB run, that savings amounts to an 8% reduction in the CPU time for that DATA step. We recommend that you explore use of the stored programs and DATA step views carefully, but proceed conservatively. The best candidates for compilation are large DATA steps which process relatively little data (i.e., when the compile to execute time ratio is high).

A disadvantage of using stored programs and DATA step VIEWS is that there is no built-in mechanism for managing the DATA step source programs. Since stored programs and views are not portable across host environments and may need to be recompiled for future SAS System releases, you will need to devise a source management mechanism for preserving your original DATA step source code.

CONCLUSION

We have examined a number of tuning controls for Version 6 of the SAS System under MVS. Some of them, such as NOSPOOL and NOIMPLMAC, do not involve resource tradeoffs and are easy to set. Others such as BUFNO, BUFSIZE, and data compression are much less straightforward and require examination of the resource tradeoffs and the resulting impact at your site.

There are some additional measurements available in Version 6 which may help you with these decisions. The FULLSTATS or FULLSTIMER options will print I/O, memory utilization, CPU time and other statistics in the log for each DATA and PROC step.

DATA Step Views and Stored Programs

Version 6 provides the ability to compile and store DATA step programs in two forms: a DATA step view and a stored program. They are kept in a SAS data library as member types VIEW and PGM respectively. DATA step views perform all the same functions of stored programs but are more flexible. Once a view is created it can then be used in place of a SAS data set in procedures and other DATA steps. There are complexities, however. DATA steps which have INFILE, FILE, SET, MERGE, MODIFY or UPDATE statements must have access to the actual external files and SAS data sets to compile properly. If this is not possible, you will have to create model files and data sets in order to compile your DATA step. Given this, it can be quite cumbersome to compile all the DATA steps in a complex program like the MXG BUILDPDB job where data is passed from one step to another. For syntax information and more details refer to SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07.

In spite of the problems associated with compiling complex DATA steps, the pay-back can be significant. For example, a stored program version of the MXG DATA step, which processes SMF data, takes about 19 CPU seconds to compile on a 3090-400S. This time is saved on subsequent executions of the stored program. On an MXG daily PDB run, that savings amounts to an 8% reduction in the CPU time for that DATA step. We recommend that you explore use of the stored programs and DATA step views carefully, but proceed conservatively. The best candidates for compilation are large DATA steps which process relatively little data (i.e., when the compile to execute time ratio is high).

A disadvantage of using stored programs and DATA step VIEWS is that there is no built-in mechanism for managing the DATA step source programs. Since stored programs and views are not portable across host environments and may need to be recompiled for future SAS System releases, you will need to devise a source management mechanism for preserving your original DATA step source code.

CONCLUSION

We have examined a number of tuning controls for Version 6 of the SAS System under MVS. Some of them, such as NOSPOOL and NOIMPLMAC, do not involve resource tradeoffs and are easy to set. Others such as BUFNO, BUFSIZE, and data compression are much less straightforward and require examination of the resource tradeoffs and the resulting impact at your site.

There are some additional measurements available in Version 6 which may help you with these decisions. The FULLSTATS or FULLSTIMER options will print I/O, memory utilization, CPU time and other statistics in the log for each DATA and PROC step.
ACKNOWLEDGMENTS

The authors wish to thank Anne Albright for technical review, Sue Tideman for editorial review, and Terry Poole, Terry Lewis, Sherri Claes and Meg Pounds for their valuable input.

REFERENCES


SAS, SAS/AF, SAS/ASSIST, SAS/FSP and SAS/GRAPH are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. IBM and MVS are registered trademarks or trademarks of International Business Machines Corporation. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.