TUNING VERSION 6 OF THE SAS® SYSTEM UNDER MVS

HELD BY: STEVE MORTON/SAS INSTITUTE

AUTHOR: DANIEL J. SQUILLACE/SAS INSTITUTE INC., CARY, NC

ABSTRACT

This paper presents information on tuning the SAS System in both interactive and batch modes under MVS. Principal areas covered include memory and program management, I/O subsystem characteristics and performance, and performance aspects of some SAS System options. Architectural and implementation differences between Versions 5 and 6 which must be understood to effectively tune the system are also discussed.

INTRODUCTION

As with any other sophisticated software system, there can be significant payback in tuning the SAS System. That payback may not 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 application(s), on how SAS software is used, and on each installation's policies regarding resource utilization and chargeback. We will cover a few "rules of thumb", but the emphasis in this paper is on describing resource utilization tradeoffs associated with tuning choices.

MEMORY MANAGEMENT

MVS/XA and MVS/ESA Considerations

MEMSIZE Option Version 6 of the SAS System uses "above the line" (greater than 16MB or 31-bit) storage for most programs and data. 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= does not limit 31-bit storage, a new SAS option has been introduced for this purpose. MEMSIZE specifies the total virtual storage the SAS system will use. 8MB has proven to be a reasonable value for MEMSIZE. You will want to specify a MEMSIZE limit. SAS version 6 is designed to use memory up to the maximum allowable amount. For example, programs are not deleted from memory until that memory is needed for another purpose. This allows procedures to be invoked multiple times without having to reload the programs.

Memory fragmentation control There is a set of options (also called superblocking options) which serve to control memory fragmentation. These work by setting aside pools or superblocks for several classes of memory allocations. These classes are for 1) the portable supervisor, 2) permanent memory which exists for the life of the session and 3) temporary memory which exists for the life of a task or procedure. Each class has an option specifying an ISA or initial storage allocation amount and an OSA or overflow storage amount. The default values for these options should suffice in most cases and should be left alone. If you wish to look up these options, they are prefixed PSU (portable supervisor), VMP (permanent storage), and VMT (temporary storage).
Out-of-memory condition options

Option SYSLEAVE reserves memory to ensure that at task or procedure termination sufficient memory is available to close data sets and clean up. Option PROCLEAVE serves a similar but less catastrophic requirement for procedures. Some procedures are designed to use memory to the extent available and then continue by opening and using work files. PROCLEAVE ensures there will be enough memory left to open these files and allocate I/O buffers for them.

MVS/370 Considerations

Many MVS/370 systems have restricted private area sizes, often less than 6MB. For these systems, the MINSTG option can be used to minimize the virtual storage used. When MINSTG is set, memory is cleaned up at task and procedure termination; that is, programs are deleted at this point rather than being retained for possible reuse. MINSTG will cause execution time to increase, sometimes significantly. On MVS/370 systems overlayed versions of some procedures are loaded. These alternate load modules start with prefix SOS.

PROGRAM MANAGEMENT

MVS/ESA and MVS/XA Considerations

There are three basic ways to execute the SAS system. The first is to execute a completely unbundled version by invoking program SASHOST. This causes many modules to be loaded separately from your SAS.LIBRARY and significantly increases library directory searches and I/O. This method is not recommended. The second method is intended for installations which do not wish to place SAS modules in the Link Pack Area. Invoking program SASXA1 will execute a partially bundled system in your address space.

LPA Bundle

The most effective way to reduce memory requirements in a multi-user SAS environment is to use the LPA modules. When this is done on MVS/ESA or XA systems, user working sets drop by at least 500K and as much as 1500K. For these systems, there are three LPA modules totaling about 1.5MB. SASXAL occupies 129K of LPA storage below the 16MB line. SASXAL2 and SABXSUPH reside in ELPA above the 16MB line. Since only 129K is required below the line where virtual storage constraint may be a consideration, using the LPA modules is a very straightforward recommendation. Incidentally, SASXAL, SASXAL2, and SABXSUPH can all be renamed to simplify maintenance and release transitions.

MVS/370 Considerations

MVS/370 systems pose a more difficult situation. Virtual storage constraint is often a problem and few installations can afford to devote 1.5MB to SAS LPA modules. So for this environment there is a base 712K LPA module and nine other modules which are eligible for LPA residence. These nine modules cover three broad areas, 1) display manager and full screen support, 2) DATA step compilation and execution, and 3) base SAS engine. This breakdown makes it possible to select LPA modules appropriate for your SAS environment. For example, a heavy batch installation would select the DATA step modules but not the display manager and full screen support modules.

The options of executing a completely unbundled system (SASHOST) and a partially bundled system not in LPA (SAS370) also exist for the MVS/370 environment.
SAS DATA LIBRARY CONSIDERATIONS

Version 6 Data Library Characteristics

The MVS SAS Version 6 data library implementation is much more device independent than that for Version 5. 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 as opposed to a physical track for Version 5 data sets. Page size is not constrained by physical track size. Therefore observations may be considerably longer in Version 6 than in Version 5.

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. (Clifford 1989) goes into considerable detail on page size considerations and the methods the SAS system uses to calculate the page size for a given data set when one is not specified.

The following example illustrates some of these points.

```
x = pages in data set X (page spans 3 blocks)
y = pages in data set Y (page spans 2 blocks)
b = physical blocks (7 per track)
```

data sets X and Y each contain 4 pages

```
Track 1 xxxxx xxxxx x
     b b b b b b b
Track 2 xxx xxxxx yyy
     b b b b b b b
Track 3 yyy yyy yyy
     b b b b b b b
```

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 is that it is relatively efficient across a range of device types and that it leads to lower storage 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. (As you will see shortly, block size is not a primary factor in I/O performance.) An MXG daily PDB, for example, takes 8% more tracks when stored using 6K physical blocks instead of half-track blocks on a 3380. My recommendation is that you put catalogs and data sets into separate libraries. Use a 6K physical block size for the catalog libraries and use full-track or half-track block sizes depending on device type for data libraries.
BUFSIZE and BUFNO tradeoffs  The unit of I/O when processing a Version 6 library sequentially is BUFSIZE*BUFNO blocks where BUFSIZE is the pagesize in use for the data set and BUFNO is the number of page buffers to allocate for the data set. This product is the primary factor 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 cost more virtual and real storage. Buffer storage is above the 16M line in XA and ESA systems.

To explore this tradeoff, data set creating/reading tests were performed on a data set containing 38,000 200-byte observations and occupying approximately 500 tracks on an IBM 3380. The code to create a data set is simply a DATA statement with a DO loop; the code to read a data set is just DATA _NULL_; SET; The metrics presented are elapsed time in seconds, I/O counts (EXCPs), and I/O connect time in seconds. I/O connect time is the time that the device is connected to the channel as measured by the MVS/XA channel subsystem. Minimizing I/O connect time is a very beneficial thing to do on mainframes because path contention is often a limiting performance factor on such systems. In each of the tests, the relative elapsed times are expressed as ratios with the shortest time set to 1.0.

### **** CREATING ****

#### 6K block and page size

<table>
<thead>
<tr>
<th>BUFNO</th>
<th>Elapsed</th>
<th>I/O Time</th>
<th>I/O Time</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>84</td>
<td>(6.0)</td>
<td>21.1</td>
<td>4230</td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td>(3.4)</td>
<td>19.0</td>
<td>2140</td>
</tr>
<tr>
<td>default</td>
<td>3</td>
<td>35</td>
<td>(2.5)</td>
<td>17.8</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
<td>(2.1)</td>
<td>16.9</td>
<td>1094</td>
</tr>
<tr>
<td>5</td>
<td>26</td>
<td>(1.9)</td>
<td>16.0</td>
<td>893</td>
</tr>
<tr>
<td>6</td>
<td>24</td>
<td>(1.7)</td>
<td>15.3</td>
<td>752</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>(1.5)</td>
<td>10.2</td>
<td>612</td>
</tr>
<tr>
<td>14</td>
<td>16</td>
<td>(1.1)</td>
<td>10.3</td>
<td>330</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>(1.1)</td>
<td>10.4</td>
<td>250</td>
</tr>
<tr>
<td>28</td>
<td>14</td>
<td>(1.0)</td>
<td>10.4</td>
<td>189</td>
</tr>
</tbody>
</table>

### **** CREATING ****

#### 22.5K block and page size

<table>
<thead>
<tr>
<th>BUFNO</th>
<th>Elapsed</th>
<th>I/O Time</th>
<th>I/O Time</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27</td>
<td>(2.5)</td>
<td>13.3</td>
<td>1035</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>(1.7)</td>
<td>8.8</td>
<td>521</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>(1.4)</td>
<td>10.4</td>
<td>385</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>(1.3)</td>
<td>8.9</td>
<td>281</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>(1.3)</td>
<td>9.8</td>
<td>248</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>(1.2)</td>
<td>9.0</td>
<td>214</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>(1.1)</td>
<td>9.6</td>
<td>184</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>(1.1)</td>
<td>9.0</td>
<td>163</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>(1.1)</td>
<td>9.5</td>
<td>157</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>(1.1)</td>
<td>9.0</td>
<td>145</td>
</tr>
<tr>
<td>30</td>
<td>11</td>
<td>(1.0)</td>
<td>9.0</td>
<td>77</td>
</tr>
</tbody>
</table>

### **** READING ****

#### 6K block and page size

<table>
<thead>
<tr>
<th>BUFNO</th>
<th>Elapsed</th>
<th>I/O Time</th>
<th>I/O Time</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>83</td>
<td>(6.0)</td>
<td>12.7</td>
<td>4224</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>(3.3)</td>
<td>11.2</td>
<td>2133</td>
</tr>
<tr>
<td>default</td>
<td>3</td>
<td>35</td>
<td>(2.5)</td>
<td>10.7</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
<td>(2.1)</td>
<td>10.5</td>
<td>1087</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>(1.8)</td>
<td>10.3</td>
<td>886</td>
</tr>
<tr>
<td>6</td>
<td>23</td>
<td>(1.6)</td>
<td>10.2</td>
<td>745</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>(1.5)</td>
<td>9.9</td>
<td>605</td>
</tr>
<tr>
<td>14</td>
<td>16</td>
<td>(1.1)</td>
<td>9.8</td>
<td>323</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>(1.1)</td>
<td>9.8</td>
<td>243</td>
</tr>
<tr>
<td>28</td>
<td>14</td>
<td>(1.0)</td>
<td>9.8</td>
<td>182</td>
</tr>
</tbody>
</table>

529
**READING**

*** 22.5K block and page size ***

<table>
<thead>
<tr>
<th>BUFNO</th>
<th>Time</th>
<th>I/O Time</th>
<th>I/O Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>26</td>
<td>8.9</td>
<td>1029</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>8.5</td>
<td>515</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>8.6</td>
<td>378</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>8.5</td>
<td>275</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>8.5</td>
<td>241</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>8.5</td>
<td>207</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>8.4</td>
<td>177</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>8.4</td>
<td>156</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>8.5</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>8.4</td>
<td>138</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>8.4</td>
<td>70</td>
</tr>
</tbody>
</table>

Examination of the preceding four tables shows a marked reduction in elapsed time and I/O count as BUFNO is increased although at a higher cost for buffer storage. Note also that with respect to I/O performance the product of BUFNO and BUFSIZE is the important factor, 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? My recommendation is to use a BUFSIZE between 22K and 24K depending on the underlying block size and to set BUFNO to at least 2. This imposes very little extra storage requirement in the simple case of processing a single SAS data set. However, if multiple data sets are open at once, storage requirements can mount rapidly. For example, an MXG daily PDB run requires 1.1MB of additional virtual storage when run with a block size of 23040, BUFSIZE=23040, and BUFNO=2 instead of the default BUFSIZE=6K and BUFNO=3. However, the elapsed time is reduced by 30%, the I/O count by 60%, and the CPU time by 3%.

Even if your memory resources are unlimited, the above tables show it is not worth going beyond BUFNO*BUFSIZE > 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.

Another consideration of note when multiple data sets are being created simultaneously in the same library is that space is allocated to each data set in discontiguous pieces, the size of each piece being BUFSIZE*BUFNO blocks. See (Bowman 1989) for details.

**Directory allocation** The amount of space reserved for directory blocks is approximately 1.5% of the sum of the primary plus all potential secondary allocations for the library. Over allocation can result for libraries which will contain a few large members or which specify an excessively large secondary extent. Any excess space reserved for directory blocks will be used for data storage, but only after the primary and all secondary extents have been exhausted.

**Data Compression Tradeoffs** Version 6 optionally performs data compression which may be requested globally or at the data set level. Intuitively, a data set containing many long character variables should be an excellent candidate for compression. As it turned out, one of our Data Center problem tracking data sets yielded excellent results. A more interesting situation is posed by data sets containing many numeric variables, such as MICSTM or MXG performance data bases. Even these yielded surprisingly good results as can be seen in the following table. PROC COPY was used to copy an MXG daily PDB and an MICS detailed time span PDB to a Version 6 data library. The CPU time difference between PROC COPY with COMPRESS=YES and COMPRESS=NO and the size of each PDB were used to calculate a per megabyte data compression cost. The tests were run on an IBM 3084.
### Data Compression Tradeoffs

<table>
<thead>
<tr>
<th>application</th>
<th>compression savings</th>
<th>compression cost/mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal</td>
<td>85%</td>
<td>0.61 cpu seconds</td>
</tr>
<tr>
<td>MICS</td>
<td>37%</td>
<td>0.78</td>
</tr>
<tr>
<td>MXG</td>
<td>32%</td>
<td>0.81</td>
</tr>
</tbody>
</table>

The overall CPU cost for data compression can vary considerably. For instance, specifying the global option COMPRESS=YES on an MXG daily PDB job added 8% additional CPU time to the DATA step which creates the PDB data sets, but an additional 32% to the overall job because of the number of times the data is compressed and decompressed in the job. It is an interesting coincidence that we achieved a 32% reduction in space in this case at a 32% cost in CPU time. Whether or not data compression is worthwhile to you depends on the cost allocation policy in your data center. In the case of the MXG example, what is more valuable to you, DASD space or CPU time?

### Version 5 Compatibility Engine Considerations

The device type block size table printed by PROC OPTIONS pertains only to Version 6 libraries. The Version 5 compatibility engine uses an internal table which retains the old Version 5 full-track and half-track defaults. This is important to note because only one block at a time is written to a Version 5 library and small blocks will have a disastrous effect on performance. All will be well unless you specify a small value for the BLKSIZE option in a session or job which will be creating Version 5 data sets. Unfortunately, the BLKSIZE option applies to both Version 5 and Version 6 libraries. Recommendation: Use 1) JCL or ALLOCATE block size parameters or 2) the BLKSIZE(devtype) option to control Version 6 library block sizes.

### OTHER TUNING ISSUES

#### Significant SAS System Options

**MAUTOSOURCE and IMPLMAC**

There is an interaction of the autocall macro facility option MAUTOSOURCE with another option, IMPLMAC, that 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 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, I 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 the WORK library for later retrieval by the %INCLUDE command. Recommendation: Set the default system option to NOSPOOL. I have seen this change reduce I/O by as much as 9% for batch jobs with a large number of input lines.
Display manager AUTOSCROLL command for the LOG and OUTPUT windows
Mainframes are not PCs, so 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 window 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.

Options no longer applicable to Version 6 Some significant tuning options which
apply to Version 5 do not apply to Version 6. These include the macro processing
options MSIZE and MLEAVE, The BLDLTABLE option, the GEN option, and the
CHKPT option. Storing source code generations in data libraries is not done in
release 6.06.

SORT Considerations

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. In previous versions, you had to explicitly choose which sort to
execute. Version 6 introduces the SORTPGM=BEST option which will invoke the
SAS sort program for 500 or fewer observations and the host sort for more than 500
observations. This cutover point is hard-coded and currently there is no way for
installations to change it. For MVS at least, this cutover point is very
conservative. When using 400-byte observations, the point at which the host sort
is more efficient is above 2000 observations. Recommendation: Use SORTPGM=BEST.

The Stored Program Facility

Version 6 provides the capability to compile and store DATA step programs as
PROGRAM members in Version 6 SAS libraries. On the surface the facility is easy
to use. To compile a program, you simply add PGM=ddname.pgmname to the RUN
statement following the DATA step. In its simplest form, you code DATA
PGM=ddname.pgmname; RUN; to execute the stored program. There are
complexities, however. DATA steps which have SET, MERGE, or UPDATE
statements must have access to the actual data sets or models of the data sets to
compile properly. Or, alternatively, the NODSNFERR option must be set and
LENGTH statements must be included for all character variables and all numeric
variables of length less than eight. When RUN PGM=ddname.pgmname is specified,
the DATA step is compiled but not executed. Therefore it is cumbersome to compile
all the DATA steps in a job such as MXG BUILDPDB where data is passed from one
step to another.

In spite of all this, for the right DATA steps the payback of compiling and storing
the code can be significant. For example, the very large MXG DATA step which
processes the SMF tape takes about 19 CPU seconds to compile on a 3090-300S.
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. Recommendation: Explore use of the stored program facility carefully
and proceed conservatively. The stored programs are not portable across host
environments and may have to be recompiled for future SAS System releases. The
best candidates for compilation are large DATA steps which process relatively little
data (i.e., compile to execute time ratio is high). Incidentally, host-dependent
FILE and INFILE options are currently supported by the stored program facility
only in the MVS environment.
SUMMARY

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 FULLSTAT or FULLSTIMER options will print I/O, elapsed time, memory utilization, and CPU time statistics in the log for each DATA and PROC step. Additionally, 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 the Version 6 installation instructions. These records are also supported in release 5.18. Contact Technical Support for further information.

References:


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

MICS is a trademark of LEGENT Corporation, Vienna, VA. USA

MXG is a registered trademark of Merrill Consultants, Dallas, TX. USA