The processing speed of mainframe computers and the capacity of storage subsystems have increased at incredible rates over the past two decades. It is often noted that today's personal computers are more powerful than the mainframes of twenty years ago. Why, then, is it becoming increasingly difficult for computer centers to provide service levels that are adequate to meet the needs of users? One part of the answer lies in the fact that the growth in applications has outpaced advances in hardware technology. There are simply more processing requirements than in the past.

Second, although the advent of fourth generation languages has certainly increased programmer productivity and has brought computing to the less technical end-user, these higher level languages often require more processing power and more storage to yield the same results.

Hardware vendors often point out that the price/performance ratios of their mainframes and peripherals have steadily improved over the same twenty years. One effect of the spread of this information has been a decrease in the emphasis put on program and storage efficiency. Beyond the brief popularity of structured programming, there has been little programmer effort to save resources. Yet the demand for subsecond online response and fast batch turnaround grows louder, amplified even more as computer chargeback bills escalate. Users must recognize that price/performance ratios measure the cost of executing a fixed number of CPU instructions in a given time period, or of storing a certain amount of data. Since today's computing generally requires more resources, these marketing claims can be misleading.

The more utilized a computer becomes, the faster its level of service degrades. A processor that averages 80% busy usually has some periods where CPU bottlenecks exist. Similarly, slowdowns can occur in the channels or storage controllers due to excessive I/O requirements. Queueing theory proves that as resource consumption increases, elapsed processing time increases exponentially. When utilization climbs above 80%, the rate of degradation is alarming.

Attempts to resolve system bottlenecks usually follow one of two courses. The first of these is system tuning. This technique may temporarily solve performance problems, but it does not significantly lower the amount of resources needed by user applications. System tuning also requires a relatively high level of technical expertise. Installations with seemingly well-tuned systems usually resort to hardware upgrades, the second method, for relief. Upgrades are certainly effective, but they are also extremely expensive. The purchase of more powerful processors and more storage capacity should obviously be delayed for as long as possible.

There is a third solution to problems caused by high resource consumption—a solution that is potentially more powerful and is certainly more cost-effective than either of the others. Though often overlooked by the data processing industry, application tuning can actually lower the consumption of CPU, I/O and storage. Applications under development as well as those already in production are targets of this methodology, which consists of recognizing areas in which resources are wasted, and then correcting them. The easiest problems to find are inefficiencies in I/O processing and disk storage, which are usually compounded by causing increased CPU and elapsed times. Processing redundancy is another cause of wasted resources that is easy to spot. Familiarizing oneself with application tuning techniques requires little time and effort; if used consistently, they yield beneficial and satisfying results.

The growth in the SAS System has paralleled that of the industry overall. Though it was initially viewed as solely a statistical package, its constant improvements and enhancements have made it the premier software...
system for data analysis of all kinds. The SAS programming language with all its features is at least as powerful as any in its class. Because of its complexity, SAS software is a good candidate on which to practice application tuning. Becoming more familiar with the language and the PROCs in the basic system gives insight into specific techniques that can have surprising resource saving results. Because of its versatility, the SAS System is also an excellent tool for measuring and reporting the resource usage of other applications and of systems in general. Note that much of the capacity planning and performance evaluation software is written using SAS software.

The following sections of this paper describe some specific areas where simple modifications in SAS coding will save resources and will thus lower elapsed processing time. Also given are some SAS programs that will help make decisions concerning sizing and blocking of disk files.

2. PROC SUMMARY vs. PROC MEANS

Data analysis often requires that information be summarized in multiple ways. An example of such a case might involve a file that contains data on a company's sales. The keys of this file would include sales date, store region and the ID of the product sold. Several reports would be needed from this data; one showing revenues by month and product, a second presenting the quantity sold by month and region, and a third giving regional year-to-date sales by product.

Using PROC MEANS to do the analysis would require two or three sorts and three executions of the PROC. But PROC SUMMARY could do the same job without a sort -- only one PROC execution and a DATA step. The key to the benefits of PROC SUMMARY lies in its CLASS variable. With one input file, the dataset produced by the PROC contains statistics summarized by all combinations of the CLASS variables. Thus, if n CLASS variables are specified, 2^n-1 summary levels (differentiated by the TYPE variable in the output dataset) are produced. A TYPE = 0 output observation summarizes the entire SAS dataset.

When the number of combinations of the CLASS variable exceeds 32,767 (a PROC SUMMARY limit), many users switch to PROC MEANS. But if the original file is split into two or more smaller ones (based on the value of the first CLASS variable), PROC SUMMARY may still be used. One must simply concatenate the output files from the multiple PROC executions to obtain the desired result.

Several tests were performed to measure the relative efficiency of each PROC. These tests, as well as others described in this paper, were executed on an IBM compatible mainframe with a MIP rating of approximately 3. Tests were run under the MVS/SP operating system and the processor was less than 50% busy with other work. The initial data files were made from files of between 50,000 and 200,000 observations, three or four summary variables and three numeric variables to be summarized. For an example like the scenario described above, with three summary levels involved, processing with PROC MEANS required about 160% more CPU time, 500% more I/Os (actually, EXCPs), and 50% more memory (at peak, due to the sorts). Perhaps most important, the jobs using PROC MEANS averaged 300% more run time than those using PROC SUMMARY. When the original data had to be split to use PROC SUMMARY, the benefit was less dramatic, but was still clear -- the jobs completed in less than half the time. Even for examples where only one summary level was extracted (where one sort was required for PROC MEANS), jobs using PROC SUMMARY (with option NWAY) used only 10% more CPU time, actually less I/O and memory, and still completed in less than two thirds the time.

3. EFFICIENT "INPUT" PROCESSING

Most SAS datasets are originally created by reading an external file with the INFILE statement, and then defining various fields through the INPUT statement. Keep in mind that much of the CPU time incurred by such processing is caused by overhead needed to define the variables to the SAS System. The amount of space needed by a SAS dataset is also affected by the way in which the variables are defined.

When only a subset of the input records are to be kept in an output SAS dataset, the subset being determined by the value of some field in the input file, that field alone should be read with a trailing @ (to hold the record) and then tested. Only after it is determined that the record will be kept should the additional variables be defined. In tests on files containing 5,000 to 25,000 10-byte records, there were savings in CPU and I/O (remember that elapsed times for the fifty-field runs were two to three times longer.

A field that is stored using a numeric (display) format but is never used in calculations should be described as a character variable in the INPUT statement. The SAS System is then spared from having to convert the field to an eight-byte floating point representation. A 25,000 record input file containing twenty-five four-byte numeric fields was read with all the variables defined as numeric, then read again with
When numeric variables are needed, use of the LENGTH statement may reduce the space required to store an output SAS dataset. The default SAS length of numeric variables is eight bytes, even if the field is read with only a one byte "informat". Eight bytes provide double precision accuracy of up to seventeen significant digits. In many cases, a length of four (specified before the INPUT statement) is sufficient since it can store seven significant digits. Even a length of two can be used for variables that will always be small integers.

4. SAS LIBRARY "COMPRESS"

SAS data libraries are similar in some ways to OS partitioned datasets. One little-known similarity is that, like partitioned datasets, SAS libraries can accumulate empty space between its members over a period of time. When a SAS member is updated, the new version does not use the same space as had been occupied initially. The unused space (where the old version had been) may be reclaimed by new members when they are small enough to fit, but many SAS libraries contain members that continue to grow over time. In these cases, the wasted space can become significant. Like PDSs, the "used" space reported in a VTOC listing for a SAS library includes this unused space. By running a PROC CONTENTS on the library, it is possible to determine the space that is actually taken up by data. The output from the PROC reports "total tracks used", the space needed by the directory and members, and "high track used", the total space actually used (and the amount reported in the VTOC).

The following test, using a 3380-type device and a SAS blocksize of 23476, shows how quickly wasted space can accumulate. Five SAS datasets were written to an empty SAS library. Each had 10,000 observations and used ten tracks. The datasets were updated, each then containing 20,000 observations. The data took up 100 tracks at this point, but 110 tracks were actually used. After two more updates (30,000 and, finally, 40,000 records per SAS dataset), there were 31 wasted tracks, causing the library to take up 15% more space than was necessary.

The SAS Institute does not supply a PROC that will "compress" a library, but the following procedure will produce the desired result:

```sas
//COMPRESS EXEC SAS OPTIONS='ERGRABEND'
//SASLIBRY DD DISP=OLD,DSN=saslibrary
//TEMPHD0LD DD DSN=holddataset,
//DISP=(NEW,DELETE,CATLG),
//UNIT=SYSDA,SPACE=(space needed)
PROC CONTENTS DATA=SASLIBRY. ALL NODS;
PROC COPY IN=SASLIBRY OUT=TEMPHD0LD;
PROC DATASETS LIBRARY=SASLIBRY KILL;
PROC COPY IN=TEMPHD0LD OUT=SASLIBRY;
PROC RELEASE DNAME=SASLIBRY;
PROC CONTENTS DATA=SASLIBRY. ALL NODS;
```

The "before" and "after" PROC CONTENTS runs can be compared to determine the amount of space that is saved. Note that the PROC RELEASE will free any unused space at the end of the library, potentially an additional savings.

5. DISK BLOCKSIZE CONSIDERATIONS

Dataset blocksizes have a tremendous effect not only on the amount of space that is needed to hold a given amount of data, but on the number of I/Os (and thus CPU and elapsed times) needed to process the file. Tests were run comparing two sequential files, each containing 5,000 100-byte records. One used an efficient blocksize (for a 3380-type device) of 23400 and required 11 tracks; the other used a blocksize of 100 and took 65 tracks. When processed, the inefficiently blocked file generated over 60 times the I/Os, twice the CPU and about four times the elapsed time.

The following formula applies to datasets stored on 3380-type devices:

```
BLKTRK=INT(1499/(15+CEIL((BLKSIZE+12)/32));
```

where BLKTRK is the number of equal length blocks per track
1499 is the number of 32-byte increments per track available for user data
15 = the number of 32-byte increments used by the block overhead, including gaps and the count area
CEIL((BLKSIZE+12)/32) = the number of 32-byte increments used by the data area

The SAS code shown in Exhibit 1 reads information about an existing dataset and computes two alternate blocksizes and the space the reblocked datasets would require. Note that one of the alternates uses half-track blocking, the most efficient user of space on a 3380-type device. The other uses a block size of approximately 6k, I/O-efficient for files that are primarily used in online applications. Sample output from this DATA step is shown in Exhibit 2 (via a PROC PRINT). Note the often tremendous space savings that are possible.

The default blocksize for output SAS libraries is set during installation and can be found by executing PROC OPTIONS. This value can be modified by specifying the BLOCKSIZE= system option. Note that the blocksize SAS uses is not necessarily the value contained in VTOC information on a SAS library (usually 32760).

6. DISK SPACE CALCULATION

When allocating new datasets, it is important to make fairly accurate estimates of the space they will require. The primary allocation should be large enough to accommodate the entire dataset, but should leave little unused space. Secondary space should not normally exceed twenty percent of the initial allocation.
For most OS sequential and partitioned datasets, if record length, blocksize and the number of records are known, space calculation is easy. SAS code for determining space needed on a 3380-type device is shown in Exhibit 3; sample output from this program is in Exhibit 4.

Calculation of space needed by SAS datasets is similarly easy. The following formula, provided by the SAS Institute, is used:

\[
\text{TRACKS} = \text{CEIL}\left(\frac{\text{NBROBS} + \text{TRKSUSED}}{\text{FLOOR}\left(\frac{\text{ TRACKLEN}}{(\text{OBSLEN} + 4)}\right)}\right) + \text{CEIL}\left(\frac{\text{GEN} \times (3000 + \text{SRCLINES} \times 86) + \text{NBRVARS} \times 78}{\text{ITRACKLEN}}\right)
\]

where
- \( \text{TRACKS} \) = the number of tracks required
- \( \text{NBROBS} \) = the number of observations in the SAS dataset
- \( \text{TRACKLEN} \) = the length, in bytes, of one track
- \( \text{OBSLEN} \) = the length, in bytes, of one observation (plus 4 bytes for descriptor information)
- \( \text{GEN} \) = generations of history information to be kept (specified as an option)
- \( 3000 \) bytes = space needed for dataset descriptor information
- \( \text{SRCLINES} \) = the number of lines of source code used to produce the dataset (86 bytes/line)
- \( \text{NBRVARS} \) = the number of SAS variables (78 bytes/variable for attribute information)

Remember in calculating the observation length to use the SAS length of numeric variables (the default is eight bytes). The above formula is only an estimate, but has been reasonably accurate in tests using 3380-type devices and a SAS blocksize of 23476.

SAS code that reads the necessary information and outputs a dataset with space calculations is shown in Exhibit 5. A PROC PRINT output from a sample run is in Exhibit 6.

DATA BKSZDATA;
INFILE DSOATA;
INPUT @01 LRECL 5.
@07 BLKSIZE 5.
@13 TRKSUSED 5.;
BLKTRK = INT(1499 / (15 + CEIL((BLKSIZE + 12) / 32))); /* BLKS/TRK */
RECORDS = BLKTRK * TRKSUSED * INT(BLKSIZE / LRECL); /* # RECS */
IF (BLKSIZE/LRECL) = INT(BLKSIZE/LRECL) THEN VARPAD=0;
ELSE VARPAD = 4;
MAXBKSZ1 = 23476 - VARPAD;
RCDBLK1 = INT(MAXBKSZ1 / LRECL);
/* TRACK */
BLKSIZA1 = (LRECL * RCDBLK1) + VARPAD;
/* (ALT. */
BLKTRKA1 = INT(1499 / 15 + CEIL((BLKSIZA1 + 12) / 32)); /* # NUMBER */
TRKSA1 = CEIL(RECORDS / RCDBLK1) / BLKTRKA1; /* 1 */
MAXBKSZ2 = 23476 - VARPAD;
RCDBLK2 = INT(MAXBKSZ2 / LRECL);
/* TRACK */
BLKSIZA2 = (LRECL * RCDBLK2) + VARPAD;
/* (ALT. */
BLKTRKA2 = INT(1499 / 15 + CEIL((BLKSIZA2 + 12) / 32)); /* # NUMBER */
TRKSA2 = CEIL(RECORDS / RCDBLK2) / BLKTRKA2; /* 2 */

EXHIBIT 1

7. SUMMARY

Several methods for saving computer resources via SAS-based application tuning have been outlined. None requires much time or effort to apply, yet savings in time and money are potentially great. The biggest benefit of presenting these few examples, however, may be the increased awareness of the reader. By making efficiency a higher priority, additional techniques will naturally come to mind. As a result of their increasing use, SAS applications -- as well as systems written in other languages -- will run faster and cheaper.

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

REFERENCES


Dan E. Lodter
Electronic Data Systems Corp.
7450 Boston Blvd.
Springfield, VA 22153
(703) 644-8251
3380 SPACE USED FOR ALTERNATE BLOCKSIZES

<table>
<thead>
<tr>
<th>LRECL</th>
<th>ORIG. BLKSIZE</th>
<th>ALT. 1 TRKS</th>
<th>ALT. 1 BLKSIZE</th>
<th>ALT. 2 TRKS</th>
<th>ALT. 2 BLKSIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>4000</td>
<td>100</td>
<td>23440</td>
<td>86</td>
<td>6080</td>
</tr>
<tr>
<td>80</td>
<td>8000</td>
<td>100</td>
<td>23440</td>
<td>86</td>
<td>6080</td>
</tr>
<tr>
<td>80</td>
<td>12000</td>
<td>100</td>
<td>23440</td>
<td>77</td>
<td>6080</td>
</tr>
<tr>
<td>80</td>
<td>16000</td>
<td>100</td>
<td>23440</td>
<td>69</td>
<td>6080</td>
</tr>
<tr>
<td>137</td>
<td>141</td>
<td>100</td>
<td>23440</td>
<td>22</td>
<td>6032</td>
</tr>
<tr>
<td>80</td>
<td>80</td>
<td>100</td>
<td>23440</td>
<td>15</td>
<td>6080</td>
</tr>
</tbody>
</table>

EXHIBIT 2

DATA NULL;
INFILE INFILE1;
INPUT #06 LRECL 5.
   #12 BLKSIZE 5.
   #18 RECS 9.
; LENGTH LINE1
LINE2
LINE3
LINE4 $ 80 ;
RCDBLK = INT(BLKSIZE / LRECL); /* RD$BLK */
BLKTRK = INT((1499 / 15 + CEIL((BLKSIZE + 12) / 32))); /* BLKS/TRK */
TRKS = CEIL(CEIL(RECS / RCDBLK) / BLKTRK); /* # OF TKS */
FILE FT22FOOl NOTITLES;
LINE1 = "3380 SPACE ANALYSIS FOR "
   TRIM(LEFT(PUT(RECS,COMMA11.))
   " \n   TRIM(LEFT(PUT(LRECL,5.)))
   "=BYTE RECORDS (BLKSIZE="
   TRIM(LEFT(PUT(BLKSIZE,5.)))
   ");";
IF RCDBLK NE 1 THEN LINE2 = TRIM(LEFT(PUT(RCDBLK,COMMA5.)))
   " RECORDS PER BLOCK"
ELSE LINE2 = "1 RECORD PER BLOCK"
IF BLKTRK NE 1 THEN LINE3 = TRIM(LEFT(PUT(BLKTRK,COMMA5.)))
   " BLOCKS PER TRACK"
ELSE LINE3 = "1 BLOCK PER TRACK"
IF TRKS NE 1 THEN LINE4 = "THE SPACE NEEDED IS "
   TRIM(LEFT(PUT(TRKS,COMMA5.)))
   " TRACKS"
ELSE LINE4 = "THE SPACE NEEDED IS 1 TRACK"
PUT @2 LINE1;
PUT @7 LINE2;
PUT @7 LINE3;
PUT @7 LINE4 //;

EXHIBIT 3

546
3380 SPACE ANALYSIS FOR 100,000 80-BYTE RECORDS (BLKSIZE=80):
  1 RECORD PER BLOCK
  83 BLOCKS PER TRACK
  THE SPACE NEEDED IS 1,205 TRACKS

3380 SPACE ANALYSIS FOR 100,000 80-BYTE RECORDS (BLKSIZE=8000):
  100 RECORDS PER BLOCK
  5 BLOCKS PER TRACK
  THE SPACE NEEDED IS 200 TRACKS

3380 SPACE ANALYSIS FOR 100,000 80-BYTE RECORDS (BLKSIZE=6400):
  80 RECORDS PER BLOCK
  6 BLOCKS PER TRACK
  THE SPACE NEEDED IS 209 TRACKS

3380 SPACE ANALYSIS FOR 100,000 80-BYTE RECORDS (BLKSIZE=6320):
  79 RECORDS PER BLOCK
  7 BLOCKS PER TRACK
  THE SPACE NEEDED IS 181 TRACKS

3380 SPACE ANALYSIS FOR 100,000 80-BYTE RECORDS (BLKSIZE=23440):
  293 RECORDS PER BLOCK
  2 BLOCKS PER TRACK
  THE SPACE NEEDED IS 171 TRACKS

3380 SPACE ANALYSIS FOR 100,000 80-BYTE RECORDS (BLKSIZE=6080):
  76 RECORDS PER BLOCK
  7 BLOCKS PER TRACK
  THE SPACE NEEDED IS 188 TRACKS

EXHIBIT 4

DATA ESTSPACE;
  INPUT @01 NBROBS 6. /** NUMBER OF OBSERATIONS **/;
  800 TRACKLEN 5. /** BLOCK LENGTH IN BYTES (47476) **;
  @14 OBSLEN 5. /* TOTAL LENGTH OF ALL FIELDS **;
  @20 GEN 2. /** NUMBER OF GENERATIONS **;
  @23 SRCLINES 5. /** LINES OF SOURCE CODE TO CREATE **;
  @29 NBRVARS 4. ; /** NUMBER OF VARIABLES **;

  TRACKS = CEIL(NBROBS / FLOOR(TRACKLEN / (OBSLEN + 4))
  * CEIL((GEN * (3000 + SRCLINES * 86) + NBRVARS * 78
  / TRACKLEN);

EXHIBIT 5

ESTIMATED SPACE NEEDED FOR SAS DATASETS

<table>
<thead>
<tr>
<th>TRACK LENGTH</th>
<th># OF GENS</th>
<th>LINES OF SOURCE</th>
<th>LENGTH PER OBS</th>
<th># OF VARS</th>
<th># OF OBS</th>
<th># OF TRKS NEEDED</th>
</tr>
</thead>
<tbody>
<tr>
<td>47476</td>
<td>5</td>
<td>30</td>
<td>240</td>
<td>60</td>
<td>10000</td>
<td>53.0</td>
</tr>
<tr>
<td>47476</td>
<td>0</td>
<td>30</td>
<td>240</td>
<td>60</td>
<td>10000</td>
<td>53.0</td>
</tr>
<tr>
<td>47476</td>
<td>5</td>
<td>500</td>
<td>500</td>
<td>80</td>
<td>10000</td>
<td>112.0</td>
</tr>
<tr>
<td>47476</td>
<td>5</td>
<td>5000</td>
<td>500</td>
<td>110</td>
<td>10000</td>
<td>153.0</td>
</tr>
<tr>
<td>47476</td>
<td>5</td>
<td>500</td>
<td>1400</td>
<td>200</td>
<td>10000</td>
<td>310.0</td>
</tr>
<tr>
<td>47476</td>
<td>0</td>
<td>500</td>
<td>1400</td>
<td>200</td>
<td>10000</td>
<td>305.0</td>
</tr>
</tbody>
</table>

EXHIBIT 6