Paper 333-2009

Troubleshoot Your Performance Issues: SAS® Technical Support Shows You How

Michael Williams, Gretel Easter, and Steve Bradsher, SAS Institute Inc., Cary, NC

ABSTRACT

When customers contact SAS Technical Support for help in improving the performance of processes running in the UNIX environment, the answer is frequently as simple as determining which SAS system options need to be modified. This paper demonstrates that if you learn how to analyze a SAS log, and if you know what portable and host SAS options are available to you and how to set them for maximum effectiveness, you can improve the performance of many of your SAS processes without resorting to making changes at the operating system level.

USING THE FULLSTIMER SAS® SYSTEM OPTION TO IDENTIFY PERFORMANCE CONSTRAINTS

When you are diagnosing performance issues, the single most useful tool that you have at your disposal is the FULLSTIMER system option, which writes additional statistics to your SAS log file. You can set this option either during invocation of a SAS session or in an OPTIONS statement within your SAS code. The wealth of information you gain by using this system option is invaluable. Consider, for example, the following note that appears in your SAS® 9.2 log if you have the default system option, STIMER, in effect in a UNIX environment:

```
NOTE: PROCEDURE SORT used (Total process time):
      real time           3:41.74
      user cpu time       1:11.11
```

Contrast that note with this example of information that appears in the log if you use the FULLSTIMER option:

```
NOTE: PROCEDURE SORT used (Total process time):
      real time           3:41.74
      user cpu time       1:11.11
      system time     1:47.47
      Memory                            2110365k
      OS Memory                         2114432k
      Page Faults                       395
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        405317          .
      Involuntary Context Switches      21080
      Block Input Operations            83625
      Block Output Operations           44513
```

Both STIMER and the FULLSTIMER options include information about how much time the SORT procedure requires. In the log, the value for the *Real Time* statistic is the total elapsed real time, also called the “wall clock time.” The value for the *User CPU Time* statistic is the total elapsed CPU time, or the actual amount of time the CPU spends processing the SAS code associated with the SORT procedure step.

The FULLSTIMER option also generates the *System CPU Time* statistic, which specifies the time that the CPU spends performing system overhead tasks on behalf of the SAS process (for example, retrieving data from RAM). This one new statistic is useful because, as a general rule, the User CPU Time value added to the System CPU Time value should be fairly close to the Real Time value. If the Real Time value is much higher, as it is in this example, then you have a problem that needs to be corrected. If, on the other hand, the User CPU Time value plus the System CPU Time value are in line with the real time value, then you know that the process is running as efficiently as it can on the server on which you are running, and only a faster server can make the step or process run faster. This topic is discussed further in the first case study, later in this paper.

The next statistic listed in the FULLSTIMER output is *Memory*, which specifies the amount of memory that is required by PROC SORT. It is important to note that in SAS® 9.1.3, this statistic does not include memory usage for the threaded kernel. PROC SORT is a threaded procedure and a case study later in this paper shows some logs in which the memory usage remains unchanged. OS Memory is a new statistic in SAS 9.2. This statistic represents the amount of memory SAS allocates from the operating system that is to be used by the threaded kernel and by SAS

DATA steps and procedures. In the FULLSTIMER summary at the end of the SAS log, the value for OS Memory reflects the peak memory usage for the entire process.

One of SAS Technical Support's most commonly reported issues is the fact that processes fail as a result of insufficient memory. If a process runs successfully, the FULLSTIMER option produces a grand total at the end of the log that closely reflects the peak memory usage for the process. If your process fails, however, you can still get an idea of how much memory was used by looking at the amount of memory used by the most memory-intensive step.

You are probably familiar with the MEMSIZE system option, which caps the amount of memory available to a single SAS process. If your system has a lot of memory available, it might not be worth the trouble to determine how much memory your process uses; you can simply set the MEMSIZE value higher. However, if memory is scarce, determining the amount helps you arrive at a reasonable value for the MEMSIZE system option.

The *Page Faults* statistic indicates the number of memory pages that SAS tries to access, but that are not in the main memory and that require I/O activity. A consistently high number of page faults that occur across multiple steps might indicate that not enough memory is available to the process. It can also mean that the application is reading a very large sequential file without doing any serious processing, causing high (but normal) values for Page Faults.

The *Page Reclaims* statistic indicates the number of page faults that are serviced without any I/O activity. I/O activity is avoided by the process of reclaiming a page frame from the list of memory pages that are awaiting reallocation. The *Page Swaps* statistic indicates the number of times a process is swapped out of main memory.

Values for the Voluntary Context Switches and Involuntary Context Switches statistics reflect the number of times the process switches from an active to inactive state, and vice versa. A *voluntary context switch* indicates that the process relinquished the CPU. This usually occurs when the process is waiting for the availability of a resource. For example, suppose you have a process that needs to read a large data set. When SAS recognizes that this process will not use a CPU during a lengthy reading, the software efficiently (and voluntarily) releases the CPU to serve another process until the reading is complete.

An *involuntary context switch* indicates that the kernel forces the process into an inactive state. For example, if you have a higher-priority process starting on the system, the kernel forces your lower-priority process into an inactive state while it runs the one with the higher priority.

If the number of involuntary context switches is consistently high, it might be an indication that the step or process is experiencing a computer-resource constraint. At this point, you should use vendor-supplied tools to monitor the CPU to determine whether a resource constraint is the cause.

If you determine that a step or process is not experiencing a resource constraint, check the number of voluntary context switches. A number that is consistently high compared with the number of involuntary context switches might indicate that the process or step is being paged to swap space. This behavior indicates a problem with the operating system's I/O subsystem, and you should also monitor it with your vendor-supplied tools.

High values for any of the statistic discussed here might indicate a performance problem. The best way to determine the cause of high values is to use system tools that are provided by the vendor of the operating system or third-party vendors. For more details on third-party tools, see the section "When SAS System Options Are Not Enough."

SAS® SYSTEM OPTIONS THAT AFFECT PERFORMANCE

When evaluating a performance problem, you need to know precisely what your current SAS option settings are so that you can determine which values might need to be modified. You can use the OPTIONS procedure to generate a complete list of your SAS option settings, as shown in this example:

```
proc options group=(performance memory);
run;
```

The list of options is written to your SAS log file. For the purposes of troubleshooting a performance problem, examine the options that relate both to performance and memory. You can find descriptions of all these options in the SAS 9.2 OnlineDoc, but you should take particular note of the settings described in the following sections because they most affect performance.

MEMSIZE SYSTEM OPTION

The MEMSIZE system option specifies the *limit* on the total amount of memory that is available to a single SAS process. Its primary purpose is to prevent a SAS process from growing too large and consuming too much memory.

UNIX is a shared environment and memory is a shared resource, so be considerate of others. Do not set a MEMSIZE value that enables your process to potentially grow larger than it needs to be. For example, if you have 32 GB of real memory on your server and 10 concurrent SAS processes running, setting MEMSIZE to **`4G`** might result in a shortage of real memory. That said, you should be aware that MEMSIZE does not *allocate* memory, nor does it automatically reserve a certain amount of memory for your process. Its function is simply to limit the size to which a process can grow. Some SAS processes only request and use the amount of memory that is needed to perform their tasks. For example, if you set MEMSIZE to `4G` but only 512 MB is required, only 512 MB will be used. Other processes are not so well-behaved. It is important that you understand this concept because if MEMSIZE is set to a high value for all processes, and other processes are running concurrently with yours, there might not be enough memory available for your process to run. In that situation, your process will fail, and you will receive an out-of-memory error in your log. In some situations, the process can end in a segmentation violation.

Therefore, be conservative when you set a default MEMSIZE value in the sasv9.cfg file. For processes that require a great deal of memory, you can set MEMSIZE at invocation for individual processes. If your process truly requires all of the real memory on the server, set MEMSIZE accordingly for that process, then use either the `cron` or the `at` command to schedule it to run at a time when no others are running.

A further reason to refrain from setting MEMSIZE to a very high value is that processes that use too much memory can cause excessive paging and adversely impact system performance. If, on the other hand, you set the value of MEMSIZE too low for your process to complete, the process fails, and you receive an error. You will have to experiment to determine the best value, and the Memory statistic in the FULLSTIMER summary provides a good idea of how much memory the process actually needs.

The default MEMSIZE value for SAS®9 in the UNIX environment is `128M`, but most likely you will find that not all of your processes can run to completion using that value. A more realistic MEMSIZE value for most processes is either `256M` or `512M`. However, some SAS procedures are memory-intensive, and you need to increase the MEMSIZE value simply to enable the process to run to completion.

Increasing the MEMSIZE value to improve performance is a different issue. In certain situations, giving a procedure more memory than the minimum it needs might enable it to hold the data in memory instead of causing it to create a temporary file. In such a case, you are trading memory for performance. This concept is examined later in "Case Study 2."

In SAS 9.2 Phase 2, you can set MEMSIZE to 0 or to MAX for test purposes. These values do not mean that all of the memory on the server becomes available. With a setting of 0 or MAX, SAS queries how much real memory is on the system, calculates how much memory can be used without swapping, and then uses that value for MEMSIZE.

In SAS 9.1.3 you can set MEMSIZE to 0 for testing purposes, but you should never leave it at that setting because doing so causes MEMSIZE to be set to an impractically high value. In SAS 9.2 Phase 2, these issues are planned to be resolved.

As a general rule, you should not specify a value for MEMSIZE that is greater than the amount of real memory on the server. However, it is possible to do this, as explained in the next section.

REALMEMSIZE SYSTEM OPTION

The REALMEMSIZE system option specifies the amount of *real* memory that the SAS process can expect to have available to it at invocation. By default, REALMEMSIZE is set to `0`, which means it is hardcoded to a value that is 80% of the MEMSIZE setting.

It is possible to set MEMSIZE to a value that is higher than the amount of real memory on the server, taking into account any additional virtual memory that is available. If you find that it is necessary to set a very large value for MEMSIZE, then you might need to lower the value of REALMEMSIZE to a practical limit for the amount of real memory that you expect to have available.

Remember: The value you choose for REALMEMSIZE influences the behavior of SAS processes; therefore, it is critical that you set it to the amount of real memory that the SAS process can expect to have available to it.

SORTSIZE SYSTEM OPTION

The SORTSIZE system option is specific to PROC SORT, but note that it is also used when sorting is performed by the SQL procedure or any other SAS procedure that performs sorting behind the scenes. SORTSIZE is similar to MEMSIZE in that it sets a limit on the amount of memory that is available to the SORT procedure. It does not allocate or reserve memory for the process.

The default value for SAS®9 is `80M` in the UNIX environment. In previous releases of SAS, setting a large SORTSIZE value was a great way to speed up sorting, particularly when the sort could be done completely in memory. Because the speed of file systems has improved, though, this advantage has largely disappeared. Values in the range of 256 MB–512 MB seem to work best, but a smaller value will probably suffice as a default value (the one specified in the sasv9.cfg file). Keep in mind that if you modify SORTSIZE from its default value, you must be sure to maintain the difference between the SORTSIZE and MEMSIZE values or even increase that difference as you use larger MEMSIZE values.

SUMSIZE SYSTEM OPTION

Like the MEMSIZE system option, this option sets a limit on the amount of memory that is available.

The SUMSIZE system option affects only the MEANS, OLAP, REPORT, SUMMARY, SURVEYFREQ, SURVEYLOGISTIC, SURVEYMEANS, and TABULATE procedures. Setting SUMSIZE properly can improve the performance of these procedures by restricting the swapping of memory that is controlled by the operating environment.

You might wonder why SAS provides additional options to control the memory that is used by individual procedures. The case studies later in this paper, illustrate that different procedures have different memory needs. Using these individual memory options, you can specify different settings for multi-step programs that contain procedures with varying needs. A good example of this is a program that contains both a SUMMARY procedure step and a PROC SORT step that has a very large input data set that cannot be sorted in memory. In such a case, you should specify a large value for MEMSIZE so that the value for SUMSIZE (the default value of which is 80% of the MEMSIZE value) is also large. However, you should set SORTSIZE to a smaller value because the data set is too large to sort in memory, and PROC SORT steps generally run faster with smaller SORTSIZE settings.

WORK SYSTEM OPTION

The WORK system option specifies the location of the directory that is to be used as the WORK library. The WORK library contains temporary SAS data sets and other temporary files that are required for SAS processing; therefore the location you choose is crucial. Pointing WORK to a file system that processes quickly, for example, will boost performance. Likewise, you should point WORK to a file system that is separate from the one where your data is stored. The more you can spread the load, the better your performance will be.

A new feature of SAS 9.2 Phase 2 is the ability to define multiple WORK libraries. Case Study 1 demonstrates how doing so can improve performance in some situations. Note that although you can define multiple WORK libraries, a single process uses a single library. SAS chooses which library is used for each process.

UTILLOC SYSTEM OPTION

The UTILLOC system option is used by thread-enabled applications, and it specifies one or more file system locations for the storage of utility files. Thread-enabled SAS applications are able to create temporary utility files that can be accessed in parallel by separate threads, so the file system or systems that you choose for UTILLOC can potentially experience a great deal of I/O activity.

By default, the UTILLOC system option points to the same file system or systems as the WORK system option. However, because of the amount of I/O activity that occurs in these locations, you should point UTILLOC to another file system, or perhaps to multiple file systems. This spreads the high I/O activity over several file systems.

To point UTILLOC to multiple directories via the sasv9.cfg file, use this syntax:

```
-UTILLOC (/location1 /location2 /location3)
```

To point UTILLOC to multiple directories at invocation, use the following command:

```
sas –utilloc \(/location1 /location2 /location3\)
```

In this command, *location1, location2,* and *location3* are paths to directories on different file systems.

As with the WORK option, SAS chooses which library is used for each process. Case Study 1 demonstrates how to bind a particular directory with an individual process and discusses why this is useful.

For Foundation SAS processes, specifying UTILLOC locations on two different file systems and specifying a WORK location on a third file system is ideal. Some other SAS procedures, such as REG and DMREG, might benefit from having additional UTILLOC locations.

THREADS SYSTEM OPTION

The THREADS system option specifies that SAS should use threaded processing if it is available. The DATA step is not threaded, and most procedures are not threaded. For an up-to-date list of which procedures support threading, see the list in “Scalable SAS Procedures” (support.sas.com/rnd/scalability/procs/index.html).

CPUCOUNT SYSTEM OPTION

SAS uses the CPUCOUNT value to calculate how many threads a thread-enabled application will create. By default, the value for CPUCOUNT is **4** or less. For example, when you run SAS on hardware with 12 CPUs, the default value of CPUCOUNT is **4**. When you run SAS on hardware with two CPUs, the default value of CPUCOUNT is **2**.

Capping the default CPUCOUNT value at **4** began in SAS 9.1.3 SP4. In previous releases, SAS set the value of CPUCOUNT to the actual number of processors available. On servers that contain a lot of CPUs, this sometimes resulted in performance degradation because the processes spawned too many threads. The new limit matches SAS Technical Support's observation that the ideal setting for CPUCOUNT is in the range of 4–6 on servers with more than six CPUs. Great performance gain has not been noted when CPUCOUNT is set higher than 6.

Be aware that the function of CPUCOUNT is to tell the thread-enabled applications how many threads to create, not to bind those threads to a specific CPU or group of CPUs. To bind a SAS process and threads to specific CPUs, you must use the operating system tools.

You should never set CPUCOUNT to a value that is higher than the actual number of CPUs on the server. Doing so can hurt performance, not improve it.

CASE STUDIES

The two case studies described in this section are based on the following system details, which include the hardware and file systems that were used, the processor speeds, and the amount of available RAM.

```
System Configuration: Sun Microsystems sun4u Sun Fire E2900
System clock frequency: 150 MHZ
Memory size: 16GB

==========================CPUs==============================
           E$        CPU          CPU
 CPU     Freq       Size        Implementation           Mask  Status   Location
 -------  --------   ---------   -------------------      -----  ------   --------
 0,512  1500 MHz  32 MB      SUNW,UltraSPARC-IV+   2.2   online   SB0/P0
 1,513  1500 MHz  32 MB      SUNW,UltraSPARC-IV+   2.2   online   SB0/P1
 2,514  1500 MHz  32 MB      SUNW,UltraSPARC-IV+   2.2   online   SB0/P2
 3,515  1500 MHz  32 MB      SUNW,UltraSPARC-IV+   2.2   online   SB0/P3

====================== IO Devices =============================
Bus     Freq      Slot +        Name +
Type   MHz       Status        Path                              Model
----      ----      ----------     ----------------------------  ----------------------------
pci      66        PCI0         SUNW,qlc-pci1077,2312 (scsi-+
                   okay         /ssm@0,0/pci@19,600000/SUNW,qlc@1

pci      66        PCI1         SUNW,qlc-pci1077,2312 (scsi-+
                   okay         /ssm@0,0/pci@19,600000/SUNW,qlc@2

pci      66           1         pci100b,35 (network)           SUNW,pci-ce
                  okay          /ssm@0,0/pci@18,700000/network@1

pci      66           2         scsi-pci1000,30 (scsi-2)        LSI,1030
                  okay          /ssm@0,0/pci@18,700000/scsi@2

pci      66           2         scsi-pci1000,30 (scsi-2)        LSI,1030
                   okay         /ssm@0,0/pci@18,700000/scsi

pci      33           2         pci1095,680 (ide)
                  okay          /ssm@0,0/pci@18,700000/pci@4/ide

pci      66           3         pci100b,35 (network)            SUNW,pci-ce
                   okay         /ssm@0,0/pci@18,700000/network@3

pci      33           3          bootbus-controller-sgsbbc     SUNW,sgsbbc
                   okay          /ssm@0,0/pci@18,700000/pci@4/bootbus-controller@3
```

The `saswork` directory is located on a Sun StorEdge 3510 that is accessed by two fiber-attached channels. Total storage is 1.5 terabytes. Drives are 10,000rpm SCSI drives at 146 GB each. There are 12 drives in the enclosure. The file system is set up as a striped partition.

CASE STUDY 1

This case study uses PROC SORT as an example because it is a threaded procedure, and users often experience performance problems with it. The tests in this study were run with SAS 9.2 Phase 2 on a mid-range Solaris 10 server.

To begin, we create a data set to sort. However, before sorting the data set, we set the FULLSTIMER option in order to see all of the statistics in the log file, as shown below. Then, as described in "Using the FULLSTIMER System Option to Identify Performance Issues," we can try different remedies to troubleshoot this performance problem.

```
1          options fullstimer;
2          libname x '/saswork1/sorttest';
NOTE: Libref X was successfully assigned as follows:
      Engine:        V9
      Physical Name: /saswork1/sorttest
3          data x.random;
4             array x (100);
5             do i=1 to 100;
6                 x(i)=ranuni(i);
7              end;
8              do i=1 to 7500000;
9                 output;
10             end;
11          run;

NOTE: The data set X.RANDOM has 7500000 observations and 101 variables.
NOTE: DATA statement used (Total process time):
      real time           1:05.64
      user cpu time       4.50 seconds
      system cpu time     20.91 seconds
      Memory                            757k
      OS Memory                         4392k
      Page Faults                       77
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        788
      Involuntary Context Switches      2375
      Block Input Operations            85
      Block Output Operations           21977
```

This data set is approximately 6 GB in size. Note that there is quite a big difference between the total CPU time (user CPU time plus system CPU time*)* and the real time. This difference occurs because the step performs a lot of I/O activity.

Next, we sort the data set on a single, random BY variable, after which we can look at performance with all of the default settings:

```
    ➢      sas sortrandom.sas


5          proc sort in=big.random out=big.sorted;
6             by x1;
7          run;

NOTE: There were 7500000 observations read from the data set BIG.RANDOM.
NOTE: SAS threaded sort was used.
NOTE: The data set BIG.SORTED has 7500000 observations and 101 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           3:45.95
      user cpu time       1:06.18
      system cpu time     1:35.73
      Memory                            83935k
      OS Memory                         87864k
      Page Faults                       2994
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        495527
      Involuntary Context Switches      9386
      Block Input Operations            53539
      Block Output Operations           31837
```

Earlier, this paper mentions that some procedures benefit from having a large amount of memory available. With that in mind, we run the same program again, this time with the MEMSIZE value increased to **3G** and the SORTSIZE value increased to **2G**:

```
      ➢      sas –memsize 3G –sortsize 2G sortrandom.sas

5          proc sort in=big.random out=big.sorted;
6             by x1;
7          run;

NOTE: There were 7500000 observations read from the data set BIG.RANDOM.
NOTE: SAS threaded sort was used.
NOTE: The data set BIG.SORTED has 7500000 observations and 101 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           3:41.74
      user cpu time       1:11.11
      system cpu time     1:47.47
      Memory                            2110365k
      OS Memory                         2114432k
      Page Faults                       395
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        405317
      Involuntary Context Switches      21080
      Block Input Operations            83625
      Block Output Operations           44513
```

The Real Time statistic decreased only by a few seconds, but the User CPU Time and the System CPU Time statistics increased. This appears to validate the recommendation against setting large values for PROC SORT, but some procedures do benefit greatly from having a larger amount of memory available to them, as is demonstrated in Case Study 2.

Next, we want to see what happens when there is more contention and the server is busier. In this test, we run two copies of the program concurrently with the default system settings to see how performance is affected. The following is a log from one of the processes:

```
      ➢      sas sortrandom.sas &
      ➢      sas sortrandom2.sas &

5          proc sort in=big.random out=big.sorted;
6          by x1;
7          run;

NOTE: There were 7500000 observations read from the data set BIG.RANDOM.
NOTE: SAS threaded sort was used.
NOTE: The data set BIG.SORTED has 7500000 observations and 101 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           7:14.24
      user cpu time       1:18.15
      system cpu time     2:01.05
      Memory                            83935k
      OS Memory                         87864k
      Page Faults                       1125
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        483350
      Involuntary Context Switches      49208
      Block Input Operations            49445
      Block Output Operations           101570
```

Here, the real-time value has increased, at just over seven minutes, and both the user and system CPU times have also increased. Still, the two processes ran in less time than it would have taken to run them sequentially.

Take note of the voluntary and involuntary context switches. The server is now busier, and processes that perform a lot of I/O are ideal candidates for being switched out because they spend a lot of time waiting for system resources.

Therefore, you see here that both of these values have increased. In the next test, we will see if that continues to be true when we run four copies of the program concurrently, again with the default system settings:

- `sas sortrandom.sas &`
- `sas sortrandom2.sas &`
- `sas sortrandom3.sas &`
- `sas sortrandom4.sas &`

```
5          proc sort in=big.random out=big.sorted;
6             by x1;
7          run;

NOTE: There were 7500000 observations read from the data set BIG.RANDOM.
NOTE: SAS threaded sort was used.
NOTE: The data set BIG.SORTED has 7500000 observations and 101 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           16:29.95
      user cpu time       1:19.02
      system cpu time     2:32.87
      Memory                            83935k
      OS Memory                         87864k
      Page Faults                       989
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        578109
      Involuntary Context Switches      96432
      Block Input Operations            209383
      Block Output Operations           248495
```

Look at the real-time value and compare it with the combined value of the user and system CPU times. This test takes longer to run than it takes to run the four copies sequentially. Notice also that the context switches do continue to escalate as the server is busier.

The easiest way to run these processes faster is to take advantage of using multiple WORK directories, which is a new feature in SAS 9.2. Keep in mind that the UTILLOC option defaults to the same location as the WORK library; that means that our utility files will also be spread around, which should help performance.

To use multiple directories, we create a file called workfiles, which contains the paths to the locations we want to use for two WORK libraries. The contents of workfiles are as follows:

```
/opt/saswork
/saswork2/sasutil1
/saswork3/sasutil2
method=random
```

We are going to run four processes concurrently again, so instead of adding the WORK system option to each **`sas`** command, it makes more sense to add the new setting for WORK to the sasv9.cfg file. The contents of sasv9.cfg are as follows:

```
-fullstimer
-msglevel i
-work /saswork1/sorttest/workfiles
```

After the four processes run, the following log is written for one of the processes:

```
5          proc sort in=big.random out=big.sorted;
6          by x1;
7          run;

NOTE: There were 7500000 observations read from the data set BIG.RANDOM.
NOTE: SAS threaded sort was used.
NOTE: The data set BIG.SORTED has 7500000 observations and 101 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           13:45.61
      user cpu time       1:14.76
      system cpu time     1:57.72
      Memory                            83935k
      OS Memory                         87864k
```

```
        Page Faults                        238
        Page Reclaims                      0
        Page Swaps                         0
        Voluntary Context Switches         506315
        Involuntary Context Switches       40954
        Block Input Operations             50340
        Block Output Operations            98874
```

Simply by writing the results to two libraries instead of one, the real-time value decreases by nearly three minutes, or a little over 20%. This automated method of avoiding contention on the WORK libraries is promising, but further improvement is possible.

If we include the UTILLOC paths in each **sas** command at invocation, we maintain complete control over where the files are written. With multiple UTILLOC paths set, I/O activity on each file system is much lighter and, therefore, faster. Similarly, it is also reasonable to include the location of the WORK directory at invocation, spreading it over multiple file systems.

If you have enough file systems, consider spreading out the libraries that contain the input data sets as well. Ideally, spread your I/O activity as much as possible by using unique paths for the WORK and the UTILLOC options as well as for the input and output libraries for each process.

The following example shows the UTILLOC paths added into the **sas** commands in an attempt to further improve performance.

```
➢   sas  -utilloc /opt/saswork  sortrandom2.sas &
➢   sas   -utilloc /opt/saswork  sortrandom.sas &
➢   sas   -utilloc /saswork2/sasutil1 sortrandom3_2.sas &
➢   sas  -utilloc /saswork3/sasutil2   sortrandom4_2.sas
```

```
5          proc sort in=big.random out=big.sorted;
6          by x1;
7          run;

NOTE: There were 7500000 observations read from the data set BIG.RANDOM.
NOTE: SAS threaded sort was used.
NOTE: The data set BIG.SORTED has 7500000 observations and 101 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           8:39.09
      user cpu time       1:09.01
      system cpu time     1:52.92
      Memory                             83942k
      OS Memory                          87864k
      Page Faults                        230
      Page Reclaims                      0
      Page Swaps                         0
      Voluntary Context Switches         509628
      Involuntary Context Switches       16778
      Block Input Operations             53728
      Block Output Operations            76692
```

The first time four concurrent copies of the step were run, the real-time value was 16:29.95.

By splitting the output libraries over two directories and spreading temporary file creation over three separate file systems using the WORK and UTILLOC options, the real-time value for each of the four processes is reduced by nearly eight minutes. That's quite an improvement.

Previously, when we sorted only one copy of the data set, we tried increasing the MEMSIZE and SORTSIZE values, but performance was not much improved.

Now, we set MEMSIZE to **3G** and SORTSIZE to **2G** and sort four copies concurrently. Everything else remains the same.

```
5          proc sort in=big.random out=big.sorted;
6             by x1;
7          run;

NOTE: There were 7500000 observations read from the data set BIG.RANDOM.
NOTE: SAS threaded sort was used.
NOTE: The data set BIG.SORTED has 7500000 observations and 101 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           10:38.85
      user cpu time       1:09.50
      system cpu time     2:12.20
      Memory                            2110372k
      OS Memory                         2114432k
      Page Faults                       6610
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        437244
      Involuntary Context Switches      27516
      Block Input Operations            62618
      Block Output Operations           107701
```

This test does not show an improvement. In fact, performance decreases by 20%. This decrease happens because the current file systems, which cache very aggressively, are better at handling I/O activity than the operating system is at managing the large amounts of memory that are made available to the process.

For more details, see the section "When SAS System Options Are Not Enough."

CASE STUDY 2

Sometimes a step has plenty of memory to run to completion, but setting MEMSIZE higher provides a performance improvement. As mentioned previously, different procedures have different memory needs, and the SUMMARY procedure is one such procedure.

This case study illustrates that the performance of PROC SUMMARY does, in fact, benefit from having additional memory available. First, we create a large data set with 1 million observations and 30 variables:

```
data sum_data (drop=j k m ind);
   array nvars (20) vv1-vv20;
   array cvars(10) $10 cv1-cv10;
   do j=1 to 1000000;
      do k=1 to 20;
         nvars(k)=ceil(50*ranuni(-1));
      end;

      do m=1 to 10;
         ind=ceil(6*ranuni(0));
         cvars(m)=put(ind,words10.);
      end;
       output;
    end;
 run;
```

In this initial test, the MEMSIZE is set to **1G**, which results in the following log:

```
17     options fullstimer;
18         proc summary data=sum_data n min max mean nway;
19         class _character_;
20         var _numeric_;
21         output out=sums;
22         run;

NOTE: Multiple concurrent threads will be used to summarize data.
NOTE: Processing on disk occurred during summarization. Peak disk usage was
approximately 2264 Mbytes. Adjusting SUMSIZE may improve performance.
NOTE: There were 1000000 observations read from the data set WORK.SUM_DATA.
NOTE: The data set WORK.SUMS has 4959465 observations and 33 variables.
NOTE: PROCEDURE SUMMARY used (Total process time):
      real time           1:13.25
      user cpu time       51.55 seconds
      system cpu time     31.60 seconds
      Memory                            711173k
      OS Memory                         716056k
      Page Faults                       100
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        98300
      Involuntary Context Switches      8390
      Block Input Operations            106
      Block Output Operations           8080
```

PROC SUMMARY issues a note saying that it was necessary to use disk space to complete processing. In this case, the recommendation is to increase the value of the SUMSIZE option, but because the default value of SUMSIZE is 80% of the MEMSIZE setting, we simply increase MEMSIZE to **4G**.

```
17
18          proc summary data=sum_data n min max mean nway;
19             class _character_;
20             var _numeric_;
21             output out=sums;
22          run;

NOTE: Multiple concurrent threads will be used to summarize data.
NOTE: There were 1000000 observations read from the data set WORK.SUM_DATA.
NOTE: The data set WORK.SUMS has 4959665 observations and 33 variables.
NOTE: PROCEDURE SUMMARY used (Total process time):
      real time           35.27 seconds
      user cpu time       43.70 seconds
      system cpu time     8.31 seconds
      Memory                            1235522k
      OS Memory                         1240600k
      Page Faults                       3
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        7896
      Involuntary Context Switches      8489
      Block Input Operations            5
      Block Output Operations           1836
```

Notice that in both cases, the real-time value is significantly smaller than the combined user CPU and system CPU times. As you can see, PROC SUMMARY takes advantage of threading very well.

Notice also that when MEMSIZE was set to **4G**, there was a marked decrease in both the real-time value and the user and system CPU times because this process was able to run entirely in memory. We know that this is the case because there is no note in the log to suggest that the availability of more memory can result in better performance.

WHEN SAS® SYSTEM OPTIONS ARE NOT ENOUGH

While the output from the FULLSTIMER option can quickly show you where your problem lies, it might also indicate that the problems are more complex and require the use of host-based monitors.

“Solving Performance Problems: Employing Host-Based Tools” (SAS Institute Inc. 2008) discusses an iterative, disciplined approach to diagnosing performance problems with the SAS System running in the UNIX and Windows environments.

The paper details the employment and interpretation of the most commonly available host-based performance monitors to help you understand what resource constraints existed as the process was running. It also explains how to overlay this information with SAS logs to reconcile episodes of bad performance on the host and on external storage systems (for example, NetApp) to the SAS steps. This process helps identify the performance constraints with SAS processes and, in doing so, leads to their quick resolution.

The paper also includes attachments that contain a log-information macro and monitoring scripts for AIX, HP-UX, Linux, Solaris, and Windows.

CONCLUSION

You can improve performance tremendously by making adjustments to SAS system options. Output from the FULLSTIMER system option offers statistics that help you determine where to look for ways to improve performance. Finally, solving performance issues requires a solid understanding of your environment: the location of the WORK library, what file systems are available to you, and so on. It also requires an understanding of the SAS procedures that you are using because they handle memory in different ways.

RECOMMENDED READING

Brown, Tony. 2008. “Solving Performance Problems: Employing Host-Based Tools.” Cary, NC: SAS Institute Inc. 2008. Available at support.sas.com/rnd/papers/sugi31/practicalperf.pdf.

ACKNOWLEDGEMENTS

We would like to thank Margaret Crevar, Tony Brown, and Clarke Thacher for their assistance and continued support. Thanks also to our colleague Tim Braam for his help with the PROC SUMMARY case study.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Michael Williams
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Phone: (919) 677-8008
Fax: (919) 531-9449
E-mail: support@sas.com
Web: support.sas.com/ts

Gretel Easter
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Phone: (919) 677-8008
Fax: (919) 531-9449
E-mail: support@sas.com
Web: support.sas.com/ts

Steve Bradsher
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Phone: (919) 677-8008
Fax: (919) 531-9449
E-mail: support@sas.com
Web: support.sas.com/ts