

Tuning Tips for Applications That Use SAS/SHARE® Software

Bill Brideson, SAS Institute Inc.
Steve Beatrous, SAS Institute Inc.
Dan Squillace, SAS Institute Inc.
Jan Squillace, SAS Institute Inc.

ABSTRACT

The SAS® System has many tuning options, most of which are left at their default values. When an application accesses data through a SAS server, sometimes the default values provide adequate performance and sometimes they do not.

The SAS System is delivered to you properly tuned for a "typical" application that uses SAS/SHARE® software. SAS Institute makes some assumptions about the kind of processing that is going to take place in a "typical" application. Recognizing that your application may not be "typical", the SAS System supplies tuning options that you can use to override default behavior.

This paper discusses programming techniques and option value adjustments that you can use to improve the performance of your applications that access data through SAS servers. The information in this paper is the result of tuning several large applications that are in use at SAS Institute.

OVERVIEW

This paper will give you some ideas to help you develop SAS applications that make the most efficient use of concurrent access to SAS files. Since this audience is composed of people with different amounts of experience developing applications that use concurrent access to data, in this first part of the paper we will take a few minutes to make sure everyone understands the principles that will be expanded in later parts of the paper.

Since introducing SAS/SHARE software in 1987, SAS Institute has compared two contrasting images to show the additional capability that SAS/SHARE software brings to the SAS System. One image shows a user's SAS session accessing files directly; the other image shows a user's SAS session connected to a SAS server's session and the server's session accessing the files directly. The essential difference is that the data in a file that is accessed through a SAS server travels through two SAS sessions whenever it is accessed. Of course, a SAS server controls concurrent access to the data that is read and written through it, so its overhead has an important purpose. But it is important to remember that data accessed through a SAS server requires more computing resources than data accessed directly.

One or more SAS servers can execute at the same time on a single computer or in a network of computers. You can use different servers for different applications, or you can use a few servers to distribute the load of many applications.

When you use more than one SAS server, each server performs only part of the work load. This allows each server to respond to requests more quickly. On the other hand, every process on a computer requires a certain amount of overhead simply to exist, and SAS servers are no different from other processes in this regard. You must balance the performance improvement that using multiple SAS servers gives your users against the increased load on your system as more SAS servers execute. The later parts of this paper

will discuss measuring how much work a SAS server is doing; you can use that information to determine when to add or delete SAS servers.

THE SAS DATA LIBRARY MODEL

You should make sure you thoroughly understand the material on pages 203 through 209 of *SAS Language: Reference, Version 6, First Edition* before you attempt to tune SAS applications and servers at your installation. Here are some of the terms defined in that material that are most important to understanding this paper:

A SAS data library can have five types of members, DATA, VIEW, CATALOG, PROGRAM, and ACCESS. This paper will deal only with the types DATA, VIEW, and CATALOG.

A library member of type DATA is a SAS data file. Through Version 5 of the SAS System, SAS Institute referred to such files as "SAS data sets." A SAS data file may be compressed, and it may have zero or more indexes.

A SAS data view is a set of directions that tells a SAS view engine how to combine data from one or more sources into observations.

A SAS catalog is a file that contains smaller files; the files contained in a catalog are catalog entries. Some types of entries you might be familiar with are PROGRAM (SAS/AF® programs), SCREEN (PROC FSEDIT screens), and FORMAT (user-written formats).

HOW DATA FLOW WHEN YOU USE SAS FILES

To tune applications that access data concurrently, it is to your advantage to understand how data are read and written in the different types of members of SAS libraries that can be accessed through a SAS server.

It is important to remember that an application cannot run any faster when it accesses data through a SAS server than it can when it accesses data directly. This may seem obvious, but it is surprisingly easy to simply blame an application's sluggish performance on "the server" without ever testing the application while accessing the data without going through a SAS server. For many applications, the difference in performance between accessing the data directly versus accessing the data through a SAS server will not be large. Whenever you develop a new application, verify that the application runs acceptably while accessing its data directly before you add a SAS server to the application's data access.

SAS Data Files

When a SAS session reads from a SAS data file that is accessed directly:

1. The procedure or DATA step requests an observation from the engine.
2. The engine requests the SAS host interface to read the page of the data file that contains the observation.

3. The engine extracts the observation from the page and returns it to the procedure.

When a SAS session updates or adds to a SAS data file that is accessed directly:

1. The procedure calls the engine to replace or add the observation.
2. The engine replaces or adds the observation in the page.
3. The engine calls the host interface to write the updated or new page to disk.

When a SAS session reads from a SAS data file that is accessed through a server:

- 1.0 The procedure or DATA step requests the observation from the REMOTE engine.
 - 1.1 The REMOTE engine determines whether the requested observation is already available in its transmission buffer in the user's SAS session. If the observation is available, it is returned to the procedure.
 - 1.2 If the observation is not already available in the user's SAS session, the REMOTE engine sends a message to the server to get a buffer full of observations, including the observation requested by the procedure.
 - 1.3 The SAS server fills the transmission buffer by requesting one or more observations from the engine that accesses the data file *in the server's SAS session*.
- 2.0 For each observation, the engine in the server's session requests the SAS host interface to read the page of the data file that contains the observation.
- 3.0 The engine in the server's SAS session extracts each observation from its page and returns it to the server.
- 3.1 After filling the transmission buffer, the server sends the buffer to the REMOTE engine.
- 3.2 The REMOTE engine extracts the desired observation from the transmission buffer and returns it to the procedure or DATA step.

When a SAS session updates or adds to a SAS data file that is accessed through a server:

- 1.0 The procedure calls the REMOTE engine to replace or add the observation.
 - 1.1 The REMOTE engine replaces the observation in its transmission buffer or adds the observation to its transmission buffer.
 - 1.2 If the data file is open for *update* access, the REMOTE engine sends a message to the SAS server that carries the new or updated observation and requests that it be updated in or added to the data file.
 - 1.3 If the data file is open for *output* access, the REMOTE engine adds observations to its transmission buffer until the buffer is full. After the transmission buffer is full, the REMOTE engine sends it to the server.
 - 1.4 The SAS server requests the engine that accesses the library *in the server's SAS session* to replace the observation in the data file or add the observation(s) to the data file..
- 2.0 The engine in the server's SAS session replaces or adds each observation by updating and creating pages in the data file.

- 2.1 The engine requests the SAS host interface to write each updated and new page to the data file.

- 3.0 The engine in the server's SAS session returns to the server.

- 3.1 The SAS server replies to the REMOTE engine indicating that the updated or new observation has been stored in the data file.

- 3.2 The REMOTE engine returns to the procedure.

SAS Data Views

The flow of data as a SAS data view is processed can be complex, because a view is a set of instructions that tells how to select and combine data from one or more sources.

A SAS data view can be interpreted in a user's SAS session or a server's SAS session. When a view is interpreted in a user's SAS session, the view file and none, some, or all of the data read by the view can be accessed *through* a SAS server. When a view is interpreted in a server's SAS session, the view file and all of the data read by the view *must* be accessed by the server.

There are three types of SAS views:

- PROC SQL views, which are interpreted by the SQL engine
- SAS/ACCESS® views, which are interpreted by SAS/ACCESS interface engines
- DATA step views, which are interpreted by the DATA step view engine

A view created by the SQL procedure can read SAS data sets (SAS data files and any kind of SAS data view).

When a SAS/ACCESS view engine is used in a SAS server's session, the view engine can only read from the database; it *cannot* update the database. The flow of data is one-way: from the database to the interface engine to the server to the user.

A DATA step view can, like a PROC SQL view, combine data from SAS data files and SAS data views. In addition, DATA step views can include sophisticated calculations and read data from external files. A DATA step view can produce data exclusively by calculation, without reading any data.

SAS Catalogs

SAS catalogs are "containers" for many different kinds of entries, and the data in each type of entry is accessed in a pattern unique to the entry type. Unlike the observations in SAS data sets, the REMOTE engine does not combine the records of a catalog entry into groups. Therefore, it is easy to cause a large number of messages to flow between a user's SAS session and a SAS server without involving a large number of catalog entries.

CONCURRENT ACCESS: UPDATE VERSUS READ-ONLY

Many applications use several SAS files. It is to your advantage if, while designing your application, you identify and segregate:

- the set of files which must be updatable by more than one user at a time
- the files that will be updated by only one user, but while other users are reading the files
- the files that will be updated so infrequently that access to those files by all users is practically read-only

The files in the first group are excellent candidates for access through a SAS server. The files in the second group are often good candidates for access through a SAS server, but for some applications the performance improvement from *not* accessing the files through a SAS server may make it worthwhile to use a more

complicated procedure to update those files "while the users are not around." The files in the third group are almost always poor candidates for access through a SAS server because all of the operating systems that the SAS System runs under provide shared read-only access to files, and that direct access is almost always faster than access through a server.

Here is a summary of the advantages and disadvantages of segregating files into read only and concurrently updated libraries.

- A SAS file that is accessed through a SAS server generally costs more, in terms of computing resources, for users of the application to use than a SAS file that is stored in a library that is accessed directly by the users.
- Reduced traffic through a server optimizes response time for the users of the concurrently updated files.
- Simpler, more direct access to read-only copies of files reduces the cost of an application's query and reporting functions. Note that such a copy may be a subset rather than the entire file.
- A SAS file that is accessed through a server can be updated while it is being queried or reported on.
- Copies of files require disk space.
- A file in a SAS library that is accessed directly by users *can not* be updated while a user executes the part of the application that uses that file.

COMPUTER RESOURCES USED BY A SAS SERVER

The information in this paper so far has been about SAS files and how they are used by an application. You will be a more effective application developer if, in addition to understanding how to make optimum use of SAS files, you also understand the computer resources that a SAS server consumes. That understanding will allow you to design your applications to make optimum use of a SAS server as well as optimum use of SAS files.

A SAS server is an independently running SAS session that brokers requests for data from other SAS sessions. There are 4 kinds of computer resources that a SAS server consumes:

- CPU (the computer's "processor")
- I/O (input from and output to the computer's permanent storage)
- Memory (the computer's working storage)
- Messages (passing data between a server and its users)

CPU, I/O, and memory resources are consumed by every SAS session. "Messages" is a name for one measurable aspect of the complex area of "communications resources;" communications resources are consumed by SAS/SHARE software and SAS/CONNECT® software because these two products enable SAS sessions to communicate with one another.

Any work done by a SAS server consumes more than one kind of resource (if you are looking for simple uncomplicated truths, you may want to skip this section). A SAS server can do several kinds of work and, as you might expect, not all kinds of work consume resources in the same relative amounts. For example, some work a SAS server can do consumes much of the CPU resource but little of the other resources, while other work consumes much of the memory resource, less of the CPU resource, and very little of the other resources.

CPU

A SAS server creates processes as users connect to it and execute DATA steps, procedures, and windows. These processes (created on users' behalf) are assigned the work that is actually performed in the server's SAS session. This allows a process in a server's

session to do work requested by one user and then yield control so that another process can do work for another user.

Most requests handled by the processes in a SAS server require small bursts of CPU time. But there are several requests that can consume especially large amounts of CPU time:

- Processing a WHERE clause
- Interpreting a SAS data view
- Processing a compressed SAS data file

When a SAS data set is accessed through a server, every WHERE clause used to select observations from that data set is evaluated by a process in the server's SAS session. This increases the server's overall use of the CPU resource to reduce its use of the messages resource. Often, evaluation of a WHERE clause can be optimized by using an index to locate the desired observations. But when an index is not used, or selects more observations than satisfy the WHERE clause, the process in the server's session must search for observations that *completely* satisfy the WHERE clause. Searching can consume a significant amount of the CPU resource. While a process conducts a search, it yields periodically to allow other processes in the server's session to do work for other users.

A PROC SQL view can consume quite a bit of the CPU resource. The SQL view engine may join tables, it may need to sort intermediate files, and there may be several WHERE clauses in the view that require evaluation. The process in which the SQL view engine executes yields periodically while a view is interpreted.

DATA step views and SAS/ACCESS views also consume the CPU resource. The process in which either of these engines executes does not yield to allow other processes to run, although the server itself allows other processes to run when a group of observations has been prepared for transmission to a user's SAS session. A DATA step view that does a great deal of calculation while preparing each observation can have a visibly harmful impact on a server's response time to other users' requests.

When a compressed SAS data file is read, processes in the server's session decompress each observation; when a compressed SAS data file is created or replaced, a process in the server's session compresses each observation. In many cases the "wall clock" time required to decompress (or compress) is shorter than the "wall clock" time required to read the additional pages of an uncompressed file. In other words, trading increased use of the CPU resource for decreased use of the I/O resource can, on balance, *reduce* the length of time users wait for a SAS server to respond. While a user processes a compressed data file through a server, other processes in the server's session may execute between groups of observations requested by that user; a SAS data file is not compressed or decompressed *in its entirety* in a single operation.

The "Programming Techniques" section of this paper offers ideas for reducing the CPU consumption of processes in a SAS server's session under the topics:

- Choose the appropriate subsetting strategy
- Index wisely
- Know your application's DATA step views

I/O

Since most work done by the processes in a server's SAS session involves I/O activity, those processes can spend a significant amount of time waiting for I/O activity to complete. (This time includes moving the head of a disk drive to the correct position, waiting for the disk to spin around to the position of the requested data, and transferring the data from the disk to the computer's working storage.) In the current release of SAS/SHARE software,

while a process in a server's session waits for I/O activity to complete, other processes in the server's session do not perform other work that uses a different (CPU, memory, or messages) resource.

That waiting could, it would seem, become a bottleneck for a SAS server, and in a few situations this problem is realized. But in practice most of a server's memory is used for I/O buffers and processes in a server's session typically satisfy most requests for data from I/O buffers that are already in memory.

A SAS server typically allocates memory for one page of a file each time the file is opened, up to the number of pages in the file. For example, if the application being executed by a user opens a file twice, enough of the server's memory to contain two pages of the file is allocated; if ten users run the application, space for 20 pages of the file is allocated in the server's memory. The number of buffers allocated for a file will not exceed the number of pages in the file.

Of course, the pages of the file maintained in memory are not the same set of pages all the time: as users request pages of the file that are not in memory, pages that are in memory are written back to the file on disk if they have been modified, or if an in-memory page has not been modified its buffer is simply used to read the new page.

A larger page size can reduce the number of I/O operations required to process a SAS data file. But it takes longer to read a large page than it takes to read a small one, so unless most of the observations in a large page are likely to be accessed by users, large page sizes can increase the amount of time required to perform I/O activity in the server's SAS session.

There are two patterns in which data is read from or written to SAS files:

- Sequential
- Random

When an application processes a SAS file in sequential order, no page of the file is read into or written from the server's memory more than once each time the file is read or written. Also, observations are transmitted to and from users' sessions in groups, which conserves the messages resource.

In many applications that are used with concurrently accessed files, data is accessed in random order, i.e., a user reads the 250th observation, then the 10,000th observation, then the 5th observation, and so forth. When a file is processed in random order, it is much more difficult to predict how many times each page of the file will be read into or written from the memory of a server's SAS session. In addition, only one observation is transmitted on each message between server and user, which does *not* conserve the messages resource.

The "Programming Techniques" section of this paper offers ideas for reducing the I/O load of a SAS server under the topics:

- Clean up your data files
- Choose the appropriate subsetting strategy
- Choose page size wisely
- Specify sequential access when an SCL program doesn't need random access

Memory

A computer's working storage is used by a SAS server to load programs, hold I/O buffers, and maintain control information. When a server's working set becomes large compared to the amount of memory installed on a computer, a significant amount of the server's working storage may be stored on disk by the operating system's virtual memory manager.

Large amounts of a server's memory are consumed by:

- A SAS data view that contains an ORDER BY clause
- Many indexes on data files accessed through a server
- A large number of files open at the same time
- Data files that have large page sizes

Since the ORDER BY clause causes the observations produced by a view to be sorted every time the view is interpreted, it requires memory to be used for a work area for the sorting step. Your application should only use this clause in its views when it has a clear benefit for your users.

When a SAS data file is opened, *all* indexes on the file are opened. Therefore, when a SAS data file has many indexes, a large amount of memory in the server's SAS session can be used to store pages of the index file and related control information. Of course, when many SAS data files that are accessed through a SAS server *each* have many indexes, this effect is multiplied.

At SAS Institute, we have observed that the majority of SAS servers' memory has been consumed by I/O buffers. Carefully selecting the number of times each file is opened by your application and the page size of each file can have considerable impact on the amount of memory required by a SAS server.

The "Programming Techniques" section of this paper offers ideas for reducing the memory requirements of a server under the topics:

- Choose page size wisely
- Index wisely
- Limit the number of files open during execution of an SCL program

Messages

Messages are the communication events between users' SAS sessions and a SAS server. Whenever a piece of information (for example, an observation) is moved from a server to a user, a message is sent from the user to the server and a reply is sent back from the server to the user.

Messages and replies are transmitted by communications access methods. The cost of a message varies greatly with access method. Memory-to-memory communication within a single computer, for example via the Cross-Memory Services (COMAMID=XMS) or Inter-User Communications Vehicle (COMAMID=IUCV) access methods is very rapid, while messages that flow on cables between computers, for example via the DECnet (COMAMID=DECNET) or TCP/IP (COMAMID=TCPIP) access methods take much longer to travel between SAS sessions.

SAS Institute has observed that the cost of sending data via most communications access methods is more directly a function of the number of messages than the amount of data. In other words, to move a million characters of data between a user and a server, it takes less time to send the data in 100 messages than to send the data in 10,000 messages.

SAS/SHARE software conserves the messages resource by:

- Transmitting observations between SAS servers and users in groups
- Evaluating WHERE clauses in SAS servers' sessions
- Interpreting SAS data views in SAS servers' sessions

The "Programming Techniques" section of this paper offers some ideas for conserving the messages resource under the topics:

- Choose the appropriate subsetting strategy
- Understand and control random access

The "Tuning Options" section shows options you can use to control the grouping of observations on messages between servers and users:

- TBUFSIZE=
- TOBSNO=

MINIMIZING AND OPTIMIZING RESOURCE CONSUMPTION

Now that you understand how the SAS System and SAS/SHARE software use files and computer resources, it's time to apply that knowledge to the design and implementation of your applications.

The most productive way to optimize the performance of your application is *programming* it to work as efficiently as possible. You can almost always realize more performance improvement by coding your application to exploit features of the SAS System than you can gain by adjusting the operation of the SAS System.

When you decide to adjust the SAS System to operate differently, remember that tuning is a balancing act and invariably requires compromise. Of course, to effectively tune the SAS System you must understand what your application's bottlenecks are.

This section will first list some programming techniques that are based on the information presented earlier in this paper. After that, the tuning options of SAS/SHARE software and the SAS System will be described.

Programming Techniques

Clean up your data files: The most obvious way to reduce the amount of work done by a SAS server is eliminating unused variables and observations from the files that are accessed through the server. To make sure that your files are no larger than they need to be, periodically remove or archive unused data.

As a SAS data file matures, users add new observations, update existing observations, and forget about old observations. In most cases the level of activity is greatest on the newest observations. If the users of your application do not frequently access older information, consider moving older observations from files that are concurrently updated to "archive" files that are accessed directly (instead of through a server).

Also as a SAS data file matures, new variables are added, some variables turn out to be larger than they need to be, and some variables lose their usefulness. Periodically check your application's SAS data files for variables that are longer than they need to be and for variables that are no longer used.

While compressing a SAS data file reduces the number of pages in it, compression can not be as efficient at eliminating unused space as you can be by deleting unused observations and variables and by shortening variables that are longer than necessary.

Smaller data files improve the performance of *all* SAS sessions by reducing the amount of disk space required by each file, by reducing the number of I/O operations required to process the data in each file, and by reducing the number and size of messages required to transmit the data in a file between a server and its users.

Choose the appropriate subsetting strategy: Creating a subset of the observations in a SAS file can consume large amounts of the I/O and messages resources. There are several subsetting techniques available in the SAS system:

- any WHERE clause that is not optimized by the use of an index
- the subsetting IF statement of the SAS DATA step
- the FIND, SEARCH, and LOCATE commands of SAS/FSP® procedures

When an index is not used to locate directly the observations that satisfy a WHERE clause, the process in the server's session must read observations from the data file until it finds one that matches the WHERE clause. This can consume a very large amount of the I/O and CPU resources. Those resource requirements can be *greatly* reduced when the variables in the WHERE clause are indexed.

The subsetting IF statement of the DATA step and the FIND, SEARCH, and LOCATE commands of SAS/FSP procedures perform the specified comparison in the user's SAS session instead of in a process in a SAS server. This requires that every observation of the SAS data set be transmitted from the server's session to the user's session, which can consume a very large amount of the messages resource, in addition to the I/O and CPU resources required to read the data file. Since the comparisons of a WHERE clause are performed in the server's session, only the desired observations are transmitted to the user's session and the message resource is conserved.

The I/O resource consumption is the same for all three cases. Using WHERE clauses is recommended, however, because the *messages* resource consumption is higher for the subsetting IF statement and the FIND, SEARCH, and LOCATE commands.

Index wisely: Indexing is a tool that optimizes WHERE clause selection of observations from SAS data sets. A WHERE clause without an index requires the process in the server to read every observation in a SAS data set to find the observations that match the WHERE selection criteria. An index often allows the server to locate the records that satisfy a WHERE clause without having to read the records that do not match.

Adding indexes may be a good idea if your application seems to be taking too long to execute WHERE clauses. However, indexes require extra memory and may present a problem for a server that is memory constrained.

A complete description of index usage may be found in the paper "Effective Use of Indexes in the SAS System," pages 605-614 of the *Proceedings of the SAS User's Group International Sixteenth Annual Conference*.

Look at a clock before you create an index: When a SAS data file is accessed through a SAS server, creating an index on it prevents the server from responding to other users' requests. While it can be useful to create an index while a data file is accessed through a SAS server, indexes on large files should be created "after hours." Indexes on large data files should *not* be created while a SAS server is expected to respond quickly to users' requests.

Choose page size wisely: Larger page sizes can be used to reduce the number of I/O operations required to process a SAS data file. But it takes longer to read a large page than it takes to read a small one and larger pages can increase the memory load on a SAS server.

Large page sizes can be useful if most of the observations on each page are likely to be accessed each time the page is read into the server's memory, or if a large page size causes all or most of a SAS data file to be kept in the server's memory. Otherwise, large page sizes can increase the amount of time required to perform I/O activity in the server's SAS session to the detriment of the server's ability to provide timely response to users' requests.

Understand and control random access: It is often worth the effort to study the order in which the users of your application access the data in your application's files. That tells you how widely dispersed your users' patterns of reference are. Sometimes you can reduce the amount of dispersal by sorting one or more files by a variable (like "date last updated") that happens to correlate (even to a small degree) with your users' pattern of access.

The components of the SAS System used most frequently to access data in a random order are:

- The "n" (position to observation number) command of SAS full-screen procedures.
- The POINT= option of the SET and MODIFY statements of the DATA step
- The KEY= option of the SET and MODIFY statements of the DATA step
- The FETCHOBS() function in Screen Control Language
- The SETKEY() function in Screen Control Language
- Using an indexed variable as a BY variable

Specify sequential access when an SCL program doesn't need random access: The SCL OPEN() function allows you to specify that a file will be sequentially accessed (the default is random access). If your application simply reads all of the observations in a SAS data set from beginning to end once, you will achieve a significant performance improvement by specifying a sequential access pattern ("IS") when the file is opened. (If your application rereads observations, you can not specify sequential access.)

Specifying an access pattern on an SCL OPEN() function is documented on page 223 of *SAS Technical Report P-216, SAS/AF Software, SAS/FSP Software, and SAS Screen Control Language: Changes and Enhancements, Release 6.07*. An example of specifying a sequential access pattern on an SCL OPEN() function is:

```
DSID = OPEN( 'MYLIB.A', 'IS' );
```

A sequential access pattern will restrict an SCL application to those functions which access data sequentially. The SCL functions which access data in a random pattern are:

- FETCHOBS()
- DATALISTC()
- DATALISTN()
- POINT()
- SETKEY()

Limit the number of files open during execution of an SCL program: An open file consumes memory in both users' and servers' SAS sessions. If a SAS server consumes "too much" memory, check the applications that access data through that server to see if any of them open files before they are needed or leave files open when they are not being used.

There are three strategies for using SAS data sets in an SCL program:

- open during initialization of the application and leave open until the application terminates
- open as needed, then leave open until the application terminates
- open as needed, then close as soon as possible

The initialization code of an application is the place to open the SAS data sets that will be used throughout the execution of the application. But if an application's initialization code must open a large number of files, the time it takes to "get started" may be long. By studying how an application is used, you may discover some SAS data sets that can be opened as functions are requested while

the application executes, which can reduce the amount of time the application takes to initialize and reduces the concentration of time required to open files.

Whether they are opened during initialization or later, lookup tables that are small should usually not be closed until an application terminates because the single I/O buffer required by such a lookup table does not require a large amount of memory. In such a case it is frequently economical to use a small amount of the memory resource to conserve the CPU resource that would be required to open and close the lookup table over and over.

Larger SAS data sets, and SAS data sets that are used *extremely* infrequently (for example, once during initialization) or during a seldom-used function (for example, a lookup table on a rarely updated field), should usually be left closed whenever they are not being used.

Evaluate each report's timeliness requirement: Consider how frequently each of your application's reports is generated and how timely the data summarized by the report must be. If a report must be based on current information, it must be based on files that are concurrently updated. A report that does not require "up to the second" information can be generated from files that are directly (and inexpensively) accessed instead of files that are accessed through a server.

For example, a travel agent making reservations or a stock broker making trades require every query to show up-to-the-second information. On the other hand, daily reports or analysis of long-term trends can use data that are "out of date" by several hours, several days, or even several weeks or months.

When copying data from a server, it may be subset horizontally with a WHERE clause and it may be subset vertically with a DROP= or KEEP= data set option. (In relational terminology, the horizontal subsetting is "selection" and vertical subsetting is "projection.") Be sure to take advantage of both methods when copying a file from a SAS server to make the copy of the file as small as possible and thus ensure that reports are generated as efficiently as possible.

Don't forget that files can be stored in users' WORK libraries. It can be most efficient to copy a file that is concurrently updated from a SAS server to a user's WORK library and then use that file more than one time to generate reports. Such a copy of a file contains very timely data yet is not especially expensive to create or use.

A SAS data file that is accessed directly is almost always less costly to use than a file that is accessed through a SAS server.

Be aware of how frequently each file is updated: Many applications contain one or more query functions that use a "lookup file" to offer a user a set of values that are valid to enter into a field. Such a file is read, but never updated, by the majority of the users of the application. Occasionally, values must be added to and removed from the lookup files as the valid input data for the application changes.

A lookup file that is used frequently and updated less often than, say, once a week is likely to be a good candidate for *not* being accessed through a SAS server if it would be easy to find some time during the week when the files can be updated because the application is not being used. On the other hand, a lookup file that is updated many times each day *should*, in many cases, be accessed through a SAS server because updating the file will be convenient: the lookup file can be updated while users use it to perform queries.

SAS catalog entries illustrate another way that update frequency can change:

An application may use only a few or many catalog entries. Like lookup files, catalog entries that are updated frequently are likely candidates for access through a SAS server. But catalog entries

that are never changed, or only changed very infrequently, *should* not be accessed through a SAS server.

The update frequency may change for some of an application's catalog entries over time. For example, while an application is under development and being tested, it can be extremely convenient for the developers of the application to be able to update *any* catalog entry while those testing the application continue with their work. During this phase, the convenience of accessing the catalog entries through a SAS server can more than pay for the cost of the overhead of server access. After the testing is completed and the application has stabilized, some or all of the application's catalogs can be moved to a SAS library that is accessed directly by the users of the application; in this phase efficient access by the users is more important than being able to update the catalog entries conveniently.

Remember that not all of an application's catalog entries must be accessed the same way. Catalog entries that must be frequently updated can continue to be accessed through a SAS server, while other catalog entries that change very seldom can be stored in SAS catalogs that are accessed directly by the users of the application.

Know your application's DATA step views: While it is creating each observation, a process in a SAS server's session that is interpreting a DATA step view does not yield control to allow other processes in the server to execute other users' requests. While DATA step views can be useful in a SAS server, they must be used carefully. A DATA step view that requires a small amount of processing to create each observation will not prevent other processes in a server's SAS session from responding to other users' requests. But a DATA step view that contains many DO loops with many calculations and reads (or even writes) many records in external files or SAS data sets can take a very long time to create each observation. Such a DATA step view should not be interpreted in a SAS server's session because it *does not* yield control until each observation is created.

If it is advantageous to your application for its DATA step views to be interpreted in a SAS server's session, be sure that any external files read by the DATA step view are available to the server's SAS session.

Tuning Options in SAS/SHARE Software

SAS/SHARE software makes some assumptions about the relative values of resources. For example, SAS/SHARE software considers messages to be more expensive than memory so it attempts to use more memory to reduce the number of messages. The default values and behavior may not be optimum for your application, so you have the opportunity to adjust:

- when and in what amounts observations are transmitted in groups instead of individually
- which SAS data views are interpreted in users' SAS sessions and which are interpreted in the server's SAS session
- how frequently a "long-running process" in a server's SAS session yields to allow other users' requests to be processed

SAS/SHARE software automatically attempts to conserve the message resource by transmitting observations in groups whenever possible. Observations can always be transmitted in groups when a data set is being created or replaced, but when a data set is opened for update access it is never appropriate to transmit more than one observation at a time. The grouping of observations when a data set is opened for input depends on the situation; you control *whether* observations are transmitted in groups according to:

- Whether the data set is opened for "random" or "sequential" access
- The control level of the data set

- The use of a BY statement, SETKEY() function, or WHERE clause to select observations from a SAS data set
- The use of the TOBSNO= data set option to override the default behavior

The factors that control *how many* observations are transmitted in each group are:

- The value specified for the TBUFSIZE= option on the PROC SERVER statement
- The value specified for the TOBSNO= data set option

TBUFSIZE= PROC SERVER option: A SAS server allocates a set of multi-observation transfer buffers, called "MOTBs", during its initialization that are shared by all SAS files accessed through that server. On all Release 6.07 and 6.08 hardware platforms the default size of each MOTB is 32,768 bytes. To change this size from the default value, use the TBUFSIZE= option on the PROC SERVER statement.

The value of the TBUFSIZE= option is used by a SAS server to automatically calculate how many observations to combine into each group for *all* of the data sets accessed through it. Therefore, this option can be the easiest way to cause a SAS server to *generally* combine more or fewer observations into each group.

When the data sets accessed through a server have large observations, the number of observations transferred in each message is small when not many observations fit into 32,768 bytes. A small number of observations per message conserves the memory resource at the expense of the messages resource, but since messages tend to be expensive relative to memory that tradeoff will probably not allow your application to perform as well as it could.

The number of observations per group is calculated by subtracting a small amount of overhead from the MOTB size and dividing the result by the length of one observation. For example, consider a SAS data set in which each observation is 10,000 bytes long. The default MOTB size is 32,768. In this case three observations are transmitted in each message when the application opens the data set for sequential access. To increase the number of observations per message to 6, specify the option "TBUFSIZE=64k" on the PROC SERVER statement.

An example of using the TBUFSIZE= option is:

```
PROC SERVER  TBUFSIZE=128K
              <other PROC SERVER options>;
```

TOBSNO= data set option: Independently of the TBUFSIZE= option's effect on a SAS server's overall behavior, you can control the number of observations per group for individual data sets that are accessed through the server. For example, if you specify TOBSNO=3, three observations will be sent in each message.

The TOBSNO= option may be specified wherever SAS data set options are accepted: as an argument to the OPEN() function of SAS Screen Control Language, on the DATA= option of a SAS procedure, and on the SET, MERGE, UPDATE, and MODIFY statements of the DATA step. It must be specified for each data set for which you want grouping behavior different from the default.

When a data set is opened for input with a sequential access pattern, a SAS server calculates the number of observations per group as the smallest of:

- the number of observations in the data set
- 100
- the number of observations that will fit into an MOTB

When a SAS data set is opened for input with a random access pattern, the default behavior is transmitting observations individually (the "group size" is one). This ensures that a user always receives up-to-date data when they position to an observation, and it reduces wasted communications bandwidth because no observations are transmitted to a user's session except the specific observations requested. While a WHERE clause or SETKEY() function is in effect, or while a BY statement is in effect that specifies an indexed variable, the TOBSNO= option is ignored.

At other times, the TOBSNO= data set option may be used to increase the number of observations transferred in each group. For example, consider an SCL program in which a SAS data set "dsid" is passed to a DATALISTC() or DATALISTN() function. The data set is read from beginning to end by the function, and then the observation chosen by the user is reread. Since by default the OPEN() function of SCL specifies a random access pattern, observations for that "dsid" are transmitted individually. But the access pattern of the DATALISTC() and DATALISTN() functions is really "skip sequential", so transmitting observations individually is not optimum. TOBSNO=4 could be specified in a case like this to reduce the number of messages by three-quarters.

The number of observations transmitted when a data set is opened for input is summarized below:

WHERE, SETKEY(), Indexed BY	Access Pattern	
	Sequential	Random
Yes	Maximum	1 (fixed)
No	Maximum	1 (tunable)

An example of using the TOBSNO= data set option is:

```
PROC FSVIEW DATA=MYLIB.A(TOBSNO=10);
```

TBUFNO= PROC SERVER option: Since MOTBs are shared by all data sets accessed through a server, it is possible not to have enough MOTBs. The number of MOTBs a SAS server allocates is specified by the TBUFNO= option on the PROC SERVER statement. By default, four MOTBs are allocated when a SAS server initializes. When WHERE clauses are used frequently to request a subset of data from a server and evaluation of those WHERE clauses is not optimized by an index, a server can spend a long time satisfying each user's request for observations that satisfy a WHERE clause.

Each MOTB is assigned when a request for data is made and is not released until the request is satisfied. Therefore, when a process in a SAS server's session takes a long time to satisfy a WHERE clause, an MOTB is assigned to that process for a long time. Such a situation reduces the server's ability to reuse MOTBs by assigning them to other processes; one solution to that problem is specifying a larger value for the TBUFNO= option. Of course, you should check the WHERE clauses being used to ensure that they can be evaluated as efficiently as possible.

When a user requests one or more observations and a server cannot assign an MOTB to the process that will execute the user's request, the server writes a message to its SAS log, waits for a brief period of time, and tries again to assign an MOTB to the process for the user's request. If there are still no MOTBs available, the sequence repeats. You can tell when the number of MOTBs is not sufficient by the presence of these messages in a server's SAS log.

An example of using the TBUFNO= option is:

```
PROC SERVER TBUFNO=8
            <other PROC SERVER options>;
```

RMTVIEW= and NORMVIEW options: Consider each SAS data view used by your application and determine whether the view should be interpreted in the server's SAS session or the users' SAS sessions. You decide where to have a view interpreted according to:

- How many observations does the view produce?
- How much data is read by the view?
- Where is the data that is read by the view?
- How much work must the computer do to interpret the view?

Some PROC SQL views are especially good candidates for interpretation in a server's SAS session because the number of observations produced by the view is much smaller than the number of observations read by the view, the data sets read by the view are available to the server and the amount of processing necessary to build each observation is not large.

Other PROC SQL views should be interpreted in users' SAS sessions because the number of observations produced by the view is not appreciably smaller than the number of observations read by the view, some of the data sets read by the view can be directly accessed by the users' SAS sessions, and the amount of processing done by the view is considerable.

By default, SAS data views are interpreted in a server's SAS session, but the RMTVIEW= option of the LIBNAME statement enables you to have the views in a library interpreted in users' SAS sessions instead. The NORMVIEW option on the PROC SERVER statement enables you to prevent all SAS data views from being interpreted in the server's session.

SAS/ACCESS views do not provide update access to the underlying database when they are interpreted in a SAS server's session, so it is usually more practical to interpret SAS/ACCESS views in users' SAS sessions.

If it is useful for your application to have a SAS/ACCESS view interpreted in a SAS server's session, make sure that all of the necessary database interface components are available to the SAS server's session.

If a user's SAS session is capable of using a SAS/ACCESS interface engine to access the underlying database, it is more efficient to execute the SAS/ACCESS interface engine in the user's SAS session. Note that in this case it may be convenient to store the view file in a SAS library that is accessed through a SAS server if the view will be updated frequently and used by more than one user.

Like SAS/ACCESS views, DATA step views are very often most useful when interpreted in users' SAS sessions. See "Know your application's DATA step views", above, for more information about interpreting DATA step views in a SAS server's session.

A complete description of the RMTVIEW= option of the LIBNAME statement is on pages 79 and 80 of *SAS/SHARE Software: Usage and Reference, Version 6, First Edition*.

Examples of specifying the RMTVIEW= and NORMVIEW options are:

```
LIBNAME MYLIB 'my SAS data library'
      RMTVIEW=YES
      <other LIBNAME options>;

PROC SERVER NORMVIEW
      <other PROC SERVER options>;
```

LRPYIELD= PROC SERVER option: Some components of the SAS System yield control periodically and can be directed to do so more or less frequently than their default rate. These components are called "long-running processes" and include evaluating WHERE clauses and interpreting PROC SQL views.

Changing the rate at which control is yielded is delicate because the act of yielding control consumes some CPU resource: increasing the frequency at which control is yielded increases a SAS server's CPU consumption all by itself. You can change the rate at which the processes in a SAS server yield control by varying the value of the PROC SERVER option LRPYIELD=. The default value of this option is 10,000; the option has no units.

To make long-running processes yield relatively more frequently, specify a value *greater* than 10,000. While a higher value may have the effect of providing more even response time to a server's users, this comes at the expense of increased consumption of the server's CPU resource. Also, the processes that run for a long time run even longer when they are asked to yield more frequently.

To make a long-running process yield less frequently, specify a value smaller than 10,000. A lower LRPYIELD= value may make some individual user requests (like an SQL join with a sort) complete sooner, but the server's other users are forced to wait as the long-running process does more work before it yields control. Response time can become more uneven when long-running processes are allowed to yield less frequently.

This option is documented on page 40 of *SAS/SHARE Software: Usage and Reference, Version 6, First Edition*.

An example of specifying the LRPYIELD= option is:

```
PROC SERVER LRPYIELD=5000
      <other PROC SERVER options>;
```

Multiple SAS servers: This is not an option you specify on a SAS program statement; instead it is a method of managing the workload of concurrent access to SAS data sets.

If you determine that a SAS server is consuming too much of a resource and you can not reduce the server's consumption of that resource any further, creating an additional SAS server allows you to divide your applications' workload among several servers.

SAS/SHARE software includes a family of SAS macros that help you manage SAS file access through multiple servers. Those

macros are documented on pages 63 through 76 of *SAS/SHARE Software: Usage and Reference, Version 6, First Edition*.

SAS System Options

The SAS System has several SAS I/O tuning options. The options that are most relevant to applications that access data through a SAS server are:

- the BUFSIZE= data set and system option
- the COMPRESS= data set and system option

BUFSIZE= option: When a file is created, use the BUFSIZE= data set option to specify the size of the pages of the file. The SAS System default page size is optimum for files that are processed sequentially, but it may not be optimum when the observations of a file are accessed in random order. PROC CONTENTS shows the page size of a SAS data file.

You might find it useful to balance the pattern in which a file is randomly accessed against the number of observations stored on each page of the file. If most random access sequences access observations in very different locations in the file, then a small page size will improve performance because most of the observations on each page are not used. On the other hand, if most random access sequences are likely to be to observations that are physically near each other in the file, you might be able to take advantage of a large page size to have many of the observations read from the file into the server's memory at once.

If you want to keep all or most of a SAS data file in memory, you can choose a very large page size. Of course, this can consume a lot of the server's memory so you should only use such a page size when you really want to. If you expect that not much data from a large file will need to be in memory at one time, choose a small page size to make reading and writing each page as fast as possible.

If you find that your server is spending a significant amount of time waiting for I/O operations to complete, consider recreating the files that are not used for sequential access with a smaller page size.

An example of using the BUFSIZE= data set option is:

```
DATA MYLIB.A(BUFSIZE=6K);
      SET MYLIB.A;
      RUN;
```

COMPRESS= option: This option is used to cause a SAS data file to be stored in compressed format. Compressing files *usually* makes them smaller, so a server is able to read more observations per I/O request when a file is compressed. The reduction in file size for a compressed file (which conserves the I/O resource) is paid for, though, by an increase in the consumption of the CPU resource (which is used to decompress the observations as the file is read). If your server is CPU-bound, compression could do more harm than good, but if your server is I/O-bound, compression could reduce its consumption of the I/O resource.

USING OPERATING SYSTEM TOOLS

Up to this point, we have been looking at SAS application and server performance from an "internal" point of view. Now we turn to an *external* point of view. By performance externals, we mean several things. First, at what rate is a SAS server consuming resources such as CPU, memory, and DASD I/O? Second, with what other workloads is a SAS server competing for these resources? And third,

what policy is being used to manage a SAS server's access to resources with respect to other work in the system?

There are several monitors available for MVS and VM to help you analyze a SAS server's resource utilization and contention with other workloads. On MVS, most sites license IBM[™]'s RMF product. RMF Monitor II and Monitor III support interactive analysis of SAS/SHARE performance. Also available on MVS are Candle Corporation's Omegamon and Landmark System's TMON for MVS. Prominent products on VM include Omegamon from Candle Corporation and XAMAP and XAMON from Velocity Software.

Questions these monitors can help you answer include:

- Are my servers getting appropriate access to resources?
- Is another workload causing a severe contention problem for one of my servers? For example, is my server fighting with another application over access to the same disk drive?
- What resource bottlenecks are most critical to my applications? Where should I direct my tuning efforts?
- Often, solutions to resource utilization problems result in making tradeoffs among resources. For example, you may be able to reduce I/O by allocating additional buffers. But the additional buffer allocation will take more memory. Use of one of these monitors can help you evaluate the effectiveness of the tradeoff.

It is beyond the scope of this paper to tell you exactly how to use specific operating system performance monitors. We are making the non-trivial assumption that you or someone else on your staff have that knowledge. Basically, every system has three principal resources: CPU, I/O, and memory. We will look at examples of managing each of these for SAS servers:

Managing CPU

The most critical factor here is assuring that your servers are getting a reasonable share of the available CPU time. Servers in general ought to run at a higher priority than clients and at the same priority as other types of server or service work on your system (transaction monitors, database servers, etc.). You can tune your SAS/SHARE application meticulously only to be foiled if batch work (for example) is preventing your servers from getting CPU time.

If CPU time is a scarce resource on your system, that is your system is generally running at very high CPU utilizations, then you need to consider SAS/SHARE tuning actions which can reduce CPU time. Two specific examples are type of server connection and whether or not to use data compression.

Managing I/O

The first thing to consider here is the amount of contention with other work on the system. Are your SAS libraries competing with other work on channels, disk controllers, or disk drives which are too busy? "Too busy" on I/O channels and control units is highly specific to each operating system and hardware vendor. But in general it is safe to say that if a disk drive is consistently above twenty percent busy, then off-loading work from that drive ought to be considered.

If there is no significant contention with other work, then you need to consider spreading application libraries using SAS/SHARE across multiple disks.

If "waiting for I/O" is still a problem for your servers, then you need to consider SAS/SHARE tuning options which can reduce I/O time. These include using smaller page sizes for randomly-accessed data, adding indexes for randomly-accessed data, and possibly using data compression. Data compression is a specific example of the resource tradeoff problem mentioned earlier. Data compression can reduce I/O and disk storage but will increase CPU time.

Managing Memory

Memory is an interesting resource in that it directly affects both CPU and I/O resource consumption. Too little memory increases both. Additional memory can reduce both. The most critical factor here is to ensure that your servers have sufficient memory to prevent excessive wait for paging. Most operating systems have controls to differentiate the amount of memory given to various workloads on the system.

If real memory is a scarce resource on your system, then you need to consider SAS/SHARE tuning actions which reduce memory consumption. Chief among these are reducing data set page sizes to reduce I/O buffer memory requirements and using shared SAS system images where possible.

CONCLUSION

The concurrent access capabilities that SAS/SHARE software adds to the SAS System give developers opportunities to create applications that allow their users to have up-to-date data and to be more productive.

Such applications use the SAS System in new ways. This paper has discussed areas to be aware of and ways to trade usage of one resource for another. This information enables developers of applications that take advantage of concurrently accessed data to write those applications to use the available computer resources in the most efficient ways possible.

WHERE TO FIND MORE INFORMATION

For more information about compressed SAS data files:

- *SAS Language: Reference, Version 6, First Edition*, pages 237, 718-719, 729-730, 744-745, and 773-774
- *SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07*, page 19
- *Proceedings of the SAS User's Group International Seventeenth Annual Conference*, "I/O Performance Improvements in Release 6.07 of the SAS System under MVS, CMS, and VMS," pages 960-964

For more information about indexes on SAS data files:

- *SAS Language: Reference, Version 6, First Edition*, pages 199, 217-225, 237, 238, and 498
- *SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07*, pages 20-21, 41-42, 82-83, 119, 177, 180, and 191
- *Proceedings of the SAS User's Group International Sixteenth Annual Conference*, "Effective Use of Indexes in the SAS System", pages 605-614

SAS, SAS/ACCESS, SAS/AF, SAS/CONNECT, SAS/FSP, and SAS/SHARE are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.

IBM is a registered trademark of International Business Machines Corporation.

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

CONTACT INFORMATION

Bill Brideson
SAS Institute Inc.
SAS Campus Drive
Cary, North Carolina 27513

email: saswib@unx.sas.com
voice: 919-677-8000 x7627
FAX: 919-677-8123