Efficiency Ideas For Large Files

J. Meimei Ma, Quintiles, Research Triangle Park, NC
Andrew H. Karp, Sierra Information Services, Inc., San Francisco, CA

INTRODUCTION

This tutorial presents options you should consider implementing when using the SAS System for handling large files. We touch on a variety of topics and approaches, as opposed to dealing with only a few in depth. Using compression and index files are among the storage techniques covered, along with ways to plan your programming strategy to incorporate appropriate efficiency concepts.

The strategies and hints discussed are directed toward programmers, data managers, project managers, and anyone else interested in cost effective solutions to SAS System programming problems. Only basic knowledge of SAS programming (DATA step and procedures) is assumed. The techniques are applicable to essentially all hardware platforms, although at times we address a few operating system-specific issues.

TERMINOLOGY

Efficiency Elements

In considering efficiency, remember that more than one component of this concept exists. Your goal should be to balance all elements, which may often involve setting priorities differently for each situation, rather than blindly applying a stock series of approaches to every programming requirement. In other words, you must understand the specific problem you are facing from many viewpoints before you can choose the best solution for that problem.

The primary efficiency elements separate into two main categories: *machine* and *human*. Machine elements include: computer processing time for calculations or data manipulations, called CPU, and processing time for reading or writing computerized information, called I/O, for Input/Output. Elements related to human efficiency include: programmer time, programmer experience level, and the priority attached to producing results quickly rather than inexpensively. Major activities that require programmer time are

- planning
- writing new code
- learning about new techniques
- revising or running existing programs
- maintaining project programs
- maintaining documentation.

One important element in choosing the most efficient solution for a given situation is *repetition value*. A program destined to be run only once is a less likely candidate for complex programming logic that emphasizes maximizing machine efficiency. Taking into account the cost of detailed testing may result in deciding that a more direct approach that saves human time is better. Conversely, the process of designing a program or set of programs that will be run once a month for the next three years against a multi-million record data set should include careful consideration of machine efficiency techniques. A program or application requirement with a higher repetition value often translates into a greater emphasis on machine efficiency over human efficiency.

Large File Characteristics

When does a file become *large*? Although the actual number of observations and/or variables that make a file "large" depends on your computing environment, we can describe common characteristics of "large" versus "small" files.

A large file may be *short and wide*, which means having relatively few observations but a large number of variables, or *long and narrow*, in which only a few variables exist for many observations. In a mainframe computing environment a large file may have as few as ten thousand records or millions of records. Batch processing using tape storage, as opposed to interactive processing, is still standard procedure in large mainframe computing environments. For computing environments using microcomputers or minicomputers, the point at which a file becomes large is heavily dependent on the particular computer system's capacity. Any data warehouse—regardless of the operating environment in which it resides—is generally composed of a number of interrelated large files, which increases the complexity of the programmer's approach to efficiently processing the data.

Working with large files is much easier when the programmer has a high level of knowledge about

- the contents of the database(s),
- SAS System capabilities
- their operating environment.

This construct applies to SAS data sets, raw files that are read with INPUT statements, or any other database files. Without enough information about a large file, just knowing where to start can be difficult, and often introduces human—in addition to machine—*inefficiency* into the programming process.

A project based on data from large files—often using programs with a high repetition value—profits greatly from deliberate planning for careful processing. Regardless of the computing environment, large file processing is expensive in terms of programmer time and computing resources. Frequently a dollar value is associated with computer processing in large projects. A chargeback system may exist that bills users for CPU utilization, tape mounts, I/O counts or other computing resources.

Any attempted production run that processes all the data in a large file becomes a significant event. The strategy of doing multiple test runs using only a portion of a data set is usually more cost-effective when working with large files than repeatedly processing the entire data set.

STRATEGIES

General Strategy

To achieve maximum efficiency,

- plan, experiment, revise the plan
- work with sample data
- stay flexible
- document everything.

Think First

Start by allowing time for planning, experimentation, and more planning. The more you know about your data, your hardware environment, and the capabilities of software available to you, the better prepared you will be to design efficient programs. There are few absolute guidelines. Experimenting with actual data is often the only way to determine if one option is better than another.

The importance of understanding the data with which you are working is often overlooked until it is too late. Insist that those with whom you are working explain the process used for data collection, key definitions and assumptions underlying the values of variables in the data set, and how these values and variables might change in future data sets that may be used in future iterations of the same project. Merely working from a copy of a file layout or CONTENTS listings is almost never enough!

Examine Multiple Options

Flexibility is another key to finding an optimal solution. Try to keep an open mind about everything, from DATA step logic options to changing hardware platforms or storage methods. Do not expect that one solution will fit all situations. Is the amount of human time required to produce the desired result more critical than computer cost? Will the program be used just once, or is it likely to evolve into a crucial part of an on-going system? Whenever possible, start by examining critically your most fundamental assumptions.

Documentation Is Important

As always, creating documentation during development and all programming stages is important. While SAS data sets are self-documenting (especially if care is taken to add LABELS and FORMATS), supplemental written documentation that describes why certain choices were made is very helpful and is often neglected during program development. All experimental benchmarking programs should be kept with at least rough notes about the results. Not only does such a history provide justification for the current project, you may benefit during the next planning cycle. Documentation of programs (comments, indentation, flow charts) and output (TITLEs, FOOTNOTEs, dates, page numbers) improves human efficiency by reducing the possibility of miscommunication. Time spent documenting as a good investment, not a necessary evil.

Programming Strategy

Writing efficient programs involves using appropriate coding techniques and thorough testing procedures. However, be cautious when considering methods that were developed with earlier releases of SAS software. Changes in architecture and internal execution logic may invalidate the advantages of certain strategies used in prior efforts. The trick is to find out about alternatives worthy of the effort required for experimention.

Consider joining a local user group or at least talking with other programmers in your company. A code review session with other SAS System users often identifies problems as well as optimal strategies to pursue. The hints and reading list provided at the end of this paper can further your search for new ideas. For longer term projects, check out the latest on new techniques and capabilities at www.sas.com if you have Internet access.

Remember that the process of testing programs, especially on large files, is an art, not a science. Try to think as much about the testing process as the coding process. One important principle is that properly defined test subsets provide more efficient test results, in both human and machine terms.

Storage Strategy

Regardless of which programming methods you use, data storage is another major aspect of developing efficient large file projects. The SAS System includes a variety of options:

- compression
- the use of indexes
- access to other database systems (ORACLE®, DB2®, etc.).

The storage hints presented here may inspire more questions than provide answers, but that is the intention. There are no shortcuts to choosing the most efficient storage method.

TIPS AND TECHNIQUES

Programming Hints

The most basic programming principle to remember is that the SAS System is composed of DATA steps and procedures. Do not assume that DATA step programming is automatically more efficient than using procedures. Even machine efficiency may be better with procedures. Using PROC SORT and PROC MEANS can often obviate the need for complex DATA step programming. PROC DATASETS helps you manage SAS data libraries and modify SAS data set descriptors without recourse to the DATA step. Complex DATA step-based reports may be replaced by judicious use of PROC PRINT, PROC REPORT, and (to a lesser extent) PROC FREQ.

We recommend you learn more about the following features, even if you have used them before (see Recommended Reading for places to start). They are basic building blocks for creating machine efficient program code.

- WHERE statement in DATA step or procedure
- WHERE data set option (DATA statement or procedure statement)
- IN operator, for clarity and less typing
- STIMER or FULLSTIMER (evaluation)
- Logical PDV
- Compiled DATA steps (PGM option)

For programmers who are familiar with SQL (Structured Query Language), human efficiency may be improved by using PROC SQL instead of more traditional DATA step code. In addition, there are situations in which SQL processing is more machine efficient than DATA step processing.

Internal Documentation of SAS Data Sets

One key feature of SAS software is the self-documenting nature of SAS data sets. Understanding the descriptor portion of a SAS data set is key to gaining human efficiency. Attaching LABELs to both the data set and its variables—as well as using FORMATs to modify the external representation of values of variables stored in SAS data sets—greatly increases the ability of SAS System programmers to understand how to manipulate and analyze SAS data sets. You should understand the features of PROC CONTENTS and PROC DATASETS, which can be used to view and manipulate the descriptor portion without a DATA step.

Save Time By Avoiding Confusion

Naming conventions help save time and money (and frustration) by preventing confusion. In Version 6, the 8-character limit on the names of SAS data sets and variables may seem to be a hindrance, especially compared to other software that allow longer names. However, you can still avoid confusion when dealing with variables in a SAS data set by adopting a common prefix for groups of variables. Also, consider sequentially numbering related groups of variables (e.g., MONTH1, MONTH2, etc.) to allow reference to the series using shortcuts in the SAS programming language.

Another potential source of confusion worth avoiding was mentioned earlier: understanding the data set itself. More than once, both of us have had to redo major portions of a project because the meaning of one or more variables was not adequately explained at project inception or changed during the course of the project. The programmer should be an integral part of the project team, not just an adjunct "worker bee" who hears about decisions after the fact. Perhaps more human efficiency has been lost because of this type of limited thinking than all others combined!

Basic Storage Hints

A SAS data set remains a more efficient storage format than a raw file, but note that two forms exist. A SAS data file (memtype=DATA) stores both descriptive information and data values within the SAS System. The other form is a SAS data view (memtype=VIEW), in which the data values may be in another database system entirely. As far as most SAS System procedures that read data sets are concerned, these two forms of a SAS data set are identical. From an efficiency standpoint, no general recommendation is possible.

Using DROP/KEEP statements or data set options to limit the number of variables in the Program Data Vector is always recommended. The only risk is in leaving out potentially useful variables when creating a subset of a larger file. Plan ahead to minimize potential problems while you benefit from increasing machine efficiency.

Keep in mind that the ATTRIB statement incorporates the functions of LABEL and FORMAT. Also, you can use the DATASETS procedure to add labels and formats to an existing SAS data set. While all SAS data files are theoretically self-documenting, additional documentation of variables still depends on programmer effort. Consider creating LABELs for the most commonly used variables. Of course, begin by choosing meaningful variable names instead of VAR1, VAR2, or X, Y. Human efficiency increases directly with the level of documentation available, especially for long-term projects.

Change LENGTH Cautiously

A common suggestion for reducing the storage requirements of a SAS data file is to specify LENGTH to change the storage space required for numeric variables. With mixed hardware platforms, this is a more difficult concept. For instance, on IBM mainframes LENGTH=2 is appropriate for small integer classification variables, e.g., 1=Yes and 2=No. However, in many hardware platforms, LENGTH=3, or even LENGTH=5, is the minimum possible length for a numeric variable. Downloading a SAS data file from a mainframe to other operating system can actually include an automatic "promotion" of numeric variable lengths to protect against losing significant digits. Reducing storage by using LENGTH is still worth considering, but now requires a deeper understanding of the potential problems, as well as operating system specific issues beyond the scope of this tutorial. Using character variables instead of numeric ones may be a simpler solution for categorical data.

Complex Storage Hints

The SAS System includes a variety of storage options. Depending on the situation, large files may be processed more efficiently if they are compressed, have index files, or are stored in another database system.

Compression

A compressed data set is often smaller than the original, but the I/O and storage space benefits should be balanced against the additional CPU needed to access records from the compressed data set. Users should be aware that under some circumstances compressing a SAS data set will result in a larger file being created, especially if there are no character variables with blank spaces to be compressed.

The compression algorithm is based on translating identical consecutive bytes into a maximum of three bytes. Blanks and binary zeros become two bytes while other repeated values shrink to three. Compressed records are variable length as opposed to the fixed length records in uncompressed data sets. As a result certain functions are no longer possible, in particular the POINT= random access ability of the SET statement to locate a record by observation number.

To save even more space, a REUSE option exists that applies when records are changed or deleted. One result of trying to use old space is that a replacement record may not fit. In such situations, the old space will contain a pointer to the new data values. Thus, records are no longer necessarily in sequential order.

To compress or not compress is also an issue for transport files. Using PROC CPORT/CIMPORT may produce significantly smaller files than relying on the specifying the XPORT engine with PROC COPY.

Storage space savings can be substantial when using compression. For instance, the savings can be 30-75% for many data sets used in clinical trials research. Little programming effort is needed to create compressed data sets if the system option COMPRESS=YES is used. In addition, all work data sets are then compressed as well as the permanent data sets (with two level names like SASD.MONTHLY). In many computer environments, compression is well worth trying.

Indexes

Using one or more indexes is well suited for large files that do not change often, but are analyzed frequently using standard selection or sort variables. WHERE statements will use an index file to read records more efficiently based on an automatic "cost" algorithm that assumes that values of index variables are uniformly distributed. If an appropriate index exists, then the SORT procedure is not necessary when doing BY processing. Although you cannot control when an index file is used, setting MSGLEVEL=I as a system option will provide a note in the SAS log about index usage.

The disadvantages of using indexes are the extra space and CPU required, along with the increased complexity of maintaining the data set. PROC CONTENTS output includes information about all index files associated with a data set. Under normal circumstances the actual updating of index files is handled automatically, but occaisonally you will need to repair an index using the DATASETS procedure. Well planned choices of which variables to use to create indexes can assure that resources are not wasted maintaining unused indexes. In general, if you expect to regularly retrieve more than one-third of the observations or the data set is smaller than three memory pages then an index is not appropriate.

Data Views

Going beyond the SAS System to store data is clearly most appropriate when the alternative database management system is already in use. By defining a data view, most SAS System procedures can be used on data stored and maintained elsewhere. If data values change constantly, then using a data view assures that results reflect the most recent information. However, there may be an added cost associated with the retrieval. When multiple procedures are planned for a relatively small subset, creating a SAS data file may still be more machine efficient.

CONCLUSION

You have many choices for where to start when you need to do thorough investigations of efficiency techniques that will apply to a specific situation. SAS software provides many options to choose from, but the variety of alternatives means that identifying the optimum decision is not always obvious.

In this era of seeking the most cost effective solution—whether your priorities are machine or human—planning is absolutely critical if your goal is to maximize efficiency. Ultimately, achieving your efficiency goals will require a careful balancing act.

TO ACHIEVE EFFICIENCY

Expect a Balancing Act

Cost

Time

Computer resources

Human resources

USEFUL SAS PROCEDURES

The following SAS procedures are useful for investigating large files. Always use appropriate TITLE statements for printed output.

COMPARE	
CONTENTS	
COPY	(converting transport files)
CPORT/CIMPORT	
DATASETS	(CONTENTS, handling index files)
FORMAT	(FMTLIB option)
FREQ	
MEANS	(incorporates SUMMARY)
PRINT	
SORT	
SQL	
SUMMARY	

RECOMMENDED READING

Books By Users

Calvert, William and Ma, J. Meimei, *Concepts and Case Studies in Data Management,* Cary, NC: SAS Institute Inc., 1996. 150 pages.

DiIorio, Frank, *SAS Applications Programming: A Gentle Introduction,* Boston, MA: PWS-KENT Publishing Company, 1991. 704 pages.

Jaffe, Jay A., *Mastering the SAS System, Second Edition*, New York, NY: Van Nostrand Reinhold, 1994. 592 pages.

Miron, Thomas, *SAS Software Solutions: Basic Data Processing*, Cary, NC: SAS Institute Inc., 1993. 234 pages.

SAS Manuals

These manuals were written and published by SAS Institute, Inc., Cary, NC, USA.

SAS Language and Procedures: Usage, Version 6, First Edition, 1989. 672 pages.

SAS Language and Procedures: Usage 2, Version 6, First Edition, 1991. 688 pages.

SAS Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition, 1989. 210 pages.

SAS Programming Tips: A Guide to Efficient SAS Programming, 1990. 176 pages.

Advanced SAS Programming Techniques and Efficiencies Course Notes, 1992. 412 pages.

Combining and Modifying SAS Data Sets: Examples, Version 6, First Edition, 1995. 197 pages.

Client/Server Computing with the SAS System: Tips and Techniques, 1995. 348 pages.

SUGI Proceedings

Beatrous, S. and Armstrong, K. (1991), "Effective Use of Indexes in the SAS System," *Proceedings of the Sixteenth Annual SAS Users Group Intl. Conference*, 16. (Also in *Client/Server Computing with the SAS System: Tips and Techniques*.)

Buffum, H.W. (1996), "Strategic Uses of SAS DATA Step Programming and SQL Passthrough to Query Oracle Databases," *Proceedings of the Twenty-first Annual SAS Users Group Intl. Conference*, 21, 573–577.

Calvert, W.S. and Swartz, L. (1995), "Using Features of SAS Software to Improve Your Documentation," *Proceedings of the 1995 North East Regional SAS Conference*, 122–127.

Dickson, A. and Pass, R. (1996), "SELECT ITEMS FROM PROC.SQL WHERE ITEMS>BASIC," *Proceedings of the Twenty-first Annual SAS Users Group Intl. Conference*, 21, 227–236.

DiIorio, F. (1994), "The Standalone Program Grows Up: Strategies for System Design," *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, 19, 1336–1345.

First, S. (1994), "Developing Easily Maintained SAS Code," *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, 19, 1350–1357.

Gilsen, B.F. (1996), "SAS Program Efficiency for Beginners," *Proceedings of the Twenty-first Annual SAS Users Group Intl. Conference*, 21, 360–369.

Howard, N. and Zender, C. (1996), "Advanced DATA Step Topics and Tips," *Proceedings of the Twenty-first Annual SAS Users Group Intl. Conference*, 21, 181–190.

Loren, J. (1996), "SAS Connections to DB2: Tools and Techniques," *Proceedings of the Twenty-first Annual SAS Users Group Intl. Conference*, 21, 498–507.

Ma, J.M. (1992), "How to Use the WHERE Statement," *Proceedings of the Seventeenth Annual SAS Users Group Intl. Conference*, 17, 259–263. (Also in *Client/Server Computing with the SAS System: Tips and Techniques.*)

Ma, J.M. (1993), "Introduction to Compressing Data Sets," *Proceedings of the Eighteenth Annual SAS Users Group Intl. Conference*, 18, 1412–1416.

Ma, J.M. (1996), "Self-Documenting Programs and Data Sets," *Proceedings of the Twenty-first Annual SAS Users Group Intl. Conference*, 21, 390–399.

Wilson, S. (1995), "Techniques for Efficiently Accessing and Managing Data," *Proceedings of the Twentieth Annual SAS Users Group International Conference*, 20, 474–483.

Yam, A.L. (1995), "Simplifying NDA Programming with PROC SQL," *Proceedings of the Twentieth Annual SAS Users Group International Conference*, 20, 133–137.

CONTACT INFORMATION

J. Meimei Ma, Ph.D.
Quintiles
P. O. Box 13979
Research Triangle Park, NC 27709
Voice: 919-941-7136
Fax: 919-941-0972
Internet: mma@quintiles.com

Andrew Karp
Sierra Information Services, Inc.
1489 Webster Street, Suite 1308
San Francisco, CA 94115
Voice: 415-441-0702
Fax: 415-441-9175
Internet: sfbay0001@aol.com