



SOS (Save Our Space) – Matters of Size

By Matthew Pearce
Amadeus Software Limited © 2001

Abstract

Disk space is one of the most critical issues when handling large amounts of data. Large data means greater processing time, more resources and therefore more money. In SAS the key to all this is the data set.

This paper will compare and contrast the various methods of minimising the physical size of a SAS data set on disk. Accessibility is an important element to be considered here, and this paper will demonstrate that sheer physical size is not the only consideration. There is little point in compressing a dataset to one tenth it's size if it takes ten times as long to read. Alternative techniques present within host operating systems will be analysed in addition to the more traditional SAS methods of data set reduction. Attention will also be given to some common sense coding methods of economising on size for existing data sets.

1. Introduction

Disk Space. The most valuable commodity when dealing with the storage of data. Storage space requires hardware to be purchased. So reducing the size of a dataset can mean a saving in financial cost, often the top priority for any business.

Time on some operating systems incurs a physical cost, for example on Mainframes or outsourcing IT generally. Larger datasets will also result in an increase in processing time. This can indirectly translate into extra human resource time - waiting for a report to be produced, for example. If the data resides on a server this effect can multiply if several people access the data simultaneously.

These issues add up to a good argument for using some method of data compression. When selecting the method to use there is more than just the physical size reduction to consider. Access times are an issue; both when reading from a dataset and writing to one. The time taken by the selected method to perform the required compression is also a factor.



2. Common Sense Coding

A dataset is made up of header Information giving details of the framework and a data portion containing the actual observations. The amount of space required for the data portion of a data set can be calculated as follows:

$(\text{Total Observation Length} * \text{Number of Observations}) + 28 \text{ bytes per 'page'}$, the prime unit of I/O.

So we need to find ways of minimising both the length and the number of observations due to this multiplier effect.

a. **Keep/Drop/Where/If**

It makes sense to keep only those variables that we are interested in when reading a dataset to create a report, for example. This is perhaps just common sense, but is often overlooked since the same end result can be produced even with redundant variables.

However this is wasting valuable space as well as taking longer to process. To keep only the relevant variables the “keep” option can be utilised:

```
data SOS.usedvars;  
    set SOS._1Gtest (keep=var1 var2 var3);  
  
run;
```

Notice how unused variables are being discarded here at the earliest opportunity to make the greatest saving in both time and disk space.

Alternatively, if only a few variables need to be dropped then the “drop” option can be utilised instead, discarding only those variables specified.

```
data SOS.usedvars;  
    set SOS._1Gtest (drop=var1 var2 var3);  
  
run;
```

A “keep” could be used here with no difference in performance, except to the programmer who has to list all the variables (bar three in this case). Since this example dataset has 638 variables this could be somewhat time consuming.



In the case of numbers of used and unused variables being equal I would recommend using a keep option. This is for the simple fact of listing those variables that you are working with, rather than the ones you have dropped. This also benefits other programmers who can see the variables of interest without running a proc contents.

Filtering out unused records also saves on time and space, and to do so at the earliest opportunity maximises this efficiency technique. An 'if' statement is one method of doing this. However, if no actions are required (such as dividing output between different datasets), then a where clause can be utilised on the input data set as a data set option.

```
data work.filtered;  
    Set sos._100mtst (where=(age > 40));  
run;
```

This difference can be explained by the actions, or lack of them, of the Program Data Vector (PDV). Since the where clause acts on the data before the observations are read into the PDV it is quicker to process data this way, in addition to saving space.

b. Data Step Views

If a snapshot of the data is required then creating a view is more efficient than creating a data set. It simply creates a one-dimensional picture of the data.

This is because computer resource usage is determined by the access pattern of the consuming task. Data access is either comprised of single passes or multiple passes, depending on what is being requested. If one pass is sufficient, no data set is created. If multiple passes are required then the view builds a spill file containing all generated observations. Subsequent passes have to read the same data contained in previous passes.

The spill file space is re-used if the data is being accessed in groups. Therefore disk space requirements are equal to the cumulative size of largest by group and not the cumulative size of all observations generated by the view. CPU time can be increased by as much as 10% due to internal host supervisor requirement.

Creation of a view is done by adding '/view=libref.dataset' to the data statement.

```
data work.filtered / view=work.filtered;  
    set sos._100mtst (where=(age > 40));  
run;
```



c. Attribute Statements

The benefits of setting the length of variables to the minimum required are best illustrated by a working example. A client was experiencing increasing problems with their data warehouse, which was already occupying a significant proportion of the disk space on an NT server.

The warehouse was growing at a rate of 0.5 GB/day and the server was down to less than 5 GB of space.

This warehouse ran each night, downloading data from oracle tables into the SAS Data Warehouse. Variables populated with data from an Oracle database have a default length of 2000.

All variables were being set and kept at this length through the various levels of the data warehouse, until being used in reports in the final layer of processing.

At this point the programmer who wrote the warehouse had realised that a particular variable was boolean, for example, and so only needed to be of length one. So the lengths of all variables were being set with various attribute statements. However up until this point certain variables contained up to 1,999 unused units of space.

Attribute Statement syntax

```
attrib agr_line_no length=8;
```

The solution was to move these attribute statements to the top of the warehouse. This resulted in a space saving of approximately 9Gb and resulted in the warehouse taking 2 hours to run instead of 5.

This example illustrates that the most basic methods of efficient coding can be overlooked. Once this had been done we looked at reducing the space further by the use of NT compression, which is covered in part 4.



3. SAS Compression: How does it work?

SAS compression is designed to:

- Treat an observation as a single string of information
- Remove repeating consecutive characters
- Add a 12 byte algorithm to each observation giving the compression details
- Add a 28 byte algorithm to each page

Version 6 is limited to compress=YES option. It is also not possible to use indexing or the point= option on compressed data sets in Version 6. Further options existing in Version 8 include:

a. Compress=BINARY

BINARY specifies that observations in a newly created SAS output data set are compressed into binary numbers. SAS uses Ross Data Compression (RDC) for this setting. This method is highly effective for compressing medium to large (several hundred bytes or larger) blocks of binary data.

b. Compress=CHAR

CHAR uses the same compression Algorithm as YES, with the same results.

4. Microsoft NTFS Compression

To activate NTFS file compression, you select the properties of the drive, directory, or file desired and set the compression attribute. When applied to a directory, the user also has the option of automatically compressing every file within the directory. This means that every file written to this directory will be compressed by default.

Another option is to use the command line to execute NT compression. This can be found via “Programs-Accessories-Command Prompt” in Windows 2000.

To compress a large data file, bigfile.txt, the command would be:

```
compact /c bigfile.txt
```

Further commands can be found by typing ‘compact /?’.



5. Compression Theory – Applied to MS NTFS Compression

Since NTFS file compression is a software solution, the following factors can be considered:

If NTFS file compression operates as a background or foreground application, it must use CPU cycles.

- If NTFS file compression manipulates data, it must use memory.
- Memory is physical.
- Lack of physical memory translates to page swapping.
- Page swapping increases disk utilization.

Hypothesis

By simple deduction, a system can read a compressed file from a disk array faster than its uncompressed counterpart—fewer bytes, less time. Less time spent at disk access, which is slow compared with memory access, speeds retrieval time. Even adding some processor cycles for expanding the file before sending it to the client can, in theory, improve on or equal the performance of retrieving and sending the original uncompressed file.

Assuming that this hypothesis holds true, the relationship between uncompressed and compressed data access is:

$$F / T > (F * C * P) / T$$

where

F = Sample file in megabytes

T = Time to read/write data to or from disk in MBps (Mb per second, constant)

C = The percentage compression achieved on the sample file type

P = Processor constant to compress or uncompress data

Multiplying through by T and dividing both sides by F gives the following necessary condition for a compressed file to be accessed faster than an uncompressed file:

$$1 > C * P.$$

So if the percentage compression (C) is 50%, for example, the processor constant (C) would have to be no greater than 200%. So provided that the processor did not require an extra 100% percent of processor utilization increase to compress data, then the above hypothesis will hold true.

The assumption is that software-based file compression depends on a fast processor (microsecond speeds) compared to hardware-based disk I/O, which is physical and slower (millisecond speeds).



6. Testing : SAS Compression vs. NT Compression

Testing was conducted on a Pentium Ø -700 processor with 256 MB of RAM and a 19 GB hard drive. Each test was replicated ten times over and an average taken of those ten processor tests.

Tests were based on the amount of time taken to read in and write a SAS dataset to disk. A simple data step loop such as the following was the main test component here:

```
data sos._10MTest;
    set sos._10MFile;
run;
```

To apply NT compression from within SAS the following code was used:

```
x "compact /c c:\Matt\NTcomp~1\_100NTcm.sd2";
```

Test Structure

Test Description	Corresponding Results	File Size	No. of Variables	No. of Observations
Small file, many variables	Table A	10Mb	638	3731
Medium file, many variables	Table B	100Mb	638	37310
Large file, many variables	Table C	1Gb	638	373100
Medium file, few variables	Table D	100Mb	10	1906541
V8 – Medium file, many variables	Table E	100Mb	638	37310
V8 – Medium file, few variables	Table F	100Mb	10	1906541

Table Structure

Table	# Characters	# Numerics
A	544	94
B	544	94
C	544	94
D	7	3
E	544	94
F	7	3



7. Results

Table A – Small File, Many Variables

Compression Used	File Size After Compression Applied	% Compression Achieved	Time Taken To Read / Write File
None	9.4 Mb	100%	1.45 s
SAS: Compress= YES	3.169 Mb	33.7%	1.17 s
NT	3.04 Mb	32.4%	2.46 s

Table B – Medium File, Many Variables

Compression Used	File Size After Compression Used	% Compression Achieved	Time Taken To Read / Write File
None	93.3 Mb	100%	17.4 s
SAS: Compress=YES	30.8 Mb	33.3%	12.09 s
NT	30.3 Mb	32.4%	16.09 s

Table C – Large File, Many Variables

Compression Used	File Size After Compression Used	% Compression Achieved	Time Taken To Read / Write File
None	932 Mb	100%	3 mins 1 s
SAS: Compress=YES	307 Mb	32.9%	1 min 58 s
NT	302 Mb	32.4%	2 mins 47 s

Table D – Medium File, Few Variables

Compression Used	File Size After Compression Used	% Compression Achieved	Time Taken To Read / Write File
None	93.5 Mb	100%	17.4 s
SAS: Compress=YES	87.6 Mb	93.7%	12.09 s
NT	39.9 Mb	42.6%	21.67 s

**Table E – Medium File, Many Variables**

Compression Used	File Size After Compression Used	% Compression Acheived	Time Taken To Read / Write File
None	93.3 Mb	100%	15.39 s
SAS V8: Compress=YES	28.52Mb	30.57%	8.7 s
SAS V8: Compress=CHAR	28.52 Mb	30.57%	9 s
SAS V8: Compress=BINARY	26 Mb	27.9%	7.68 s
NT	302 Mb	32.4%	2 mins 47 s

Table F – Medium File, Few Variables

Compression Used	File Size After Compression Used	% Compression Acheived	Time Taken To Read / Write File
None	93.3 Mb	100%	20.54 s
SAS V8: Compress=YES	88.28 Mb	94.02%	19.54 s
SAS V8: Compress=CHAR	88.28 Mb	94.02%	19.52 s
SAS V8: Compress=BINARY	112.7 Mb	120.8%	26.76 s
NT Compression	44.9 Mb	42.7%	24.2 s

8. Analysis

Looking at the first 3 tables you can clearly see that high compression levels of up to 33% (i.e. 67% reduction) are attained by both compression methods for this particular dataset. The significant difference is the time taken to read in the file, perform the compression and write it to disk. In this particular example, SAS compression is the clear winner in terms of performance. Whilst negligible for the smaller 10Mb file (Table A - only 1.2 second and 41% faster) the percentage performance gap in time is clearly reflected for the 100Mb file (Table B - 5 seconds and 44% faster) and significant for the 1Gb file (Table C - 49 seconds and 30% faster).

Table D demonstrates how a different structured file can affect the effectiveness of SAS's compression algorithm. The greater read/write access speed is still present in SAS compression, but the compression achieved is 93% of the original uncompressed file. NT Compression maintains it's high compression ratio, (42.6%) whilst taking only 4 seconds longer to read/write to disk. So whilst there is slight performance degradation in terms of compression speed, the major objective of minimising the physical size of the dataset is still attained.

A possible explanation for this can be found by analysing the structure of the selected dataset (fig.1). Five of the selected variables are boolean, and so take only 1 unit of data even when uncompressed. So these variables will actually take up more space when compressed, due to the extra compression information added (even though this will simply be illustrating that uncompressed=compressed).

ACC_TYPE	Char	1
ADDREKEY	Char	1
ADDTYPE	Num	8
AGE	Num	8
AGREEMNT	Char	1
APPDATE	Num	8
BANKRUPT	Char	1
BKACCNO	Char	12
BKPTFLAG	Char	1
BKSORTCD	Char	9

fig. 1 – header information for the dataset tested

At this point Version 8 compression was introduced into the frame, to see if SAS has improved in the next generation. Methods of data access have clearly improved, as can be seen in Table E. The same 10Mb file had read/write times of 2 seconds faster in Version 8 compared to Version 6 (Table B) – an improvement of 11.5 %. So it could be expected that compression would be faster in version 8, which is the case (under 10 seconds) .



The compression ratio has also improved by 2% (30.5% against Version 6's 32.9%), so the compression header information has been made more compact.

Additional compression methods have also been added, notably the method of compressing numeric data into binary code. Indeed, the Binary option gives both the best compression ratio (27.9%) and the fastest performance time (7.68 seconds).

Table F illustrates how this new option must be approached with caution, however. Only three of the ten variables in this dataset are numeric, and so they will be the only variables to have been compressed better than usual. However, four others are boolean characters, which will not compress at all. In fact they increase in space occupied due to the compression information being added. This combines to produce a dataset 20% larger when SAS runs its compression algorithm!

NT compression maintains its consistently good compression ratio of 42.7%, compressing the V8 dataset as well as the equivalent V6 dataset (see Table D). There is some performance degradation but again this is negligible compared to the space saving produced.



9. Other Host – dependant methods

Alternatives to using the DATA step COMPRESS option are as follows:

Unix

- **compress [-cv] [-b bits] [Filename]**
The amount of compression obtained depends on the size of the input, the number of 'bits' per code, and the distribution of common sub-strings. Typically, text such as source code or English is reduced by 50-60%. The 'bits' parameter specified during compression is encoded within the compressed file, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is subsequently allowed.
- **uncompress [-cfv] [Filename]**
The uncompress utility will restore files to their original state compression. If no files are specified, the standard input will be uncompressed to the standard output.
- **zcat [Filename]**
The zcat utility will write to standard output the uncompressed form of files that have been compressed.

OPTIONS

The following options are supported:

- c Write to the standard output; no files are changed and no .Z files are created. The behaviour of zcat is identical to that of 'uncompress -c'.
- f When compressing, force compression of file. Even if it does not actually reduce the size of the file, or if the corresponding file already exists.
- v (Verbose). Write to standard error messages concerning the percentage reduction or expansion of each file.
- b (Bits). Set the upper limit (in bits between 9 and 16) for common substring codes. Lowering the number of bits will result in larger, less compressed files.



Mainframe

- From the Interactive System Productivity Feature (ISPF) menu, option 3.1 allows you to compress library members.
- For programmable techniques, the following Job Control Language (JCL) utilities are available :
 - o IEBCOPY - To compress a PDS (A partitioned data set is effectively one file composed of many members with the same characteristics and is equivalent to a library).
 - o ICEGENER - For removing records marked for deletion from flat files
 - o IDCAMS For doing the same thing with VSAM (Virtual Storage Access Method) files and Non VSAM files.



10. Conclusion - Space reduction vs. efficiency trade-off

My results show that the structure of a dataset needs to be carefully examined before selecting a method of compression to use, if any. Datasets containing many variables and fewer observations compress more compactly in SAS than datasets with few variables and many observations.

If any doubt exists then a host-operating system method may prove to be the safer option. I have found NT compression to consistently compress SAS datasets to 30-40% of their original size on disk. The slight performance degradation when doing so for certain SAS datasets does not outweigh the benefits of saving over 50% of the original space.

Zippping a file is another method I could have looked at. This is acknowledged as the best method for saving space (up to 10% of the original file, however the time taken to compress is significantly greater. I used Winzip to compress a 1Gb file and it took well over 20 minutes. This could be the best option for archived files.



Acknowledgements

Information obtained from the following websites was utilised in the creation of this document.

www.sas.com

www.storageadmin.com

Contact Information

Matthew Pearce
Amadeus Software Ltd
Orchard Farm
Witney Lane
Leafield
OX28 5PG
England

Telephone	+44 (0) 1993 878287
Fax	+44 (0) 1993 878042
E-mail	Info@amadeus.co.uk
Web Page	www.amadeus.co.uk

Copyright Notice

No part of this material may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Amadeus Software Ltd. © Amadeus Software. June 2001. All rights reserved.

Trademark Notice

Microsoft products are registered trademarks of Microsoft Inc, USA. Base SAS Software is a registered trademarks of SAS Institute, Cary, NC, USA.