Working efficiently with SAS

Paul W. Dickman
Department of Medical Epidemiology and Biostatistics
Karolinska Institutet

paul.dickman@meb.ki.se

November 18, 2003

How is efficiency defined?

- Programming time
  - Time required to write the original code
  - Time required to update/maintain the code
- Run time
  - CPU time
  - I/O time (input/output)
  - Network time
- Resource usage (e.g. using disk space efficiently)

Overview

- How we use SAS at MEP and how we can use it more efficiently.
- The two main dimensions of efficiency of concern to us are disk space usage and your time (time to write the code and time to run it).
- Efficiency can be improved by:
  - Reducing the size of SAS data sets.
  - Writing efficient SAS code.
  - Reducing the amount of network traffic
    - ∗ Working on the local rather than network drive.
    - ∗ Reducing the size of SAS data sets.
- I will present some simple methods (e.g. `length` and `compress`) for reducing the size of SAS data sets and increasing efficiency.

There is always a trade-off

- It is possible to use the `length` statement to reduce the storage space required by some variables and hence the size of the SAS data set.
  - Requires extra programming and has the potential to give incorrect results if used incorrectly.
  - Reduces the size of data sets.
  - This reduces network traffic (you get your results sooner).
  - A net gain in terms of time spent by you.
- Using the SAS `compress` option will often reduce the size of SAS data sets.
  - Requires extra CPU time to perform the compression and decompression (and some extra programming).
  - Reduces the size of data sets.
  - This reduces network traffic (you get your results sooner).
  - Usually a net gain in terms of time spent by you.

How we work at MEP

- Running SAS under UNIX, either by logging in directly or via SAS/CONNECT.
  - See 'Tips for Using Large Data Files with SAS on UNIX Systems' `http://www.utexas.edu/cc/newsletter/aug95/sastips.html`
- Working under windows with data files stored on `H:\`.
- This seminar is aimed primarily at those running SAS under Windows, but most issues are relevant to SAS under UNIX.

Why store files on the server (`H:`)?

- Security — all data containing individual records (even where PNRs have been removed) should be stored on `H:\` since the physical security of the server room is greater than that of the room where your PC is stored.
- `H:\` is backed up
  – A good reason to also store your SAS files on `H:\`.
- The disadvantage is that every time you access the SAS data set (i.e. each PROC step), the file must be copied from the network drive to the local drive.
- I'll describe two approaches to reducing the amount of network traffic (using the WORK library and the SASFILE statement).

The WORK data library

- SAS data sets are identified by names of the form LIBREF.FILENAME where LIBREF is the name of the data library and FILENAME the name of the file.
- The WORK data library is a default data library defined by SAS at the start of each session.
- Whenever you use a one-level name, SAS assumes that the library is WORK.

```
375  data temp;
376  x=3;
377  run;

NOTE: The data set WORK.TEMP has 1 observations and 1 variables.
```

- Under SAS version 8, the WORK library is assigned by default to a directory in the "SAS Temporary Files" subdirectory in the 'temp' directory, for example, `C:\WINNT\Temp\SAS Temporary Files\`.
- If your `C:\` drive is short of space then WORK can be re-assigned (in `config.sas`) to another drive, but I recommend always using a local drive for WORK if possible.
- At the end of your SAS session (i.e. when you exit SAS), the contents of the work directory are deleted.
- If SAS ends abnormally (e.g. a system crash), the contents of the work directory are not removed automatically.
  — You should check your work directory occasionally and remove old files.
- If you are performing a series of file operations, you can improve efficiency by first writing the data to the work directory.

Using the work directory

```
/* Sort the data and write the results to the WORK directory */
proc sort data=h.mydata out=mydata; by pnr; run;

data mydata;
merge mydata otherdata;
by pnr;
....
run;

more file operations;

/** Write the data back to H: **/
data h.mydata;
set mydata;
...
run;
```

Reading a data set into memory using the SASFILE statement

- In a program with several PROC steps, each operating on the same data set, SAS rereads the data for each procedure.
- If the data set is stored on a server, this means the data must be read across the network multiple times within the same program.
- Using the SASFILE statement (new in release 8.1) it is possible to read a data set into memory and keep it open.
- A limitation is that there must be sufficient available memory to store the data set.
- An illustration of the syntax is shown on the following slide.

```
sasfile h.big load;

proc means data=h.big;
var parity age height bmi;
run;

proc freq data=h.big;
var case eox parity f2 bmi mpage mpty f1;
format mpage mpage. f1 age. bmi bmi. f2 height.;
run;

proc logistic data=h.big;
class mpage f1 bmi agefb f2 \ param=ref;
model case=eox parity f2 bmi mpage mpty f1;
format mpage mpage. f1 age. bmi bmi. f2 height.;
run;

sasfile h.big close;
```

SAS data sets are not stored efficiently

- A SAS data set for a MEP study was 53.3Mb.
  — when compressed with WinZip it was 2.9Mb (94% smaller).
- A bit is a basic of computer information, valued at either 0 or 1.
- A byte is a collection of 8 bits.
- Using 8 bits (1 byte) it is possible to represent 2^8=256 unique integers.
- If 1 bit is used for the sign (plus or minus) then with the same 1 byte we can represent the integers from -132 to 132.
- By default, SAS uses 8 bytes to store a numeric variable (the default length of a SAS variable is 8).

```
-----Alphabetic List of Variables and Attributes-----

  #    Variable    Type    Len     Pos
---------------------------------------
  1    X1          Num       8       0
  2    X2          Num       8       8
  3    X3          Num       8      16
```

- A data set containing 1000 observations and 1000 variables would require 8Mb (plus a little extra for data set information).
- We do not need 8 bytes to store integers.
- By telling SAS to use less than the default 8 bytes to store integers, we can significantly reduce the size of SAS data sets.
- Data sets containing primarily integer variables can be reduced in size by around 50% by assigning appropriate lengths.

The LENGTH statement

- The length used to store variables in a SAS data set can be set using the LENGTH statement (or the ATTRIB statement).
- Variables containing real numbers should be left with the default length of 8.

Length in bytes	Largest integer represented exactly
3	8,192
4	2,097,152
5	536,870,912
6	137,438,953,472
7	35,184,372,088,832
8	9,007,199,254,740,990

Table 1: Largest integer represented exactly by length for SAS variables under Windows

- Examples using the LENGTH statement:

```
data one;
length sex age 3 pnr 6;
...
run;

length pnr 6 default=3;

length pnr6 fat--age 3 nut1-nut56 8;
```

- Note that specifying a length less than that required will result in a loss of precision without any warning being given (see the example on page 92 of the version 6 SAS Language: Reference manual).

Be careful when changing the length!

```
data temp;
length x 4 y 3;
do x=9000 to 9010;
y=x;
output;
end;
run;

proc print;
run;
```

```
OBS   X     Y

  1  9000  9000
  2  9001  9000
  3  9002  9002
  4  9003  9002
  5  9004  9004
  6  9005  9004
  7  9006  9006
  8  9007  9006
  9  9008  9008
 10  9009  9008
 11  9010  9010
```

Compressing data sets with the COMPRESS option

- The possible values for COMPRESS are
  - no (the default value)
  - yes
  - binary (a new option available in SAS version 8 which is especially efficient for numeric data)
- COMPRESS is both a system option and a dataset option.
- COMPRESS as a data set option

```
data h.mydata(compress=yes);
infile ...;
...
run;
```

- COMPRESS as a system option

```
options compress=yes;
```

All newly created SAS data sets will be compressed.

- SAS will automatically uncompress compressed data sets when it needs to read them.
- The SAS system viewer cannot read compressed data sets.
- There are some other limitations with compressed data sets but most users at MEP will not encounter them.
  — for example, you cannot access a compressed data set using the `point=` option.

An example

```
12   data c.yyy(compress=yes);
13   length i 4 x1-x500 3;
14   array x x1-x500;
15     do i=1 to 10000;
16       do over x;
17       x=mod(i,2);
18       end;
19     output;
20     end;
21   run;

NOTE: The data set C.YYY has 10000 observations and 501 variables.
NOTE: Compressing data set C.YYY decreased size by 49.75 percent.
      Compressed is 504 pages; un-compressed would require 1003 pages
NOTE: DATA statement used:
      real time           5.32 seconds
      cpu time            3.75 seconds
```

Summary of the file size and time to write the file to disk using various combinations of options

			Real time (s)		CPU time (s)	
Compressed	Length	Size (Mb)	Local	Server	Local	Server
No	No	40.0	10.8	46.4	3.6	4.0
Yes	No	10.1	5.1	18.7	4.5	4.7
No	Yes	16.0	5.8	18.3	3.2	3.2
Yes	Yes	8.1	5.9	15.6	3.8	3.6

- When using `compress=binary` (only available in version 8) the file size was 5.8Mb (both with and without using the length statement).
- When compressed with WinZip, the 40Mb file was compressed to 122Kb!
  – this is exceptional because all 10,000 records are identical
- In general, WinZip is more efficient than SAS compression.

Subsetting Observations More Efficiently

- The following approach for performing an analysis on a subset of observations is inefficient.

```
data elderly;
set patients;
if age > 65;
run;

proc print data=elderly;
run;
```

- This approach (using a subsetting IF statement) results in SAS first reading the entire data set (patients), writing the reduced data set (elderly), and then reading the reduced data set again to perform the procedure.

- Use, instead, a WHERE data set option:

```
proc print data=patients (where=(age > 65));
run;
```

or a WHERE statement

```
proc print data=patients;
where age > 65;
run;
```

- Both the WHERE statement and the WHERE data set option first validate the condition to see whether the observation is to be kept before it is read into the Program Data Vector (PDV).
- Using the WHERE approach avoids creating an additional data set.

Other considerations

- Use drop and keep statements to delete unwanted variables (but use these statements efficiently).

```
data merge;                       data merge;
merge maindata canreg;            merge maindata(keep=pnr x y z)
by pnr;                                 canreg(keep=pnr x y z);
keep pnr x y z;                   by pnr;
run;                              run;
```

- The code on the right is more efficient because only those variables that are used are read into the PDF (program data vector).
- When storing completed projects, compress them into a ZIP archive using WinZip.
- Make as few copies of your data set as possible (the subject of an upcoming seminar).

- Store integers as numeric variables.
- Think about how you store free text fields (assigning a length of 100 characters to a character field can be very inefficient if this length is only required for a few observations).
- Tips on sorting more efficiently
  `http://support.sas.com/sassamples/quicktips/03apr/tunesort.html`