Paper 008-2009

Using Formats, MP Connect®, and Other SAS® Efficiency Techniques to save Time and Disk Space

John Charles Gober, Arumugam Sutha
U.S. Bureau of the Census, American Community Survey

ABSTRACT

Working with historic household Census data can be quite challenging. Semi-annual data sets can exceed 50 gigabytes and when working with multi-year data it is often necessary to extract information from yearly clusters of data sets exceeding 300 gigabytes. These yearly clusters, until recently, have been processed sequencally, each independent process relying on the completion of the previous process before continuing. The sorting alone of some of these data sets can take over an hour of clock time. Processing six data set representing three years of data can take six hours. By using format tables and eliminating sorts and merges, reducing repeated passes through the data, and the use of MP connect for independent processes processing time and temporary storage can be reduced by over 50 percent. This paper will explore several techniques using format tables, combining data steps, and MP connect to achieve these results.

KEYWORDS

CNTLIN, CPUCOUNT, THREADS, Format, FMTSEARCH, MP connect, PROC DATA SETS, PROC FORMAT, rsubmit, SASCMD, %bquote, %syslput.

INTRODUCTION

The American Community Survey (ACS) was fully implemented in 2005. When the 2005 data was released only single year processing could be undertaken. With the introduction of 2007 data the Bureau is now is now able to process not only single year statistics but also three year statistics. In 2010 five years of data will be ready for analysis.

Sorting, merging, and analyzing large data sets can be quite challenging when CPU time and disk space are at a premium. Insertions of a single variable from one data set into another can double or triple processing time. Using user generated formats as an alternative to the commonplace technique of sort-sort-merge can eliminate a significant amount of time. Eliminating multiple passes through large data sets can also reduce processing time. It is also more economical to process data sets in parallel, not sequentially, when the processes are independent of each other. This paper will present several simple but effective techniques that can increase performance and save resources for the average power user.

SIMPLE FORMATS AND EFFICIENCIES

In order to save disk space an attempt has been made to normalize the ACS data into a relational structure. This works well for most of the end users who only need to retrieve and process small segments of data, i.e. Data at the county, zip code, or community level. However, at the state and national level, the re-integration of data from multiple data sources can be extremely cumbersome. Especially when the user only uses PROC SORTs and data step MERGEs.

The following is a code excerpt from one of our programs before any efficiency enhancements were made (*extraneous code has been removed to reduce clutter*).

```
proc sort data=temp.m00
   (keep=fipst fcnty bst bcnty btrat          132,763,331 observations temp.m00
         block blks1 blks2)                   real time 38:22.80
    out=m00;                                  cpu time  13:16.31
    by fipst fcnty bst bcnty btrat block
       blks1 blks2;
run;
(continued)
```

```
data temp2;
   merge m00
         temp.g00
         (keep=fipst fcnty bst bcnty          132,763,331 observations m00
               btrat block blks1 blks2          9,504,014 observations temp.g00
               sbstr);                       real time 4:02.77
   by fipst fcnty bst bcnty btrat block      cpu time  3:08.98
      blks1 blks2;
run;

proc sort data=temp2;                        132,763,331 observations temp2
   by sbstr;                                 real time 2:09.98
run;                                         cpu time  3:15.88

data temp3(keep=sbstr count);
   retain count 0;
   set temp2;                                132,763,331 observations temp2
   by sbstr;                                           5 observations temp3
   if first.sbstr then count = 0;            real time 24.29 seconds
   count = count + 1;                        cpu time  23.74 seconds
   if last.sbstr then output;
run;

data wk.cnt00(keep=cnt1-cnt5);
   array cnt{5} (5*0);                                 5 observations wk.cnt00
   set temp3   end=last;                     real time 0.04 seconds
   cnt[input(sbstr,1.)] = count;             cpu time  0.00 seconds
   if last;
run;
```

The M00 data set has over one hundred thirty two million records and takes over 10 gigabytes of disk space. The G00 dataset has only nine million and is considerably smaller in size, fourteen times smaller than M00. In addition note that there were four passes made through the M00 data set. Considering the record count difference alone between these two datasets an intellectual flag for an alternative programming solution should have been triggered. In addition, this process needed to be repeated three times in order to collect data from three distinct and independent cycles of data. Total temporary disk space for the three cycles exceeded one hundred twenty gigabytes. CPU usage typically exceeded typically one hour, real time three hours. This was unacceptable when the final outcome was only a single record, five variable dataset per cycle.

After a brief and independent analysis of the existing code it was discovered that by using format tables all of the PROC SORTs could be eliminated. In addition the repeated reading and re-reading of the same dataset only to produce summary counts could be combined into a single data step thus simplifying the code and reducing clock time, cpu usage, and i/o costs. In addition eliminating multiple copies of the data would save precious disk space. The code would also be cleaner and easier to manage,

The first efficiency item that was addressed was the sorting of the large TEMP.M00 data set which took over thirteen minutes of cpu time and the consequent merging of the relatively small TEMP.G00 data set which took an additional four minutes, all done to add a single variable to TEMP.M00. It would be good at this time to point out that this process is a many to one match where the two datasets contain a disproportionate number of observations, a good use for format tables. Below is the code that was used to create a simplified version of the code.

Custom format tables are not hard to produce. There are numerous conference papers and SAS technical example and tips on formats so this paper will not go into great detail. The primary skepticism of creating formats is the idea that a table cannot be created when there are multiple classification variables or the resulting label has multiple variables. This is easily resolved with concatenation as shown in the below code.

```
libname myfmtlib "/work1";
/*format routine control dataset */
data currfmt(keep=fmtname start end
                  label hlo type);
   length fmtname label $12. start end $22.;
   retain type "C";
   set temp.g00
        (keep=fipst fcnty bst bcnty btrat
              block blks1 blks2 sbstr) end=last;
   by fipst fcnty bst bcnty btrat block
      blks1 blks2 sbstr;
   fmtname = "$currmstrfmt";                    9,504,015 observations currfmt
   start = fipst||fcnty||{…}||blks2;            real time 1:35.33
   end = start; label = sbstr;                  cpu time    21.64
   output;
   if last then do;
      start = ""; end = "";
      hlo = "O"; label = "X";
      output;
   end;
run;

proc format cntlin=currfmt library=myfmtlib;   9,504,015 observations currfmt
run;                                           real time 1:08.18
                                               cpu time  1:03.00
```

Notice the use of a permanent format library. This was necessary since the format had to be available for separate but simultaneous processing in an MP/connect environment.

In addition to using format tables a number of other efficiencies were made. In the original program the M00 and G00 were merged in one data step process to add the classification variable sbstr. The resulting data set was then sorted in preparation for tallying that variable (sbstr contains only the character values “1” through “5”). Another pass through the data was made to generate a data set containing one record for each of the five classifications and their associated counts. Finally the data set was normalized in another data step process to create a single record data set containing five separate variables, one for each of the classifications. Using the format created above and creating a summary variable up front for each classification of SBSTR the code was reduced.

```
data wk.cnt00(keep=cnt1 - cnt5);
   set temp.m00(keep= fipst fcnty bst bcnty btrat
                      block blks1 blks2 valdf)
       end=last;
   sbstr=put(var1||{…}||var8,                 132,763,331 observations temp.m00
         $currmstrfmt.);                                1 observations wk.cnt00
   if sbstr="1" then cnt1+1;                  real time 26:06.73
   else if sbstr="2" then cnt2+1;             cpu time  14:21.87
   else if sbstr="3" then cnt3+1;
   else if sbstr="4" then cnt4+1;
   else if sbstr="5" then cnt5+1;
   if last then output;
run;
```

Again, the resulting data set was placed in a permanent location because this portion of code was to be used in an MP/connect environment and repeated three times for different data cycles.

It was hoped that by creating and using a format table created from TEMP.G00 thus elimination the PROC SORT and reading the table directly into TEMP.M00 considerable cpu savings would be realized. However it ended up that cpu savings were significant but minimal. Clock time and I/O were significantly reduced, nineteen minutes versus sixteen minutes cpu, forty four minutes verses twenty eight minutes clock time, one gigabyte I/O versus five hundred thirty two million I/O.

NESTED FORMATS

Some programmers have concerns over generating a format table with over nine million entries. They feel that once a table becomes too large there is not any efficiency gain over using a PROC SORT and a data step merge routine. By comparing the original code with the enhanced code it was discovered cpu processing time for the data step routine using the format table actually was increased by two percent. This could partially be explained by the combining of code from other data steps into the data step using the format, there was a noticeable decrease when comparing the entire processes, not just a single process. Over all though the savings in space alone made the new code beneficial having saved over 100 gigabytes.

Knowing that further changing the code to use nested formats would probably not result in any more efficiency it was decided to go ahead with the change as an education and training tool for future projects.

Nested formats are useful when several formats share the same associations. By removing these common associations and creating a new format that is then reference by the original formats written code can be reduced creating a smaller and cleaner program.

Creating a nested format is very similar to a simple format when they are created using the PROC FORMAT routine. However when creating a control data set to be used as input to PROC FORMAT it was discovered there are several nuances that have to be considered before the format will function properly.

```
/*nested format routine control dataset*/
data currfmt(keep=fmtname start end label hlo type);
   length fmtname label $12. start $22;
   retain starta " " type "C";
   set temp.g00(keep=fipst fcnty {…} blks2 sbstr) end=last;
   by fipst fcnty {…} blks2;
   fmtname = "$x"||fipst||fcnty||"x";
   start = fipst||fcnty||{…}||blks2;
   end  = fipst||fcnty||{…}||blks2;
   label = sbstr;
   output;
   if last then  do;
      start = ""; end = "";
      fmtname = "$x"||fipst||fcnty||"x";
      hlo = "O"; label = "X";                    9,504,014 observations temp.g00
      output;                                    9,507,234 observations currfmt
   end;                                          real time 20.94 seconds
   if first.fcnty then do;                       cpu time  20.64 seconds
    fmtname  = "$currmstrfmt"; hlo = "F";
      starta = fipst||fcnty||{…}||blks2;
      label = "$x"||fipst||fcnty||"x1.";
   end;
   if last.fcnty then do;
      fmtname  = "$currmstrfmt"; hlo = "F";
      start    = starta;
      end      = fipst||fcnty||{…}||blks2;
      label  = "$x"||fipst||fcnty||"x1.";
      output;
   end;
run;

proc sort data = currfmt;                        9,507,234 observations currfmt
   by fmtname label;                             real time 1:39.87
                                                 cpu time  1:38.36

proc format cntlin=currfmt lib=myfmtlib;         real time 1:21.30
run;                                             cpu time  1:07.06
```

In the above code the target (nested) formats and the referencing format were both created using one data step. Hindsight now dictates that even though cpu costs would go up (negligible for application) it would have been better to have used two separate data steps for better readability.

When creating a label referencing a format using PROC FORMAT and internally assigning values and labels the label assignment for formats are enclosed in brackets as in the following statements.

```
proc format;
   value $currmstrfmt
       "0100100000000000" - "0100101001021100" = "[$x01001x.]"
        …
       "7215100000000000" - "0201302013000100" = "[$x72151x.]"
       other = "X";
   value $x01001x
       "0100101001020100" = "1"
        …
       "0100101001021100" = "9";
run;
```

In the improved code where a control dataset was used to populate the format contents the brackets are not necessary. But tagging the label as containing a format reference was necessary. This was accomplished by assigning the HLO variable to the value of "F" instead of "O". The PROC SORT procedure was also necessary. Because of the technique used to generate the values and labels the records for the nested formats were intermixed with the calling format. This caused the calling format to be loaded again and again after each loading of the nested formats. This in turn cause the controlling format to only contain records associated with the last nested format.

MULTI PROCESSING

The change of SAS/Connect from a synchronous to an asynchronous client server environment in SAS 8 now allows programmers to submit tasks with independent parallelism to execute simultaneously instead of consecutively. As mentioned earlier this application requires the input and processing of three extremely large but independent cycles of data. Granted three separate programs could have been written and a fourth to collect, combine, and process the resulting data but this would also require a program or code to constantly monitor the independent processes for completion. This has all been built into MP/connect. The below code was developed base on information primarily found on SAS technical support web pages, tech tips, and conference paper. The code is very similar to the existing code. Only some pre and post processing parameters needed to be set along with the creation of a few macro variables.

```
options sascmd="sas9 -memsize 56k -threads -cpucount 4";
options autosignon threads cpucount=5 source source2 mprint;

signon remote=task1; signon remote=task2; signon remote=task3;
%let locdir1=/cycle1; %let locdir2=/cycle2; %let locdir3=/cycle3;

%syslput remdir1=%bquote(&locdir1) /remote=task1;
%syslput remdir2=%bquote(&locdir2) /remote=task2;
%syslput remdir3=%bquote(&locdir3) /remote=task3;

rsubmit process=task1 wait=no;
   options fmtsearch=(myfmtlib work);
   libname myfmtlib "/temp1/sastmp";
   libname temp     "&remdir1";
   libname wk       "/temp1/sastmp";

   data wk.cnt00(keep=cnt1-cnt5);
      set temp.m00(keep=fipst fcnty bst bcnty btrat block blks1 blks2)
          end=last;
      sbstr = put(fipst||fcnty||{…}||blks2,$currmstrfmt.);
      if sbstr = "1" then cnt1 + 1;
      else if sbstr="2" then cnt2 + 1;          136,123,017 observations temp.m00
      else if sbstr="3" then cnt3 + 1;          1 observations wk.cnt00
      else if sbstr="4" then cnt4 + 1;          real time 36:13.91
      else if sbstr="5" then cnt5 + 1;          cpu time  16:31.78
      if last then output;
   run;
endrsubmit;
```

```
rsubmit process=task2 wait=no;                  132,763,331 observations
  {repeat code for second cycle}                real time 29:19.18
endrsubmit;                                     cpu time  15:34.41

rsubmit process=task3 wait=no;                  135,049,806 observations
 {repeat code for third cycle}                  real time 36:53.10
endrsubmit;                                     cpu time  17:01.98

waitfor _all_ task1 task2 task3;

signoff remote=task1; signoff remote=task2; signoff remote=task3;
```

Notice the use of permanent data sets allowing access of the remote processed data by the parent process. These data sets can always be deleted using PROC DATASETS at the end of the parent process. The option AUTOSIGNON eliminates the need for a userid and password. The option SASCMD contains the name of the SAS call and any options needed for the remote sessions. %SYSLPUT was used to pass and create macro variables to each remote session. The use of %BQUOTE was necessary because the argument to %SYSLPUT contained special characters. Finally WAITFOR was used to suspend processing of the parent program until all remote tasks were completed. Total cpu time for all remote processes was forty seven minutes (one hour forty minutes clock time). The maximum cpu time used by any one remote process was seventeen minutes. Unfortunately cpu time cannot be reduced, just extended across processors. However because clock time was shared it was reduced to thirty six minutes., a reduction of over sixty percent.

CONCLUSION

SAS has always been criticized as being a resource hog. In most cases this is not the fault of the software but of the programmer. Programmers tend to seek out the simplest and fastest coding methods causing excessive use of cpu and disk space. With a little effort, determination, and due diligence it is possible to increase efficiency by a factor of five. In this application, by eliminating and combining processes and using efficient coding techniques, this application now runs in under an hour instead of two and a half hours. Just as important temporary disk space has gone from over one hundred gigabytes to practically nothing.

AKNOWLEGMENTS

Kimberly Dee Wortman, Census Bureau, Technical Support Group, SAS Branch.

AUTHOR CONTACT

John Charles Gober
Bureau of the Census
4710 Silver Hill Road
HQ 3H460E
Suiltland, Maryland 20746-1912
301-763-5964

Arumugam Sutha
Bureau of the Census
4710 Silver Hill Road
HQ 3H465
Suiltland, Maryland 20746-1912
301-763-5400

REFERENCES

SAS Institute Inc. 2008.*SAS® 9.2 Companion for UNIX Environments*. Cary, NC: SAS Institute Inc.
SAS Institute Inc. 2008.*Base SAS® 9.2 Procedures Guide*. Cary, NC: SAS Institute Inc.
SAS Institute Inc., 2008.*Communications Access Methods for SAS/CONNECT® 9.2 and SAS/SHARE® 9.2*. Cary, NC: SAS Institute Inc.
Multiprocessing with Version 8 of the SAS System, Cheryl Doninger, SAS Institute Inc., http://support.sas.com/techsup/technote/ts632.pdf, April 18,2000.
Building and Using User Defined Formats, Art Carpeneter, SUGI29 Proceedings, Paper 236-29, http://www2.sas.com/proceedings/sugi29/236-29.pdf, May2004.

TRADEMARK INFORMATION

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the United States of America and other countries. ® Indicates USA registration. Other brand or product names are registered trademarks or trademarks of their respective companies.