Paper CC-01

A View Toward Performance

Ed Heaton, Data & Analytic Solutions, Inc., Fairfax, VA

ABSTRACT

Often we need to preprocess our data through a DATA step before we submit it to some SAS® Procedure. When our data are very large – billions of rows – this processing of reading the entire dataset, making the modifications to each row, writing the entire dataset back out to disk, and then reading then entire dataset back into the PROC can take a very long time.

The author proposes the use of a DATA view to process the data one row at a time and then to send that row directly to the PROC for processing. With very large datasets the time savings can be on the order of 60%.

THE PROBLEM

This paper will discuss multi-threaded PROCS that SAS provides and the theory behind such – and the limited number of multi-threaded tasks that SAS provides. It will talk briefly about the SASFILE statement and how it can eliminate the high cost of I/O when our data files are small enough. Then it will introduce the above concept and will present examples and performance data to back up our claims.

MULTI-THREADING IN SAS

On a single-core processor, multi-threading involves time slices. That is, tasks are broken down into smaller units called threads and the CPU processes the threads sequentially. Threads can originate from different programs; that is why your Microsoft® Outlook® application can post a meeting reminder while your SAS DATA step is running.

Multiprocessing uses multiple CPUs – or cores – to crunch threads simultaneously. This can be more efficient because your SAS DATA step does not have to stop to let your CPU process a thread from Outlook.

But, let's stick with SAS. We have a dual-core CPU and we want to simultaneously – not sequentially process SAS threads. SAS does have some procedures with built-in multithreading capability. The Base SAS® multi-threaded procedures are

- MEANS (SUMMARY),
- REPORT,
- SORT,
- SQL, and
- TABULATE.

THE I/O PROBLEM

We often process massive amounts of data with SAS and our processes are usually I/O bound. That is, our CPUs spend a lot of time waiting for data. Multi-processing does not really help here – it just gives us more CPUs hanging around in the break room waiting for data! We need a way to access our data faster.

THE SASFILE STATEMENT

One solution is to pre-load our data file into memory. Reads from disk are slow. If you are reading across a network, it can be even worse. If your CPU can't process the data as quickly as the network sends it, it will stop sending your data and send data that someone else requested to them. Then, when you want more data, the network might take its own sweet time before it sends more data to you!

One solution is to bulk-load the data file into memory. That way, we are not waiting on the CPU to process the data; the network police do not sense that the data is back-logging in our workstation; and our data will more likely keep flowing to us – down those infamous tubes. Once the data is in our memory, reads from that memory are incredibly faster.

The SASFILE statement will load a SAS data file into memory.

```
sasFile myLib.myData load ;
```

This works wonders – if your data file is small enough to fit in your available memory.

After you are done with the data file, you have to explicitly remove it from memory.

```
sasFile myLib.myData close ;
```

THE PROBLEM – A CASE STUDY

Suppose we want to find the percentage of office visits where the total cost is over a certain threshold – over the past five years. Furthermore, suppose we have a data file for each quarter for each state for those five years and that each of those files have the costs of the office visit parsed into five categories but without a total cost. We need to combine all (50 states + DC) × 5 years × 4 quarters/year = 1020 data files, sum the five costs for each row, add a flag to specify if the sum is over a prescribed limit, and pass that on to PROC FREQ. That might mean that we read on the order of five billion rows, write all five billion rows back to disk keeping, e.g., the quarter, the state, and the flag, and then read that data back into PROC FREQ. Even though the calculation is extremely quick (see below), the I/O involved with those two reads and one write of the five billion rows in enormous and the total processing time is hours.

```
1    Data _null_ ;
2        Retain c1-c5 123.45 ;
3        Do i=1 to 5e9 ;
4            x =  sum( of c1-c5 ) ;
5        End ;
6    Run ;

NOTE: DATA statement used (Total process time):
      real time           3:22.38
      cpu time            3:22.24
```

THE SOLUTION

The solution to speed up the processing of much smaller datasets is to use the SASFILE statement to bulk-load the dataset into memory at the very beginning of the process. But for these large tasks, we propose to use a temporary data view stored in the WORK library. The view can be created by the DATA step or PROC SQL – whichever best suits the needs. The time to create the view is nil. That view is then used as the dataset for the PROC.

SUPPORTING MACRO

We need a list of the 1020 input datasets to use in a SET statement. A macro will help this and make the code more flexible.

```
%macro allDataFiles( firstYear= , lastYear= ) ;
    %local year quarter fips st ;
    %do year=&firstYear %to &lastYear ;
        %do quarter=1 %to 4 ;
            %do fips=1 %to 56 ;
                %let st = %sysFunc( fipState(&fips) ) ;
                %if ( "&st" ne "--" ) %then libRef.&st&year.Q&quarter ;
            %end ;
        %end ;
    %end ;
%mEnd allDataFiles ;
```

This is not a paper about macro; the macro code is just included to give context to the other code in this paper.

THE HARD WAY

Let's process some data the conventional way.

```
Data forProcFreq ;
    Set %allDataFiles( firstYear=2004 , lastYear=2008 ) ;
    OverLimit = ( sum( of CostOf: ) gt 200 ) ;
Run ;
Proc freq data=forProcFreq ;
```

```
    Tables ST*OverLimit / noPct noCol noCum ;
Run ;
```

With only 10,000 rows and 58 columns in each table, the DATA step takes about an hour and a half to run. That's because it reads 10,200,000 rows from disk and writes 10,200,000 rows back to disk. Each of the 1020 data files is 13,665,280 bytes. That's 14 gigabytes read from disk and 14 gigabytes written back to disk.

```
NOTE: DATA statement used (Total process time):
      real time           1:32:15.34
      user cpu time       43.01 seconds
      system cpu time     18:16.01
      Memory                           36240k
```

After this was done, the data had to be read back into the FREQ procedure. This is much faster because SAS does not have to parse the data into the Program Data Vector (PDV) and it does not have to write the data back to disk.

```
NOTE: There were 10200000 observations read from the data set WORK.FORPROCFREQ.
NOTE: The PROCEDURE FREQ printed pages 1-4.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           3:53.54
      user cpu time       4.78 seconds
      system cpu time     52.14 seconds
      Memory                           169k
```

The whole process took an hour and 36 minutes. Now imagine a more realistic two to ten million rows for each of the 1020 files!

EASE THE I/O BOTTLENECK WITH A DATA VIEW

It's really easy to turn a DATA step that creates a data file into one that creates a data view.

```
Data forProcFreq / view=forProcFreq ;
    Set %allDataFiles( firstYear=2004 , lastYear=2008 ) ;
    OverLimit = ( sum( of CostOf: ) gt 200 ) ;
Run ;
```

With this technique, one row from the first dataset (AL2004Q1) is read into the PDV of the DATA step, the costs are summed and compared against the threshold to set the OverLimit flag, and then the row is passed to PROC FREQ for processing. Then the next row from AL2004Q1 is read, processed, and passed to the PROC. This process continues until all of the data is processed.

The DATA step only needs the memory to process one row of data. That's true whether it's creating a data file or a data view. The system needs the memory to pass one row of data from the DATA step to the PROC, and PROC FREQ needs the memory to process 51 pairs of values – keeping the state USPS code. The data is read only once and never written back to disk – not even to the WORK library. The savings both in time and disk space are substantial.

```
NOTE: DATA statement used (Total process time):
      real time           1:18.56
      user cpu time       0.67 seconds
      system cpu time     2.01 seconds
      Memory                           8131k
...
NOTE: View WORK.FORPROCFREQ.VIEW used (Total process time):
      real time           35:39.92
      user cpu time       48.46 seconds
      system cpu time     1:04.62
      Memory                           36262k
...
NOTE: PROCEDURE FREQ used (Total process time):
      real time           35:40.46
      user cpu time       48.54 seconds
```

```
      system cpu time     1:04.98
      Memory                           230k

NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513
NOTE: The SAS System used:
      real time           36:59.71
      user cpu time       49.32 seconds
      system cpu time     1:07.35
      Memory                           75879k
```

CONCLUSION

While the code that creates a DATA step view normally runs in a second or two, it took over a minute to create this data view because it combines so many tables. It then took almost 36 minutes for the data view to read all of the datasets and pass the data to the FREQ procedure.

However, the FREQ procedure finished about half a second after the data view finished its work – that time was probably used to write the report. The whole process took less than 40 minutes. Time savings: 60%.

CONTACT INFORMATION (HEADER 1)

Your comments and questions are valued and encouraged. Contact the author at:

Ed Heaton
Project Manager, Sr. SAS Developer
Data and Analytic Solutions, Inc.
3057 Nutley Street, #602
Fairfax, VA 22031
Office: 301-520-7414
Fax: 703-991-8182
eheaton@dasconsultants.com
http://www.dasconsultants.com
SBA 8(a) & SDB, WBE (WBENC), MBE (VA & MD)