Paper PO-10

TIPS AND TRICKS OF EFFICIENT SAS® PROGRAMMING FOR SDTM DATA

Eric Qi, Merck & Co., Upper Gwynedd, Pennsylvania
Fikret Karahoda, Merck & Co., Upper Gwynedd, Pennsylvania

ABSTRACT

Dataset sizes increased dramatically after the pharmaceutical industry adopted the SDTM data model. For example, SUPPQUAL, CF, and LB domains can easily reach several gigabytes (GB) in size. The need to process this data efficiently becomes more important even when using today's high-speed computer resources. In this paper, we will discuss a case to show how efficient programming plays an important role in handling large SAS datasets.

INTRODUCTION

Dataset size increased dramatically after the pharmaceutical industry adopted the SDTM data model. SUPPQUAL, CF, and LB domains, for example, can easily reach several GB. One SAS program can sometimes run for several hours. In this paper, we point out performance bottlenecks and propose reasonable improvement solutions.

EFFICIENCY

Efficiency can come from several categories that includes:

- CPU time - the time the Central Processing Unit spends performing the operations assigned
- I/O time - the time the computer spends on the two tasks of input and output. Input refers to moving the data from storage areas such as disks or tapes into memory. Output refers to moving the results out of memory to storage or to a display device
- Memory - the size of the work area that the CPU must devote to the operations in the program
- Data storage - how much space on disk or tape the data occupies

The sample data used in this paper is a 6.4 GB SUPPQUAL domain containing 1,676,389 observations. We will use this data to investigate the following case and identify the bottleneck impeding performance.

CASE AND TIPS

CASE: SUBSET A DOMAIN DATA FROM THE SUPPQUAL DATA

Finding the bottleneck

There are several methods to subset a domain data from the SUPPQUAL data; the following four methods were tested:

1. Hash object
2. Data step with WHERE statement
3. Data step with IF statement
4. PROC SQL

Following are the codes we used:

```
**** Hash object ;
data domainlist;
  length rdomain $2 const 8;
  rdomain='AE';
  const=1;
  output;
run;

data new;
   if 0 then set domainlist ; * match parms types ;

   DCL hash hh   (dataset: 'domainlist') ;
   hh.DefineKey  ('rdomain'          ) ;
   hh.DefineData ('const') ;
   hh.DefineDone () ;

   do until (eof);
   set data.&indata end=eof;
     if hh.find()=0 then output;
   end;
   stop;
run;

**** Data step with WHERE statement;
data new;
  set data.&indata;
  where rdomain='AE';
run;

**** Data step with IF statement;
data new;
  set data.&indata;
  if rdomain='AE';
run;

**** PROC SQL;
proc sql;
  create table new as
  select *
  from data.&indata
  where rdomain='AE';
quit;
```

We used PC SAS 9.1.3 and data residing in our company network drive. The real time, User CPU time, System CPU time, and Memory used for each of these four methods listed in the following table:

Table 1. Comparison of real time, User CPU time, System CPU time and Memory using data from network drive.

Method	Real Time (hh:mm:ss)	User CPU (Second)	System CPU (Second)	Memory (K)
Hash object	1:01:28	9.07	9.64	592
Data step with WHERE statement	1:03:32	2.48	9.17	190
Data step with IF statement	1:05:26	4.81	10.10	184
PROC SQL	1:09:02	2.53	10.49	162

Among these four methods, the real times are very similar. Although the hash object method used more CPU time and memory than the other three methods, the difference was small. Additionally, the one hour plus running time was too long to run jobs from network drive.

If we copy SUPPQUAL data into a local PC and run, will improved performance be achieved? Surprisingly, it is much better! Table 2 below shows the average real time, User CPU time, System CPU time, and Memory used.

Table 2. Comparison of real time, User CPU time, System CPU time and Memory using data from local PC.

Method	Real Time (hh:min:ss)	User CPU (Second)	System CPU (Second)	Memory (K)
Hash object	0:02:23	4.54	5.87	592
Data step with WHERE statement	0:02:31	1.55	6.98	190
Data step with IF statement	0:02:34	2.60	6.88	180
PROC SQL	0:02:34	1.34	8.27	159

Noticeably, the CPU times differ by only a few seconds, and the big difference is in the real time. Real time reduced from hours to minutes with the majority of the real time being I/O time. We found that I/O is the bottleneck that primarily affects performance that brings us to the challenge of how to find a .solution.

Truncate off excessive trailing blanks

We know that I/O time is the time a computer utilizes to move data from storage areas such as disks or tapes into memory, move results out of memory into storage, or moving data to a display device. If the file size can be reduced by truncating excessive trailing blanks in character variables, improved I/O time may be achieved.

PROC CONTENT of both original SUPPQUAL and truncated one are listed below:

Content of original SUPPQUAL

```
#    Variable    Type     Len    Label

4    IDVAR       Char       8    Identifying Variable
5    IDVARVAL    Char     200    Identifying Variable Value
0    QEVAL       Char     200    Evaluator
7    QLABEL      Char     200    Qualifier Variable Label
6    QNAM        Char     200    Qualifier Variable Name
9    QORIG       Char     200    Origin
8    QVAL        Char    2000    Data Value
2    RDOMAIN     Char       2    Related Domain Abbreviation
```

1	STUDYID	Char	200	Study Identifier
3	USUBJID	Char	200	Unique Subject Identifier

Content of truncated SUPPQUAL

#	Variable	Type	Len	Label
4	IDVAR	Char	8	Identifying Variable
5	IDVARVAL	Char	19	Identifying Variable Value
10	QEVAL	Char	1	Evaluator
7	QLABEL	Char	40	Qualifier Variable Label
6	QNAM	Char	8	Qualifier Variable Name
9	QORIG	Char	1	Origin
8	QVAL	Char	235	Data Value
2	RDOMAIN	Char	2	Related Domain Abbreviation
1	STUDYID	Char	8	Study Identifier
3	USUBJID	Char	18	Unique Subject Identifier

After truncating, the dataset size reduced from 6.4 GB to 144 MB, resulting in a 98% reduction in data size. Realizing that different methods generally do not affect performance and to simplify the process, we decided to proceed using only Data step with the WHERE statement. We ran the Data step with the WHERE statement using truncated SUPPQUAL data located in a network drive. The following is the result:

Table 3. Comparison of real time, User CPU time, System CPU time and Memory using truncated data from network drive.

Method	Real Time (seconds)	User CPU (Second)	System CPU (Second)	Memory (K)
Data step with WHERE statement	1.23	1.03	0.14	183

The performance displayed in Table 3 is much better than using the original data in a network drive, and real time decreased from one hour to one second.

Compress the data

SAS has a good quality feature to compress the data, and if data is compressed, the size of the data will be smaller and I/O time will be less. Therefore, if the SUPPQUAL data is compressed and the compressed SUPPQUAL data is used as input for the above code, what is the size of the compressed data and what will be the resulting performance?

```
data data.suppqual_c(compress=yes);
  set data.suppqual;
run;
```

Tthe following code was used to compress the data:

The SUPPQUAL data was compressed by 97% and the size reduced to 164 MB. We ran the Data step with the WHERE statement using compressed SUPPQUAL data located in the network drive. The following is the result:

Table 3: Real time, User CPU time, System CPU time and Memory using compressed data from network drive

Method	Real Time (Second)	User CPU (Second)	System CPU (Second)	Memory (K)
Data step with WHERE statement	2.22	1.99	0.17	200

The performance is much better than un-compressed data in the network drive and is similar to using truncated data; real time reduced from one hour to several seconds.

DISCUSSION:

Both truncating and compressing large size data can dramatically improve I/O time and overall performance, and both approaches are easy to implement. However, the one drawback of truncating the character variables is that data attributes will change.

There are two ways to apply compress options - one is global options with compress=yes, the other is applying compress=yes in the data step as shown in the above code. Both approaches will make the output datasets created in the data steps or procedures compressed. However, to improve the programs on which you are working efficiently, you may need to compress the large input data **first**.

CONCLUSION:

In today's clinical trial data represented in the SDTM model, the dataset sizes can easily reach to several gigabytes requesting excessive amount of time to process these datasets within SAS programming environment. Our experiment shows that the real bottleneck in the process is the I/O time in network. By pre-compressing the large input data, the SAS programming performance can be improved dramatically.

REFERENCES

[1] Keiko I. Powers,2001. Efficient Statistical Programming? - Let SAS® Do the Work. SUGI 80-26.
[2] Kirk Paul Lafler. 2000. Efficient SAS® Programming Techniques. SUGI 146-25.
[3] Rick Langston. 2005. Efficiency Considerations Using the SAS® System. SUGI 002-30.
[4] SAS Programming Efficiencies. http://www.ssc.wisc.edu/sscc/pubs/4-3.pdf
[5] Leigh Ihnen and Mike Jones, 2009. Improving SAS® I/O Throughput by Avoiding the Operating System File Cache. http://support.sas.com/resources/papers/proceedings09/327-2009.pdf

ACKNOWLEDGMENTS

The authors greatly acknowledge the review and candid feedback from Donna Usavage, Ellen Asam, Margaret Coughlin and Maryann Williams,.

TRADEMARKS

Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

CONTACT INFORMATION

Eric Qi
Scientific Programming
Biostatistics and Research Decision Sciences
Merck & Co. (UG1D-88)
North Wales, PA 19454-2505
Office: (267) 305 6902
E-mail: Eric_Qi@merck.com

Fikret Karahoda
Scientific Programming
Biostatistics and Research Decision Sciences
Merck & Co. (UG1D-88)
North Wales, PA 19454-2505
Office: (267) 305 1370
Email: fikret_karahoda@merck.com