***Sequential Processing***

SAS is built for sequential processing and is very efficient in this manner. Think about this as in a low level language, such as C. Jiangtang Hu [1] once wrote a blog to compare the similarity of file processing in SAS DATA step and in C. It turns out to be almost identical in nature. This is no incidence. SAS was designed for classical statistical analysis such as ANOVA back in time when analysts used punch cards when sequential processing was the nature and Summary Statistics was the key concept [2]. In linear models, Summary Statistics are usually in a format of summation of the data. For example, in OLS, the necessary information is all in Sum of Square Cross Product matrix. Sequential process naturally fits into the context. In fact, in any matrix languages such as R or MATLAB, there is a mechanism working at the background that enables all these element wise calculation works.

Without doubt, this sequential nature will lower productivity of the programmer since all these element wise calculation has be to handled explicitly by the programmer. But, that is the way to program in SAS, just like C programmers in old time

On the other hand, exactly because of this sequential processing nature, SAS was criticized to be not productive when dealing with problems involving non sequential processing. This is not true.

Here, several examples will be used to demonstrate how SAS programmers can work as comfortably as R programmers in solving a problem with non sequential nature. These examples address issues that were heavily criticized by both mediocre SAS users and other analysts.

1. Calculation between rows:

In many cases, records from different rows are involved in computation. This can be further classified into three categories that will employ different techniques in SAS programming.

1.1 Using values from lagged records of fixe interval. For instance, to calculate the difference between current records and the records two steps before, the lag2() function in a data step can be used. Be cautious that the value from any lag function won’t be initiated until it is first called. SAS-List archive had an extensive discussion on this issue [3].

1.2 Using values from leap forward records. For instance, some computation involves current records and another records several steps ahead. In this case, a classical method is to use FIRSTOBS= data step option, or treat it reversely as the first case, taking the leap forward records as current one while the current record as the lag record. SAS-List archive also had a heated discussion on this issue [4] and sasCommunity.org has a well documented article about the solutions [5].

1.3 Both cases above involve calculation between two records of fixed row difference. In some cases, conditional access is required. This type of work is a special case of so called Table Lookup tasks. There are many very well written papers discussing this topic and I will simply leave it to readers.

In the next example, I will show an example that is also deemed difficult to do in SAS, namely rolling window calculation. Suppose a SAS programmer wants to calculate cumulative statistics summary statistics over a rolling time window for a given data, say total revenue during certain time interval while the rolling time window may contain different number of records and the maximum number is unknown without pass the data once. This is actually a recent question on SAS-L [6]. One quick solution is available via PROC SQL leveraging Cartesian self join. Another more “*SASsy*” approach is to use Multi-Label Format in SAS. Of course there are many other alternatives.

First, we want to summarize for a rolling window of 5 time units.

**data** fmt;

retain fmtname 'rollwindow' type 'n' hlo 'M';

do start=**1** to **10**;

end=start+**5**;

label=cats('time', start);

output;

end;

hlo='O'; label='Out-Of-Bound';

output;

**run**;

**data** dsn;

do time=**1** to **20**;

x=rannor(**0**);

y=ranuni(**0**);

output;

end;

**run**;

**proc** **format** cntlin=fmt; **run**;

**proc** **means** data=dsn noprint;

class time /preloadfmt mlf;

format time rollwindow.;

var x y;

output out=summary\_roll mean(x y)= std(x y)= /autoname;

**run**;

**data** dsn2;

do time=**1** to **20**;

k=ranpoi(**10**, **10**);

do j=**1** to k;

time=time+j/(k+**1**);

x=rannor(**0**); y=ranuni(**0**);

output;

end;

end;

drop k j;

**run**;

**proc** **means** data=dsn2 noprint;

class time/preloadfmt mlf exclusive;

format time rollwindow.;

var x y;

output out=summary\_roll2 mean()= std()=/autoname;

**run**;

As a side exercise, suppose now we want to calculate with a time window of shrinkage number of time units [7]. Solution using Multi-label format is as the following:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

non-rolling but shrinking time window,

similar for growing time window

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**data** fmt2;

retain fmtname 'winx' type 'n' hlo 'M';

do start=**1** to **10**;

end=**18**;

label=cats('time', start);

output;

end;

hlo='O'; label='Out-Of-Bound';

output;

**run**;

**proc** **format** cntlin=fmt2 cntlout=fmt\_all;

**run**;

**proc** **means** data=dsn2 noprint;

class time/preloadfmt mlf exclusive;

format time winx.;

var x y;

output out=summary\_roll3 mean()= std()=/autoname;

**run**;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* An example \*\*\*\*\*\*\*\*\*\*\*\*\*\*;

**data** TradeDate;

input TradeDate yymmdd10.;

format TradeDate yymmdd10.;

cards;

2007-01-04

2007-01-05

2007-01-08

2007-01-09

2007-01-10

2007-01-11

2007-01-12

2007-01-15

2007-01-16

2007-01-17

2007-01-18

2007-01-19

2007-01-22

2007-01-23

2007-01-24

2007-01-25

2007-01-26

2007-01-29

2007-01-30

2007-01-31

;

**run**;

**data** raw;

input id $ Date\_S yymmdd10. +**1** Date\_e yymmdd10. Buy;

format Date\_S Date\_E yymmdd10.;

cards;

A001 2007-01-09 2007-01-24 24.5

A001 2007-01-12 2007-01-16 56.6

;

**run**;

/\*------------------------------ Desired Output --------------------------\*

id Date\_S Date\_E Buy Hold\_Days

A001 2007-01-09 2007-01-24 24.5 12

A001 2007-01-12 2007-01-30 56.6 3

--------------------------------------------------------------------------------\*/

**data** fmt;

set raw;

retain fmtname 'tdate' type 'n' hlo 'M';

start=Date\_S; end=Date\_e;

label=cats(ID, \_n\_);

**run**;

**proc** **format** cntlin=fmt out=fmt\_ref;

**run**;

**proc** **means** data=tradeDate noprint nway;

class TradeDate/mlf exclusive preloadfmt ;

format TradeDate tdate.;

var TradeDate;

output out=\_test n()=\_freq\_;

**run**;

1. Matrix multiplication

Another operation that average SAS programmers feel difficult is a matrix multiplication type work. This job in fact involves Cartesian join which is not readily available in DATA Step. In lower level language, explicitly element-wise operation is required. For example, a pseudo code for Triangular Matrix Multiplication (Algorithm 1.2.1) in *Matrix Computation* [8] looks like this:

***C=0***

for I = 1:n;

***for j=1:n;***

***for k=i:j;***

***C[I,j]=A[i,k] B[k,j] + C[I,j];***

***end;***

***end;***

***end;***

In SAS, such element-wise operation is almost inevitable. But due to the nature of how SAS handle end of file flag, any computation involving Cartesian product is not an easy job. One work around is to use random access with POINT= option, or use the HASH object newly available in SAS9.

Of course, this task can be easily solved using PROC SCORE by understanding that matrix multiplication is exactly what scoring function does for linear model. The syntax is super simple:

**PROC SCORE DATA=MYDATA SCORE=MYSCORE TYPE=PARMS OUT=SCORE;**

**VAR x1-x10;**

**RUN;**

In above code, the scoring data set MYSCORE should have a variable called \_MODEL\_ or \_NAME\_, and a variable called \_TYPE\_ with a value of ‘PARMS’. Of course, the value can be others such as ‘RIDGE’ if a ridge regression was invoked and the estimates are output to a scoring data set. In such case the TYPE= option should be changed accordingly to TYPE=RIDGE.

“A wise man gets more use from his enemies than a fool from his friends”, says a cliché.

The sequential processing nature of SAS makes it very capable in implementing Streaming algorithms [9]. Stream Algorithm itself deserves more than a book length introduction, but I will give a simple example to demonstrate how a stream algorithm naturally fits into SAS.

Jeffrey Scott Vitter explored a fast algorithm to sample K data points without replacement from a data stream of N data points without prior knowledge of N [10]. We will only concentrate on Algorithm R, the simplest one, but implementing more complex Algorithm X, Z in SAS imposes no extra difficulties.

Algorithm R:

***Make an array /auxiliary storage C of size K***

***for j = 0 to K-1, read in the next record into C[j]***

***t:=K***

***while not End-Of-File;***

***t:=t+1;***

***M:=TRUNC(t \* RANDOM());***

***If M<K then***

***read in next record to C[M]***

***else***

***skip current record;***

***end;***

This can be directly translated into SAS, almost line by line:

%inc sasautos(sysrc);

**run**;

%let n\_samp=100000;

**data** samp(index=(idx));

set bigdata(obs=&n\_samp);

where c1='C';

idx=\_n\_;

**run**;

%let start\_obs=%eval(&n\_samp + 1);

**data** samp;

set bigdata(firstobs=&start\_obs) end=eof;

where c1='C';

if \_n\_=**1** then total=&start\_obs;

idx=int(total\*ranuni(&seed))+**1**;

total+**1**;

modify samp point=idx;

if \_iorc\_=%***sysrc***(\_sok) then do;

set bigdata point=total;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* use an auxiliary array upfront \*/

/\* instead of using random access \*/

/\* will save time \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

replace;

end;

else \_error\_=**0**;

**run**;

Note that in this code, we simplified further the process by directly setting a SAS data called BIGDATA. This simplification enables using FIRSTOBS= option in SET statement. Even though in reality, there are cases, such as existing logic deletion, that the data descriptor portion does not contain correct number of observations. Other natural applications include, but not limited to:

1. Using two-way pipe to transfer data to a client SAS machine that further process incoming data stream
2. Online updating of a sample of web activities

In an era of big data, stream processing is becoming ever popular and IMHO, SAS is in a good position to take on the challenges. It is only up to the talent of SAS analysts.

Reference:

[1] Jiangtang Hu; <http://www.jiangtanghu.com/blog/2011/01/03/sas-data-steps-built-in-loop-an-illustrated-example/>

[2] David A. Larson; “Analysis of Variance with Just Summary Statistics as Input”, *The American Statistician*, Vol. 46, No. 2 (May, 1992), pp. 151-152

[3]SAS-L; <http://listserv.uga.edu/cgi-bin/wa?A2=ind1001C&L=sas-l&P=R23292&m=303081>

[4] SAS-L; <http://www.listserv.uga.edu/cgi-bin/wa?A2=ind0503B&L=sas-l&P=R16276>

[5] sasCommunity; <http://www.sascommunity.org/wiki/Four_methods_of_performing_a_look-ahead_read>

[6] SAS-L; <http://listserv.uga.edu/cgi-bin/wa?A2=ind1010C&L=sas-l&P=R12729&m=303081>

[7] Liang Xie; <http://www.sas-programming.com/2010/10/summary-numerical-data-in-rolling.html>

[8] Gene H Golub, Charles F Von Loan (1996); *Matrix Computation*, Ed.3, The John Hopkins University Press

[9] Charu C. Aggarwal *Editor* (2007), *Data Streams: Models and Algorithms*, IBM T. J. Watson Research Center, Yorktown Heights, NY, USA

[10] Jeffrey S Vitter (1985); “Random Sampling with a Reservoir”, ACM Transactions on Mathematical Software, Vol 11, No.1, 37-57