Further Discussion of Summarizing Impossibly Large SAS® Data Sets

Sheng Luo, AEGON Corporation, Frazer, PA
Xinsheng Lin, IMS America, Plymouth Meeting, PA

Abstract

Michael Raithel presented at the Twenty Second SUGI Conference a very interesting concept of the horizontally summarizing the large data set in SAS. The concept is innovative and important at the age of data warehousing. Since both authors have been extensively involved in processing large data sets, we feel especially interested in this concept. Combining Raithel's horizontal summary algorithm with author's own experiences, this paper examines two other algorithms in SAS to further improve performance of the horizontal summarization.

The paper will first review Raithel's horizontal summary algorithm. When the SUMMARY procedure fails to perform on a huge data set with many CLASS variables, his horizontal algorithm would work by sorting by and summarizing at one particular combination of the CLASS variables at a time, and doing so as many times as necessary to reach out all possible combinations of the CLASS variables. While we greatly appreciate for Raithel's concept, we do believe that his algorithm can be changed for better. Our first SAS algorithm is the revision of Raithel's. The revised SAS codes will be displayed and explained. Our specific experiment demonstrated 31% to 48% reduction in CPU time by using the revised algorithm. Next, we discuss our second SAS algorithm, which we have created from our experience in processing large data sets over years. In Raithel's SAS algorithm, the number of SORT procedures required is ((2**N)+1) for N CLASS variables. In ours, it is C(N, K), where the K is INT(N,2), and C(N, K) is the number of combinations of K elements out of N samples. Therefore, to summarize for 6 CLASS variables, the number of PROC SORT required in our algorithm is 20, while in Raithel's, it is 65. The critical part of our algorithm is to generate a combination table, which can then be used to generate the SAS program to do the horizontal summary in more efficient mode. Our specific experiment shows 49% to 72% decease in CPU time by applying the second algorithm. The paper will present the complete SAS codes for implementation of the algorithm and the concept of the combination table.

In summary, the purpose of this paper is to describe two SAS algorithms to improve the horizontal summarization. We wish that the paper could benefit other SAS users working with large data sets.

Review of the Horizontal Summary Technique

The task is to summarize a SAS data set at every level or _TYPE_ of CLASS variables. In SAS expression, we have:

```
PROC SUMMARY SUM DATA=IN.DATA;
CLASS X1-X&N;
VAR C1-C&M;
OUTPUT OUT=NEW.DATA SUM=;
```

where &N and &M are the numbers of CLASS variables and computation variables. The number of combinations of CLASS variables, or _TYPE_'s, in the output data set is 2**&N. For example, there are 8 _TYPE_'s for 3 CLASS variables, and 256 combinations for 8 CLASS variables. When &N is large, or when CLASS variables have many different values, or when the data set is large, the SUMMARY procedure with a 'CLASS X1-X&N' statement can fail.

Mr. Raithel presented an excellent solution, called the horizontal summarization. Instead of summarizing all levels of combinations at once, the horizontal algorithm processes one by one a particular level of summarization at a time until all combinations of CLASS variables are exhausted. His SAS algorithm is as follow.

1. Concatenate all CLASS variables into a new variable STRING. This requires a DATA step in SAS.

2. Sort the output data set from step 1 and summarize using NWAY option by STRING.

3. Loop 2**N times to perform steps 4 to 6 in sequence. For 6 CLASS variables, for example, loop 64 times from 0 to 63 as follow:

```
%MACRO BIGLOOP;
%DO BICOUNT=0 %TO 63 %BY 1;
.....codes for steps 4 to 7;
%END;
%MEND BIGLOOP;
%BIGLOOP;
```

Here the &BICOUNT is equal to the value of the _TYPE_ variable in the SUMMARY procedure.

4. Read in the data set from step 2, redefine the value of STRING so that subsequent summary processes only CLASS variables corresponding to current _TYPE_ as defined by &BICOUNT. The example below is taken from Raithel's with some change in variable names.

```
DATA TEMP;
SET NEW.BASEDATA;
```

```
FORMAT X BINARY6.;
X=&BICOUNT;
IF X='0.....'B THEN SUBSTR(STRING,01,04)=' ';   /*X1*/
IF X='.0....'B THEN SUBSTR(STRING,05,02)=' ';   /*X2*/
......
IF X='.....0'B THEN SUBSTR(STRING,21,10)=' ';   /*X6*/
```

5. Sort the output data set from step 4 and summarize by STRING.

6. Restore all CLASS variables from STRING.

Using this algorithm, we can summarize impossibly large data set as long as the SORT procedure in step 1 can succeed. The algorithm can also be modified easily for any number of CLASS variables. It does not require heavy programming effort.

While we applaud for this concept, we do however believe that his algorithm carries some unnecessary I/O extensive steps. In his SAS algorithm, there are 2**(N+1)+1 DATA steps; 1 for step one; 2**N for step 4; another 2**N for step 6. When dealing with the impossibly large data set, these DATA steps are CPU consuming and expensive. Also, the lowest level or highest _TYPE_ of classes is summarized twice; once in steps 1 and 2 and once in steps 4 to 6, an unnecessary duplication. We can improve the performance of the horizontal summary by eliminating and revising some steps of his algorithm, and adding a new step. This is what resulted in our first SAS algorithm, the revised version of the original.

First Algorithm, Revised from Raithel's

The first algorithm is a revision of the Raithel's. Specifically, we eliminated his steps 1, 4 and 6. We also replaced his step 3 with our codes to generate names of CLASS variables so to avoid many I/O extensive DATA step. Appendix One displays a complete sample codes of our SAS algorithm for 6 CLASS variables. Here are descriptions of the algorithm.

1. Sort the input data set and summarize using NWAY option BY X1-X&N, where &N is the number of CLASS variables. The output data is the summary at the highest _TYPE_.

2. Loop (2**N)-1 times to perform steps 3 and 4 in sequence. For 6 CLASS variables, for example, loop 63 times from 0 to 62. In the sample program, the &BICOUNT is equal to the value of the _TYPE_ variable in the SUMMARY procedure. Please note that _TYPE_ 63 is omitted from the loop because it is already produced from step 1.

3. For each _TYPE_ in the loop, assign names of corresponding CLASS variables to a macro variable &X. We use the DATA _NULL_ because there is no input or output of the data sets for this step. The only output is the macro variable &X. The variable KEYS is to contain the names of selected CLASS variables. Its length should be at least total length of all CLASS variable names, plus the number of CLASS variables and then minus one. In the sample program, the total length of all CLASS variable names is 12. The number of CLASS variables is 6. Therefore the length of KEYS should be at least 17 (=12+6-1).The names of CLASS variables 'X1' to 'X6' are initiated to a temporary array XX. The next DO loop runs through every element of the array XX. The BAND (bitwise AND) function evaluates the correspondence between current _TYPE_ defined as &BICOUNT and the Ith CLASS variable defined by 2**(6-I). If BAND equals to 0, meaning no correspondence, then the ith element of XX is blacked out. Otherwise, the ith element of the XX is the name of the ith CLASS variable. Then all elements of the XX are assigned to the variable KEYS. The last 'CALL SYMPUT' statement sends the value of KEYS to the macro variable &X. In subsequent SORT and SUMMARY procedures, the 'BY &X' statements are used to carry the names of corresponding CLASS variables to process. Thus the composition and decomposition of the variable STRING in Raithel's algorithm are no longer necessary, neither are those related DATA steps.

4. Sort the output data set from step 2 and summarize by &X, where &X gives a list of CLASS variables corresponding to the current value of &BICOUNT or _TYPE_. The output data set from this step is the summary at the level corresponding to the current value of _TYPE_.

The major difference between our and Raithel's algorithms is in that ours does not have any DATA steps to manipulate values of the CLASS variables through variable STRING. so that 2**(N+1) I/O extensive DATA steps are avoided. Of cause our algorithm still have to process 2**N-1 times on the DATA _NULL_ in step 3. But compared with CPU required for steps 1, 4 and 6 in Raithel's algorithm, the CPU used for step 3 in our algorithm is extremely trivial. As Raithel's algorithm, the revised can also be modified easily for any number of CLASS variables. It does not require heavy programming effort.

Authors compared the performance on UNIX system of the 2 algorithms. The numbers of CLASS variables used are 6 and 7. The numbers of observations of data sets are 1.3 million and 10.5 million respectively. We found the revised algorithm results in 31% to 48% saving in CPU time.

Second Algorithm, From Our Experience

Our second algorithm is based on our premise that in SAS, the number of SORT procedures required for horizontally summarizing N CLASS variables is no more than C(N, K), where K is the integer part of N divided by 2, and C(N,K) is the number of combinations of K elements out of N samples. For example, the number of SORT procedures required for 6 CLASS variables is 20 (here K=INT(6,2)=3, and C(6,3)=20). In Raithel's algorithm, the number of SORT procedure required for N CLASS variables is at least 2**N, that would translate to 64 for 6 CLASS variables. Therefore, 45 SORT procedures can be saved by applying the premise in place of Raithel's algorithm for 6 CLASS variables. This is where we can reach for further improvement.

To demonstrate the premise, lets use an example of 3 CLASS variables, X1 to X3. The number of SORT procedures required should be 3 by the premise. Here is how we process.

1.0 Sort the original data by X1, X2 and X3;
1.1 Summarize the output of step 1.0 by X1, X2 and X3, for _TYPE_ 7;

1.2 Summarize the output of step 1.1 by X1 and X2, for _TYPE_ 6;
1.3 Summarize the output of step 1.1 by X1, for _TYPE_ 4;

2.0 Sort the output data from step 1.0 by X2 and X3;
2.1 Summarize the output from step 2.0 by X2 and X3, for _TYPE _ 3;
2.2 Summarize the output from step 2.1 by X2, for _TYPE_ 2;

3.0 Sort the output data from step 1.0 by X3 and X1;
3.1 Summarize the output of step 3.0 by X3 and X1, for _TYPE_ 5;
3.2 Summarize the output of step 3.1 by X3, for _TYPE_ 1;
3.3 Summarize the output of step 3.2 without BY statement, for _TYPE_ 0.

In steps 1.1, 1.2, 2.1, 2.2, 3.1, 3.2 and 3.3, there are no SORT procedures because the input data sets are already in sorting order.

In Raithel's algorithm, the process is:

1 Sort from the original data set, and summarize by X1 X2 X3, for _TYPE_ 7;

2 Sort from the output from step 1 and summarize by nothing, for _TYPE_ 0;

3 Sort from the output from step 1 and summarize by X3, for _TYPE_ 1;

4 Sort from the output from step 1 and summarize by X2, for _TYPE_ 2;

5 Sort from the output from step 1 and summarize by X2 X3, for _TYPE_ 3;

6 Sort from the output from step 1 and summarize by X1, for _TYPE_ 4;

7 Sort from the output from step 1 and summarize by X1 X3, for _TYPE_ 5;

8 Sort from the output from step 1 and summarize by X1 X2, for _TYPE_ 6;

9 Sort from the output from step 1 and summarize by X1 X2 X3, for _TYPE_ 7.

Therefor his algorithm needs 8 SORT procedures at least without step 9, because step 9 is a duplication of step 1.

By comparison, our algorithm not only saves 6 SORT procedures, but also the subsequent SUMMARY procedures use ever smaller input data sets for each SORT block. Thus it improves CPU consumption.

The CPU reduction of our algorithm is achieved through rearranging order of CLASS variables in By statements. When the number of CLASS variables is relatively small, such as the case of 3 CLASS variables, one can easily figure out the necessary order of CLASS variables and then simply type in the SAS codes as the example described. However, if the number of CLASS variables is large, such as 5 or more, not only the order of CLASS variable for the minimum sorting becomes less obvious. but also the number of SAS statements becomes too many for one to type. Therefore, it is good to have some tools to facilitate implementation of our algorithm. The first of these tools is to create a combination table. An example of the combination table for 4 CLASS variables is as follow:

Table 1. Combination Table For 4 Class Variables

1	**2**	**3**	**4**
1	**4**	**2**	
1	**3**	**4**	
2	**3**	**4**	
4	**2**		
3	**4**		

In this table, the numbers 1 to 4 represent CLASS variables X1 to X4 respectively. Any row of the table represent the order of CLASS variables in a SORT procedure. For instance, the 2nd row has values 1, 4, 2, which represents a 'BY X1 X4 X2' statement in SORT procedure. Once you sort the data set by X1 X4 X2, you can summarize by X1 X4 X2, then by X1 X4, and then by X1. If you look closely, the first column includes all combinations of C(4,1); the first two columns include all combinations of C(4,2); the first 3 columns include all combinations of C(4,3); and first 4 columns include all combinations of C(4,4). In general the first j columns should include all combinations of C(N, j). In this way, summary at all combinations of CLASS variables can be reached through minimum SORT procedures.

After creation of the combination table, we can use the SAS program displayed in Appendix 2 to quickly generate another SAS program for horizontal summary. In the sample program of Appendix 2, the variables X1 to X4 are simply the combination table in Table one. The variables Y1 to Y4 are corresponding _TYPE_ values for CLASS variables in horizontal summary. For example, in row 4, X1 is 2 and Y1 equals 4, which means summarizing at second Class variable is for _TYPE_ 4. In row 4, X2 is 3 and Y2 is 6, which means summarizing the second and the third CLASS variables corresponds _TYPE_ 6. The variable MISS is the number of missing values of X1 to X4 in current row. The variable THROW represents how many columns of X1 to X4 need to drop off from summarizing. In rows 2 and 3, the values of THROW are 1, which means not to summarize at X1 because it is already done in row 1.

Appendix 3 displays the SAS codes, the output from the sample program in Appendix 2, to execute the second algorithm. As you can see from these SAS codes, the number of SORT procedures used is 6, which is the number of combinations of 2 elements out of 4. The number of SUMMARY procedures is 16, which is the result of 2 raised to a power of 4.

We conducted experiment on UNIX system for 6 and 7 CLASS variables, and 1.3 million and 10.5 million observation data sets. The CPU is 49% and 72% less than Raithel's algorithm. However creation of the combination table may consume a lot of time for large numbers of CLASS variables. This is a disadvantage of our algorithm. Authors have worked out a systematic way to create the combination table for any number

of CLASS variables, but we are still in effort to put it in program codes. But if you frequently run such impossibly large data sets it is worth investing time on creating combination tables even manually.

Summary

We presented 2 SAS algorithms for improving the horizontal summarization presented by Mr. Michael Raithel. We provided description analysis and results of experiments to compare algorithms. From theoretical point of view, our algorithms appear to be able to enhance Raithel's algorithm. Results of our experiment also supported the theoretical analysis. We wish this paper could benefit to SAS users who deals with large data sets.

References

SAS Institute Inc. (1991) SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07, Cary, NC: SAS Institute Inc.

Michael Raithel, "Summarizing Impossibly Large SAS® Data Sets For The Data Warehouse Server Using Horizontal Summarization", Proceedings Of The 22nd Annual SUGI Conference, PP514-518.

SAS Institute Inc. (1990), SAS Language, Reference Version 6, First Edition, Cary, NC:SAS Institute Inc.

SAS Institute Inc. (1990), SAS Procedure Guide Version 6, Third Edition, Cary, NC:SAS Institute Inc.

Author Contact

Sheng Luo
AEGON Corporation
20 Moores Road
Frazer, PA 19355
Phone: (610) 648-5065
Email: sheng_luo@pvn.com

Xinsheng Lin
IMS Amerrica LTD
660 West Germantown Pike
Plymouth Meeting, PA 19462
Phone: (610)832-5532
Email: xinshl@imsint.com

Appendix 1: Sample SAS Program for The First Algorithm

```
%MACRO BIGLOOP;

*Step 1: Sort the input data set*;
PROC SORT DATA=IN.DATA OUT=SORTED;
BY X1-X6;
PROC SUMMARY MISSING NWAY SUM DATA=SORTED;
BY X1-X6;
VAR C1-C4;
OUTPUT OUT=IN.S63(DROP=_TYPE_) SUM=;
*End of step 1*;

*Step 2: Loop steps 3 and 4*;
%DO BICOUNT=0 %TO 62 %BY 1;

*Step 3: Generate BY variable list*;
DATA _NULL_;
LENGTH KEYS $ 17;
ARRAY XX{6} $ _TEMPORARY_ ('X1 ','X2 ','X3 ','X4 ','X5
','X6');
DO I=1 TO 6;
IF BAND((2**(6-I)), &BICOUNT)=0 THEN XX{I}=' ';
KEYS=LEFT(TRIM(KEYS)||' '||LEFT(XX{I}));
END;
CALL SYMPUT('X',KEYS);
*End of step 3*;

*Step 4: Sort & summarize the output from step 1*;
PROC SORT DATA=IN.S63 OUT=TEMP;
BY &X;
PROC SUMMARY MISSING NWAY SUM DATA=TEMP;
BY &X;
VAR C1-C4;
OUTPUT OUT=IN.S&BICOUNT(DROP=_TYPE_) SUM=;
*End of step 4*;

%END;
*End of step 2*;
%MEND BIGLOOP;

*Execute the macro*;
%BIGLOOP;
```

Appendix 2: Sample SAS Program For Implementating The Second Algorithm

```
PROC FORMAT;
VALUE FMT 1='X1' 2='X2' 3='X3' 4='X4' .=' ';
RUN;

DATA P;
LENGTH X1-X4 T1-T4 3 Y1-Y4 $ 50 TYPE $ 10;
ARRAY X{*} X1-X4;
ARRAY T{*} T1-T4;
ARRAY Y{*} Y1-Y4;
INFILE CARDS EOF=LAST;
INPUT JUNK X1-X4 T1-T4 MISS THROW;
FILE 'SAMPLE.SAS';
IF _N_=1 THEN DS='IN.DATA;';
ELSE DS='IN.S15;';
PUT 'PROC SORT OUT=TEMP DATA=' DS $8.;
Y1=PUT(X1,FMT.);
```

```
DO I=2 TO (4-MISS);
Y{I}=TRIM(Y{I-1})||' '||PUT(X{I}, FMT.);
END;
PUT 'BY ' Y{4-MISS} $ '; RUN;';
TYPE='TEMP';
DO I=(4-MISS) TO (1+THROW) BY -1;
PUT 'PROC SUMMARY DATA=' TYPE $  'MISSING SUM
NWAY;';
PUT 'BY ' Y{I} $ ';';
PUT 'VAR C1-C9;';
TYPE='IN.S'||LEFT(T{I});
PUT 'OUTPUT OUT=' TYPE $ 'SUM=; RUN;';
END;
PUT;
PUT;
RETURN;
LAST:
FILE 'SAMPLE.SAS';
PUT 'PROC SUMMARY DATA=IN.S1 MISSING SUM
NWAY;';
PUT 'VAR C1-C9;';
PUT 'OUTPUT OUT=IN.S0 SUM=; RUN;';
CARDS;
 1     1     2     3     4     8     12     14     15     0  0
 2     1     4     2     .     8     9     13     .     1  1
 3     1     3     4     .     8     10     11     .     1  1
 4     2     3     4     .     4     6     7     .     1  0
 5     4     2     .     .     1     5     .     .     2  0
 6     3     4     .     .     2     3     .     .     2  0
;
RUN;
```

Appendix 3. The Second SAS Algorithm, Output From the Sample Program In Appendix 2

```
PROC SORT OUT=TEMP DATA=IN.DATA;
BY X1 X2 X3 X4 ; RUN;
PROC SUMMARY DATA=TEMP MISSING SUM NWAY;
BY X1 X2 X3 X4 ;
VAR C1-C9;
OUTPUT OUT=IN.S15 SUM=; RUN;
PROC SUMMARY DATA=IN.S15 MISSING SUM NWAY;
BY X1 X2 X3 ;
VAR C1-C9;
OUTPUT OUT=IN.S14 SUM=; RUN;
PROC SUMMARY DATA=IN.S14 MISSING SUM NWAY;
BY X1 X2 ;
VAR C1-C9;
OUTPUT OUT=IN.S12 SUM=; RUN;
PROC SUMMARY DATA=IN.S12 MISSING SUM NWAY;
BY X1 ;
VAR C1-C9;
OUTPUT OUT=IN.S8 SUM=; RUN;


PROC SORT OUT=TEMP DATA=IN.S15;
BY X1 X4 X2 ; RUN;
PROC SUMMARY DATA=TEMP MISSING SUM NWAY;
BY X1 X4 X2 ;
VAR C1-C9;
OUTPUT OUT=IN.S13 SUM=; RUN;
PROC SUMMARY DATA=IN.S13 MISSING SUM NWAY;
BY X1 X4 ;
VAR C1-C9;
OUTPUT OUT=IN.S9 SUM=; RUN;


PROC SORT OUT=TEMP DATA=IN.S15;
BY X1 X3 X4 ; RUN;
PROC SUMMARY DATA=TEMP MISSING SUM NWAY;
BY X1 X3 X4 ;
VAR C1-C9;
OUTPUT OUT=IN.S11 SUM=; RUN;
PROC SUMMARY DATA=IN.S11 MISSING SUM NWAY;
BY X1 X3 ;
VAR C1-C9;
OUTPUT OUT=IN.S10 SUM=; RUN;


PROC SORT OUT=TEMP DATA=IN.S15;
BY X2 X3 X4 ; RUN;
PROC SUMMARY DATA=TEMP MISSING SUM NWAY;
BY X2 X3 X4 ;
VAR C1-C9;
OUTPUT OUT=IN.S7 SUM=; RUN;
PROC SUMMARY DATA=IN.S7 MISSING SUM NWAY;
BY X2 X3 ;
VAR C1-C9;
OUTPUT OUT=IN.S6 SUM=; RUN;
PROC SUMMARY DATA=IN.S6 MISSING SUM NWAY;
BY X2 ;
VAR C1-C9;
OUTPUT OUT=IN.S4 SUM=; RUN;


PROC SORT OUT=TEMP DATA=IN.S15;
BY X4 X2 ; RUN;
PROC SUMMARY DATA=TEMP MISSING SUM NWAY;
BY X4 X2 ;
VAR C1-C9;
OUTPUT OUT=IN.S5 SUM=; RUN;
PROC SUMMARY DATA=IN.S5 MISSING SUM NWAY;
BY X4 ;
VAR C1-C9;
OUTPUT OUT=IN.S1 SUM=; RUN;


PROC SORT OUT=TEMP DATA=IN.S15;
BY X3 X4 ; RUN;
PROC SUMMARY DATA=TEMP MISSING SUM NWAY;
BY X3 X4 ;
VAR C1-C9;
OUTPUT OUT=IN.S3 SUM=; RUN;
PROC SUMMARY DATA=IN.S3 MISSING SUM NWAY;
BY X3 ;
VAR C1-C9;
OUTPUT OUT=IN.S2 SUM=; RUN;


PROC SUMMARY DATA=IN.S1 MISSING SUM NWAY;
VAR C1-C9;
OUTPUT OUT=IN.S0 SUM=; RUN;
```