WUSS 40-2018

How to Handle Big Files and Reduce Execution times using *SAS*®

Kaiqing Fan, Mastech Digital Inc.

ABSTRACT

As a SAS Developer or SAS user, we are always struggling with the long execution times of our SAS engines because of large data files. Sometimes it would be a couple hours, some other times, it may be more than 20 or 30 hours, or longer. In my eyes, too long execution times are not acceptable. Actually we have many SAS technical skills, and if we can use them properly, we can hugely shorten the execution time. I did it. I successfully shortened the execution time from 36 hours to around 1-2 hours; from 4 hours to 6 minutes. Here I want to summarize most of the technical skills I used and share them with you.

Key Words: Execution time, efficiency of execution, SAS Techniques

INTRODUCTION

As a SAS Developer or SAS user, we are always struggling with the long execution time of our SAS model engines, sometimes it would be couple hours, some other times, it may be more than 20 or 30 hours, or longer, our longest engine was 4 whole days. In my eyes, too long execution time is not acceptable, it would cause waiting in queue, burning money to buy servers and memories. Actually we have rich SAS technical skills and experiences, if we can use them properly, we can hugely shorten the execution time. I did it. I successfully shortened the execution time from 36 hours to around 1-2 hours; from 4 hours to 6 minutes. Here I want to summarize most of the technical skills and experience I used and share with you.

1, USING SUBSET OR SAMPLING FOR YOUR SAS CODE DEVELOPMENT

During developing engines, we would like to suggest you use sampling from big data to reduce the size of data set, and remember that millions of millions of observations and computations take a long time to finish execution!

Please remember sampling has a potential risk, we can use sampling for engine development, but validation must be full size of data.

```
1)  Of the options firstobs=  and  obs=
    (1) Data new;
    Set old (obs=1000);
    run;
    (2) option obs=100;
    PROC IMPORT DATAFILE="&inpth./&txtfilename..txt" OUT=txt_sas7bdat_filename
     DBMS=dlm  REPLACE; DELIMITER='|'; GETNAMES=NO;  guessingrows=1;
    run;
    option obs=100;
    (3)  data COMPT.LEAD_txt;
    infile "&D_IN/&txtname" dlm='|' dsd missover truncover firstobs=2 lrecl=32767;
    format &fmtlist.  ;
    input &inptlst.;
    if _N_ = 100 then stop;
    run;
2)  Proc surveyselect
3)  Seed ( )
```

2, DROP YOUT BURDEN

Using `keep` =, `drop`=, or their statements to only keep the useful columns or only required columns, or drop useless columns when you read the big data files. This way can drop a heavy burden especially when data is big with too many useless columns. Usually in the development requirements, columns that would be required, and columns that would not be useful are not explained clearly or are ignored. But as a developer, we should ask the business analyst to make it as clear as possibly.

One extreme example is that I developed one PII column masker engine. One of the files was 8 GB with 263 columns, but only required 6 columns to be masked 1000 times for safety. If we loop 1000 times with mask function with the whole data, it would take 30-40 hours, but I picked the 6 columns out, after masking, then merged back, and it only took 20-25 minutes. **[1]**

3, WHERE OPTION/STATEMENT PROCESSES MORE EFFICIENT THAN IF OPTION/STATEMENT

The where statement/option is more efficient than the if statement/option since the where statement makes judgments when reading in, whereas the if statement makes judgments after completing reading the data set. But please remember that if can be used for newly defined variables, where cannot.

Two types of where statement/option, as option and as where statement.
Example 1:

```
Data new;
Set old(where = (type in ('line','bar','pie')));
where type in ('line','bar','pie');
Run;
```

Example 2:

```
Data new;
Set old;
New_variable = strip(type)||strip(_N_);
/*we can only use the if statement or option here*/

Run;
```

For the new created New_variable, you cannot use where in the same data step.

4, PROC DATASETS

When you want to change or modify the label or formats of variables, **Proc datasets** is faster than data steps and procedures.

5, PROC APPEND IS MORE EFFICIENT DATA STEP

When you want to append a dataset, **proc append** is more efficient than **data** step; **set** statement.

But proc append has its tricks. If the appended data sets have different data structure, different column order, or different lengths for the same columns, it will cause troubles.

6, OPTIONS COMPRESS=YES;

Be careful when using it especially during development, since it will compress the new created data file, and not show the results on your SAS window. So if you want to use it, make sure that your engines are 100% correct!

7, CLASS STATEMENT IS MORE EFFICIENT THAN GROUP STATEMENT

In proc procedures, when we need to group variables, the class statement is more efficient than the by statement because the by statement performs sort too.

Examples: **Proc means, Proc univariate.**

8, FREE THE SPACE

When you have to use PROC IML, and it stores the dataset into a SAS library or memory, don't forget to release the space. Example 1,

```
SASfile test2 load;
… …
SASfile test close;
```

Example 2: any time you don't want to use the dataset, close it.

```
Use datasetname;
… …
Close datasetname;
```

9, DATA STEP IS MORE EFFICIENT THAN PROC IML

In a matrix/vector language such as SAS/IML or R, developers should use matrix and vector operations instead of scalar computations whenever possible.

But I utilize SAS data step to implement vector calculations instead of proc iml matrix calculation since proc iml is time-consuming in SAS.

10, AVOID OUTPUTTING DATA TO FOLDER

If not required, please try not to output the immediate data files into the perminant library because it takes more time. Example: The first data step is faster than the below one because of temporary file created.

```
%let Path = /sas/model/business_model/market/sell/temporary;
LIBNAME Path "&Path";
Data new;      set old; run;  ------ faster than the below one because of temporary file created.
Data Path.new; set old; run;
```

11, HASH OBJECT IS MORE EFFICIENT THAN DATA STEP MERGE

Hash can quickly query and read datasets and merge data files without sorting. Hash is more efficient than the merge statement but it consumes lots of memory storage. Many times, a company does not give you enough memory for big data.

In my PII column masker engine, the data file with 8 GB, when I merged back with hash object, it failed because of not enough memory.

```
data OUT_PATH.outfile_name(drop=rc);
retain &original_retain_list. SALT_:;
if 0 then set COMPUT.masked_required_fields_regular;
declare hash hh_add(dataset:"COMPUT.masked_required_fields_regular");
rc=hh_add.defineKey("SALT_KEY");
rc=hh_add.defineData(all:'yes');
rc=hh_add.defineDone();
do until(eof);
set original_columns(drop=&mask_list.) end=eof;
rc=hh_add.find();
if rc=0 then output;
end;
```

```
stop;
run;
```

12, PROC DS2

I tried this method, but need lots of time because "DS2 language is too complex for many SAS-users." I studied it for more than 20 hours but still am confused about how to use it correctly. What I need is the following:

Data step code like
`Data new; Set old;`
`If money > 0 then type="profit";`
`Run;`
	Please anyone can provide one Proc ds2 example:
	How could we code through proc ds2 ??????

13, VIEW OPTION IN DATA STEP

Using the view option in data step or procedures can save lots of space and running time.

Example,

```
data test1/view=test1;
Do i = 1 to 1000000000000000000000;     y=i**2;    Output;    End;
Run;
```

14, USE _NULL_; IF POSSIBLE

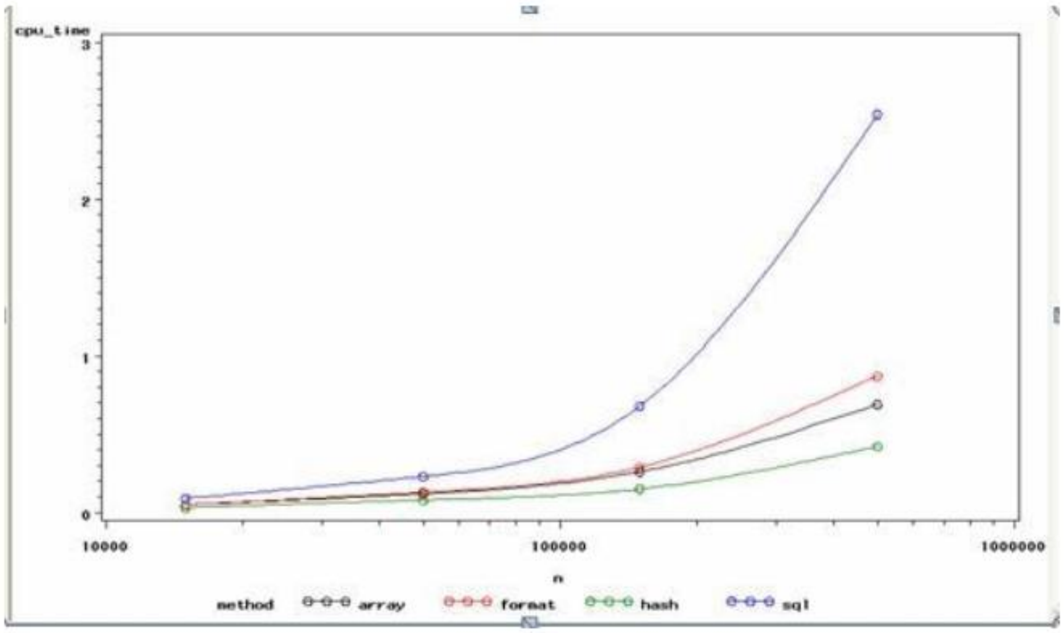
Any time if you don't need to create a dataset in a data step, please use `_NULL_;`

```
data _null_;    set old;     run;
```

15, DATA STEP INSTEAD OF PROC SQL

The data step merge is much faster than proc sql. If proc sql is not needed or the **BEST** choice, use the data step merge as much as you can.

Comparison of Speed among Hash Object, Merge via Format, Data Step, Proc SQL



Performance Analysis of different equivalents of Proc SQL in SAS **[2]**
https://www.quora.com/Are-there-any-performance-benchmarks-on-PROC-SQL-SAS-vs-SQL

16, PARALLEL PROCESSING METHOD

Add more multiple threads when using servers such as linux/unix, but burning money, especially most times, the number of threads or servers you can use is limited because of budget. There are many parallel processing formats. I would like to provide two examples.

```
%macro parallel_script(taskname=);
       systask command
       "sasgsub -gridsubmitpgm &path./&taskname..sas -METAPASS yourpwd. -gridwait"
       taskname="&taskname.";
       data _null_;        x = sleep(3,1);      run;
%mend parallel_script;

%macro parallel;
       %parallel_script(taskname=step1_inputs1);
       %parallel_script(taskname=step1_inputs2);
waitfor _ALL_    step1_inputs1   step1_inputs2;
%mend parallel;
%parallel;

%macro run_parallel;
%let src=%sysfunc(grdsvc_enable(_all_, resource=SASDev));
options autosignon;

%macro six_parallel_grid;
%do runid=1 %to 6;
%syslput _ALL_ / remote=task&runid;
       rsubmit task&runid wait=no;
           %prediction(runid=&runid);
       endrsubmit;
%end;
%mend six_parallel_grid;
%six_parallel_grid;
waitfor _all_ task1 task2 task3 task4 task5 task6;
signoff _all_;
%mend run_parallel;
%run_parallel;
```

17, COMBINATION PROCESSING METHOD [3] [4]

It is the **inverse process of parallel processing method**. Usually when the number of certain data files with same data structure in a folder or sub-groups in a big data is huge, as per business requirements, businessmen prefer to explain them one file or sub-group by at a time because it is very easy to clearly explain the requirements. However, it needs endless looping, and is very time consuming. Considering this issue, we propose the combination processing method.

Parallel method is to distribute files into different servers or threads. It is like distributing 10 files to 10 servers, so the time can decrease 90% off. It needs more and more servers or threads when data become bigger and bigger or execution time would be longer and longer because the number of servers are limited.

The combination processing method is that: in the very beginning, we combine or stack or append all of hundreds of thousands data files with same data structure or sub-groups with unique IDs into a single combined file, then the SAS engine only needs to process this single combined file through all SAS steps once. We can get the correct solutions, and significantly decrease execution time. Our experience was that we reduced more than 90% of execution time.

It does not require additional server memory or additional money. It can be used together with all other SAS skills. The more the data files or sub-groups, the more time it can save compared with one by one processing.

```
%macro one_by_one(file_name=);
Data &file_name._step1;
Set &file_name;
Run;
/*there are many procedures such as proc sort,  data step merge, proc transpose,
proc means, proc freq, proc sql with sum function inside of this one_by_one macro */
%mend;
%one_by_one(file_name=file1);
… …
%one_by_one(file_name=file100000);
```

For the combination method, append all files into one single files in the very beginning

```
%macro Combined_method(file_name=);
Data &file_name._step1;
Set &file_name;
UNIQUE_FILE_ID ="&file_name";
Run;

Proc append base=combined_files data=&file_name._step1  force;  run;
%mend Combined_method;
%Combined_method(file_name=file1);
… …
%Combined_method(file_name=file100000);
```

We have automation method to process it.
/*All procedures such as proc sort, data step merge, proc transpose, proc means, proc freq,

proc sql with sum function are outside of this Combined_method macro now*/

Once appending all files, we don't need SAS macro anymore.

Proc sort data=&file_name;
By sorted_groups_keys;
run;
	Proc sort data=combined_files;
	By UNIQUE_FILE_ID sorted_groups_keys;
	Run;

Data each_merged_data;
Merge &file_name1(in=a)
&file_name2(in=b);
By sorted_keys;
If a=1 and b=1;
Run;
	Data merged_data;
	Merge combined_files1(in=a)
	combined_files2(in=b);
	By UNIQUE_FILE_ID sorted_keys;
	If a=1 and b=1;
	Run;

proc transpose data=&file_name
out=each_group_transpose (drop=_LABEL_)
name=PERIOD;
id ID_VARIABLE;   var _NUMERIC_;
by sorted_groups_keys;
run;
	proc transpose data=Combined_files
	out=Combined_Files_transpose (drop=_LABEL_)
	name=PERIOD;
	id ID_VARIABLE; var _NUMERIC_;
	by UNIQUE_FILE_ID sorted_groups_keys;
	run;

CONTACT INFORMATION

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

Kaiqing Fan

Sr. Data Scientist

Sr. SAS Developer Lead

Mastech Digital Inc.

Address: 6750 Miller Road, Brecksville, OH-44141

Mobile: 504.344.7267

Email: fankaiqinguw@gmail.com

Linkedin: https://www.linkedin.com/in/fan-kaiqing-81776940/

REFERENCES

[1] Four Solutions to Mask Any Target Columns In Your Data Using SAS Enterprise Guide

Global Big Data Conference 2018, Santa Clara, CA

[2] *Performance Analysis of different equivalents of Proc SQL in SAS*

https://www.quora.com/Are-there-any-performance-benchmarks-on-PROC-SQL-SAS-vs-SQL

[3] Kaiqing Fan

Optimize the Logic A Little, Shorten the Execution Time A Lot ---- A New Combination Processing Method to Handle Big Complex Files

Global Big Data Conference 2018, Santa Clara, CA

[4] Kaiqing Fan

A More Efficient Big Data Processing Method - The Reverse Processing to Parallel Processing - Introduction to the Combination Processing Method

Submitted to IEEE BigData 2018, Seattle, WA

THANKS LETTER

Thanks so much for the corrections and the valuable patience from Schembri, Michael to make the paper more readable!