SAS® GLOBAL FORUM 2018

USERS PROGRAM

Using In-Database Technology to Boost the Efficiency of Data Analytics

April 8 – 11 | Denver, CO
#SASGF

Using In-Database Technology to Boost the Efficiency of Data Analytics

Don McCarthy, Mike Santema, and Qing Yuan

Kaiser Permanente Department of Research and Evaluation

Introduction

Data volume is growing at an unprecedented pace in the health care industry, and working with Big Data can be both time consuming and challenging for SAS programmers. One solution to the problems of Big Data that is offered by SAS to users with access to data within a Relational Database Management System (RDBMS) such as Teradata® is In-Database processing. In this paper, we use a subset of Kaiser Permanente's Electronic Health Record (EHR) data to test the efficiency gains offered by In-Database processing for several popular Base SAS procedures analytic processes within the Teradata Database. Sample code is provided for each BASE SAS Procedure explored and the logs from Conventional Processing and In-Database Processing are compared.

Data

For this paper, we use a specially prepared subset of Kaiser Permanente's EHR data related to utilizations (e.g. encounters between patient and healthcare providers, procedures performed by providers, and diagnoses made by providers). Kaiser Permanente has the largest private-sector EHR dataset in the U.S. and EHR data for Kaiser Permanente Southern California includes more than 900 million encounters from data inception though 2017. We selected data related only to hospital setting encounters (100 million records) and their related diagnoses and procedures.

In-Database Processing

Conventional Processing of data located within a RDBMS involves the SAS/ACCESS® engine generating a SQL SELECT * statement that fetches all rows and columns from the table referenced and bringing them into SAS where the processing occurs. With a large table; this results in substantial network latency and can take much time. SAS offers In-Database Processing as a solution for this problem. In-Database processing offers advantages over Conventional Processing as the SAS/ACCESS engine delegates more processing to the RMBS and fetches the minimum amount of data into SAS. In SAS 9.4, the following Base SAS Procedures are available for In-Database processing using the SAS/ACCESS engine with BASE SAS and Teradata :

PROC FREQ, PROC MEANS, PROC REPORT, PROC SUMMARY, PROC TABULATE. Additional SAS STAT Procedures are available for In-Database Processing, and PROC TRANSPOSE is available for In-Database Processing with the SAS In-Database Code Accelerator.

The SQLGENERATION system option or LIBNAME statement option controls whether and how in-database procedures are run inside the database. By default, the in-database procedures are run inside the databases when possible.

Conclusion

It is useful to compare the real and CPU times of the above Procedures as submitted Conventionally and In-Database and to consider the In-Database Processing Times as a percentage of the Conventional Processing times. These results are presented in tabular form below. While we were pleasantly surprised by how quickly SAS Conventional Processing provided results from a reasonably-large sized Teradata table, In-Database Processing times were much faster and, as a share of Convectional Processing times ranged from 1 to 48 percent. Clearly, substantial gains in efficiency can be had by leveraging In-Database processing.

PROC	Conventional Processing	In-Database Processing		In-Database Processing as % of Conventional	
		Using PROC	Using Pass-Through	Using PROC	Using Pass-Through
EXPAND	04:04.01	n/a	00:07.68	n/a	3%
FREQ	06:20.83	00:04.46	00:05.66	1%	1%
MEANS	03:42.10	00:02.11	00:00.73	1%	1%
MEANS (PERCENTILES)	03:38.72	03:38.10	01:15.14	100%	34%
TABULATE	06:01.21	00:03.19	n/a	1%	n/a
TRANSPOSE	14:47.71	n/a	07:08.02	n/a	48%

References

SAS Institute. 2017. SAS/ACCESS® 9.4 for Relational Databases: Reference. 9th ed. Cary, NC: SAS Institute.

SAS Institute. 2017. SAS Analytical Products 13.2 Documentation. Cary, NC: SAS Institute.

SAS Institute. 2008. SAS In-Database Processing with Teradata: An Overview of Foundation Technology. Cary, NC: SAS Institute.

Using In-Database Technology to Boost the Efficiency of Data Analytics

Don McCarthy, Mike Santema, and Qing Yuan

Kaiser Permanente Department of Research and Evaluation

Procedures

PROC EXPAND allows the programmer to convert time series data from one frequency or time step to another (e.g. monthly to annual), to replace missing data with several interpolation schemes, to perform certain rolling calculations, and to generate lagging or leading values of numerical data.

PROC EXPAND is not available in SAS 9.4 for In-Database processing, however, explicit pass though allows one to take advantage of Teradata's massively parallel processing to replicate certain of PROC EXPAND's functionality. We compare calculating two moving averages using PROC EXPAND with Conventional Processing and explicit pass though In-Database Processing.

Our data has 20 columns including ones for: the number of procedures performed per encounter (px_n) , the number of diagnoses made per encounter (dx_n), the unique identifier for each encounter (util_id), a datetime admission date (adate), and a admission year. We will make 6 month moving averages of dx_n and px_n using data from 1980 to 2017. This is common analysis for a SAS programmer working healthcare to perform as it can be useful in exploring patterns of data capture and in provider behavior.

The BASE SAS Conventional Processing syntax is a combination of a PROC SQL query that fetches the data from Teradata (libname=td_work) and makes and monthly date and monthly means and the PROC EXPAND procedure.

```
proc sql;
  create table dxpx1
    as select
       mdy(month(datepart(adate)),1,adyr) as dat format=mmddyy10.
      ,mean(dx_n) as dx_n
      ,mean(px_n) as px_n
    from td_work.hsp_util
      where adyr>1979
    group by 1 order by 1;
quit;
proc expand DATA = dxpx1 OUT = dxpx;
  convert dx_n = ma_dx_n / METHOD = none TRANSFORMOUT = (cmovave 6);
  convert px_n = ma_px_n / METHOD = none TRANSFORMOUT = (cmovave 6);
run;
```

Procedures (contd)

We next perform the same calculations using explicit pass through to use In-Database Processing.

```
proc sql;
connect to teradata (user=&terauser password=&terapwd
connection=global mode=TERADATA);
execute (create multiset table  td_work.dxpx as
    (select
        a.dat
       ,MAVG(a.dx_n,6,a.dat) as dx_n
       ,MAVG(a.px_n,6,a.dat) as px_n from
        (
          select
              (adyr-1900)*10000 + extract(month from adate) * 100 + 1
                    (DATE)  as dat
             ,average(dx_n) as dx_n
             ,average(px_n) as px_n
          from td_datamart.hsp_util
             where adyr>1979
          group by 1
        ) a
   )
    with data no primary index  ;
  ) by teradata; disconnect from teradata;
quit;

data util_detail;
  set td_work.util_detail;
run;
```

Like Conventional, In-Database Processing must first make monthly dates and monthly means; then it is able to use MAVG() to generate the moving averages.

Procedures (contd)

Finally, the data are brought down to SAS using a DATASTEP. A comparison of logs is interesting. The log for the Conventional Processing is:

```
NOTE: Table WORK.DXPX1 created, with 461 rows and 3 columns.
NOTE: PROCEDURE SQL used (Total process time):
      real time           4:03.96
      cpu time            4:22.50

NOTE: PROCEDURE EXPAND used (Total process time):
      real time           0.05 seconds
      cpu time            0.01 seconds
```

The SQL query brings down the entire table from Teradata and uses it to calculate the 461 rows of monthly sums and takes just over four minutes: quite good performance for a reasonably large table. The PROC EXPAND procedure takes less than a second to return results.

The log for the In-Database Processing is:

```
NOTE: PROCEDURE SQL used (Total process time):
      real time           2.18 seconds
      cpu time            0.00 seconds

NOTE: Teradata connection: TPT FastExport has read 461 row(s).
NOTE: The data set WORK.DXPX has 461 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           5.50 seconds
      cpu time            0.38 seconds
```

The initial Teradata SQL query takes a bit over 2 seconds and the DATASTEP brings down the resulting table to SAS in a bit under 6 seconds. What took 4 and 22.51 second in Conventional Processing took 7.68 seconds in In-Database processing. While the time for Conventional Processing is not unreasonable, the programmer can improve efficiency markedly using In-Database Processing even with a dataset of around 100 million rows and 20 columns.

Using In-Database Technology to Boost the Efficiency of Data Analytics

Don McCarthy, Mike Santema, and Qing Yuan

Kaiser Permanente Department of Research and Evaluation

Procedures (contd)

PROC FREQ computes descriptive statistics based on unique values in the data set and produces n-way tabular reports. It is an essential procedure within BASE SAS used primarily for counting, displaying and analyzing categorical type data.

These tools are used extensively by the programmer working in a healthcare setting, as for counting, doing error checking of the data or categorizing data. It produces one-way to n-way frequency and cross tabulation (contingency) tables. For two-way tables, PROC FREQ computes tests and measures of association. For n-way tables, PROC FREQ does stratified analysis, computing statistics within, as well as across, strata.

We compare performing a two-dimensional table between care setting type and care setting subtype to calculate the frequencies, percentages, cumulative frequencies and cumulative percentages using the built-in intuitive syntax present in PROC FREQ, which is already optimized in SAS 9.4 for In-Database processing:

```
** FREQ **;
proc freq data=dluser.hsp_util;
          table care_type * care_subtype / list;
Run;
```

PROC FREQ will by default add several statistics to the table when we define two dimensions. We choose to keep the default statistics here in the example.

We next perform the same calculations using explicit pass through method:

```
** FREQ **;
proc sql;
connect to teradata (user=&terauser password=&terapwd
connection=global mode=TERADATA);
execute (create multiset table  td_work.freq as
    (select
            care_type
          , care_subtype
          , count(*) as Frequency
          , count(*) * 100.00 / sum(count(*)) over () as Percentage
```

Procedures (contd)

```
** FREQ **;
          , sum(count(*)) over (order by care_type, care_subtype
            rows unbounded preceding) as CumulativeFrequency
          , 100.00 * CumulativeFrequency / sum(count(*)) over ()
            as CumulativePercent
      from dl_res_user.hsp_util
           group by care_type, care_subtype
           )
           with data no primary index;
           ) by teradata;
disconnect from teradata;
quit;

data util_freq;
  set td_work.freq;
run;
```

A simple 2-dimensional table is generated. Conventional Processing must download the data to SAS then do the counting by care_type. The log for the Conventional Processing is:

```
NOTE: There were 53214875 observations read from the data set
DLUSER.hsp_util
NOTE: PROCEDURE FREQ used (Total process time):
      real time           6:20.83
      cpu time            3:30.78
```

The same code, with In-Database processing turned on showed significant improvement for the same procedure:

```
NOTE: PROCEDURE FREQ used (Total process time):
      real time           4.46 seconds
      cpu time            0.48 seconds
NOTE: SQL generation will be used to construct frequency and
crosstabulation tables.
```

Procedures (contd)

In-database processing is evident from the log note regarding SQL generation. The SASTRACE option setting allows the generated SQL query to be printed to the SAS log, as well.

```
TERADATA_2: Prepared: on connection 4
 select COUNT(*) as "ZSQL1", case  when COUNT(*) >
COUNT(TXT_1."CARE_TYPE") then ' ' else MIN(TXT_1."CARE_TYPE") end as
"ZSQL2",
case  when COUNT(*) > COUNT(TXT_1."CARE_SUBTYPE") then ' ' else
MIN(TXT_1."CARE_SUBTYPE") end as "ZSQL3" from
"DL_RES_USER"."hsp_util" TXT_1 group by TXT_1."CARE_TYPE",
TXT_1."CARE_SUBTYPE“
```

The generated SQL query was passed to DBMS for fetching data. And bring them back to SAS for processing.

Finally, Explicit pass through Teradata appeared to perform similarly as in-Database processing:

```
NOTE: PROCEDURE SQL used (Total process time):
      real time           4.76 seconds
      cpu time            0.09 seconds
NOTE: The data set WORK.UTIL_FREQ has 39 observations and 6
variables.
NOTE: DATA statement used (Total process time):
      real time           0.90 seconds
      cpu time            0.06 seconds
```

Enabling in-database procedures to delegate more of the processing to the Teradata Database by generating more complex SQL and passing it to Teradata optimizes the PROC FREQ processing. More work is done in the database and less data movement occurs. The SAS procedures can be modified to dynamically generate SQL queries that reference Teradata SQL functions. This approach provides performance improvements.

Using In-Database Technology to Boost the Efficiency of Data Analytics

Don McCarthy, Mike Santema, and Qing Yuan

Kaiser Permanente Department of Research and Evaluation

Procedures (contd)

PROC MEANS and PROC SUMMARY are easily-recognized by most SAS programmers as basic tools for deriving simple statistics about a dataset. These procedures offer a wide array of arithmetic functions like means and standard deviations, quickly calculated by basic calculators. PROC MEANS is also for examining the distribution of data, relying on orders and to yield medians and percentiles of numeric data.

These tools are particularly important to the programmer working in a healthcare setting, as descriptive statistics need to be gathered on how a large population of patients encounter the healthcare provider overall.

We compare performing means, standard deviation, and percentile calculations on a length of stay variable, marking how many hours patients are in the hospital (los_hrs) using the built-in intuitive syntax present in PROC MEANS, which is already optimized in SAS 9.4 for In-Database processing (and the same code run again with PROC SUMMARY instead of PROC MEANS):

```
** Mean/SD **;
proc means noprint
          data=dluser.hsp_util;
          var los_hrs;
          output out=mean_imp
          mean=mean_los
          stddev=sd_los;
Run;

** Percentiles **;
proc means noprint
          data=dluser.hsp_util;
          var los_hrs;
          output out=percentiles_imp
          min=min_los
          p25=p25_los
          median=median_los
          p75=p75_los
          max=max_los;
run;
```

Procedures (contd)

We next perform the same calculations using explicit pass through:

```
** Mean/SD **;
proc sql;
connect to teradata (tdpid=tdp2 user=&terauser password=&terapwd );
          create table mean_exp as
          select * from connection to teradata(
                    select avg(los_hrs) as mean_los
                    ,         stddev_samp(los_hrs) as sd_los
                    from dl_res_user.hsp_util
          );
disconnect from teradata;
quit;

** Percentiles **;
proc sql; connect to teradata (tdpid=tdp2 user=&terauser
password=&terapwd);
  create table percentiles_exp as
  select * from connection to teradata(
     select min(los_hrs) as min_los
,    percentile_disc(0.25) within group (order by los_hrs) as p25_los
,    percentile_disc(0.5) within group (order by los_hrs) as
          median_los
,    percentile_disc(0.75) within group (order by los_hrs) as p75_los
,    max(los_hrs) as max_los
     from dl_res_user.hsp_util
  );
disconnect from teradata;
quit;
```

Explicit Teradata code also has functions built in for these simple ranges, like AVG( ) for mean, and PERCENTILE_DISC( ) for percentiles.

Procedures (contd)

A simple 1-record results table is generated in both the means and percentile tables. Conventional Processing must download the data to SAS then do the means. The log for the Conventional Processing is:

```
NOTE: The data set MEAN_IMP has 1 observations and 4 variables.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           3:42.10
      cpu time            2:23.51
NOTE: The data set PERCENTILES_IMP has 1 observations and 7 variables
NOTE: PROCEDURE MEANS used (Total process time):
      real time           3:38.72
      cpu time            2:21.76
```

The same code, with In-Database processing turned on showed improvement only for the means and standard deviation, percentiles was about the same:

```
NOTE: PROCEDURE MEANS used (Total process time):
      real time           2.11 seconds
      cpu time            0.31 seconds
NOTE: The data set PERCENTILES_IMP has 1 observations and 7 variables
NOTE: PROCEDURE MEANS used (Total process time):
      real time           3:38.10
      cpu time            2:17.45
```

Explicit pass through Teradata code appeared to be faster for both means and percentile situations, although:

```
NOTE: Table MEAN_EXP created, with 1 rows and 2 columns.
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.73 seconds
      cpu time            0.07 seconds
NOTE: Table PERCENTILES_EXP created, with 1 rows and 5 columns.
NOTE: PROCEDURE SQL used (Total process time):
      real time           1:15.14
      cpu time            0.07 seconds
```

Results using PROC SUMMARY instead of PROC MEANS were the same.

Using In-Database Technology to Boost the Efficiency of Data Analytics

Don McCarthy, Mike Santema, and Qing Yuan

Kaiser Permanente Department of Research and Evaluation

Procedures (contd)

PROC TABULATE allows the programmer to display descriptive statistics in tabular format. It computes many statistics that are computed by other procedures, such as MEANS, FREQ, and REPORT. It then displays the results of these statistics in a table format. It is very useful for programmers in a healthcare setting to generate report, it can produce tables in up to three dimensions and allows, within each dimension, multiple variables to be reported one after another hierarchically.

TABULATE is also available in SAS 9.4 for In-Database processing, however, explicit pass though method is excessively cumbersome to generate.

We compare average and maximum length of stay variables, marking how many hours patients are in the hospital (los_hrs) by care sub type and admission year using PROC TABULATE with Conventional Processing and In-Database Processing.

```
proc tabulate data=dluser.hsp_util;
class adyr care_subtype;
var los_hrs;
keylabel N = ‘N’;
tables
          adyr all = “Total” * f = comma10.,
          los_hrs * care_subtype * (N Mean Max) * f = comma10. /
          misstext=‘0’;
run;
```

The log for conventional processing in SAS is:

```
NOTE: There were 53214875 observations read from the data set
DLUSER.hsp_util

NOTE: PROCEDURE TABULATE used (Total process time):
      real time           6:01.21
      cpu time            4:39.71
```

Procedures (contd)

Input data is stored as a table or view in a database management system (DBMS), the PROC TABULATE procedure can use in-database processing to perform most of its work within the database.

The log for the In-Database Processing is:

```
NOTE: SQL generation will be used to perform the initial
summarization.
NOTE: PROCEDURE TABULATE used (Total process time):
      real time           3.19 seconds
      cpu time            1.45 seconds
```

The results returns results in less than 5 seconds, it provides the advantages of faster processing and reduced data transfer between the Teradata and SAS.

PROC TABULATE performs in-database processing by using SQL implicit pass-through. The procedure generates SQL queries that are based on the classifications and the statistics that specify in the TABLE statement. The Teradata executes these SQL queries to construct initial summary tables, which are then transmitted to PROC TABULATE.

The In-Database processing recorded in the log is:

```
TERADATA_0: Prepared: on connection 2
SELECT * FROM DL_RES_USER."hsp_util"

        proc tabulate data=dluser.hsp_util;
        class adyr care_subtype ;
        var los_hrs;
        keylabel N='N' ;
        tables
        adyr all="Total"*f=comma10.,
             los_hrs*care_subtype * (N Mean Max)
*f=comma10./misstext='0';
         run;
```

Procedures (contd)

```
TERADATA: tryoeinf()
TERADATA: tryoeinf()
TERADATA: tryoeinf()
NOTE: SQL generation will be used to perform the initial
summarization.

TERADATA_1: Prepared: on connection 3
SELECT * FROM DL_RES_USER."hsp_util"

TERADATA_2: Prepared: on connection 4
 select COUNT(*) as "ZSQL1", MIN(TXT_1."ADYR") as "ZSQL2",
MIN(TXT_1."CARE_SUBTYPE") as "ZSQL3", COUNT(*) as "ZSQL4",
COUNT(TXT_1."los_hrs") as "ZSQL5", MAX(TXT_1."los_hrs") as "ZSQL6",
SUM(CAST( TXT_1."los_hrs" AS DOUBLE PRECISION)) as "ZSQL7" from
"DL_RES_USER"."hsp_util" TXT_1 group by TXT_1."ADYR",
TXT_1."CARE_SUBTYPE"

TERADATA: trforc: COMMIT WORK
ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching
data.

TERADATA_3: Executed: on connection 4
 select COUNT(*) as "ZSQL1", MIN(TXT_1."ADYR") as "ZSQL2",
MIN(TXT_1."CARE_SUBTYPE") as "ZSQL3", COUNT(*) as "ZSQL4",
COUNT(TXT_1."los_hrs") as "ZSQL5", MAX(TXT_1."los_hrs") as "ZSQL6",
SUM(CAST( TXT_1."los_hrs" AS DOUBLE PRECISION)) as "ZSQL7" from
"DL_RES_USER"."hsp_util" TXT_1 group by TXT_1."ADYR",
TXT_1."CARE_SUBTYPE"

TERADATA: trget - rows to fetch: 849
TERADATA: trforc: COMMIT WORK
TERADATA: trforc: COMMIT WORK
```

Explicit passthrough with Teradata SQL syntax is less attractive as it requires, when there are a large number of categories to summarize (here our data have 39 categories and more than 30 years) a cumbersome transpose of the data involving numerous UNION ALL joins or a complicated CROSS JOIN. Syntax for a simpler transpose and its performance will be shown in the next section.

Using In-Database Technology to Boost the Efficiency of Data Analytics

Don McCarthy, Mike Santema, and Qing Yuan

Kaiser Permanente Department of Research and Evaluation

Procedures (contd)

PROC TRANSPOSE allows the programmer convert rows to columns and columns to rows (e.g. observations to variables and variables to observations). PROC TRANSPOSE is not available in SAS 9.4 for In-Database processing, however, explicit pass though allows one to transpose data in Teradata. We compare transposing the variables: procedures performed per encounter (px_n) and the number of diagnoses made per encounter (dx_n) using PROC TRANSPOSE with Conventional Processing and explicit pass though In-Database Processing.

The BASE SAS Conventional Processing syntax is PROC TRANSPOSE and fetches the entire table from Teradata (libname=td_work) which is then transposed in SAS.

```
proc transpose data=td_work.hsp_util keep= util_id dx_n px_n)
out=util_detail (keep=util_id _label_ col1 rename= (_label_=
util_detail col1= counts));
    by util_id;
run;
```

We next perform the same calculations using explicit pass though.

```
proc sql; connect to teradata (user=&terauser password=&terapwd
connection=global mode=TERADATA);
execute (create multiset table  td_work.util_detail as
  (select
       a.util_id
     ,'dx_n' as util_detail
     ,a.dx_n as counts
   from td_db.hsp_util a
   union all
   select
      b.util_id
     ,'px_n' as util_detail
     ,b.px_n as counts
   from td_db.hsp_util b
  )
with data primary index(util_id) ; by teradata;
disconnect from teradata; quit;
```

Procedures (contd)

Teradata SQL does not have a transpose function; however, it is relatively straightforward to transpose columns using a combination of a SELECT query and a UNION ALL. A comparison of logs is interesting. The log for the Conventional Processing is:

```
NOTE: The data set WORK.UTIL_DETAIL has 106429750 observations and 3
variables.

NOTE: PROCEDURE TRANSPOSE used (Total process time):
      real time           14:47.71
      cpu time            8:55.98
```

The 3 columns needed from the table of 100 million rows are brought down to SAS using a DATASTEP and then its transposed using PROC TRANSPOSE. It is quite time-consuming. The log for the In-Database Processing is:

```
NOTE: PROCEDURE SQL used (Total process time):
      real time           8.91 seconds
      cpu time            0.00 seconds
NOTE: Teradata connection: TPT FastExport has read 106429750 row(s).
NOTE: The data set WORK.UTIL_DETAIL has 106429750 observations and 3
variables.
NOTE: DATA statement used (Total process time):
      real time           6:59.11
      cpu time            3:19.49
```

The SQL query returns results in less than 10 seconds, but the time to bring down the 100 million rows of transposed data erodes the In-Database Processing advantage to some extent as it takes almost 7 minutes.

SAS® GLOBAL FORUM 2018

April 8 – 11 | Denver, CO
Colorado Convention Center

#SASGF

Paper 2823-2018

Using In-Database Technology to Boost the Efficiency of Data Analytics

Don McCarthy, Mike Santema, and Qing Yuan

Kaiser Permanente Department of Research and Evaluation

ABSTRACT

To gain efficiency in a big data environment, we are able to run and optimize key SAS procedures analytic processes within the Teradata® Database. The scope of this paper is to explore the In-Database processing advantages, the most frequently used SAS analytical and reporting In-Database procedures, which are executed inside the Teradata database directly, and the comparison of efficiency between direct SAS analytic processing and SAS In-Database processing with Teradata system. Finally, sample programs are provided for each explored In-Database analytical procedures.

INTRODUCTION

Data volume is growing at an unprecedented pace in the healthcare industry, and working with big data can be both time consuming and challenging for SAS programmers. One solution to the problems of big data that is offered by SAS to users with access to data within a Relational Database Management System (RDBMS) such as Teradata® is In-Database processing. In this paper, we use a subset of Kaiser Permanente's Electronic Health Record (EHR) data to test the efficiency gains offered by In-Database processing for several popular Base SAS procedures analytic processes within the Teradata Database. Sample code is provided for each BASE SAS Procedure explored and the logs from Conventional Processing and In-Database Processing are compared.

For this paper, we use a specially prepared subset of Kaiser Permanente's EHR data related to utilizations (e.g. encounters between patient and healthcare providers, procedures performed by providers, and diagnoses made by providers). Kaiser Permanente has the largest private-sector EHR dataset in the U.S. and EHR data for Kaiser Permanente Southern California includes more than 900 million encounters from data inception though 2017. We selected data related only to hospital setting encounters (100 million records) and their related diagnoses and procedures.

IN-DATABASE PROCESSING

Conventional Processing of data located within a RDBMS involves the SAS/ACCESS® engine generating a SQL SELECT * statement that fetches all rows and columns from the table referenced and bringing them into SAS where the processing occurs. With a large table; this results in substantial network latency and can take much time. SAS offers In-Database Processing as a solution for this problem. In-Database processing offers advantages over Conventional Processing as the SAS/ACCESS engine delegates more processing to the RDBMS and fetches the minimum amount of data into SAS. In SAS 9.4, the following BASE SAS Procedures are available for In-Database processing using the SAS/ACCESS engine with Base SAS and Teradata: PROC FREQ, PROC MEANS, PROC REPORT, PROC SUMMARY, and PROC TABULATE. Additional SAS STAT Procedures are available for In-Database Processing, and PROC TRANSPOSE is available for In-Database Processing with the SAS In-Database Code Accelerator.

PROCEDURES: CONVENTIONAL AND IN-DATABASE

We compare the performance of PROC EXPAND, PROC FREQ, PROC MEANS, PROC SUMMARY, PROC TABULATE, and PROC TRANSPOSE with Conventional Processing and two types of In-Database Processing: the first using the PROC syntax with In-Database Processing enabled and the second using Teradata SQL syntax with SAS/ACCESS explicit passthrough. For BASE SAS PROCs where In-Database Processing is possible in SAS 9.4, we perform both Conventional and In-Database Processing by editing the program OPTIONS.

Conventional: `OPTIONS sqlgeneration=none;`

In-Database: `OPTIONS sqlgeneration=dbms;`

PROC EXPAND

PROC EXPAND allows the programmer to convert time series data from one frequency or time step to another (e.g. monthly to annual), to replace missing data with several interpolation schemes, to perform certain rolling calculations, and to generate lagging or leading values of numerical data.

PROC EXPAND is not available in SAS 9.4 for In-Database processing, however, explicit passthrough allows one to take advantage of Teradata's massively parallel processing to replicate certain of PROC EXPAND's functionality. We compare calculating two moving averages using PROC EXPAND with Conventional Processing and explicit passthrough In-Database Processing.

Our data has 20 columns including ones for: the number of procedures performed per encounter (px_n), the number of diagnoses made per encounter (dx_n), the unique identifier for each encounter (util_id), a datetime admission date (adate), and an admission year. We will make 6 month moving averages of dx_n and px_n using data from 1980 to 2017. This is common analysis for a SAS programmer working healthcare to perform as it can be useful in exploring patterns of data capture and in provider behavior.

The BASE SAS Conventional Processing syntax is a combination of a PROC SQL query that fetches the data from Teradata (libname=td_work) and makes and monthly date and monthly means and the PROC EXPAND procedure.

```
proc sql;
  create table dxpx1
    as select
       mdy(month(datepart(adate)),1,adyr) as dat format=mmddyy10.
      ,mean(dx_n) as dx_n
      ,mean(px_n) as px_n
    from td_work.hsp_util
       where adyr>1979
  group by 1 order by 1;
quit;


proc expand DATA = dxpx1 OUT = dxpx;
   convert dx_n = ma_dx_n / METHOD = none TRANSFORMOUT = (cmovave 6);
   convert px_n = ma_px_n / METHOD = none TRANSFORMOUT = (cmovave 6);
run;
```

We next perform the same calculations using explicit passthrough to use In-Database Processing.

```
proc sql noerrorstop;
connect to teradata
  (user=&terauser password=&terapwd connection=global mode=TERADATA);
      execute
      (create multiset table  td_work.dxpx as
        (select
               a.dat
              ,MAVG(a.dx_n,6,a.dat) as dx_n
              ,MAVG(a.px_n,6,a.dat) as px_n from
              (
                     select
                      (adyr-1900)*10000 +
                      extract(month from adate) * 100 +
                      1 (DATE)  as dat
```

```
                        ,average(dx_n) as dx_n
                        ,average(px_n) as px_n
                              from td_datamart.hsp_util
                              where adyr>1979
                        group by 1
                  ) a
            )
            with data no primary index  ;
        ) by teradata;
    disconnect from teradata;
    quit;


    data dxpx;
          set dluser.dxpx;
    run;
```

Like Conventional, In-Database Processing must first make monthly dates and monthly means; then it is able to use MAVG() to generate the moving averages. Finally, the data are brought down to SAS using a DATA step. A comparison of logs is interesting. The log for the Conventional Processing is:

```
    NOTE: Table WORK.DXPX1 created, with 461 rows and 3 columns.

    NOTE: PROCEDURE SQL used (Total process time):

          real time           4:03.96

          cpu time            4:22.50

    NOTE: PROCEDURE EXPAND used (Total process time):

          real time           0.05 seconds

          cpu time            0.01 seconds
```

The SQL query brings down the entire table from Teradata and uses it to calculate the 461 rows of monthly sums and takes just over four minutes: quite good performance for a reasonably large table. The PROC EXPAND procedure takes less than a second to return results.

The log for the In-Database Processing is:

```
    NOTE: PROCEDURE SQL used (Total process time):

          real time           2.18 seconds

          cpu time            0.00 seconds

    NOTE: Teradata connection: TPT FastExport has read 461 row(s).

    NOTE: The data set WORK.DXPX has 461 observations and 3 variables.

    NOTE: DATA statement used (Total process time):

          real time           5.50 seconds

          cpu time            0.38 seconds
```

The initial Teradata SQL query takes a bit over 2 seconds and the DATASTEP brings down the resulting table to SAS in a bit under 6 seconds. What took 4 and 22.51 second in Conventional Processing took 7.68 seconds in In-Database processing. While the time for Conventional Processing is not unreasonable, the programmer can improve efficiency markedly using In-Database Processing even with a dataset of around 100 million rows and 20 columns.

PROC FREQ

PROC FREQ computes descriptive statistics based on unique values in the data set and produces n-way tabular reports. It is an essential procedure within BASE SAS used primarily for counting, displaying and analyzing categorical type data.

These tools are used extensively by the programmer working in a healthcare setting, as for counting, doing error checking of the data or categorizing data. It produces one-way to n-way frequency and cross tabulation (contingency) tables. For two-way tables, PROC FREQ computes tests and measures of association. For n-way tables, PROC FREQ does stratified analysis, computing statistics within, as well as across, strata.

We compare performing a two-dimensional table between care setting type and care setting subtype to calculate the frequencies, percentages, cumulative frequencies and cumulative percentages using the built-in intuitive syntax present in PROC FREQ, which is already optimized in SAS 9.4 for In-Database processing:

```
proc freq data= td_work.hsp_util;
   table care_type * care_subtype / list;
run;
```

PROC FREQ will by default add several statistics to the table when we define two dimensions. We choose to keep the default statistics here in the example.

We next perform the same calculations using explicit passthrough method:

```
proc sql;
connect to teradata (user=&terauser password=&terapwd connection=global
mode=TERADATA);
execute (create multiset table  td_work.freq as
     (select
               care_type
             , care_subtype
             , count(*) as Frequency
             , count(*) * 100.00 / sum(count(*)) over () as Percentage
             , sum(count(*)) over (order by care_type, care_subtype
               rows unbounded preceding) as CumulativeFrequency
             , 100.00 * CumulativeFrequency / sum(count(*)) over ()
               as CumulativePercent
       from td_datamart.hsp_util
              group by care_type, care_subtype
            )
              with data no primary index;
    ) by teradata;
disconnect from teradata;
quit;

data util_freq;
  set td_work.freq;
run;
```

A simple two-dimensional table is generated. Conventional Processing must download the data to SAS then do the counting by care_type. The log for the Conventional Processing is:

```
NOTE: There were 53214875 observations read from the data set
TD_WORK.hsp_util

NOTE: PROCEDURE FREQ used (Total process time):
      real time           6:20.83
      cpu time            3:30.78
```

The same code, with In-Database processing enabled showed significant improvement for the same procedure:

```
NOTE: PROCEDURE FREQ used (Total process time):
      real time           4.46 seconds
      cpu time            0.48 seconds
NOTE: SQL generation will be used to construct frequency and
crosstabulation tables.
```

In-database processing is evident from the log note regarding SQL generation. The SASTRACE option setting allows the generated SQL query to be printed to the SAS log, as well. The generated SQL query was passed to RDBMS for fetching data. And bring them back to SAS for processing.

Finally, Explicit pass through Teradata appeared to be similar as in-Database processing:

```
NOTE: PROCEDURE SQL used (Total process time):
      real time           4.76 seconds
      cpu time            0.09 seconds
NOTE: The data set WORK.UTIL_FREQ has 39 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.90 seconds
      cpu time            0.06 seconds
```

Enabling In-Database Procedures to delegate more of the processing to the Teradata Database by generating more complex SQL and passing it to Teradata optimizes the PROC FREQ processing. More work is done in the database and less data movement occurs. The SAS procedures can be modified to dynamically generate SQL queries that reference Teradata SQL functions. This approach provides performance improvements.

PROC MEANS AND SUMMARY

PROC MEANS and PROC SUMMARY offer a wide array of arithmetic functions like means and standard deviations, quickly calculated by basic calculators. PROC MEANS is also useful for examining the distribution of data, relying on orders and to yield medians and percentiles of numeric data.

We compare performing means, standard deviation, and percentile calculations on a length of stay variable, marking how many hours patients are in the hospital (los_hrs) using the built-in intuitive syntax present in PROC MEANS, which is already optimized in SAS 9.4 for In-Database processing (and the same code run again with PROC SUMMARY instead of PROC MEANS).

We first calculate means and standard deviations.

```
proc means noprint
  data= td_work.hsp_util;
  var los_hrs;
  output out=mean_imp
  mean=mean_los
  stddev=sd_los;
run;
```

We then calculate statistics for interquartile range, median, and extreme outliers:

```
proc means noprint
  data= td_work.hsp_util;
  var los_hrs;
  output out=percentiles_imp
  min=min_los
  p25=p25_los
  median=median_los
  p75=p75_los
  max=max_los;
run;
```

As with PROC FREQ, this code can be run with Conventional and In-Database Processing. We do both.

We then perform the same calculations using In-Database Processing through explicit passthrough. First we calculate means and standard deviations

```
proc sql;

connect to teradata (user=&terauser password=&terapwd connection=global
mode=TERADATA);
  create table mean_exp as
  select * from connection to teradata(
        select avg(los_hrs) as mean_los
        ,     stddev_samp(los_hrs) as sd_los
        from td_datamart.hsp_util
  );
disconnect from teradata;
quit;
```

We then calculate statistics for interquartile range, median, and extreme outliers:

```
proc sql; connect to teradata (user=&terauser password=&terapwd
connection=global mode=TERADATA);
  create table percentiles_exp as
  select * from connection to teradata(
     select
        min(los_hrs) as min_los
      ,percentile_disc(0.25) within group (order by los_hrs) as p25_los
      ,percentile_disc(0.5) within group (order by los_hrs) as median_los
      ,percentile_disc(0.75) within group (order by los_hrs) as p75_los
      ,max(los_hrs) as max_los
     from td_datamart.hsp_util
  );
disconnect from teradata;
quit;
```

Explicit passthrough also allows the programmer to access Teradata functions for these simple ranges, such as AVG( ) for mean, and PERCENTILE_DISC( ) for percentiles. A simple 1-record results table is generated in both the means and percentile tables.

Conventional Processing must download the data to SAS then calculate the means. The log for the Conventional Processing is:

```
NOTE: The data set MEAN_IMP has 1 observations and 4 variables.

NOTE: PROCEDURE MEANS used (Total process time):

      real time           3:42.10

      cpu time            2:23.51
```

```
NOTE: The data set PERCENTILES_IMP has 1 observations and 7 variables
NOTE: PROCEDURE MEANS used (Total process time):
      real time           3:38.72
      cpu time            2:21.76
```

The same code, with In-Database processing enabled showed improvement only for the means and standard deviation, percentiles was similar to Conventional Processing:

```
NOTE: PROCEDURE MEANS used (Total process time):
      real time           2.11 seconds
      cpu time            0.31 seconds
NOTE: The data set PERCENTILES_IMP has 1 observations and 7 variables
NOTE: PROCEDURE MEANS used (Total process time):
      real time           3:38.10
      cpu time            2:17.45
```

Explicit passthrough Teradata code appeared to be faster for both means and percentile situations, however:

```
NOTE: Table MEAN_EXP created, with 1 rows and 2 columns.
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.73 seconds
      cpu time            0.07 seconds
NOTE: Table PERCENTILES_EXP created, with 1 rows and 5 columns.
NOTE: PROCEDURE SQL used (Total process time):
      real time           1:15.14
      cpu time            0.07 seconds
```

Results using PROC SUMMARY instead of PROC MEANS, in both Conventional and with In-Database Processing environments, were the same.

PROC TABULATE

PROC TABULATE allows the programmer to display descriptive statistics in tabular format. It computes many statistics that are computed by other procedures, such as MEANS, FREQ, and REPORT. It then displays the results of these statistics in a table format. It is very useful for programmers in a healthcare setting to generate reports, it can produce tables with more than two dimensions and allows, within each dimension, multiple variables to be reported one after another hierarchically. TABULATE is also available in SAS 9.4 for In-Database processing, however, an explicit SQL passthrough method is excessively cumbersome to generate.

We compare average and maximum length of stay variables, marking how many hours patients are in the hospital (los_hrs) by care sub type and admission year using PROC TABULATE with Conventional Processing and In-Database Processing.

```
proc tabulate data=td_work.hsp_util;
  class adyr care_subtype;
  var los_hrs;
  keylabel N = 'N';
  tables
        adyr all = "Total" * f = comma10.,
            los_hrs * care_subtype * (N Mean Max) * f = comma10. /
```

```
                 misstext='0';
run;
```

The log for the Conventional Processing is:

```
NOTE: There were 53214875 observations read from the data set
DLUSER.hsp_util


NOTE: PROCEDURE TABULATE used (Total process time):
      real time           6:01.21
      cpu time            4:39.71
```

Input data is stored as a table or view in a database management system (DBMS), the PROC TABULATE procedure can use in-database processing to perform most of its work within the database.

The log for the In-Database Processing is:

```
NOTE: SQL generation will be used to perform the initial summarization.
NOTE: PROCEDURE TABULATE used (Total process time):
      real time           3.19 seconds
      cpu time            1.45 seconds
```

The results return results in less than 5 seconds. It provides the advantages of faster processing and reduced data transfer between the Teradata and SAS.

PROC TABULATE performs in-database processing by using SQL implicit pass-through. The procedure generates SQL queries that are based on the classifications and the statistics that specify in the TABLE statement. The Teradata executes these SQL queries to construct initial summary tables, which are then transmitted to PROC TABULATE.

Explicit passthrough with Teradata SQL syntax is less attractive as it requires, when there are a large number of categories to summarize (here our data have 39 categories and more than 30 years) a cumbersome transpose of the data involving numerous UNION ALL joins or a complicated CROSS JOIN. Syntax for a simpler transpose and its performance will be shown in the next section.

PROC TRANSPOSE

PROC TRANSPOSE allows the programmer to convert rows to columns and columns to rows (e.g. observations to variables and variables to observations). PROC TRANSPOSE is not available in SAS 9.4 for In-Database processing, however, explicit passthrough allows one to transpose data in Teradata. We compare transposing the variables: procedures performed per encounter (px_n) and the number of diagnoses made per encounter (dx_n) using PROC TRANSPOSE with Conventional Processing and explicit passthrough In-Database Processing.

The BASE SAS Conventional Processing syntax is PROC TRANSPOSE and fetches the entire table from Teradata (libname=td_work) which is then transposed in SAS.

```
proc transpose
     data=td_work.hsp_util
         (keep= util_id dx_n px_n)
     out=util_detail
         (keep=util_id _label_ col1 rename=(_label_=util_detail
         col1=counts));
  by util_id;
run;
```

We next perform the same calculations using explicit passthrough.

```
proc sql noerrorstop;
connect to teradata
  (user=&terauser password=&terapwd connection=global mode=TERADATA);
      execute
      (create multiset table  td_work.util_detail as
              (select
                     a.util_id
                    ,'dx_n' as util_detail
                    ,a.dx_n as counts
               from td_datamart.hsp_util a
               union all
               select
                     b.util_id
                    ,'px_n' as util_detail
                    ,b.px_n as counts
              from td_datamart.hsp_util b
        )
        with data primary index(util_id);
    ) by teradata;
disconnect from teradata;
quit;


data util_detail;
  set tdwork.util_detail;
run;
```

Teradata SQL does not have a transpose function; however, it is relatively straightforward to transpose columns using a combination of a SELECT query and a UNION ALL. A comparison of the logs shows a stark contrast. The log for the Conventional Processing is:

```
NOTE: The data set WORK.UTIL_DETAIL has 106429750 observations and 3
variables.

NOTE: PROCEDURE TRANSPOSE used (Total process time):

      real time           14:47.71

      cpu time            8:55.98
```

The 3 columns needed from the table of 100 million rows are brought down to SAS using a DATASTEP and then is transposed using PROC TRANSPOSE. It is quite time-consuming. The log for the In-Database Processing is:

```
NOTE: PROCEDURE SQL used (Total process time):

      real time           8.91 seconds

      cpu time            0.00 seconds

NOTE: Teradata connection: TPT FastExport has read 106429750 row(s).

NOTE: The data set WORK.UTIL_DETAIL has 106429750 observations and 3
variables.

NOTE: DATA statement used (Total process time):

      real time           6:59.11

      cpu time            3:19.49
```

The SQL query returns results in less than 10 seconds, but the time to bring down the 100 million rows of transposed data erodes the In-Database Processing advantage to some extent as it takes almost 7 minutes.

CONCLUSION

It is useful to compare the real and CPU times of the above Procedures as submitted Conventionally and In-Database and to consider the In-Database Processing Times as a percentage of the Conventional Processing times. These results are presented in tabular form below. While we were pleasantly surprised by how quickly SAS Conventional Processing provided results from a reasonably-large sized Teradata table, In-Database Processing times were generally much faster. In-Database using the SAS optimized PROCs ranged, as a share of Convectional Processing times, from 100% to 1% percent. In-Database using explicit passthrough ranged, as a share of Convectional Processing times, from 48% to 1% percent. Clearly, substantial gains in efficiency can be had by leveraging In-Database processing.

PROC	**Conventional Processing**	**In-Database Processing**		**In-Database Processing as % of Conventional**	
		Using PROC	Using Passthrough	Using PROC	Using Passthrough
EXPAND	04:04.01	n/a	00:07.68	n/a	3%
FREQ	06:20.83	00:04.46	00:05.66	1%	1%
MEANS (Means)	03:42.10	00:02.11	00:00.73	1%	1%
MEANS (IQR)	03:38.72	03:38.10	01:15.14	100%	34%
TABULATE	06:01.21	00:03.19	n/a	1%	n/a
TRANSPOSE	14:47.71	n/a	07:08.02	n/a	48%

REFERENCES

SAS Institute. 2017. SAS/ACCESS® 9.4 for Relational Databases: Reference. 9th ed. Cary, NC: SAS Institute.

SAS Institute. 2017. SAS Analytical Products 13.2 Documentation. Cary, NC: SAS Institute.

SAS Institute. 2008. SAS In-Database Processing with Teradata: An Overview of Foundation Technology. Cary, NC: SAS Institute.

ACKNOWLEDGMENTS

The authors would like to thanks Wansu Chen and Fagen Xie for the support and assistance.

CONTACT INFORMATION

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

Don McCarthy
Kaiser Permanente Department of Research and Evaluation
donald.p.mccarthy@kp.org

Mike Santema
Kaiser Permanente Department of Research and Evaluation
michael.l.santema@kp.org

Qing Yuan
Kaiser Permanente Department of Research and Evaluation
qing.yuan@kp.org