Paper SAS329-2014

Parallel Data Preparation with the DS2 Programming Language

Jason Secosky and Robert Ray, SAS Institute Inc., Cary, NC and
Greg Otto, Teradata Corporation, Dayton, OH

ABSTRACT

A time-consuming part of statistical analysis is building an analytic table for statistical procedures. Whether it is recoding input values, transforming variables, or combining data from multiple data sources, the work to create an analytic table can take time. The DS2 programming language in SAS® 9.4 simplifies and speeds data preparation with user-defined methods, storing methods and attributes in shareable packages, and threaded execution on multi-core symmetric multiprocessing (SMP) and massively parallel processing (MPP) machines. Come see how DS2 makes your job easier.

INTRODUCTION

Creating analytic tables takes time. You can reduce that time by using the DS2 programming language. DS2 provides user-defined methods, packages, and thread programs that make it easier to manage complex programs using modular coding techniques. DS2 speeds execution with parallel MapReduce style programs, dubbed "thread" and "data" programs after the statements that begin those programs. A table is processed in parallel by partitioning it among thread programs. The result of the thread programs can be combined and further reduced by a data program.

Parallel execution of the thread and data programs is supported on SMP hardware and MPP hardware. In an MPP database environment, like Teradata, parallel execution of threads is managed by the database in conjunction with the SAS Embedded Process. In this paper, we show how to program DS2 thread and data programs and how to execute that program on SMP and MPP hardware. In particular, we highlight Teradata execution to show significant improvement in execution time using a case study that includes data preparation and custom scoring code. Processing time for this case study was improved by 19X, from over 13 minutes using traditional methods including DATA step to less than one minute using DS2 and the SAS In-Database Code Accelerator for Teradata.

DS2, INSPIRED BY DATA STEP

DS2 is a procedural programming language with variables and scope, methods, packages, control flow statements, table I/O statements, and parallel programming statements. DS2 features both fixed and varying length character types along with floating, integer, and arbitrary precision numeric types. The data types supported by DS2 map well to other database systems so that data elements move more efficiently between the database engine and DS2 without loss of precision.

Setting aside the new capabilities of DS2, at its core DS2 is similar to the SAS DATA step language. The DATA, SET, IF…THEN…ELSE, and DO statements are shared between the languages and behave the same. DATA step expressions operate the same in DS2. Most DATA step functions can be called from DS2. Here is "hello world" written in DS2; the code in the INIT method is familiar to a DATA step programmer:

```
proc ds2;
data _null_;
  method init();
    dcl varchar(26) str;
    version = 2.0;
    str = 'Hello World - version ' || put(version, d3.1) ||'!';
    put str;
  end;
enddata;
run; quit;
```

DS2 programs can run in Base SAS, SAS High-Performance Analytics, SAS In-Database Code Accelerator, and SAS In-Memory Analytics. DS2 became a production feature in SAS 9.4. For detailed documentation on DS2, please see the *SAS 9.4 DS2 Language Reference*.

The focus for DS2 is parallel execution. Some DATA step features, like reading text files, do not translate well to parallel execution. For this reason, DS2 does not replace DATA step. The following sections introduce DS2 syntax and features.

METHODS

Solving difficult programming problems is made possible by combining smaller program units. DS2 provides the ability to define methods and packages. Methods group related program statements and variables in a named unit. A method can be invoked multiple times by name. All executable code in DS2 resides in a method. In the "hello world" program, all executable code resides in the INIT method. Here is a program that defines a Celsius to Fahrenheit method and calls it multiple times within a loop.

```
proc ds2;
data _null_;
  method c2f(double Tc) returns double;
    /* Celsius to Fahrenheit */
    return (((Tc*9)/5)+32);
  end;

  method init();
    do DegC = 0 to 30 by 15;
      DegF = c2f(degC);
      put DegC= DegF=;
    end;
  end;
enddata;
run; quit;
```

Multiple methods can have the same name as long as the number or type of parameters is different. Methods and DO blocks support variable scope. In the "hello world" program, the variable STR is declared in the INIT method, is scoped to the INIT method, and can be accessed only from within the INIT method.

DS2 has three system methods, INIT, RUN, and TERM, which are automatically called when a DS2 program executes. INIT runs once on program start, RUN runs once for every row in the input table, and TERM runs once on program termination.

PACKAGES

Methods and variables can be grouped into a package. Packages allow more complex code to be built from simpler parts. Packages are stored on disk and can be shared and reused in other programs. A package is similar to a class in object-oriented languages. The difference is packages do not support inheritance. This program defines a package that includes the Celsius to Fahrenheit method, declares an instance of the package, and calls the method.

```
proc ds2;
package conversions;
  method c2f(double Tc) returns double;
    /* Celsius to Fahrenheit */
    return (((Tc*9)/5)+32);
  end;
endpackage;

data _null_;
  method init();
    dcl package conversions conv();
    do DegC = 0 to 30 by 15;
      DegF = conv.c2f(degC);
      put DegC= DegF=;
    end;
  end;
enddata;
run; quit;
```

PARALLEL EXECUTION

DS2 supports parallel execution of a single program operating on different parts of a table. This type of parallelism is classified as Single Program, Multiple Data (SPMD) parallelism. In DS2, it is the responsibility of the programmer to identify which program statements can operate in parallel. The THREAD statement is used to declare a block of variables and program statements that can execute in parallel. The block of code declared with the THREAD statement is called a thread program. The block of code declared with the DATA statement is called a data program.

Thread programs can be declared along with the data program in one DS2 procedure step, or they can be permanently stored as a table for reuse.

The following example shows a thread program that computes row sums in parallel. In this case, four thread programs are started. Each program operates on a different set of rows from the input table, EMPLOYEE_DONATIONS. The SET statement reads a row of data, which contains the contributions for a member. The VARARRAY statement groups the contribution variables so that we can easily iterate over the variables to sum them.

```
proc ds2;
thread threadpgm_donations;
  vararray double contrib[*] qtr1-qtr4;
  dcl double total;
  method run();
    set employee_donations;
    total = 0;
    do i = 1 to dim(contrib);
      total + contrib[i];
    end;
  end;
endthread;

data totals;
  dcl thread threadpgm_donations t;
  method run();
    set from t threads=4;
  end;
enddata;
run; quit;
```

The thread program designates the sections of the code that can be run in parallel. The thread program is started in 4 threads when it is declared using the DCL THREAD statement and used on a SET FROM statement. The rows output by the thread program are read by the data program's SET FROM statement.

Note that the program is completely specified in DS2. DS2 manages starting the threads on different parts of the input data. There are no macros involved, and you do not have to manually partition your data for parallel execution.

For programs that are CPU bound, using a thread program on SMP hardware can improve performance. For programs that are either CPU or I/O bound, MPP hardware can improve performance. The next two sections describe using MPP hardware, in particular Teradata, to improve the performance of DS2 thread programs with parallel execution.

SAS EMBEDDED PROCESS IN TERADATA

Processing data stored in a database often requires extracting the data into SAS, operating on the data, and then loading the result back into the database. The SAS Embedded Process (EP) enables SAS processing of data in a database without extracting the data into SAS. The result is faster execution time by eliminating data movement across an external network between the database and the DS2 thread program and BY-group processing the data in parallel.

To run a DS2 program in parallel in an MPP database, we need to install and start an EP in the database. The EP is fully integrated with the Teradata SQL parser and with Teradata resource management for setting thread priority, handling query aborts, and logging, The EP is able to efficiently read and write data to and from the database and execute DS2 programs.

When an EP is started on each database node, it operates on data that is managed by that database node. With many nodes and an EP running a DS2 program on partitions of data within the node, the DS2 program operates on all of the data in parallel, resulting in faster execution time.

Each of the units of parallelism within a Teradata database node is called an AMP. Each Teradata AMP manages a partition of a table, based on a hashing algorithm applied to one or more columns in each row. Teradata systems often include 24-36 AMPs per node with hundreds of AMPs across all nodes in a single system. Each Teradata AMP coordinates work with a corresponding EP thread, resulting in a high level of parallelism in the EP as well.

Figure 1 shows DS2 operating in parallel on several nodes in a Teradata database.

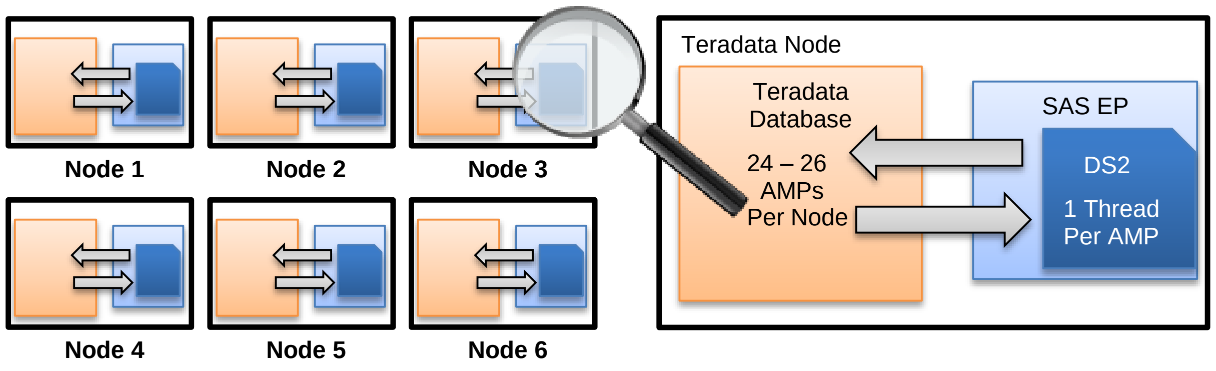


Figure 1. Teradata Executing the SAS Embedded Process (EP) and DS2 in Parallel

DS2 thread and data programs can run in the EP. You can also use a BY statement in the thread and data programs. The BY statement specifies a partitioning of the data and defines how the data is sorted before it is processed by DS2. The BY variables might or might not correspond to any physical partitioning of the data in the data source. On Teradata, starting the EP and partitioning the data is performed by the database and is based on the Teradata SQL that interacts with the EP. Here is a representation of the Teradata SQL for starting the EP. The SQL identifies which DS2 program to execute, what data to use as input, and how to partition the data that is read by the DS2 program:

```
insert into customer_scores
select * from
   sas_ep( on(select * from transaction_history)
           partition by customer_id
           ds2(select model_ds2 from customer_segmentation_models
               where model_name = 'holiday_sales')
         ) sep;
```

When executing a DS2 program in Teradata, the necessary SQL to execute the program is generated and submitted by the SAS In-Database Code Accelerator. The SAS In-Database Code Accelerator also instructs the database about how to partition and sort the data before it is presented to the EP. The next section discusses how to use the SAS In-Database Code Accelerator to run your code in Teradata.

SAS® IN-DATABASE CODE ACCELERATOR

The SAS In-Database Code Accelerator enables you to publish a DS2 data and thread program to the database and execute those programs in parallel inside the database. A thread program is required to define the sections of the DS2 code that can be processed in parallel.

Examples of programs that can benefit from parallel thread processing include large transpositions, computationally complex programs, scoring models, and BY-group processing.

THREAD PROGRAM IN-DATABASE

When the SAS In-Database Code Accelerator detects the following conditions, it generates SQL to move the DS2 processing inside the database:

- A thread program is defined to the parallel sections of code.
- The Code Accelerator is licensed. – Run the SETINIT procedure and look for the following entry:

```
---SAS In-Database Code Accelerator for Teradata
```

- SAS EP is installed on the database server in the LIBNAME referenced in the thread program SET statement.
- The database user has the necessary GRANT access to SAS EP functions in the database.
- The DS2ACCEL system option is set to ANY or the PROC DS2 DS2ACCEL option is set to YES. The default is YES for SAS 9.4, but this must be specified for SAS 9.4M1 and later. Note, in SAS 9.4, the DS2ACCEL option is named INDB.

The NOTE in this excerpt from a SAS log indicates that the thread program was processed inside the database:

```
...
126  enddata;
127  run;
NOTE: Running thread program in database.
NOTE: Execution succeeded. No rows affected.
128  quit;
...
```

This code template can be used as a starting point for implementing a thread program that runs inside the database with BY-group processing. If your data does not require BY-group processing, then remove the BY statement in the thread program and remove the references to FIRST. and LAST.

```
PROC DS2 DS2ACCEL=YES;
  thread indb_thread;
    /* declare global variables, included in output if not dropped */
    method run();
      /* declare local variables, not included in output */
      set db.input_table_or_view;
      by by_column;
      if first.by_column then do;
        /* initialized retained values used for aggregation across rows */
      end;
      /* row level processing */
      if last.by_column then do;
        /* code to finish forming the output row */
        output;
      end;
    end;
  endthread;
  run;

  /* main program - with only "set" in main program everything runs InDB */
  data db.output_table (dbcreate_table_opts='PRIMARY INDEX(by_variable)');
    dcl thread indb_thread indb; /* instance of the thread */
    method run();
      set from indb;
      /* final processing in data program - optional with BY-group processing */
      output;
    end;
  enddata;
run;
quit;
```

BY PROCESSING

When there is a BY statement in the thread program, an entire BY group is processed by a single thread. The SAS In-Database Code Accelerator relies on the database to gather the rows of a BY group on the right AMP, and then sort the rows on the BY group variables. Once the rows are grouped and sorted, each AMP delivers the rows to a corresponding DS2 thread running within the SAS EP. Each instance of the thread program processes one or BY groups depending on the distribution of the input data.

When using a BY statement, "FIRST.by_variable" and "LAST.by_variable" are used to test for changes in the BY group variables, just like with DATA step. Using these tests the thread program can control how data is pivoted or aggregated, and when it is appropriate to produce an output row.

DATA DISTRIBUTION AND BY-GROUP PROCESSING

Teradata distributes data across the AMPs using a hash function on one or more columns. For permanent tables this hash distribution is defined by the PRIMARY INDEX clause of the CREATE TABLE statement. For dynamic data, like a view that is used in a larger query, the database optimizer chooses a distribution based on the SQL and statistics on the data (collected or estimated).

Data distribution is a key consideration with an MPP database like Teradata because it affects table storage, resource balance, and ultimately response time. Poorly distributed data can lead to lumpy or skewed processing, where one thread operates on much more data than the others. In a parallel processing environment, a task cannot complete until the last unit of parallelism for that task is done.

The next sections illustrate several cases on how data distribution affects the balance of processing with the SAS EP.

Data Distribution - Row at a Time Processing (Scoring)

When DS2 running inside the database is processing all rows in the input without any regard to ordering, the key to good performance is balancing the distribution of rows evenly across all AMPs. Distribution of the input for permanent tables is based on the Teradata PRIMARY INDEX (PI).

A unique PI or a PI with many more unique values than AMPs results in even data distribution. Work with your Teradata data base administrator (DBA) to learn more about choosing good PI values and about how to test a proposed PI before actually restructuring or reloading data.

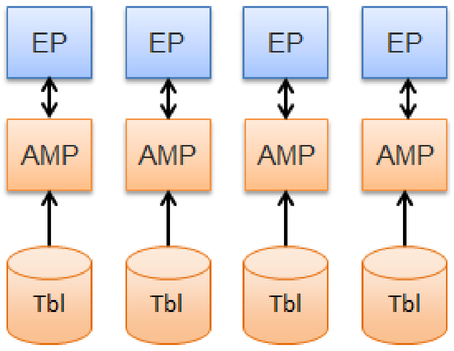


Figure 2. Direct Retrieval from Permanent Table to EP

Data Distribution - BY Processing Same Columns as the PRIMARY INDEX

When the thread BY group is the same as the PRIMARY INDEX of the input table, rows for the same BY values are already located on the AMP where the DS2 processing occurs.

The SAS In-Database Code Accelerator still tells the database to redistribute and sort the input based on the BY group variables, but the database recognizes that rows with the same BY values are already on the same AMP so that the rows need to be sorted only locally before they are sent to the EP.

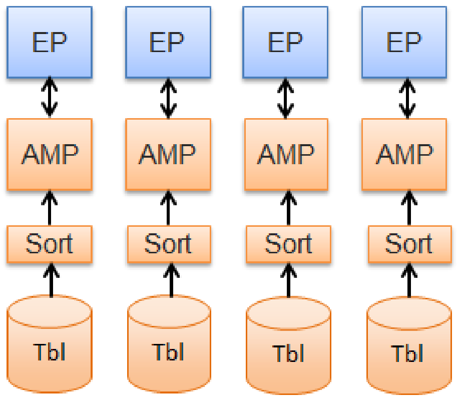


Figure 3. BY Group Variables Equal to the PRIMARY INDEX - Results in an AMP-Local Sort

Data Distribution - BY Processing on Different Columns

When the thread BY group is not the same as the PRIMARY INDEX of the input table, the database must redistribute and sort rows for the same BY group values before sending them to the EP.

Row redistribution on the Teradata BYNET interconnect is very fast and efficient, and is common during database processing. Nevertheless, row redistribution requires more processing to set up the rows for input to the EP compared to the cases where rows can remain AMP-local.

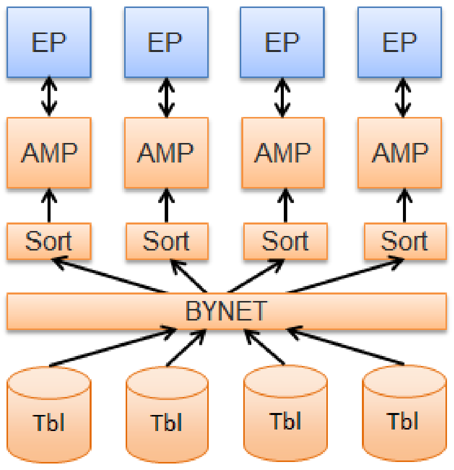


Figure 4. BY Group Variables Not Equal to the PRIMARY INDEX - Requires Row Redistribution and Sort

Data Distribution – Skewed BY Values

If BY group values have a small number of unique values across the entire table, processing can become unbalanced or "skewed" because some AMPs have more rows to process than others. In extreme cases, some AMPs might have no rows to process because there are fewer values than AMPs. Consider GENDER as an example. There are only two values, 'M' and 'F', so only two AMPs out of potentially hundreds perform any work. Extreme data skew can also lead to out of space conditions on the overloaded AMPs.

Scenarios like this should be avoided. Rethink the data flow to take advantage of a two stage process with local aggregation in the thread program that accumulates for both gender values, and then do a final BY group processing in the data program to produce the desired output.

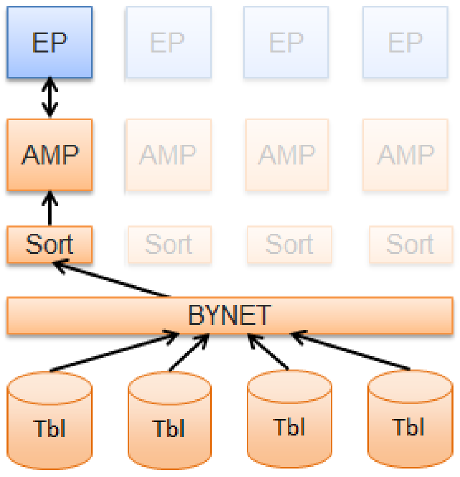
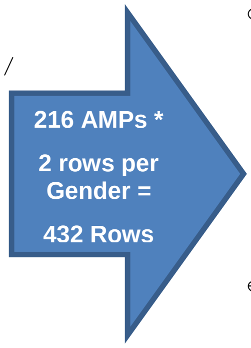


Figure 5. Skewed BY Group Variables - Leads to Unbalanced Processing

Data Distribution – Addressing Highly Skewed BY Group Values

Using the GENDER example, the thread program can perform an intermediate local aggregation on the data based on how it was originally spread across the AMPs and then pass the intermediate aggregation to the DATA program for final aggregation:

```
thread indb_thread;
  method run();
    set db.input_table; /* No BY */
    if gender = 'M' then
      /* accumulate for Male */
    if gender = 'F'
      /* accumulate for Female */
  end;
  method term();
    /* output for male */
    /* output for female */
  end;
endthread;
```



```
data db.output_table;
  dcl thread indb_thread indb;
  method run();
    set from indb; by gender;
    if first.gender then
      /* clear totals */
    /* Accumulate */
    if last.gender then
      /*output for this gender*/
  end;
enddata;
```

Data Distribution – Data Program Output in Teradata

By default, the table created by the SAS In-Database Code Accelerator distributes the data program output in the database on the first output column. This default might not be the most efficient distribution scheme. To specify a different output partitioning, the DBCREATE_TABLE_OPTS table option is used. Here is an example of using this option to distribute the data by CUSTOMER_ID:

```
data db.output_table (dbcreate_table_opts='PRIMARY INDEX(customer_id)');
  dcl thread indb_thread indb;
  method run();
    set from indb;
    output;
  end;
enddata;
```

This results in a balanced data distribution and structures the output table in a way that it can be used efficiently for joins to other tables based on CUSTOMER_IDs.

Data distribution is a key consideration with an MPP database like Teradata because it affects table storage, resource balance, and ultimately response time. Poorly distributed data can lead to inefficient processing. Please keep data distribution in mind when programming with an MPP database.

DATA AND THREAD PROGRAM INSIDE THE DATABASE

As we saw in the previous section with the gender example, some data preparation tasks might require final aggregation processing across all of the rows or all of the BY variable groups from the thread program. When additional code is inserted in the data program's RUN method, the data program can also be sent to the SAS EP for processing inside the database.

Depending on your version of SAS, additional processing in the data program is performed as follows:

- SAS 9.4 – Code other than the SET FROM statement in the data program is run in SAS. The output of the thread programs is extracted from the database into SAS.
- SAS 9.4M1 – Code in the data program is run on one AMP, passing from the thread program to the data program in a database temporary table. If a BY statement is used, the entire temporary table is sorted. Use this capability carefully, large result sets coming from the thread programs can potentially exhaust available temporary space on the one AMP. There is no parallelism with this data program.
- Future Capability – Data programs that contain a BY statement will be analyzed to determine if all the logic is isolated within each BY group. If so, BY groups may be executed in parallel on separate AMPs. If there is logic that traverses BY groups to produce overall results, the data program will be run on a single AMP.

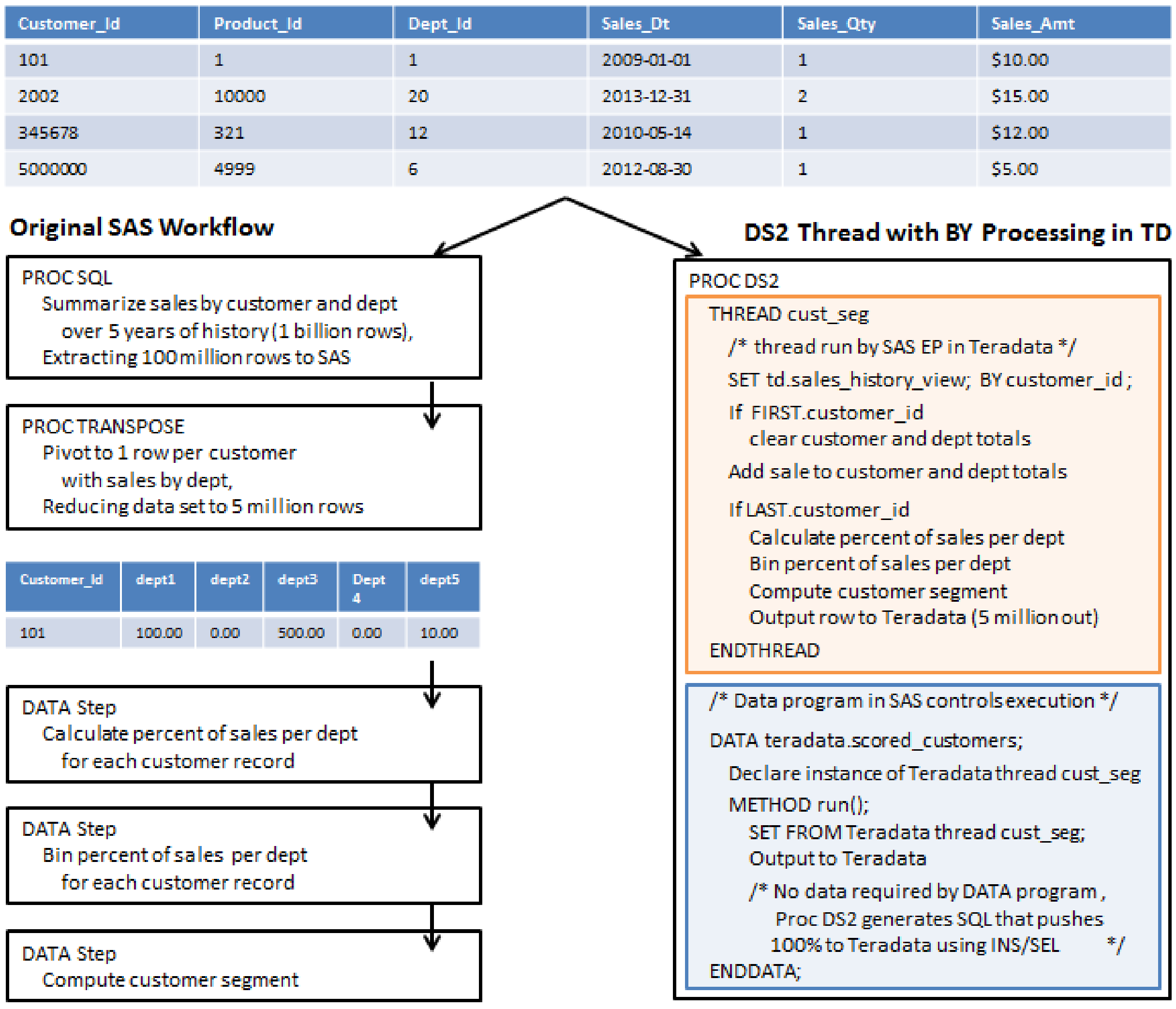
CASE STUDY – DATA PREPARATION AND SCORING

The case study introduced here is adapted from a scoring program used by a SAS customer in the retail industry. It demonstrates the following concepts:

- how the DS2 language can perform multiple data transformations in a single pass through a table, using BY-group processing to row pivot and reduce data for modeling or scoring
- how running DS2 threads in parallel using the SAS In-Database Code Accelerator for Teradata reduces elapsed time and uses resources efficiently

The original workflow uses a combination of the SQL procedure, the TRANSPOSE procedure, and multiple DATA steps to transform a large sales transaction table into a format that is appropriate for scoring. First, it extracts a partial aggregation from the database. Then it makes several passes over the aggregated data using different SAS program steps to transform the data. Finally, it computes a customer score that is used to segment customers for marketing purposes. This process is illustrated in Figure 6.

This pattern of data preparation followed by statistical analysis is frequently encountered. Focusing on this workflow provides a good model to show how to apply DS2 to many problems across different industries.



Customer_Id	Product_Id	Dept_Id	Sales_Dt	Sales_Qty	Sales_Amt
101	1	1	2009-01-01	1	$10.00
2002	10000	20	2013-12-31	2	$15.00
345678	321	12	2010-05-14	1	$12.00
5000000	4999	6	2012-08-30	1	$5.00

Original SAS Workflow

```
PROC SQL
  Summarize sales by customer and dept
    over 5 years of history (1 billion rows),
  Extracting 100 million rows to SAS
```

```
PROC TRANSPOSE
  Pivot to 1 row per customer
    with sales by dept,
  Reducing data set to 5 million rows
```

Customer_Id	dept1	dept2	dept3	Dept 4	dept5
101	100.00	0.00	500.00	0.00	10.00

```
DATA Step
  Calculate percent of sales per dept
    for each customer record
```

```
DATA Step
  Bin percent of sales per dept
    for each customer record
```

```
DATA Step
  Compute customer segment
```

DS2 Thread with BY Processing in TD

```
PROC DS2
  THREAD cust_seg
    /* thread run by SAS EP in Teradata */
    SET td.sales_history_view; BY customer_id ;
    If FIRST.customer_id
      clear customer and dept totals
    Add sale to customer and dept totals
    If LAST.customer_id
      Calculate percent of sales per dept
      Bin percent of sales per dept
      Compute customer segment
      Output row to Teradata (5 million out)
  ENDTHREAD
  /* Data program in SAS controls execution */
  DATA teradata.scored_customers;
    Declare instance of Teradata thread cust_seg
    METHOD run();
      SET FROM Teradata thread cust_seg;
      Output to Teradata
      /* No data required by DATA program ,
           Proc DS2 generates SQL that pushes
          100% to Teradata using INS/SEL        */
  ENDDATA;
```

Figure 6. Case Study – Data Preparation and Scoring Using DS2 with BY-Group Processing

CASE STUDY - IMPLEMENTATION

These basic steps were followed to convert the original workflow to a single PROC DS2 step with a parallel thread program:

1. Analyze the existing workflow, identifying reusable code blocks. Including mapping existing variables to DS2 arrays to reduce changes to existing DATA step code.
2. Create a Teradata view to replace the PROC SQL step that originally extracted data from Teradata to SAS. This provides a database object that can be defined as input to the DS2 thread. The database resolves the view as part of the query that runs DS2 in the SAS EP.
3. Convert the existing code, starting with a DS2 template for an in-database thread program with BY-group processing:
    a. Convert data transformation code at the beginning of the thread program RUN method including counter resets when the CUSTOMER_ID changes (IF FIRST.CUSTOMER_ID is TRUE THEN initialize variables).
    b. Merge the original DATA step scoring code into the thread program RUN method when the last row has been read (IF LAST.CUSTOMER_ID is TRUE THEN finish computation and output a row).
4. Test and compare results to the original.

Careful choice of variable names and array definitions meant that 75-80% of the original scoring logic in the DATA step version of the code was not modified in the DS2 version.

CASE STUDY - PERFORMANCE

To measure the performance of the original SAS workflow and the equivalent DS2 version with the SAS In-Database Code Accelerator for Teradata we tested with a large simulated sales history table on a multi-node Teradata system:

Data Characteristics:

5 years of sales history

5,000,000 unique customers

1,000,000,000 sales transactions across 20 departments

Teradata System:

Teradata Data Warehouse Appliance Model 2750

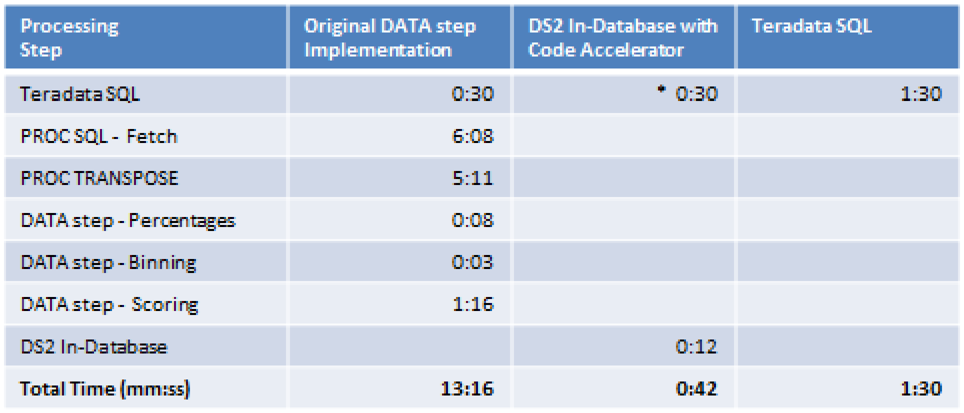
6 database nodes

216 AMPs

SAS Server:

Windows Server 2008 running SAS 9.4 for 64-bit Windows

Performance results confirm that the SAS In-Database Code Accelerator for Teradata implementation is much faster in terms of elapsed time with a speedup of 19X compared to the original SAS implementation with the majority of the processing on the SAS server:



Processing Step	Original DATA step Implementation	DS2 In-Database with Code Accelerator	Teradata SQL
Teradata SQL	0:30	* 0:30	1:30
PROC SQL - Fetch	6:08		
PROC TRANSPOSE	5:11		
DATA step - Percentages	0:08		
DATA step - Binning	0:03		
DATA step - Scoring	1:16		
DS2 In-Database		0:12	
Total Time (mm:ss)	**13:16**	**0:42**	**1:30**

** The Teradata view that feeds the EP is the same query as in the original PROC SQL*

Figure 7. Case Study Performance Comparison

Some key observations about the performance results:

- In the original approach, approximately 50% of the time is spent extracting data from Teradata to SAS.
- The transpose logic to pivot sales by department is much more efficient in DS2 compared to PROC TRANSPOSE.
- The DS2 implementation makes very efficient use of Teradata resources, out-performing an SQL implementation with better elapsed time and less CPU usage.
- The in-database DS2 implementation is 19X faster than the original on this test configuration.
- The in-database DS2 implementation is 64X faster than the original excluding the 30 seconds taken by the database to join and aggregate 1 billion transactions down to 100 million rows as into the SAS transformation and scoring process.
- The Teradata SQL implementation was split into two steps with an intermediate temporary table due to the complexity of the SQL.
- Conversion to Teradata SQL is non-trivial for multi-step programs that start from a procedural flow, requiring SQL coding expertise that might not be readily available to data analysts using SAS
- A future experiment is to compare these results with DS2 in SMP mode, varying the number of threads.

CONCLUSION

DS2 and the SAS In-Database Code Accelerator technology provides the SAS user with a powerful new tool to directly leverage a Teradata database platform using familiar procedural syntax. The SAS In-Database Code Accelerator executes the DS2 data and thread programs in Teradata and generates Teradata SQL to drive BY-group processing in a partitioned, highly parallel data environment. The DS2 thread/data programming paradigm allows the user to easily express MapReduce type operations using syntax that is similar to DATA step.

RESOURCES

- SAS Institute, Inc. 2013. *SAS® 9.4 DS2 Language Reference*. Available at http://support.sas.com/documentation/onlinedoc/base/index.html
- SAS Institute, Inc. 2013. *SAS® 9.4 In-Database Products: User's Guide, Third Edition*. Available at http://support.sas.com/documentation/onlinedoc/indbtech/index.html
- SAS Institute, Inc. 2013. *DS2 Programming: Essentials (training course)*. Available at https://support.sas.com/edu/schedules.html?id=1798

ACKNOWLEDGMENTS

The authors would like to thank Cindy Wang, Kanthi Yedavalli, David Wiehle, Tom Weber, and Austin Swift for their efforts in the development of the SAS In-Database Code Accelerator.

CONTACT INFORMATION

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

Jason Secosky, Robert Ray
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
919-677-8000
Jason.Secosky@sas.com
Robert.Ray@sas.com
http://www.sas.com

Greg Otto
100 SAS Campus Drive
Cary, NC 27513
Teradata Corporation
919-531-2661
Greg.Otto@teradata.com