Parallel Data Preparation with the DS2 Programming Language

John Cunningham, Teradata Corporation, Danville, California

Paul Segal, Teradata Corporation, San Diego, California

Tho Nguyen, Teradata Corporation, Raleigh, North Carolina

ABSTRACT

A time consuming part of statistical analysis is building an analytic data set for statistical procedures. Whether it is recoding input values, transforming variables, or combining data from multiple data sources, the work to create an analytic data set 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 SMP and MPP machines.

INTRODUCTION

Creating analytic data sets takes time. You can reduce that time by using the DS2 programming language. DS2 provides user-defined methods, packages, and thread programs that simplify coding. DS2 speeds execution with parallel map/reduce style programs, dubbed "thread" and "data" programs after the statements that begin those programs. A data set 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 and MPP hardware. In a parallel 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.

DS2, SIMILAR TO 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 data elements move more efficiently between the database engine and DS2 without loss of precision.

Setting aside the new capabilities of DS2, at it's 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 will be 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 Grid, 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 DecC = 0 to 30 by 15;
      DegF = c2f(degC);
      put DecC= 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 only be accessed from within the INIT method.

DS2 has three system methods, INIT, RUN, and TERM that 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 DecC = 0 to 30 by 15;
      DegF = conv.c2f(degC);
      put DecC= DegF=;
    end;
  end;
enddata;
run; quit;
```

PARALLEL EXECUTION

DS2 supports parallel execution of a single program operating on different parts of a data set. This kind 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. Thread programs can be declared along with the data program in one Proc DS2 step, or they can be permanently stored as a data set/table for resuse. The block of code declared with the DATA statement is called a data program.

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 data set, 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 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;
    end;
  end;
endpackage;

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

The THREAD statement defines the thread program. 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 symmetric multiprocessing hardware (SMP) can improve performance. For programs that are either CPU or I/O bound, massive parallel processing hardware (MPP) 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, 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

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 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. The Teradata database units of parallelism within each node are called Teradata AMPs. 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 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.

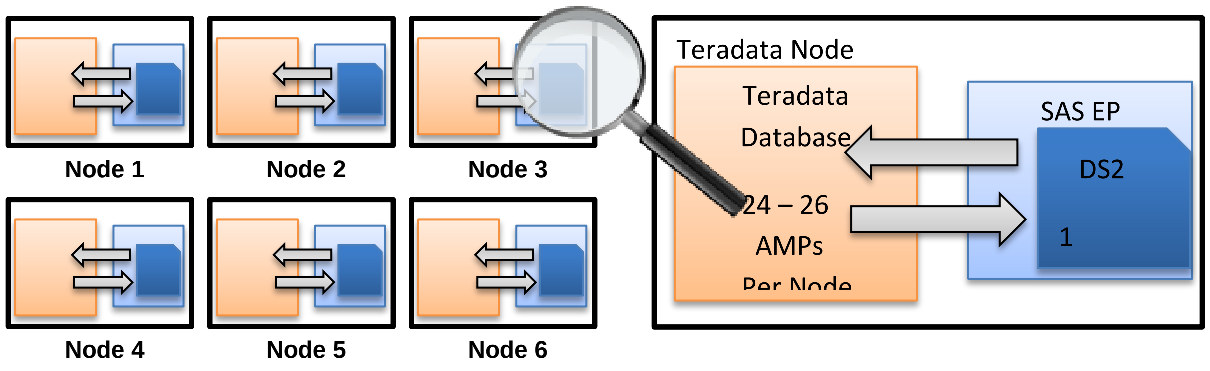


Figure 1. Teradata Executing the SAS Embedded Process (EP) and DS2 in Parallel

DS2 thread and data programs can run in the EP. You can also use a BY statement in the thread and data programs, which specifies a partitioning of the data that defines how the data will be sorted before it is processed by DS2. The BY variables may or may 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 as defined in the Teradata SQL that interacts with the EP. Here is a representation of the Teradata SQL for starting the EP, specifying a DS2 program to execute, and partitioning the data read by the DS2 program.

```
insert into customer_scores
select * from
   sas_ep( on(select * from transaction_history)
           partition by customer_id
           model_table(customer_segmentation_models)
           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 DS2 Code Accelerator. The DS2 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 Code Accelerator to run your code in Teradata.

SAS DS2 CODE ACCELEARTOR

The DS2 Code Accelerator enables you to publish a data and thread program to the database and execute those programs in parallel inside the database. 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 DS2 Code accelerator detects the following conditions it will generate SQL to move the DS2 processing In-Database:

- DS2 Code Accelerator is licensed -- check with PROC Setinit looking for:

  ```
  ---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
- InDB=YES option is specified -- default is YES for SAS 9.4, but this must be specified for SAS 9.4_M1

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

```
PROC DS2 InDB=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;
  endthread;
  run;

  ds2_options trace;  /* optional - print SQL trace to SAS log */

  /* main program - with only "set" in main program everything runs InDB */
  data db.output_table;
    dcl thread indb_thread indb; /* instance of the thread */
    method run();
```

```
            set from indb;
            /* final processing in data program - optional with BY processing */
            output;
          end;
        enddata;
      run;
      quit;
```

BY PROCESSING

DS2 Code Accelerator uses the database to distribute rows to the right AMP and then sort on the BY column(s). When there is a BY statement in the thread program input rows is delivered by each Teradata AMP to a corresponding DS2 thread ordered on the BY variable(s).

When the thread contains a BY statement the thread can use “FIRST.by_variable” and “LAST.by_variable” to test for changes in the BY 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 PROCESSING

Teradata distributes data across the AMPs using a hash function on one or more column(s). 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 (skewed) processing. In a parallel processing environment any task cannot complete until the last unit of parallelism for that task is done.

There are several cases to consider with regard to how data distribution affects the distribution of data and the balance of processing with the SAS EP.

Data Distribution - Row At A Time Processing (Scoring)

When DS2 running In-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 will result in even data distribution. Work with your Teradata DBA to learn more about choosing good PI values, and about how to test a proposed PI before actually restructuring or reloading data.

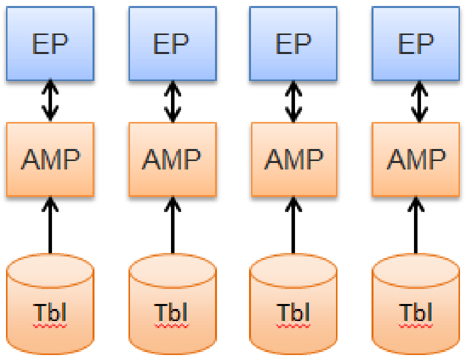


Figure 2. Direct Retrieve from Permanent Table to EP

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 will occur. The DS2 Code Accelerator still tells the database to redistribute and sort the input based on the BY variables, but the database recognizes that rows with the same BY values are already on the same AMP so the rows only need to be sorted locally before they are sent to the EP.

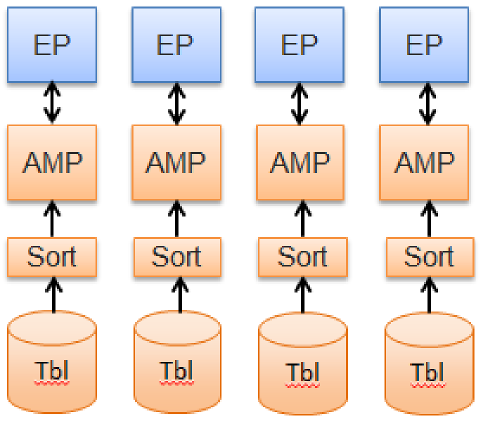


Figure 3. BY Variables Equal to the PRIMARY INDEX Results in an AMP-Local Sort

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 values before sending them to the EP. Row redistribution on the Teradata BYNET interconnect is very fast and efficient, this happens all the time during database processing. But it will require more processing to set up the rows for input to the EP compared to the cases where rows can remain AMP-local.

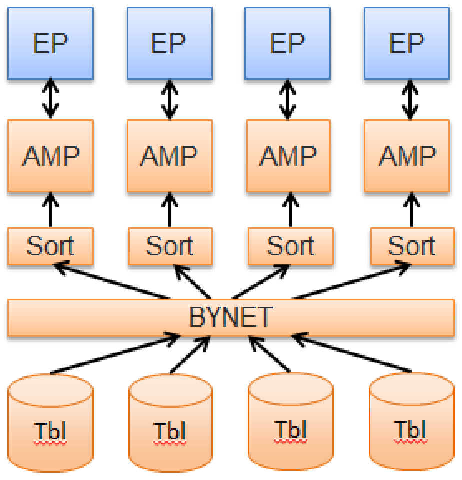


Figure 4. BY Variables Not Equal to the PRIMARY INDEX Requires Row Redistribution and Sort

Data Distribution – Skewed BY Values

If BY 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 may have no rows to process because there are fewer values than AMPs. Consider gender as an example, when there are only two values ‘M’ and ‘F’ then only two AMPs (out of potentially hundreds) will have 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, then do a final BY group in the data program to produce the desired output.

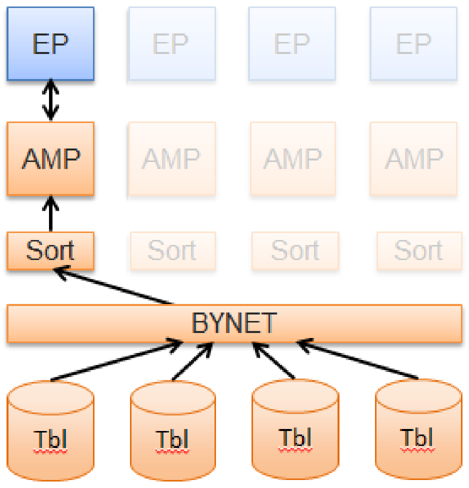


Figure 5. Skewed BY Variables Leads to Unbalanced Processing

Data Distribution – Output

The DS2 Code Accelerator distributes the thread program output in the database based on the BY column(s) by specifying the Teradata PRIMARY INDEX. This results in good data distribution and structures the output table in a way that it can be used efficiently for joins to other tables based on the BY column(s).

When there is no BY statement, the Code Accelerator uses a special designator NO PRIMARY INDEX (NOPI) for the output table. NOPI causes the AMP to store output rows locally as they are produced by the corresponding EP thread. In this case, if the output volume from each EP thread is balanced then the output table will be balanced.

DATA AND THREAD PROGRAM IN-DATABASE

Building on the example in the previous section, some data preparation tasks may require final aggregation processing across all of the rows or all of the BY groups from the thread program. When additional code is inserted in the data program RUN method, that code can also be sent to the SAS EP for processing in the database.

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

- SAS 9.4 TS1M0 – Code other than SET FROM <thread> in the data program will run in SAS. The output of the thread programs will be extracted from the database into SAS.
- SAS 9.4 TS1M1 – Code in the data program will run on one AMP, passing from the thread program to the data program in a database temporary table. 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 in this step.
- Future Capability – Code in the data program can also have a BY statement, on the column(s) that are different than the thread program, or the same. Placing a BY statement in the data program means the data program will run in parallel on multiple AMPs just like the thread program.

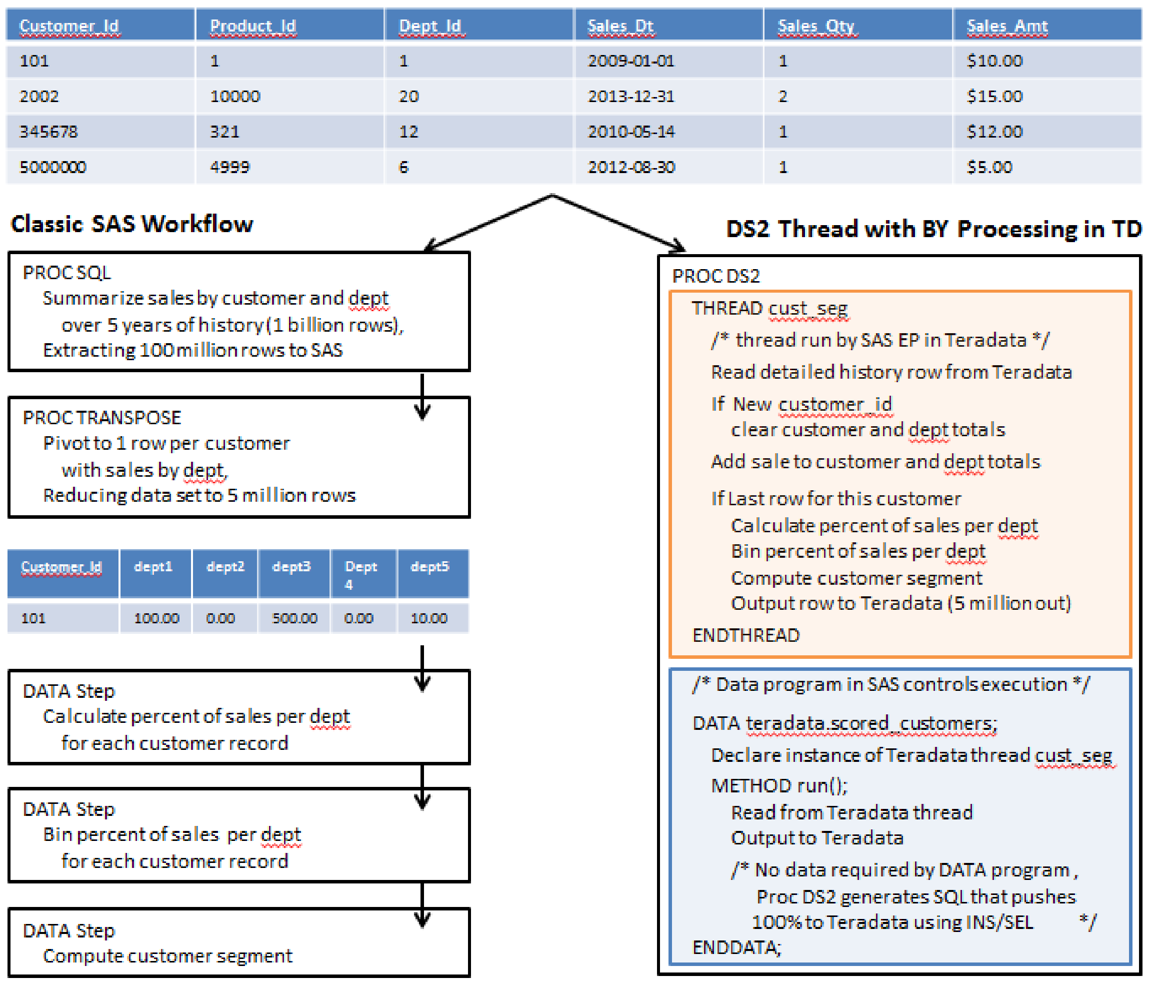
CASE STUDY – DATA PREPARATION AND SCORING

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

- How the DS2 language can perform multiple data transformations in a single pass through a table, using BY processing to ro pivot and reduce data for modeling or scoring.
- How running DS2 threads in parallel using DS2 Code Accelerator on Teradata reduces elapsed time and uses resources efficiently.

The original workflow uses a combination of PROC SQL, PROC Transpose, 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 pattern of data preparation followed by statistical analysis is encountered frequently. Focusing on this workflow provides a good model for 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

Classic 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 */
    Read detailed history row from Teradata
    If New customer_id
      clear customer and dept totals
    Add sale to customer and dept totals
    If Last row for this customer
      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();
      Read from Teradata thread
      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 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. Map 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 the DS2 in the SAS EP.
3. Convert the existing code, starting with a DS2 template for an In-Database thread program with BY processing:
    a. Convert data transformation code at the beginning of the thread run() method including counter resets when the customer_id changes (IF FIRST.customer_id is TRUE do block).
    b. Merge the original Data step scoring code into the thread run() method when the last row has been read (IF LAST.customer_id is TRUE do block).
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 reused without modification in the DS2 version.

CASE STUDY - PERFORMANCE

To measure the performance of the original SAS workflow and the equivalent DS2 version with Code Accelerator 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 with 6 nodes and 216 AMPs

SAS Server:

Windows Server 2008 running SAS 9.4 TS1M0 for 64-bit Windows

Performance results confirmed that the In-Database DS2 implementation is much faster in terms of elapsed time with a speedup of 13X over the original SAS workflow processed on the SAS server. The DS2 implementation also makes very efficient use of Teradata resources for a process that requires a combination of data transformation and complex calculations, out-performing a SQL implementation with better elapsed time and less CPU usage:

<Results needs to be formatted as a basic table… Move relevant observations to analysis points after the table>

- E.g. Long time to extract
- DS2 transpose logic much more efficient that PROC Transpose

- "Classic" SAS workflow (mm:ss) = 13:16

PROC SQL 6:38
TRANSPOSE 5:11
DATA steps (3) 1:27
Total 13:16

  - Data transfer time is the largest cost at ~ 50% of elapsed time
  - Proc Transpose is also a heavy hitter

- DS2 in Teradata

Total 0:42

  - 19 times faster end-to-end
  - 64 times faster excluding the ~30 seconds execution time on the query to summarize 1 billion rows down to 100 million entering DS2

- SQL (2 steps)

Total 1:30

    - Had to break this into two steps because the complexity of a single request blew out Teradata parser/dispatcher memory limits (still working on getting it back into one request…)
    - Very CPU intensive in the second step that does the math, I think the CPU cost of this approach will be much more than 2X the cost of the DS2 implementation
- It would also be interesting to run the same process with DS2 in SMP mode varying the number of threads (TBD)

SQL AS AN ALTERNATIVE

Imagine what it would take to write SQL to perform the same operations.

- ➢ Advanced SQL expertise required (may not be available within SAS community)
- ➢ May be slower, more resource intensive (CPU and/or IO) converting a procedural flow to set processing (pending final clean up and run time analysis of Case Study SQL conversion)

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 threaded 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.

CONTACT INFORMATION

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

Name: John Cunningham
Enterprise: Teradata
E-mail: john.cunningham@teradata.com

Name: Tho Nguyen
Enterprise: Teradata
E-mail: tho.nguyen@teradata.com
Web: www.teradata.com/sas