Decision Management with DS2

Helen Fowler, Teradata Corporation, West Chester, Ohio
Tho Nguyen, Teradata Corporation, Raleigh, North Carolina

ABSTRACT

We all make tactical and strategic decisions every day. With the presence of big data, are we making the right or the best decisions possible as data volume, velocity and variety continue to grow? As businesses become more targeted, personalized and public, it is imperative to make precise data-driven decisions for regulatory compliance and risk management.

INTRODUCTION

How we make business decisions varies from gut instinct to those driven by data, or some combination of the two. Many organizations wonder if they are making the right, or best possible, decisions. And, as data volumes and variety grow exponentially, the tactical decisions that drive operations become even more important.

Operational decisions are less strategic than deciding whether or not to open a new division or develop a new product line, but they still have a significant impact on the bottom line and, ultimately, the organization's success.

How do you know if your services staff always follows company policy? Do the pricing decisions comply with regulations? As business becomes more targeted, personalized and public, it is vital to make precise, fact-based (data-driven), transparent operational decisions. These decision need to have an auditable history to show regulatory compliance and risk management.

SAS and Teradata have partnered to deliver SAS® In-Database Decision Management for Teradata – an in-database solution that executes analytic logic and business rules within the database leveraging DS2 technology. This enables you to:

- Streamline and automate the analytic decision lifecycle for data-driven decisions.
- Perform decision calculations in a scalable, high-performance platform.
- Improve performance, cost and governance.

SAS In-Database Decision Management for Teradata is a comprehensive solution allowing you to automate operational decision processes founded on advanced analytics so that you can make the best decision for any scenario. There's no need to replicate data and extract it for analysis, or recode decisions into operational systems. The entire process takes place inside your Teradata® Database.

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 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 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.

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

SAS CODE ACCELERATOR FOR TERADATA

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 Code accelerator detects the following conditions it will generate SQL to move the DS2 processing In-Database:

- 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

The 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.

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 need 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)

CONCLUSION

DS2 and the SAS In-Database Decision Management for Teradata provide the SAS user with a powerful new tool to directly leverage a Teradata database platform using familiar procedural syntax. The SAS In-Database Decision Management for Teradata executes the business rules in Teradata and generates data-driven decision decisions.

CONTACT INFORMATION

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

Name: Helen Fowler
Enterprise: Teradata
E-mail: helen.fowler@teradata.com

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