Paper 142-2011

Teradata Parallel Transporter: Loading Your SAS® Data Just Became Easier

Jeff Bailey, SAS Institute Inc., Cary, NC

ABSTRACT

Teradata Parallel Transporter (TPT) provides a full-featured application programming interface (API) that enables you to access the Teradata loading feature. If you use SAS® with Teradata, you need to know about this exciting technology. This paper will help you start using the SAS/ACCESS® Interface to Teradata and Teradata Parallel Transporter to efficiently move SAS data to a Teradata Server. Although very important, restarting load jobs is beyond the scope of this paper and is not fully covered. It is mentioned where required. The good news is that this paper will give you the head start that you need to learn how to restart jobs on your own.

WHAT IS TERADATA PARALLEL TRANSPORTER?

The Teradata Parallel Transporter is an object-oriented client application that enables rapid loading, updating, and extraction. It is designed to be highly customizable and works well with third-party applications, such as SAS.

There are many advantages to using the TPT API over the older methods (FastLoad, MultiLoad, and Multi-Statement) in your SAS environment. Let's discuss some of the more important advantages.

Advantages of using the TPT interface to SAS:

- **Future enhancements:** New features and performance enhancements will be made to the TPT API interface.
- **Consistent interface:** TPT uses a similar set of LIBNAME and data set options for FastLoad, MultiLoad and Multi-Statement, which makes coding your SAS jobs much easier. This code similarity is particularly true when you need to restart jobs. This paper does not discuss restarting jobs in detail, but once you understand the basics of TPT you will be able to quickly learn how to restart jobs.
- **TPT API libraries are loaded directly into SAS:** There are no separate utility processes to launch which saves processing time and makes configuration simpler.
- **Simple configuration:** There are no utilities that SAS must find and launch, so setting up the environment is a snap. If you aren't installing and configuring the software, you don't have to worry about this. See the SAS configuration guide for your environment if you would like to learn more about configuration.
- **Control over utility slot usage:** This one is huge! There are a small number of utility slots defined on your Teradata Servers. The ability to control the use of these slots from your SAS jobs will make your life easier and keep your database administrator (DBA) happy. For more information about utility slots, see the "Utility Slots" section in this paper.
- **No directory security issues:** The TPT API does not require an application to create scripts and control files during Update operator (MultiLoad) processing. As a result, there is no need to worry about creating intermediate files and directories and the privileges they will need.
- **Integration with Teradata Workload Manager:** Workload Manager is a DBA tool that monitors the processes running in the Teradata Server. Using the TPT API allows you to create jobs that can be monitored. This integration will make your DBA much happier.

There is a FastExport operator included in the TPT API. It isn't covered in this paper. For more information on FastExport, see *SAS/ACCESS 9.2 for Relational Databases*. This document is available here: http://support.sas.com/documentation/onlinedoc/access.

REQUIRED SOFTWARE

In order to use the TPT API with SAS, you must have SAS 9.2 and the SAS/ACCESS Interface to Teradata installed in your environment. TPT was not supported in SAS releases prior to SAS 9.2.

In addition to the SAS software, you must have the following Teradata software installed and configured in your environment:

- Teradata CLIv2 client libraries, Teradata Tools and Utilities (TTU) 12 (12.00.00.3) or higher (CLIv2, TeraGSS and tdicu).

- Teradata Parallel Transporter API, TTU 12.0 (12.00.00.3 or) or higher. The TTU and TPT API releases must be at the same level.

TTU and TPT must be installed on any machine where SAS will access data from a Teradata Server. It is highly recommended that your TTU and TPT software is at the most current patch level. Make certain that the TTU and TPT patches are applied.

REQUIRED DATABASE PRIVILEGES

In order to use SAS to load data into Teradata you will need a user account and password on a Teradata Server. Contact your Teradata database administrator (DBA) if you need assistance connecting to the Teradata Server.

For this example, imagine that you have a database named MYDB that contains an empty table named MYTABLE. You need to load a table named MYTABLE. In this example, you are allowed to create TPT API utility tables in the database where your target table lives, so you run code similar to the following:

```
libname MYDB teradata server=TDServ DATABASE=MYDB user=TDUser password=mypw1;

proc append base=MYDB.MYTABLE (TPT=YES FASTLOAD=YES)
            data=work.MYTABLE;
run;
```

Here are the Teradata privileges that the TDUser must have in order to load the MYDB.MYTABLE Teradata table:

- CREATE TABLE on MYDB
- DROP TABLE on MYDB
- CREATE MACRO on MYDB (Stream operator – Multi-Statement)
- DROP MACRO on MYDB (Stream operator – Multi-Statement)
- INSERT on MYDB
- DELETE on MYDB
- SELECT on MYDB

Loading into a production environment presents many challenges. It is best to learn in a non-production environment. Once you understand how SAS and the TPT API work together, you can use these techniques in your production environment. For more information about working in production environment, see the “Separating Utility Tables from Production Tables” section in this paper.

TPT LOAD OPERATOR (FASTLOAD)

The TPT Load operator uses the Teradata FastLoad protocol to quickly move large amounts of data into an empty table.

The Teradata FastLoad protocol has many restrictions. Some of these restrictions are specific to SAS:

- The target table must be empty. (The table can be created as part of the load.)
- SAS can’t FastLoad into a view.
- Only one table can be loaded for a FastLoad job in SAS.
- Duplicate data is not allowed. (SAS discards the duplications, so the job doesn’t fail.)
- No referential Integrity (RI) can be defined on the target table. (Soft RI is OK.)
- No secondary, join, or hash indexes can be defined on the target table.
- No triggers can be defined on the target table.

If you run a FastLoad job that violates these restrictions, the load step will fail. The TPT facility in SAS/ACCESS logs any errors about the database tables, which makes diagnosing the problems easier.

Insufficient space in your database and loading a non-empty table are two common causes of FastLoad failure. On the other hand, using the Load operator with SAS to load duplicate data into your Teradata table will not cause the step to fail. Many people find this surprising. However, using the Load operator can cause subtle problems because the duplicates are not loaded into the table. If you must load duplicate rows in the table, you need to find another way to add them. You can use the Update operator (MultiLoad) to add duplicate rows to a table.

Let’s take a look at a simple SAS job that uses PROC APPEND to load the data contained in the WORK.FASTLOAD SAS data set into a Teradata table named TPT_LD_TEST. This is a very simple job. We will use the FASTLOAD=YES option to invoke bulk loading.

We will submit the following code to load the TPT_LD_TEST table. Keep in mind, TPT=YES is the default. If you don't include that option, SAS will still attempt to use the TPT API.

```
LIBNAME TDServ TERADATA SERVER=TDServ USER=tduser PASSWORD=tdpasswd1;

PROC APPEND BASE=TDServ.tpt_ld_test (FASTLOAD=YES TPT=YES)
            DATA=work.FastLoad;
RUN;
```

Since the TPT API is installed and properly configured on our machine, it is used for the load. You can see the note in the following SAS log.

```
83   proc append base=tdserv.tpt_ld_test (FASTLOAD=YES TPT=YES)
84               data=work.FastLoad;
85   run;

NOTE: Appending WORK.FASTLOAD to tdserv.tpt_ld_test.
NOTE: There were 2000000 observations read from the data set WORK.FASTLOAD.
NOTE: 2000000 observations added.
NOTE: The data set tdserv.tpt_ld_test has . observations and 11 variables.
NOTE: Teradata connection: TPT Fastload has inserted 2000000 row(s).
NOTE: PROCEDURE APPEND used (Total process time):
      real time           2:40.38
      cpu time            27.62 seconds
```

Output 1. Output from a PROC APPEND Statement Showing That TPT Was Used

It is a good idea to look for the TPT messages in the SAS log. A TPT API install or configuration issue will not cause the step to fail. If there is a problem with TPT, then the traditional CLIv2 interface will be used instead.

Although you can run you load jobs without specifying many data set options, there are some you will want to include. These options are covered in detail in the SAS TPT Options section of this paper. Here is a code example that more closely matches what you will see in production environments.

```
proc append base=MYTera.LOADTEST (TPT=YES FASTLOAD=YES
                                  TPT_MIN_SESSIONS=2
                                  TPT_MAX_SESSIONS=6
                                  TENACITY=1
                                  SLEEP=5
                                  TPT_ERROR_TABLE_1 = LOADTEST_ET
                                  TPT_ERROR_TABLE_2 = LOADTEST_UV
                                  TPT_LOG_TABLE     = LOADTEST_RS
                                  TPT_TRACE_LEVEL=2
                                  TPT_TRACE_LEVEL_INF=12
                                  TPT_TRACE_OUTPUT='C:\SAS_TPT_fastload.txt')
            data=WORK.LOADDATA;
run;
```

In production jobs, you may want to limit the number of sessions that your jobs will use. This is one area where the TPT API functionality in SAS/ACCESS really shines. You can limit the number of sessions using the TPT_MIN_SESSIONS= and TPT_MAX_SESSIONS= data set options.

The SLEEP= and TENACITY= options impact utility slot usage. We will discuss these options in greater detail later in this paper.

The TPT_ERROR_TABLE_1=, TPT_ERROR_TABLE_2= and TPT_LOG_TABLE= options explicitly name the utility tables that will be created during execution, so these tables are easy to find. If your load job completes with no issues, these tables are dropped. It is a great idea to verify that the utility tables were dropped after the load completes.

The logging information produced by the TPT API can be captured and written to a file. The TPT_TRACE_LEVEL= and TPT_TRACE_LEVEL_INF= options determine the logging levels used. TPT_TRACE_LEVEL= appears to be the most useful of the two options. Setting this option to 2 is a good idea. Setting TPT_TRACE_LEVEL_INF=12 never seems to produce output. It is a good practice to save this information to disk using the TPT_TRACE_OUTPUT= option. You specify a filename for this option. (The documentation for this option mistakenly says to specify an integer.)

Make sure you trap the TPT API output using the TPT_TRACE_OUTPUT= option. This output contains a lot of useful information, which is very useful when you encounter problems with your load job.

When you are learning to use the TPT API with SAS/ACCESS, you will encounter some issues that require restarting a job. A detailed discussion of job restarting is beyond the scope of this paper, but let's discusses it briefly. Typically, there is enough information in the SAS log to get you started. Here is an example of a FastLoad failure caused by insufficient space in the database containing the target table.

```
204  proc append base=TDserv.tpt_test (TPT=YES FASTLOAD=YES)
205              data=work.FastLoad;
206  run;

NOTE: Appending WORK.FASTLOAD to TDSERV.tpt_test.
NOTE: There were 1 observations read from the data set WORK.FASTLOAD.
NOTE: 0 observations added.
NOTE: The data set RUFUS.tpt_test has . observations and 1 variables.
ERROR: Teradata connection: No more room in database TINYDB. Correct error and
restart as an APPEND process with option TPT_RESTART=YES. Since no checkpoints were
taken, if the previous run used FIRSTOBS=n, use the same value in the restart.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           1.32 seconds
      cpu time            0.04 seconds

NOTE: The SAS System stopped processing this step because of errors.
```

Output 2. Output from a PROC APPEND Statement Showing a FASTLOAD Failure

It is obvious from this SAS log that there is not enough space in the database TINYDB. To fix this issue, we need to get our DBA to increase its size. After the database is sized properly, you can resubmit your code. In test environments, it may be easier to recreate the target table and drop the utility tables. For more information about the TPT_RESTART= option, see the *SAS/ACCESS 9.2 for Relational Databases Reference*. This document is available here: http://support.sas.com/documentation/onlinedoc/access.

TPT UPDATE OPERATOR (MULTILOAD)

The Update operator can be used to load a non-empty Teradata table, and it will load duplicate rows (if the Teradata table is defined as a multi-set table).

The Teradata MultiLoad protocol has many restrictions. Some restrictions are specific to SAS:

- You must load into an empty or non-empty table. (The table can be created as part of the load.)
- SAS can't MultiLoad into a view.
- Only one table can be loaded in a MultiLoad job in SAS.
- No references to foreign keys.
- No unique secondary, hash, or join indexes.
- No triggers.

Let's take a look at a simple SAS job that uses PROC APPEND to load the data in the WORK.MULTILOAD data set into a Teradata table named TPT_UD_TEST. This is a very simple job. We will use the MULTILOAD=YES option to invoke the Update operator.

We will submit the following code to load the TPT_UD_TEST table. Keep in mind, TPT=YES is the default. If you don't include that option, SAS will still attempt to use the TPT API.

```
LIBNAME TDServ TERADATA SERVER=TDServ USER=tduser PASSWORD=tdpasswd1;

PROC APPEND BASE=TDServ.tpt_test (MULTILOAD=YES TPT=YES)
            DATA=work.MultiLoad;
RUN;
```

Since the TPT API is installed and properly configured on our machine, MultiLoad is used. See the note in the following SAS log.

```
30   proc append base=TDServ.tpt_test (TPT=YES MULTILOAD=YES)
31               data=work.MultiLoad;
32   run;

NOTE: Appending WORK.MULTILOAD to TDSERV.tpt_test.
NOTE: There were 100000 observations read from the data set WORK.MULTILOAD.
NOTE: 100000 observations added.
NOTE: The data set TDSERV.tpt_test has . observations and 1 variables.
NOTE: Teradata connection: TPT MultiLoad has inserted 100000 row(s).
NOTE: PROCEDURE APPEND used (Total process time):
      real time           5.87 seconds
      cpu time            0.07 seconds
```

Output 3. Output from a PROC APPEND Statement Showing That TPT Was Used

It is a great idea to check the SAS log for the TPT messages. A TPT API install or configuration issue will not cause the step to fail. If there is a problem with using MultiLoad (TPT Update operator), then the traditional MultiLoad utility will be used instead. You will want to ensure that this doesn't happen because the MultiLoad utility is much slower than the TPT API.

Although you can run your load jobs without specifying many data set options, there are some you will want to include. These options are covered in detail in the SAS TPT Options section of this paper. The following code closely matches what you will see in production environments. This code should look very familiar to you. It is very similar to the code shown for FASTLOAD=YES. The only difference is that the Update operator creates a work table (_WT) that holds a temporary copy of the data to be loaded.

The work table has the potential to cause space problems on TPT API MultiLoad jobs. Make certain that the database (or user) containing the work table has enough space for the work table.

```
proc append base=MYTera.loadtest (TPT=YES MULTILOAD=YES
                                  TPT_BUFFER_SIZE=64
                                  TPT_MIN_SESSIONS=1
                                  TPT_MAX_SESSIONS=6
                                  SLEEP=2
                                  TENACITY=1
                                  TPT_ERROR_TABLE_1 = LOADTEST_ET
                                  TPT_ERROR_TABLE_2 = LOADTEST_UV
                                  TPT_LOG_TABLE     = LOADTEST_RS
                                  TPT_WORK_TABLE    = LOADTEST_WT
                                  TPT_TRACE_LEVEL=2
                                  TPT_TRACE_LEVEL_INF=12
                                  TPT_TRACE_OUTPUT='C:\SAS_TPT_multiload.txt')
            data=work.adddata;
run;
```

During your experimentation you will probably encounter errors. In a testing environment, you may not need to restart. Instead, dropping tables and starting over may be enough to remove the errors. Specifying the names of the utility tables helps a great deal here. You can look at your SAS code and know exactly which tables you need to drop. Restarting a production MultiLoad job is beyond the scope of this paper. If you need to do this, contact your DBA.

There is a trick to rerunning a failed MultiLoad job. The problem is that there will be a lock on the table that you tried to load. You will need to clear this lock in order to continue. This lock can be cleared with a SQL command.

Here is an example SQL statement that will clear an MLOAD lock on a table.

```
RELEASE MLOAD mydb.mytable;
```

Let's assume that our PROC APPEND step failed. Here is some SAS code that we can use to clear the problem.

Warning: This technique is for testing environments only. In a production environment, you will want to restart the job.

```
/* Clearing an MLOAD lock in a testing environment */
proc sql;
   connect to teradata (server=tdserv user=tduser password=tdpasswd1);
```

```
   execute (drop table tduser.loadtest_et) by teradata;
   execute (commit) by teradata;
   execute (drop table tduser.loadtest_uv) by teradata;
   execute (commit) by teradata;
   execute (drop table tduser.loadtest_rs) by teradata;
   execute (commit) by teradata;
   execute (drop table tduser.loadtest_wt) by teradata;
   execute (commit) by teradata;
   disconnect from teradata;
quit;

proc sql;
   connect to teradata (server=tdserv user=tduser password=tdpasswd1);
   execute (release mload tduser.tpt_test) by teradata;
   disconnect from teradata;
quit;
```

For more information about restarting, see the TPT_RESTART data set option in *SAS/ACCESS 9.2 for Relational Databases*.

TPT STREAM OPERATOR (MULTI-STATEMENT)

The TPT Stream operator uses the Multi-Statement protocol to quickly insert data into a table. If the table is defined using the MULTISET keyword you can insert duplicate rows into it. This method offers some opportunities for making the inserts more efficient, but it should be used only for relatively small amounts of data. In this case, you will need to do some performance testing to find out what "relatively small" means in your environment.

One of the great things about the Stream operator is that it is so easy to use. There are not many options that you must set. Plus, the performance gain in using this technique is nice. Before we discuss the details, let's take a look at the restrictions to using this technique. Fortunately, there are not many.

Teradata Stream operator restrictions:

- You must insert into an empty or non-empty table. (The table can be created as part of the job.)
- Only one table can be loaded in a MULTISTMT= job in SAS.

Here is a simple example of using the Multi-Statement.

```
proc append base=MYTera.loadtest (TPT=YES MULTISTMT=YES
                                  TPT_PACKMAXIMUM=YES)
            data=work.insdata;
run;
```

The TPT_PACK= and TPT_PACKMAXIMUM= option are specific to the Multi-Statement protocol.Multi-Statement enables SAS to send multiple insert statements to Teradata, which can result in substantial performance increases. The TPT_PACK= option allows you to set this packing factor. The default is 20. You can set this value as high as 600. But you don't have to set it at all. The TPT_PACKMAXIMUM= option tells the TPT API to determine the best value and use it. If you include TPT tracing, you can see what the value was set to. Here is a portion of a trace file (generated by the TPT_TRACE_OUTPUT= option) that shows the packing factor chosen by the TPT API.

```
TESTPACKFACTOR: entering
TESTPACKFACTOR: TestPack: 600
TESTPACKFACTOR: Testing pack factor 600
TESTPACKFACTOR: DML group 1 Array Support: ON
TESTPACKFACTOR:   IndicLen: 600
TESTPACKFACTOR:    DataLen: 15600
TESTPACKFACTOR: MaxRowSize: 64260
TESTPACKFACTOR: nDataBytes: 26
<deleted lines…>
TESTPACKFACTOR: Prepare returns: 0
TESTPACKFACTOR: my_oper_area->TemplateInfo.bDMLUse: FALSE
**** 14:59:18 Packing set to 600 statement(s) per request
TESTPACKFACTOR: leaving
```

Output 4. Trace Output from TPT Showing the Packing Factor That Was Chosen

Like the other examples that we have seen, Multi-Statement is best used with session management, utility slot and tracing options enabled. Here is an example.

```
proc append base=MYTera.loadtest (TPT=YES MULTISTMT=YES
                                  TPT_PACKMAXIMUM=YES
                                  TPT_MIN_SESSIONS=1
                                  TPT_MAX_SESSIONS=6
                                  SLEEP=2
                                  TENACITY=1
                                  TPT_LOG_TABLE=LOADTEST_RS
                                  TPT_TRACE_LEVEL=2
                                  TPT_TRACE_LEVEL_INF=12
                                  TPT_TRACE_OUTPUT='C:\SAS_TPT_multistmt.txt')
            data=work.insdata;
run;
```

For more information about restarting, see the TPT_RESTART= data set option in the *SAS/ACCESS 9.2 for Relational Databases.* This document is available here: http://support.sas.com/documentation/onlinedoc/access.

WORKING IN PRODUCTION ENVIRONMENTS

UTILITY SLOTS

The FastLoad, MultiLoad, and FastExport protocols use a Teradata utility slot when performing their work. If there is not an available utility slot for your job, your load may fail. Fortunately, we have a way of mitigating this issue. First, let's take a closer look at the utility slots.

There are a finite number of utility slots available in a Teradata Server. The number of slots is based on the Teradata Server configuration. In other words, your DBA determines the optimal number of slots in the environment. The default number of slots available is 5, but it is commonly increased to 15 (the maximum).

The MAXLOADTASKS parameter in DBSControl can determine the number of utility slots available in a Teradata Server. In current versions of Teradata, MAXLOADTASKS can be overridden by Teradata Active Server Management (TASM). There is no easy way for an end user to query the system to learn the current setting of MAXLOADTASKS. (Contact your DBA for more information.) However, you might be able to guess the value by querying the DBC.SessionInfo view :

```
select count(*) as UtilityCount
  from dbc.sessioninfo
where partition in ('MLOAD','FASTLOAD','EXPORT');
```

There will be times when your load job cannot immediately get an available utility slot. . The SLEEP= and TENACITY= options help protect our production jobs from failing because of unavailable utility slots.

The SLEEP= option tells the TPT API to wait *n* number of minutes before trying to logon again.

The TENACITY= option tells the TPT API to continue trying to logon for *h* number of hours.

Here is an example. This Update operator (MultiLoad) job will try to logon every 5 minutes (sleep) for the next 2 hours (tenacity).

```
proc append base=MYTera.loadtest (TPT=YES MULTILOAD=YES
                                  SLEEP=5
                                  TENACITY=2)
            data=work.adddata;
run;
```

It is a great idea to discuss these settings with your DBA when coding a production load job.

SESSIONS

Your DBA may want you to limit the number of sessions that your jobs use. In fact, it is quite common for a DBA to ask SAS users to use fewer sessions for load jobs.

There are many reasons for limiting the number of sessions used by your jobs. Some of the more common reasons include:

- Requesting a large number of sessions increases the chance that your job will have to wait for resources.
- Logging into each session takes time and can delay execution.
- Excessive jobs consume resources unnecessarily, which could prevent other jobs from running.

By default, your TPT API jobs in SAS will create one session per available Access Module Processor (AMP). In many cases, the result is an excessive number of sessions being used. Fortunately, you can limit the number of sessions by specifying the TPT_MAX_SESSIONS= option for your job. Choosing an appropriate value for this option may require a conversation with your DBA.

The following SQL query displays the number of AMPs available to your Teradata Server.

```
select count(*) Number_of_AMPs
  from (select distinct vproc
          from dbc.ampusage) a;
```

The TPT_MIN_SESSIONS= option allows you to specify the minimum number of sessions that your job will use. The default value is 1, so your job will use a single session. If having one session works for your job, you are set. Chances are that you will want to set this value higher, so your job will perform better.

If you set TPT_MIN_SESSIONS= to a higher value, then also including TPT_MAX_SESSIONS= your code will not work. You will see a very descriptive error in your SAS log. TPT_MIN_SESSIONS= and TPT_MAX_SESSIONS= can be set to the same value.

Choosing appropriate values for these options is best accomplished by trial and error. It is best to start at lower values and increase the maximum number of sessions until performance no longer increases substantially. Choose the smallest minimal value that delivers acceptable performance.

Best Practice: Meet with your DBA to discuss how to determine the best values for the TPT_MIN_SESSIONS= and TPT_MAX_SESSIONS= options and always code them on your load jobs. Don't rely on the defaults.

SEPARATING UTILITY TABLES FROM PRODUCTION TABLES

By default, the TPT utilities will create tables in the database where your target table is located. This poses a huge problem because most, if not all, production shops prohibit creating tables in a production database. Fortunately, it is possible to load a production table while creating the utility's working table in a separate database. The solution to this dilemma requires some setup on the database side, which means that you will have to enlist the aid of your DBA. Unfortunately, to my knowledge, this technique isn't documented anywhere – until now.

This technique is best shown with an example. This paper includes two scripts (SQL and SAS) that you can use to recreate this example environment on your Teradata Server.

Here is a simple TPT Load operator (FastLoad) job that loads a table named LOADTEST in the production database (TPTLOAD).

```
libname tdserv teradata server=tdserv user=tptuser password=tptuser1 database=TPTLOAD;

proc append base=tdserv.loadtest (TPT=YES FASTLOAD=YES
                                  TPT_ERROR_TABLE_1 = LOADTEST_ET
                                  TPT_ERROR_TABLE_2 = LOADTEST_UV
                                  TPT_LOG_TABLE     = LOADTEST_RS)
            data=work.loaddata;
run;
```

This job will load the TPTLOAD.LOADTEST table and attempt to create three tables: TPTLOAD.LOADTEST_ET, TPTLOAD.LOADTEST_UV, and TPTLOAD.LOADTEST_RS. Chances are TPTUSER will not have CREATE TABLE privileges in the production database TPTLOAD, so this job will fail.

It is possible to fully qualify the table names in the error table options, but that alone will not fix the problem.

Here is an example of the TPT_ERROR_TABLE_1 option being set with a fully qualified table name.

```
TPT_ERROR_TABLE_1 = 'TPTLOAD.LOADTEST_ET'
```

Here is what we want to do. Our Teradata user (TPTUSER) needs to be able to load a table in the production database (TPTLOAD). Because this is a production environment, the utility tables created by the TPT API Load operator should be created in the TPTUTIL database. This environment is depicted in Figure 1.

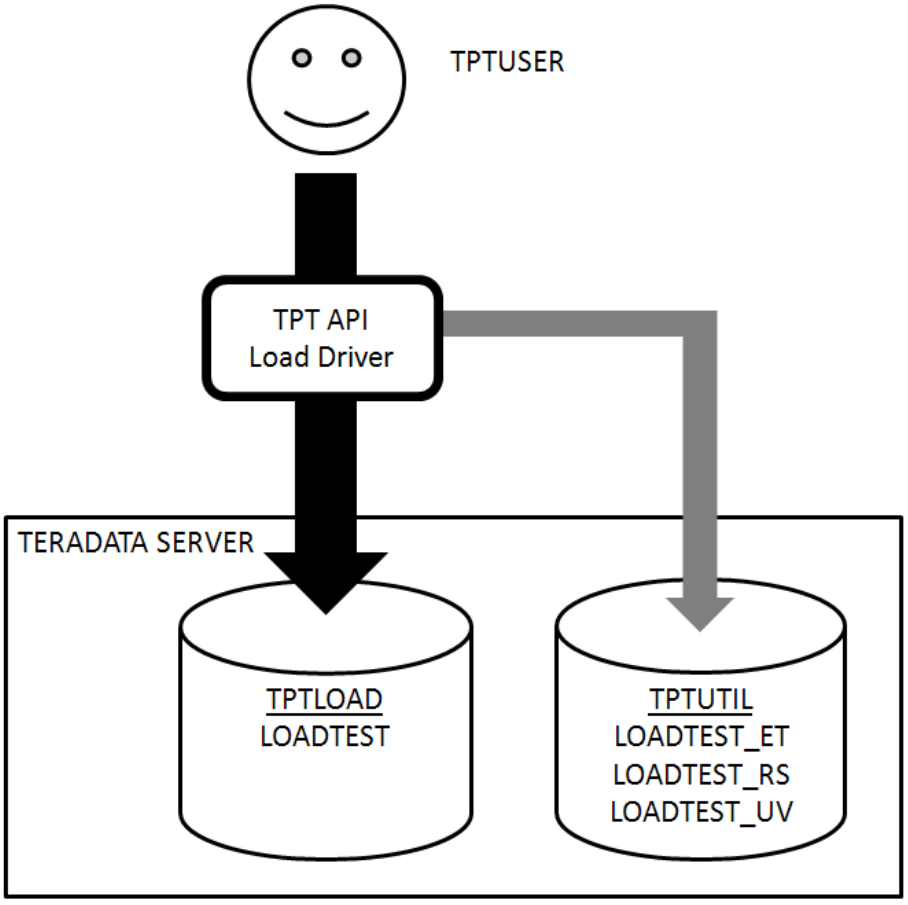

Figure 1. Example Production Environment Configuration for TPT API Load Operator Job

In order for this plan to work, we need to alter the TPTUSER, so that TPTLOAD is its default database. Setting the database using the DATABASE= option will not work. Fortunately, it is easy to set this up, but you will need to enlist the aid of your DBA. Your DBA can configure the user so that TPTLOAD is the default database. In our example, the TPTUSER has the proper default database defined when it is created.

Your DBA must grant the proper privileges to TPTUSER, TPTLOAD, and TPTUTIL. In order to grant these privileges, the DBA connects to the Teradata Server as a privileged user and issues the appropriate GRANT commands. These GRANT commands must be issued in our example environment.

Grants on the TPTLOAD Database

```
grant insert on TPTLOAD to tptuser;

grant select on TPTLOAD to tptuser;

grant delete on TPTLOAD to tptuser;

-- Allow DBA to create/drop the TPTLOAD.LOADTEST table.

grant create table on TPTLOAD to dbc;

grant drop table   on TPTLOAD to dbc;

-- This one is tricky.

grant insert on tptload to tptutil with grant option;
```

Grants on the TPTUTIL Database

```
grant create table on tptutil to tptuser;

grant drop   table on tptutil to tptuser;

grant insert       on tptutil to tptuser;

grant delete       on tptutil to tptuser;

grant select       on tptutil to tptuser;

grant create macro on tptutil to tptuser;

grant drop macro   on tptutil to tptuser;
```

The next step is to set the error and log table options properly in your SAS code. This is not difficult.

```
/* DBA has altered TPTUSER so that TPTLOAD is the default database */
/* TPTUSER has been granted the required privileges.               */

libname tdserv teradata server=rufus user=tptuser password=tptuser1;

proc append base=tdserv.loadtest (TPT=YES FASTLOAD=YES
                                  TPT_BUFFER_SIZE=64
                                  TPT_ERROR_TABLE_1 = 'TPTLOAD.LOADTEST_ET'
                                  TPT_ERROR_TABLE_2 = 'TPTLOAD.LOADTEST_UV'
                                  TPT_LOG_TABLE     = 'TPTLOAD.LOADTEST_RS')
            data=work.loaddata;
run;
```

If everything has been configured properly, the TPTLOAD.LOADTEST table will be loaded, and no errors encountered. It is a good idea to make sure that there are no utility tables left in the TPTUTIL database. You can verify this using SAS, Basic Teradata Query Facility (BTEQ), or Teradata SQL Assistant. Checking this with SAS is easy. You can submit a LIBNAME statement and use the Explorer window to look at the table list.

You can list the tables in the TPTUTIL database using the following PROC SQL code:

```
/* List utility tables in the TPTUTIL database */
proc sql;
   connect to teradata (server=tdserv user=tptuser password=tptuser1);
   select * from connection to Teradata
      (select tableName from dbc.tables where DatabaseName='TPTUTIL');
   disconnect from teradata;
quit;
```

Sample Environment SQL Script

This BTEQ script will help your DBA create the database objects used in this example. Don't worry if you don't know how to run this script. Your DBA will. This script connects to Teradata, creates the two databases (TPTLOAD and TPTUTIL), the LOADTEST table, and the TPTUSER user. In addition to creating the objects the script issues the proper grants for the environment. Please feel free to contact me (Jeff.Bailey@sas.com) for an electronic copy of this script.

```
-- The logon command will need to be altered for your environment.
-- Your DBA will know how to run this script.
--
.logon tdserv/dbc,dbcPW1;

-- run as DBC
--

-- Drop objects so the script can be rerun.
--
DROP TABLE TPTLOAD.LOADTEST;

DROP DATABASE TPTLOAD;
DROP DATABASE TPTUTIL;
DROP USER TPTUSER;
```

```
-- Create the objects
--
CREATE DATABASE TPTLOAD
   FROM DBC
AS
   PERM  = 1073741820
   SPOOL = 1073741820
   NO FALLBACK
   NO AFTER JOURNAL
   NO BEFORE JOURNAL;

CREATE DATABASE TPTUTIL
   FROM DBC
AS
   PERM  = 1073741820
   SPOOL = 1073741820
   NO FALLBACK
   NO AFTER JOURNAL
   NO BEFORE JOURNAL;

-- Pay special attention to the DEFAULT DATABASE clause
--
CREATE USER TPTUSER
   FROM DBC
AS
   PASSWORD=tptuser1
   PERM  = 1073741820
   SPOOL = 1073741820
   TEMPORARY  = 1073741820
   DEFAULT DATABASE = TPTLOAD
   NO FALLBACK;

-- Grant proper privileges
--
grant insert on TPTLOAD to tptuser;
grant select on TPTLOAD to tptuser;
grant delete on TPTLOAD to tptuser;
grant create table on TPTLOAD to dbc;
grant drop table   on TPTLOAD to dbc;

grant create table on tptutil to tptuser;
grant drop   table on tptutil to tptuser;
grant insert       on tptutil to tptuser;
grant delete       on tptutil to tptuser;
grant select       on tptutil to tptuser;
grant create macro on tptutil to tptuser;
grant drop macro   on tptutil to tptuser;

-- this one is tricky!
grant insert on tptload to tptutil with grant option;

-- Create a very simple test table
--
create table TPTLOAD.loadtest(i integer, j varchar(20));

.exit;
```

Sample SAS program

After the database objects have been created, and the proper privileges granted, you will want to load the new table using SAS. This sample program creates test data and loads it into the environment created by the SQL script. This code creates test data and moves it into Teradata using the Load operator (FastLoad), Update operator (MultiLoad), and Stream operator (Multi-Statement). Please feel free to contact me (Jeff.Bailey@sas.com) for an electronic copy

of this SAS program.

```
libname tdserv teradata server=tdserv user=tptuser password=tptuser1;

OPTIONS fullstimer;

data work.loaddata;
   format j $20.;
   do i = 1 to 10000;
      j = '01234567890123456789';
      output;
   end;
run;

data work.adddata;
   format j $20.;
   do i = 20001 to 30000;
      j = '01234567890123456789';
      output;
   end;
run;

data work.insdata;
   format j $20.;
   do i = 30001 to 40000;
      j = '01234567890123456789';
      output;
   end;
run;

proc append base=tdserv.loadtest (TPT=YES FASTLOAD=YES
                                  TPT_BUFFER_SIZE=64
                                  TPT_ERROR_TABLE_1 = 'TPTUTIL.LOADTEST_ET'
                                  TPT_ERROR_TABLE_2 = 'TPTUTIL.LOADTEST_UV'
                                  TPT_LOG_TABLE     = 'TPTUTIL.LOADTEST_RS'
                                  TPT_TRACE_LEVEL=2
                                  TPT_TRACE_LEVEL_INF=12
                                  TPT_TRACE_OUTPUT='C:\SAS_TPT_fastload.txt')
            data=work.loaddata;
run;

proc append base=tdserv.loadtest (TPT=YES MULTILOAD=YES
                                  TPT_BUFFER_SIZE=64
                                  TPT_ERROR_TABLE_1 = 'TPTUTIL.LOADTEST_ET'
                                  TPT_ERROR_TABLE_2 = 'TPTUTIL.LOADTEST_UV'
                                  TPT_LOG_TABLE     = 'TPTUTIL.LOADTEST_RS'
                                  TPT_WORK_TABLE    = 'TPTUTIL.LOADTEST_WT'
                                  TPT_TRACE_LEVEL=2
                                  TPT_TRACE_LEVEL_INF=12
                                  TPT_TRACE_OUTPUT='C:\SAS_TPT_multiload.txt')
            data=work.adddata;
run;

/* This is the one that is causing the abend. */
proc append base=tdserv.loadtest (TPT=YES MULTISTMT=YES
                                  TPT_PACKMAXIMUM=YES
                                  TPT_LOG_TABLE='TPTUTIL.LOADTEST_RS'
```

```
                                    TPT_TRACE_LEVEL=2
                                    TPT_TRACE_LEVEL_INF=12
                                    TPT_TRACE_OUTPUT='C:\SAS_TPT_multistmt.txt')
            data=work.insdata;
run;
```

SAS TPT OPTIONS (LISTED ALPHABETICALLY)

FASTLOAD=YES|NO

This option tells SAS to use the Teradata FastLoad capability. When used with the TPT=YES option, the TPT API is used. If TPT=NO, then the CLI version of FastLoad is used. This option is an alias for the BULKLOAD= option. By specifying the FASTLOAD= option in your code, you can easily see which Teradata protocol is being used. The default value for the FASTLOAD option is NO.

MULTILOAD=YES|NO

This option tells SAS to use the Teradata MultiLoad capability. When used with the TPT=YES option, the TPT API is used. If TPT=NO, then the MultiLoad utility is used. Using the MutilLoad utility is a hassle because configuration files are needed to set many of the options. It's not fun. The TPT API is much easier to use and faster because there is no overhead required to start the utility. The default value for the MULTILOAD option is NO.

MULTISTMT=YES|NO

If you need to insert rows into Teradata and you are not going to use the FASTLOAD= or MULTILOAD= options, then you want to set the MULTISTMT option to YES. By default, SAS inserts one row at a time into Teradata. Setting the MULTISTMT option to YES causes SAS to insert multiple rows at a time (as many rows as will fit in a 64K buffer). If multiple rows will not fit into the 64K buffer, then SAS goes back single row inserts.

SLEEP=6 (HOW OFTEN)

There are a finite number of utility slots available on a Teradata Server. There are times when there are no slots available to a load job. That is where the SLEEP= and TENACITY= options are useful. The SLEEP= option specifies the number of minutes that a stalled utility waits before it attempts to logon again. The value that you specify for this option must be greater than 0.

This option works in conjunction with the TENACITY= option, which enables you to specify the time (in hours) that utilities will attempt to connect.

For example, if SLEEP=5 and TENACITY=2, then the utilities will attempt to logon every 5 minutes for the next 2 hours.

TENACITY=4 (HOW LONG)

The TENACITY= option tells the TPT API operators how long (in hours) to continue logon attempts when there are no utility slots available. The value that you specify for this option must be greater than 0.

This option is usually used in conjunction with the SLEEP= option.

For example, if SLEEP=5 and TENACITY=2, then the utilities will attempt to logon every 5 minutes for the next 2 hours.

TPT=YES|NO

This option determines if SAS will use the TPT API. If you want to use the TPT API for your loads, you are not required to set this option because the default value is YES. However, it is a good idea to specify this option in your SAS program, because then it is obvious in the code that you want to use TPT.

TPT_BUFFER_SIZE=64

This option specifies the size of the FastLoad and MultiLoad output data parcels in kilobytes. Valid values for this option range from 1 to 64. Setting this to a small size, say 4, really slows down processing. Sometimes slowing things down is a good idea. It allows you to see that the utility tables are being created properly. When you test with small tables, there is a good chance that the job will run properly and delete the utility tables so fast that you cannot see them.

The default value of 64 yields great performance. TPT=YES must be specified in order to use this option.

TPT_DATA_ENCRYPTION= YES|NO

There are times when you must encrypt all communication between the Teradata client and server. The TPT_DATA_ENCRYPTION option allows you to do that when using the TPT API. The default value of NO tells SAS not to encrypt communications. This option is valid only if you have specified TPT=YES.

TPT_ERROR_TABLE_1=

FastLoad and MultiLoad jobs require error tables. This option is valid only if you have specified TPT=YES. The TPT_ERROR_TABLE_1= option allows you to name the first error table (the acquisition error table) that Teradata uses to hold rows that were rejected during processing. By default, the name used for this table is the target table's name with "_ET" appended to the end of it. ET is shorthand for error table. The TPT_ERROR_TABLE_1= option allows you to use a specific name for this table, but there is a more important reason to use it.

In many organizations, there are rules governing where you can create a table. Chances are you will need to create your temporary tables in a specific database. This option allows you to do that. This technique is not covered in the SAS documentation. Here is an example:

```
TPT_ERROR_TABLE_1='ERRTEMP.LOAD_ET'
```

If there are no errors encountered, the tables are dropped at the end of the job. Here is where the TPT_BUFFER_SIZE= option is useful. Setting this option to a low value (4 or 8) slows the load down so that you can verify that your error tables are being created properly. Don't forget to remove the option or set it back to 64 when you are done.

The error table you specify must be new. You cannot use an existing table unless you are restarting a paused Load operator job.

TPT_ERROR_TABLE_2=

This option is similar to TPT_ERROR_TABLE_1. This option is valid only if you have specified TPT=YES. The TPT_ERROR_TABLE_2= option allows you to specify the name of the second error table (the application error table). These are errors encountered during the application phase of load processing. If you have data that violates primary index constraints, that data is placed in this table. By default, the name used for this table is the target table's name with "_UV" appended to the end of it. UV is shorthand for uniqueness violations.

In many organizations, there are rules governing where you can create a table. Chances are you will need to create your temporary tables in a specific database. This option allows you to do that. This technique is not covered in the SAS documentation. Here is an example:

```
TPT_ERROR_TABLE_2='ERRTEMP.LOAD_UV'
```

If there are no errors encountered, the tables are dropped at the end of the job. Here is where the TPT_BUFFER_SIZE= option is useful. Setting this option to a low value (4 or 8) slows the load down so that you can verify that your error tables are being created properly. Don't forget to remove the option or set it back to 64 when you are done.

The error table you specify must be new. You cannot use an existing table unless you are restarting a paused Load operator job.

TPT_LOG_TABLE=

The restart log table contains information used by Teradata to restart failed load (FastLoad and MultiLoad) jobs. Like the previously discussed error tables (first error table and second error table), the restart table is dropped after the job successfully completes. This option is valid only if you have specified TPT=YES.

If you don't explicitly set a value for the TPT_LOG_TABLE option, then the table is named tablename_RS. Chances are you will not be allowed to create a table in the database where the target load table lives. Fortunately, we can place the table in a specific database. Here is an example.

```
TPT_LOG_TABLE='ERRTEMP.LOAD_RS'
```

If there are problems encountered during the load process, this table can be used to restart the job. If there are no issues encountered during the load, then the table is dropped at the end of the job. The log table must be new. If the name specified (or chosen by default) for the work table exists, then the job fails. You must restart or drop and create the tables in order for the job to run.

TPT_MAX_SESSIONS=1

Here is a compelling reason to use the TPT API: DBAs are very concerned with session use, and the TPT_MAX_SESSIONS option enables you to limit sessions. Using this option helps you keep your DBA happy. The TPT_MAX_SESSIONS option allows you to limit the number of sessions your FastLoad, MultiLoad or Multi-Statement jobs use. This option is valid only if you have specified TPT=YES.

The default value for this option is 1, which means one session per Access Module Process (AMP). This value is typically too high. You will want to set it to a more reasonable (lower) value. For more information, see the "Session" section of this paper.

TPT_MIN_SESSIONS=1

This is a sister option to the TPT_MAX_SESSIONS= option, but it isn't quite as dangerous. This option is valid only if you have specified TPT=YES. Increasing the minimum number of sessions may result in better performance. There are potential problems possible. For more information, see the "Session" section of this paper.

Also, the TPT_MIN_SESSIONS= option must be set to a value that is equal to or lesser than the TPT_MAX_SESSIONS= option.

TPT_PACK=20

This option allows you to specify the number of statements to pack into a Multi-Statement insert requests. Currently, the maximum value you can set is 600. Altering the pack size can greatly improve the performance of your jobs, but before you set this to 600, see the description of the TPT_PACKMAXIMUM= option. It is the next option in this list. The TPT_PACK option is valid only if you have specified TPT=YES.

TPT_PACKMAXIMUM=YES|NO

This is a really important option for Multi-Statement TPT jobs. Unfortunately, the SAS documentation (SAS 9.2 TS 2M3) is incorrect. Here is what this option really does.

Setting this option to YES (which is not the default) tells the Teradata Stream operator to dynamically determine the best pack factor (specified by the TPT_PACK= option) for the current job. By allowing the Teradata Stream operator to determine the pack factor, you can greatly increase the performance of Multi-Statement jobs, and you don't have to worry about determining a great pack size. You can use the trace options to see what is used for your job.

These trace options will allow you to see the value chosen for the pack size:

- TPT_TRACE_LEVEL=2
- TPT_TRACE_LEVEL_12
- TPT_TRACE_OUTPUT='C:\TPT_TRACE_OUTPUT.txt'

After you have run your job search, your trace file for the phrase "Packing set to," and you will see the packing factor used.

TPT_RESTART=YES|NO

This option specifies that a failed FastLoad, MultiLoad, or Multi-Statement job is being restarted. The default is NO which means that the job is not being restarted. A detailed discussion of restarting is beyond the scope of this paper. For more information, see the TPT_RESTART option in *SAS/ACCESS 9.2 for Relational Databases Reference*. This document is available here: http://support.sas.com/documentation/onlinedoc/access.

TPT_TRACE_LEVEL=1

This option is quite useful because it specifies the tracing level for sending data to Teradata using the TPT API. This option is valid only if you have specified TPT=YES. Level 1, the default, is no tracing. Valid values are the integer numbers between 1 and 9. Trace level 2 (operator-level general trace) and trace level 3 (operator-level command-line interface (CLI) trace) are the useful. You will need to experiment with this option. For more information, see the TPT_TRACE_LEVEL data set option in *SAS/ACCESS 9.2 for Relational Databases Reference.* This document is available here: http://support.sas.com/documentation/onlinedoc/access.

TPT_TRACE_OUTPUT=

This option specifies where to write the TPT_TRACE_LEVEL= output. This option is valid only if you have specified TPT=YES. Here is an example:

```
TPT_TRACE_OUTPUT='C:\SAS_TPT_Trace_Output.txt'
```

CONCLUSION

Using the SAS/ACCESS Interface to Teradata with the Teradata Parallel Transporter API is a painless way to significantly increase the performance of your load jobs. As you have seen, using SAS with the TPT API is very straight forward. The code samples in this paper are a great starting point for experimentation in your environment. In addition to the code examples, this paper includes some of the most important things that you should know (utility slot usage and how to load a production table while creating the mandatory utility tables in a non-production database). You will encounter these issues in your production environment.

This paper covers a great deal of information in a short amount of time. Hopefully, you are contemplating your next step. Here are some suggestions:

- Pass this paper along to your Teradata DBA and discuss utility slot usage. Make a plan regarding where your DBA would like you to create your utility tables.
- Create SAS programs which use the TPT API to FastLoad, MultiLoad and insert using Multi-Statement. Run these programs in your test environment. This is a great time to play with the options that were discussed in this paper. It is also an opportunity to experiment with TPT_ options that were not discussed.

If you would like sample code or have any questions about using SAS with Teradata, please feel free to contact me.

REFERENCES

Otto, Greg. Swift, Austin. 2010. “SAS/ACCESS Options for Optimal Teradata Integration.” 2010 Teradata PARTNERS User Group Conference & Expo. Miamisburg, Ohio: Teradata PARTNERS.

SAS Institute Inc., 2010. Configuration Guide for SAS 9.2 Foundation for Microsoft Windows. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/installcenter/en/ikfdtnwincg/62175/PDF/default/config.pdf

SAS Institute Inc., 2009. Configuration Guide for SAS 9.2 Foundation for UNIX Environments. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/installcenter/en/ikfdtnunxcg/61994/PDF/default/config.pdf

Teradata, 2010. Teradata Parallel Transporter Reference, Release 13.10. Miamisburg, Ohio: Teradata. Available at http://www.info.teradata.com/templates/eSrchResults.cfm?prodline=&txtpid=&txtrelno=&txtttlkywrd=TBUILD&rdsort=Title&srtord=Asc&nm=Teradata+Parallel+Transporter

Teradata, 2010. Teradata Parallel Transporter Users Guide, Release 13.10. Miamisburg, Ohio: Teradata. Available at http://www.info.teradata.com/templates/eSrchResults.cfm?prodline=&txtpid=&txtrelno=&txtttlkywrd=TBUILD&rdsort=Title&srtord=Asc&nm=Teradata+Parallel+Transporter

ACKNOWLEDGMENTS

The author extends his heartfelt thanks and gratitude to the following individuals:

Pravin Boniface, SAS Institute

John Cunningham, Teradata

Marie Dexter, SAS Institute

Greg Otto, Teradata

Austin Swift, SAS Institute

CONTACT INFORMATION

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

Jeff Bailey
SAS Campus Drive
SAS Institute Inc.
E-mail: Jeff.Bailey@sas.com