Teradata Connector for Hadoop Tutorial

Version: 1.0
April 2013
7.7.8 Existing Error table error received when exporting to Teradata in internal.fastload method ...................................................... 56
7.7.9 No more room in database error received when exporting to Teradata ..... 57
7.7.10 "No more spool space" error received when exporting to Teradata .......... 57
7.7.11 Separator is wrong or absent ................................................. 58
7.7.12 Date / Time / Timestamp format related errors ................................ 58

8 FAQ .................................................................................................................. 60
8.1 How about the Connector’s Fault Tolerance & Failover Behaviors? .......... 60
  8.1.1 Import methods ...................................................................................... 60
  8.1.2 Export methods ..................................................................................... 60
8.2 How to use User Customized Text Format Parameters? .......................... 60
8.3 How to use Unicode character as the separator? ....................................... 61
8.4 Why is the actual number of mappers less than the value of -nummappers? .... 61
8.5 Why don’t decimal values in Hadoop exactly match the value in Teradata? ..... 61
8.6 When should charset be specified in JDBC URL? ........................................ 61
8.7 How to configure the dependent jar files of TDCH? .................................... 62
8.8 How to configure capacity scheduler to prevent task skew? ....................... 62
1 Introduction

1.1 Overview

The Teradata Connector for Hadoop (TDCH) is a set of API and tools that support high-performance parallel bi-directional data movement between Teradata systems and Hadoop ecosystem of products.

TDCH can function as an end user tool with its own CLI (command line interface). It can also serve as a building block for integration with other end user tools, through a JAVA API (application programming interface). For example, it can be integrated with Sqoop so Sqoop users can use the Sqoop command line interface to move data between Teradata and Hadoop. The scope of this tutorial is limited to TDCH as an end user tool. All the examples in the tutorial use TDCH’s CLI. The JAVA API will be covered in a separate document.
1.2 Audience

TDCH is designed and implemented for the Hadoop user audience. Users in this audience are familiar with HDFS (Hadoop Distributed File System) and MapReduce. They are also familiar with the widely used tools in the Hadoop ecosystem, such as Hive and Sqoop. They are comfortable with the command line style of interfaces many of these tools support. They may have only limited knowledge about the Teradata database system.
1.3 Architecture

TDCH builds on MapReduce as its execution engine.

MapReduce is a framework for processing parallelizable problems across huge datasets using a large number of computers (nodes). Computational processing can occur on data stored either in a file system or in a database. MapReduce can take advantage of locality of data, processing data on or near the storage assets to decrease transmission of data.

The parallel and distributed nature of MapReduce offers extreme scalability as well as fault tolerance, which is capable of recovering from partial failure of servers or storage during the operations.
1.4 Features

1.4.1 Supported Hadoop Data Objects

TDCH supports loading data into and extracting data from various types of Hadoop data objects, including:

- **HDFS files**
  
  HDFS is a distributed, scalable file system designed to run on commodity hardware. HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size.

- **Hive tables**
  
  Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis. It defines a simple SQL-like query language, called QL, which enables users familiar with SQL to query the data. Hive executes queries via MapReduce.

- **Hive tables registered via HCatalog**
  
  HCatalog is a table and storage management service for data created using Apache Hadoop. HCatalog’s table abstraction presents users with a relational view of data in HDFS and ensures that users need not worry about where or in what format their data is stored.

  For Hive tables, TDCH supports loading data into a single partition in a Hive partitioned table. It also supports extracting data from a single partition in Hive partitioned table.

1.4.2 Supported File Formats

TDCH supports loading data into and extracting data from Hadoop file objects of various different formats:

- **TextFile**
  
  TextFile is structured as a sequence of lines of text, and each line consists of multiple fields. Lines and fields are delimited by separator. TextFile is easier for humans to read.

- **SequenceFile**
  
  SequenceFile is a flat file consisting of binary key/value pairs. It is extensively used in MapReduce as input/output formats.

- **RCFile**
  
  RCFile (Record Columnar File) is a data placement structure designed for MapReduce-based data warehouse systems, such as Hive. RCFile applies the concept of “first horizontally-partition, then vertically-partition”. It combines the advantages of both row-store and column-store. RCFile guarantees that data in the same row are located in the same node, and can exploit a column-wise data compression and skip unnecessary column reads.
1.4.3 Supported Methods for Loading Data into Teradata

A Teradata Connector Export Job makes use of multiple mappers to load data into a Teradata table. We currently support the following 3 methods of execution:

- **batch.insert**
  
  If batch.insert is specified as the execution method, each mapper starts a session to insert rows into the stage table via JDBC batch execution.

- **multiple.fastload**
  
  If multiple.fastload is specified as the execution method, each mapper starts a fastload job to load rows into the stage table via JDBC fastload.

- **internal.fastload**
  
  If internal.fastload is specified as the execution method, each mapper has its own session but all mappers are participating in one fastload job. All sessions are coordinated via an internal protocol.

1.4.4 Supported Methods for Extracting Data from Teradata

A Teradata Connector Import Job makes use of multiple mappers to extract data from a Teradata table. We currently support the following 3 methods of execution:

- **split.by.value**
  
  If split.by.value is specified as the execution method, each mapper starts a session to retrieve data in a given value range from a source table in Teradata.

- **split.by.hash**
  
  If split.by.hash is specified as the execution method, each mapper starts a session to retrieve data in a given hash value range, from a source table in Teradata.

- **split.by.partition**
  
  If split.by.partition is specified as the execution method, each mapper starts a session to retrieve a subset of partitions from a source table in Teradata, if the source table is already a partitioned table.

  If the source table is not a partitioned table, a partitioned stage table will be created with a partition key the same as the distribution key.

1.4.5 Controlling the Degree of Parallelism

TDCH is capable of moving massive amount of data between two extremely scalable systems – Teradata database and Hadoop cluster. Therefore, it is very important to be able to control the degree of parallelism while doing such a data movement. TDCH achieves this by controlling the number of mappers used in a MapReduce job. The number of mappers can be configured via a parameter called **nummappers**. How to set this parameter appropriately to optimize the performance of data movement will be discussed in the Performance Tuning section.
2 Installing Connector

2.1 Prerequisites

- Teradata Database 13.0
- Hadoop 1.0.3
- Hive 0.9.0 (optional)
- HCatalog 0.4.0 (optional)

The prerequisite versions listed above have been extensively tested with this version of Teradata Connector for Hadoop. We recommend using these versions of the prerequisites even though later versions may also work if there are no interface changes.

2.2 Software Download

Currently, the website for downloading the latest software release of the Teradata Connector for Hadoop (Command Line Edition, Sqoop Integration Edition, and Teradata Studio Edition) is under development. It will be available soon on Developer Exchange (DevX).

Various Hadoop distributions such as Hortonworks and Cloudera will distribute the TDCH that is integrated with Sqoop through their website and repackage it as the Hortonworks Connector for Teradata and Cloudera Connector for Teradata, respectively.

2.3 Installation

TDCH is installed with tar.gz package, which has 3 simple steps:

**Step 1:** Extract tar.gz package

```
tar -xvzf teradata-hadoop-connector-all-<version>.tar.gz
```

**Step 2:** Copy teradata-hadoop-connector-all-<version>.jar into

`<HADOOP_INSTALL>/lib/`, remove any previous version if exists.
Step 3: Copy these properties files into `<HADOOP_INSTALL>/conf/` (optional):
- teradata-export-properties.xml
- teradata-import-properties.xml

2.4 Required System Configuration

- `<HADOOP_INSTALL>/conf/mapred-site.xml`

Open the file in the editor of your choice and add the following value to `mapred.child.java.opts` property:

```
<property>
  <name>mapred.child.java.opts</name>
  <value>-Djava.security.egd=file:/dev/.urandom</value>
</property>
```

- Make changes on all Hadoop data and client nodes, and then restart Hadoop MapReduce.
3 Using Connector

3.1 Recommended System Configuration

- /proc/sys/net/core/somaxconn (optional)
  Open the file in the editor of your choice and modify value to 1024.

- /proc/sys/net/ipv4/tcp_max_syn_backlog (optional)
  Open the file in the editor of your choice and modify value to 2048.

3.2 Common Parameters for both Export and Import

TDCH supports a number of parameters for performing export and/or import jobs.

Please keep in mind that all parameters should be in **lower case**.

In this section, we will first introduce the parameters that are applicable for both export and import.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>URL for connecting to a Teradata database.</td>
<td>yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>User name for connecting to a Teradata database.</td>
<td>yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>Password for connecting to a Teradata database.</td>
<td>yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Supported Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobtype</td>
<td>The type of export/import job.</td>
<td>hcat, hive, hdfs</td>
<td>hdfs</td>
</tr>
</tbody>
</table>
### Parameter: `fileformat`

**Description:** The format of to-be-imported or to-be-exported data file in HDFS. 'hcat' and 'hive' job types support 'rcfile', 'sequencefile', and 'textfile' file formats, and 'hdfs' job type supports only 'textfile' format.

**Required:** no

**Supported Values:** rcfile, sequencefile, textfile

**Default Value:** textfile

### Parameter: `separator`

**Description:** The field separator to use with the imported/exported files. This parameter is only applicable with the 'textfile' file format.

**Required:** no

**Supported Values:** string

**Default Value:** \t

### 3.3 Export Parameters

For data export into Teradata from Hadoop, TDCH supports a number of parameters.

#### 3.3.1 Hadoop related export parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>sourcelocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The directory of to-be-exported source files in HDFS. It is required for an 'hdfs' job, optional with a 'hive' job, and not valid with an 'hcat' job. For a 'hive' job, either specify this or the 'sourceschema' parameter but not both.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>sourcedatabase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The name of the source database in Hive or HCatalog from which the data is exported. It is not valid with an 'hdfs' job.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sourcetable</td>
<td>The name of the source table in Hive or HCatalog from which the data is</td>
</tr>
<tr>
<td></td>
<td>exported. It is required for an 'hcat' job, optional with a 'hive' job, and</td>
</tr>
<tr>
<td></td>
<td>not valid with an 'hdfs' job. For a 'hive' job, either specify this or the</td>
</tr>
<tr>
<td></td>
<td>'sourcepaths' parameter but not both.</td>
</tr>
<tr>
<td>sourcepartitionschema</td>
<td>The full partition schema of the source table in Hive, in comma-separated</td>
</tr>
<tr>
<td></td>
<td>format. It is valid with a 'hive' job only. When this parameter is used, the</td>
</tr>
<tr>
<td></td>
<td>'sourcetableschema' parameter must also be specified.</td>
</tr>
<tr>
<td>sourcefieldnames</td>
<td>The names of fields to export from the source HDFS files, or from the source</td>
</tr>
<tr>
<td></td>
<td>Hive and HCatalog tables, in comma separated format. The order of the source</td>
</tr>
<tr>
<td></td>
<td>field names need to match the order of the target field names for schema</td>
</tr>
<tr>
<td></td>
<td>mapping. This parameter must be provided when the 'targetfieldnames' parameter</td>
</tr>
<tr>
<td></td>
<td>is specified.</td>
</tr>
<tr>
<td>hiveconfigurationfile</td>
<td>The path to the Hive configuration file in the HDFS. It is required for a</td>
</tr>
<tr>
<td></td>
<td>'hive' or 'hcat' job launched through remote execution or on data nodes.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Supported Values</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
</tr>
<tr>
<td>nullstring</td>
<td>string</td>
</tr>
<tr>
<td>nullnonstring</td>
<td>string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Supported Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>enclosedby</td>
<td>character</td>
<td>&quot; (double quotes)</td>
</tr>
<tr>
<td>escapedby</td>
<td>character</td>
<td>\ (back slash)</td>
</tr>
</tbody>
</table>

**enclosedby**

**Description**
A character, when specified, will be extracted from each field’s text on both end (first and last character) if it exists on both end. This parameter is only supported with an 'hdfs' job.

**Required**
no

**Supported Values**
character

**Default Value**
" (double quotes)

**escapedby**

**Description**
A character, when specified, is used to un-escape all instances of the enclosed-by and the escaped-by characters in the field's text. This parameter is only supported with an 'hdfs' job.

**Required**
no

**Supported Values**
character

**Default Value**
\ (back slash)
### 3.3.2 Teradata related export parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required</th>
<th>Supported Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>targettable</td>
<td>The name of target table in Teradata system.</td>
<td>yes</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>targetfieldnames</td>
<td>The names of fields to export to the target table in Teradata system, in comma separated format. The order of the target field names must match the order of the source field names for schema mapping. This parameter must be provided when the 'sourcefieldnames' parameter is specified.</td>
<td>no</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>targetfieldcount</td>
<td>The number of fields to export to the target table in Teradata system. Either specify this or the 'targetfieldnames' parameter but not both.</td>
<td>no</td>
<td>integer</td>
<td>0</td>
</tr>
<tr>
<td>stagedatabase</td>
<td>The database with which Teradata Connector for Hadoop uses to create staging tables.</td>
<td>no</td>
<td>the name of a database in Teradata system</td>
<td>the current logon database in JDBC connection</td>
</tr>
<tr>
<td>stagetablename</td>
<td>The name in Teradata system with which Teradata Connector for Hadoop uses to create staging table, if staging is required. Its length cannot exceed 20 characters. It should be used when the target Teradata table has name exceeding 20 characters.</td>
<td>no</td>
<td>string less than 20 characters</td>
<td></td>
</tr>
</tbody>
</table>
### 3.3.3 Miscellaneous Export parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required</th>
<th>Supported Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>method</strong></td>
<td>The method Teradata Connector for Hadoop uses to export data to Teradata system.</td>
<td>no</td>
<td>batch.insert, multiple.fastload, internal.fastload</td>
<td>batch.insert</td>
</tr>
<tr>
<td><strong>nummappers</strong></td>
<td>The maximum number of output mapper tasks. If the value is zero, then the number of mappers will be same as the number of file blocks in HDFS. Use either this parameter or 'numreducers', but not both.</td>
<td>no</td>
<td>an integer greater than or equal to zero</td>
<td>2</td>
</tr>
<tr>
<td><strong>numreducers</strong></td>
<td>The maximum number of output reducer tasks if export is done in reduce phase. Use either this or the 'nummappers', but not both.</td>
<td>no</td>
<td>an integer greater than or equal to zero</td>
<td>0</td>
</tr>
<tr>
<td><strong>fastloadsockethost</strong></td>
<td>The job client host name or IP address that fastload tasks communicate with to synchronize its states. This parameter is valid with 'internal.fastload' method only. If this parameter is not specified, Teradata Connector for Hadoop will automatically lookup for the node that the job is launched on, the configuration values of the 'dfs.datanode.dns.interface' parameter or the 'mapred.tasktracker.dns.interface' parameter if these are configured. Otherwise, Connector will select the IP address of the node's first network interface.</td>
<td>no</td>
<td>resolvable host name or IP address</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>fastloadsocketport</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>The host port that fastload tasks will communicate with to synchronize its states. This parameter is valid with the 'internal.fastload' method only. If this parameter is not specified, Teradata Connector for Hadoop will automatically select an available port starting from 8678.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supported Values</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>fastloadsockettimeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The timeout value for the server socket that listens to the fastload tasks connection. The unit is millisecond. If this parameter is not specified, Teradata Connector for Hadoop will use the default value of 480000.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>480000</td>
</tr>
</tbody>
</table>

One can specify such export parameters as command line arguments via the command line interface. For example:

```bash
hadoop com.teradata.hadoop.tool.TeradataExportTool
-url jdbc:teradata://dbserver/database=testdb
-username dbc
-password dbc
-classname com.teradata.jdbc.TeraDriver
-fileformat textfile
-jobtype hdfs
-method internal.fastload
-sourcepaths /user/hduser/export_sample
-nummapper 20
-targettable export_data
```

The above commands illustrate that the job type is 'hdfs', and all data fields from ‘/user/hduser/export_sample’ in Hadoop are exported to an ‘export_data’ table in Teradata system using the ‘internal.fastload’ method. The storage format of the files is ‘textfile’.
3.4 Import Parameters

For data import into Hadoop from Teradata, TDCH supports a number of parameters.

3.4.1 Hadoop related import parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>targetpaths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The directory with which to place the imported data. It is required for a 'hdfs' job, optional for a 'hive' job, and not valid for a 'hcat' job. For a 'hive' job, either specify this or the 'targettable' parameter but not both.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td>The value of property 'mapred.output.dir'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>targetdatabase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The name of the target database in Hive or HCatalog. It is optional with a 'hive' or 'hcat' job and not valid with an 'hdfs' job.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td>default</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>targettable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The name of the target table in Hive or HCatalog. It is required with a 'hcat' job, optional with a 'hive' job, and not valid with a 'hdfs' job. For a 'hive' job, specify either this parameter or the 'targetpaths' parameter but not both.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>targettableschema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The column schema of the target table, including the partition schema, in comma-separated format.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>targetpartitionschema</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Description</td>
<td>The partition schema of the target table in Hive or HCatalog, in comma-separated format. This parameter is applicable with 'hive' job only, and 'targettableschema' must be specified with it.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>targetfieldnames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The names of fields to write to the target file in HDFS, or to the target Hive or HCatalog table, in comma separated format. The order of the target field names must match exactly the order of the source field names for schema mapping. This parameter must be provided when the 'sourcefieldnames' parameter is specified.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>hiveconfigurationfile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The path to the Hive configuration file in the HDFS. It is required for a 'hive' or 'hcat' job launched through remote execution or on data nodes.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>enclosedby</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>A character, when specified, is used to enclose each field's text on both end. This parameter is only supported with an 'hdfs' job.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>character</td>
</tr>
<tr>
<td>Default Value</td>
<td>&quot; (double quotes)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>escapedby</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>A character, when specified, is used to escape all instances of the enclosed-by and the escaped-by characters in the field's text. This parameter is only supported with an 'hdfs' job.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>character</td>
</tr>
<tr>
<td>Default Value</td>
<td>\ (back slash)</td>
</tr>
<tr>
<td>Parameter</td>
<td>nullstring</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</td>
</tr>
<tr>
<td>Description</td>
<td>A string, when specified, is used to replace all instances with which the field’s value is null and the fields’ data. When the job type is ‘hdfs’, this parameter is only applied on fields with data type related to string (VARCHAR, CHAR, LONGVARCHAR, and CLOB). When the job type is ‘hive’ or ‘hcat’ and the target table does not yet exist, this field is applied to all fields.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>nullnonstring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>A string, when specified, is used to replace all instances with which the field’s value is null and the fields’ data and on Fields with data type unrelated to string (VARCHAR, CHAR, LONGVARCHAR, and CLOB). This parameter is not supported with 'hive' or 'hcat' jobs.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

### 3.4.2 Teradata related import parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>sourcequery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The SQL query to select data from Teradata database, either specify this or the 'sourcetable' parameter but not both.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>The select SQL query Teradata database supported</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>sourcetable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The name of the source table in Teradata system from which the data is imported. Either specify this or the 'sourcequery' parameter but not both.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>sourcefieldnames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>The names of columns to import from the source table in Teradata system, in comma separated format. The order of the source field names must match exactly the order of the target field names for schema mapping. This parameter must be present when the 'targetfieldnames' parameter is specified. If not specified, then all columns from the source table will be retrieved.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>batchsize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The number of rows each time Teradata Connector for Hadoop will fetch from Teradata system, up to 1MB buffer size limit. However, when the batch.insert method is used, the batchsize should not be greater than 13683, which is a limitation of Teradata database.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>an integer greater than 0 (less than or equal to 13683 when using the batch.insert method)</td>
</tr>
<tr>
<td>Default Value</td>
<td>10000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>splitbycolumn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The name of a table column to be used for splitting import tasks. It is optional with 'split.by.hash' and 'split.by.value' methods, and not valid with 'split.by.partition' method. If this parameter is not specified, the first column of the table’s primary key or primary index will be used.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>a valid table column name</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>forcestage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>If set to true, then staging is used even if source table is a PPI table. It is valid with 'split.by.partition' method only.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>true false</td>
</tr>
<tr>
<td>Default Value</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>stagedatabase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The database in Teradata system with which Teradata Connector for Hadoop uses to create staging table.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
</tbody>
</table>
### Supported Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>The name of a database in Teradata system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>The current logon database in JDBC connection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>stagetablename</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The name in Teradata system with which Teradata Connector for Hadoop uses to create staging table, if staging is required. Its length cannot exceed 20 characters. It should be used when the source Teradata table has name exceeding 20 characters.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>string</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

### Miscellaneous Import parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The method Teradata Connector for Hadoop uses to import data from Teradata system.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
</tbody>
</table>
| Supported Values | split.by.hash  
|                 | split.by.partition  
|                 | split.by.value |
| Default Value | split.by.hash |

<table>
<thead>
<tr>
<th>Parameter</th>
<th>nummappers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The number of mappers for the import job. It is also the number of splits Teradata Connector for Hadoop will attempt to create.</td>
</tr>
<tr>
<td>Required</td>
<td>no</td>
</tr>
<tr>
<td>Supported Values</td>
<td>an integer greater than 0</td>
</tr>
<tr>
<td>Default Value</td>
<td>2</td>
</tr>
</tbody>
</table>
One can specify such import parameters as command line arguments via our command line interface. For example:

```
hadoop com.teradata.hadoop.tool.TeradataImportTool
-url jdbc:teradata://dbserver/database=testdb
-username dbc
-password dbc
-classname com.teradata.jdbc.TeraDriver
-fileformat textfile
-jobtype hdfs
-method split.by.hash
-targetpaths /user/hduser/import_sample
-nummapper 20
-sourcetable testable
```

The above commands illustrate that the job type is 'hdfs' and all columns of a 'testtable' table from Teradata system are imported to the path of /user/hduser/import_sample in Hadoop using the 'split.by.hash'. The storage format of the files is 'textfile'. The number of mappers for the import job is 20.
4 Use Case Examples

4.1 Environment Variables

Execute the following on the Hadoop cluster client node; adapt the library names and their paths to the actual installed libraries on the cluster.

```bash
export HADOOP_HOME=/usr/lib/hadoop
export HIVE_HOME=/usr/lib/hive
export HCAT_HOME=/usr/lib/hcatalog-0.4.0
export LIB_JARS=$HCAT_HOME/share/hcatalog/hcatalog-0.4.0.jar
   ,$HIVE_HOME/lib/hive-metastore-0.9.0.jar
   ,$HIVE_HOME/lib/libthrift-0.7.0.jar
   ,$HIVE_HOME/lib/hive-exec-0.9.0.jar
   ,$HIVE_HOME/lib/libfb303-0.7.0.jar
   ,$HIVE_HOME/lib/jdo2-api-2.3-ec.jar
   ,$HIVE_HOME/lib/slf4j-api-1.6.1.jar
   ,$HIVE_HOME/lib/hive-cli-0.9.0.jar
   ,$HIVE_HOME/lib/hive-builtins-0.9.0.jar
export HADOOP_CLASSPATH=$HCAT_HOME/share/hcatalog/hcatalog-0.4.0.jar
   :$HIVE_HOME/lib/hive-metastore-0.9.0.jar
   :$HIVE_HOME/lib/libthrift-0.7.0.jar:$HIVE_HOME/lib/hive-exec-0.9.0.jar
   :$HIVE_HOME/lib/libfb303-0.7.0.jar:$HIVE_HOME/lib/jdo2-api-2.3-ec.jar
   :$HIVE_HOME/conf:$HADOOP_HOME/conf
   :$HIVE_HOME/lib/datanucleus-core-2.0.3.jar
   :$HIVE_HOME/lib/datanucleus-rdbms-2.0.3.jar
   :$HIVE_HOME/lib/datanucleus-connectionpool-2.0.3.jar
   :$HIVE_HOME/lib/mysql-connector-java-5.1.17-bin.jar
   :$HIVE_HOME/lib/commons-dbcp-1.4.jar
   :$HIVE_HOME/lib/commons-pool-1.5.4.jar
   :$HIVE_HOME/lib/hive-cli-0.9.0.jar
   :$HIVE_HOME/lib/hive-builtins-0.9.0.jar
```
4.2 Use Case: Import from Teradata Table to HDFS File

4.2.1 Setup: Create a Teradata table and insert data

Execute the following in a Teradata SQL client (e.g. BTEQ)

```
.LOGON testsystem/testuser
    DATABASE testdb;
    CREATE MULTISET TABLE example1_td (  
        c1 INT  
        , c2 VARCHAR(100)  
    );
    INSERT INTO example1_td VALUES (1,'foo');
```

4.2.2 Run: TeradataImportTool command

Execute the following on the client node command line

```
hadoop com.teradata.hadoop.tool.TeradataImportTool  
    -libjars $LIB_JARS  
    -url jdbc:teradata://testsystem/database=testdb  
    -username testuser  
    -password testpassword  
    -jobtype hdfs  
    -sourcetable example1_td  
    -nummappers 1  
    -separator ',,'  
    -targetpaths /user/mapred/ex1_hdfs  
    -method split.by.hash  
    -splitbycolumn c1
```

- Set job type as hdfs
- Set source Teradata table name
- Set separator, e.g. comma
- Set target paths (not exist yet)
- The import job uses the split.by.hash method
- The column used to make data split
4.3 Use Case: Export from HDFS File to Teradata Table

4.3.1 Setup: Create a Teradata table

Execute the following in a Teradata SQL client (e.g. BTEQ)

```
.LOGON testsystem/testuser
DATABASE testdb;
CREATE MULTISET TABLE example2_td (  c1 INT  ,c2 VARCHAR(100) );
.LOGOFF
```

4.3.2 Setup: Create a HDFS file

Execute the following on the console command line

```
echo "2,acme" > /tmp/example2_hdfs_data
hadoop fs -mkdir /user/mapred/example2_hdfs
hadoop fs -put /tmp/example2_hdfs_data /user/mapred/example2_hdfs/01
rm /tmp/example2_hdfs_data
```
4.3.3 Run: TeradataExportTool command

Execute the following on the client node command line

```bash
hadoop com.teradata.hadoop.tool.TeradataExportTool
-libjars $LIB_JARS
-url jdbc:teradata://testsystem/database=testdb
-username testuser
-password testpassword
-jobtype hdfs
-sourcepaths /user/mapred/example2_hdfs
-nummappers 1
-separator ','
-targettable example2_td
-forcestage true
-stagedatabase testdb
-stagetablename export_hdfs_stage
-method internal.fastload
-fastloadsockethost 153.65.179.70
-fastloadsocketport 8988
```

- Set job type as hdfs
- Set source HDFS path
- Set separator, e.g. comma
- Set target Teradata table name
- Force to create stage table
- Database to create stage table
- Name of the stage table
- Use internal.fastload method
- Set the socket host and port for internal fastload
4.4 Use Case: Import from Teradata table to Hive Table

4.4.1 Setup: Create a Teradata table and insert data

Execute the following in a Teradata SQL client (e.g. BTEQ)

```sql
.LOGON testsystem/testuser
DATABASE testdb;
CREATE MULTISET TABLE example3_td (  
c1 INT  
    ,c2 VARCHAR(100)  
);  
INSERT INTO example3_td VALUES (3,'bar');
.LOGOFF
```

4.4.2 Setup: Create a Hive table

Execute the following on the Hive command line

```sql
CREATE TABLE example3_hive (  
h1 INT  
    , h2 STRING  
) STORED AS RCFILE;
```
4.4.3 Run: TeradataImportTool command

Execute the following on the client node command line

```plaintext
hadoop com.teradata.hadoop.tool.TeradataImportTool 
-libjars $LIB_JARS 
-url jdbc:teradata://testsystem/database=testdb 
-username testuser 
-password testpassword 
-jobtype hive 
-fileformat rcfile 
-sourcetable example3_td 
-nummappers 1 
-targettable example3_hive
```

- Set job type as hive
- Set file format as rcfile
- Set source TD table name
- Set target Hive table name

4.4.4 Run: TeradataImportTool command

Execute the following on the client node command line, which uses the SQL query to select data from Teradata database

```plaintext
hadoop com.teradata.hadoop.tool.TeradataImportTool 
-libjars $LIB_JARS 
-url jdbc:teradata://testsystem/database=testdb 
-username testuser 
-password testpassword 
-jobtype hive 
-fileformat rcfile 
-sourcequery "select * from example3_td" 
-nummappers 1 
-targettable example3_hive
```

- Set job type as hive
- Set file format as rcfile
- Use a SQL query to get source data
- Set target TD table name
4.5 Use Case: Import from Teradata table to Hive Table (Table not exist)

4.5.1 Setup: Create a Teradata table and insert data

Execute the following in a Teradata SQL client (e.g. BTEQ)

```sql
.LOGON testsystem/testuser
DATABASE testdb;
CREATE MULTISET TABLE example4_td ( 
   c1 INT,
   c2 FLOAT,
   c3 VARCHAR(100)
);
INSERT INTO example4_td VALUES (3,2.35,'bar');
.LOGOFF
```

4.5.2 Run: TeradataImportTool command

Execute the following on the client node command line. A new hive partitioned table (named “example4_hive” in database of “default”) with the specified table schema and partition schema will be created in the specified database at the end of import.

```
hadoop com.teradata.hadoop.tool.TeradataImportTool
-libjars $LIB_JARS
-url jdbc:teradata://testsystem/database=testdb
-username testuser
-password testpassword
-jobtype hive
-fileformat rcfile
-sourcetable example4_td
-sourcefieldnames "c1,c2,c3"
-nummappers 1
-targetdatabase default
```

- Set job type as hive
- Set file format as rcfile
- Set source TD table name
- The Hive database name to create table
4.6 Use Case: Export from Hive Table to Teradata Table

4.6.1 Setup: Create a Teradata table

Execute the following in a Teradata SQL client (e.g. BTEQ)

```
.LOGON testsystem/testuser
DATABASE testdb;
CREATE MULTISET TABLE example5_td (  
   c1 INT  
   , c2 VARCHAR(100)
);
.LOGOFF
```

4.6.2 Setup: Create a Hive table and load data

Execute the following on the Hive command line

```
CREATE TABLE example5_hive (  
   h1 INT  
   , h2 STRING
) row format delimited fields terminated by ',' stored as textfile;
```

Execute the following on the console command line

```
echo "4,acme">/tmp/example5_hive_data
```
hive -e "LOAD DATA LOCAL INPATH '/tmp/example5_hive_data' INTO TABLE example5_hive;"
rm /tmp/example5_hive_data

4.6.3 Run: TeradataExportTool command

Execute the following on the client node command line

```
hadoop com.teradata.hadoop.tool.TeradataExportTool
    -libjars $LIB_JARS
    -url jdbc:teradata://testsystem/database=testdb
    -username testuser
    -password testpassword
    -jobtype hive
    -fileformat textfile
    -sourcetable example5_hive
    -nummappers 1
    -targettable example5_td
```

4.7 Use Case: Import from Teradata PPI table to Hive Partitioned Table

4.7.1 Setup: Create a Teradata PPI table and insert data

Execute the following in a Teradata SQL client (e.g. BTEQ)

```
.LOGON testsystem/testuser
DATABASE testdb;
CREATE MULTISET TABLE example6_td (  
    c1 INT  
    , c2 DATE  
) PRIMARY INDEX (c1)
PARTITION BY RANGE_N(c2 BETWEEN DATE '2006-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' MONTH);
```
### 4.7.2 Setup: Create a Hive Partitioned Table

Execute the following on the Hive command line

```sql
CREATE TABLE example6_hive (  
    h1 INT  
) PARTITIONED BY (h2 STRING)  
STORED AS RCFILE;
```

### 4.7.3 Run: TeradataImportTool command

Execute the following on the client node command line

```bash
hadoop com.teradata.hadoop.tool.TeradataImportTool  
-libjars $LIB_JARS  
-url jdbc:teradata://testsystem/database=testdb  
-username testuser  
-password testpassword  
-jobtype hive  
-fileformat rcfile  
-sourcetable example6_td  
-sourcefieldnames "c1,c2"  
-nummappers 1  
-targettable example6_hive  
-targetfieldnames "h1,h2"
```

Specify both source and target field names so TeradataImportTool knows how to map Teradata column Hive partition columns.
4.8 Use Case: Export from Hive Partitioned Table to Teradata PPI table

4.8.1 Setup: Create a Teradata PPI table

Execute the following in a Teradata SQL client (e.g. BTEQ)

```
.LOGON testsystem/testuser
DATABASE testdb;
CREATE MULTISET TABLE example7_td ( 
   c1 INT,
   c2 DATE
) PRIMARY INDEX (c1)
PARTITION BY RANGE_N(c2 BETWEEN DATE '2006-01-01' AND DATE '2012-12-31' EACH INTERVAL '1' MONTH);
.LOGOFF
```

4.8.2 Setup: Create a Hive Partitioned Table and load data

Execute the following on the command line

```
echo "6,2012-02-18" > /tmp/example7_hive_data
```

Execute the following on the Hive command line

```
CREATE TABLE example7_tmp (h1 INT, h2 STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
CREATE TABLE example7_hive ( 
   h1 INT
) PARTITIONED BY (h2 STRING)
STORED AS RCFILE;
```
LOAD DATA LOCAL INPATH '/tmp/example7_hive_data' INTO TABLE example7_tmp

    INSERT INTO TABLE example7_hive PARTITION (h2='2012-02-18') SELECT h1 FROM example7_tmp;
    DROP TABLE example7_tmp;

4.8.3 Run: TeradataExportTool command

Execute the following on the client node command line

```
hadoop com.teradata.hadoop.tool.TeradataExportTool
    -libjars $LIB_JARS
    -url jdbc:teradata://testsystem/database=testdb
    -username testuser
    -password testpassword
    -jobtype hive
    -fileformat rcfile
    -sourcetable example7_hive
    -sourcefieldnames "h1,h2"
    -nummappers 1
    -targettable example7_td
    -targetfieldnames "c1,c2"
```

Specify both source and target field names so TeradataExportTool knows how to map Hive partition column to Teradata column.

4.9 Use Case: Import from Teradata Table to HCatalog Table

4.9.1 Setup: Create a Teradata table and insert data

Execute the following in a Teradata SQL client (e.g. BTEQ)

```
.LOGON testsystem/testuser
DATABASE testdb;
CREATE MULTISET TABLE example8_td ( 
    c1 INT
    , c2 VARCHAR(100)
```
4.9.2 Setup: Create a Hive Table

Execute the following on the Hive command line

```sql
CREATE TABLE example8_hive (
    h1 INT,
    h2 STRING
) STORED AS RCFILE;
```

4.9.3 Run: TeradataImportTool command

Execute the following on the client node command line

```
hadoop com.teradata.hadoop.tool.TeradataImportTool
    -libjars $LIB_JARS
    -url jdbc:teradata://testsystem/database=testdb
    -username testuser
    -password testpassword
    -jobtype hcat
    -sourcetable example8_td
    -nummappers 1
    -targettable example8_hive
```

Set job type as hcat
4.10 Use Case: Export from HCatalog Table to Teradata Table

4.10.1 Setup: Create a Teradata table

Execute the following in a Teradata SQL client (e.g. BTEQ)

```
.LOGON testsystem/testuser

DATABASE testdb;

CREATE MULTISET TABLE example9_td (  
c1 INT  
, c2 VARCHAR(100)
);

.LOGOFF
```

4.10.2 Setup: Create a Hive Table and load data

Execute the following on the Hive command line

```
CREATE TABLE example9_hive (  
h1 INT  
, h2 STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
```

Execute the following on the console command line

```
echo "8,acme">/tmp/example9_hive_data

hive -e "LOAD DATA LOCAL INPATH '/tmp/example9_hive_data' INTO TABLE example9_hive;"

rm /tmp/example9_hive_data
```

4.10.3 Run: TeradataExportTool command

Execute the following on the client node command line
hadoop com.teradata.hadoop.tool.TeradataExportTool
-libjars $LIB_JARS
-url jdbc:teradata://testsystem/database=testdb
-username testuser
-password testpassword
-jobtype hcat
-sourcetable example9_hive
-nummappers 1
-targettable example9_td

Set job type as hcat
5 Storage Space Requirements

5.1 Storage Space Required for Loading Data into Teradata

This section describes the storage space requirement for the three export methods to load data into Teradata tables:

- **batch.insert**

  This execution method creates a NOPI stage table unless the target table is already a NOPI table. Data is loaded into the stage table first before an INSERT-SELECT is executed to move the data into the target table. Please make sure there is a sufficient amount of permanent space, normally the same size as the source data, allocated to accommodate data in the stage table. Please note this is in addition to the storage space required for the target table and the spool space required for the INSERT-SELECT execution.

- **multiple.fastload**

  The export job creates a NOPI stage table, regardless what the target table type is. Data is loaded into the stage table first before an INSERT-SELECT is executed to move the data into the target table. Please make sure there is a sufficient amount of permanent space, normally the same size as the source data, allocated to accommodate data in the stage table. Please note this is in addition to the storage space required for the target table and the spool space required for the INSERT-SELECT execution.

- **internal.fastload**

  The export job creates a NOPI stage table unless the target table is an empty NOPI or PI table. Data is loaded into the stage table first before an INSERT-SELECT is executed to move the data into the target table. Please make sure there is a sufficient amount of permanent space, normally the same size as the source data, allocated to accommodate data in the stage table. Please note this is in addition to the storage space required for the target table and the spool space required for the INSERT-SELECT execution.
5.2 Storage Space Required for Extracting Data from Teradata

This section describes the storage space requirement for the three import methods to extract data from Teradata tables:

❖ split.by.value

Each mapper starts a session to retrieve data in a given value range via a SELECT SQL statement. Please make sure there is a sufficient amount of spool space required for processing the SELECT statement.

❖ split.by.hash

Each mapper starts a session to retrieve data in a given hash value range via a SELECT SQL statement. Please make sure there is a sufficient amount of spool space required for processing the SELECT statement.

❖ split.by.partition

- Source table not partitioned:

  Execute an INSERT-SELECT to move data from the source table into a partitioned stage table. Then each mapper starts a session to retrieve a partition from the stage table via a SELECT SQL statement. Please make sure there is a sufficient amount of permanent space, the same size as the source data, allocated to accommodate data in the stage table. Please note this is in addition to the spool space required for the INSERT-SELECT execution and for processing the SELECT statement.

- Source table already partitioned:

  No stage table is required. Each mapper starts a session to retrieve a partition from the source table via a SELECT SQL statement. Please make sure there is a sufficient amount of spool space required for processing the SELECT statement.
6  Performance Tuning

6.1  How to choose a nummappers Value for Data Export

- **batch.insert**

  As mentioned previously, each mapper starts a session to insert rows into a NOPI stage table. For loading a small amount of data, one can use the number of data blocks as the number of mappers. For a large amount of data, one can use a number that is smaller than the number of concurrent sessions allowed in a system.

- **multiple.fastload**

  As mentioned previously, each mapper starts a fastload job to load rows into a separate stage table via JDBC fastload. Each fastload job consumes a utility slot. There is a limit on the number of utility slots in the system, which is typically in the range of 15 to 30. One should use a number that is smaller than the number of utility slots in a system.

- **internal.fastload**

  As mentioned previously, each mapper has its own session but all mappers are participating in one fastload job. The number of sessions in this case cannot exceed the number of AMPs in a system. Therefore, one should set nummappers to be less than or equal to the number of AMPs in a system.
6.2 How to choose a nummappers Value for Data Import

- **split.by.value**

  As mentioned previously, each session is normally an all-AMP operation in this case. One should normally set the nummappers value to 20-30, as there is a limit on the number of all-AMP operations running concurrently in a system.

- **split.by.hash**

  As mentioned previously, each session is normally an all-AMP operation in this case. One should normally set the nummappers value to 20-30, as there is a limit on the number of all-AMP operations running concurrently in a system.

- **split.by.partition**
  - Source table not partitioned:

    As mentioned previously, each session is a one-AMP operation. Therefore, one may use a large number of mappers as it is not subject to the maximum number of all-AMP operations in the system.

  - Source table already partitioned:

    As mentioned previously, each session is normally an all-AMP operation in this case. One should normally set the nummappers value to 20-30, as there is a limit on the number of all-AMP operations running concurrently in a system.
6.3 How to Select Export Methods for Loading Data into Teradata

This section provides suggestions on how to choose from the three export methods for loading data into Teradata tables:

- **batch.insert**
  
  One should use this export method for loading a small amount of data. One could also use this method for loading a large amount of data in situations where multiple.fastload and internal.fastload methods are not appropriate.

- **multiple.fastload**
  
  One can use this export method for loading a large amount of data into a small Teradata system (i.e., a system with a small number of nodes and AMPs), especially in a situation where internal.fastload is not appropriate. Please note, as discussed previously, one could use only a small number of mappers with this export method.

- **internal.fastload**
  
  One should use this export method for loading a large amount of data into a large Teradata system (i.e., a system with a large number of nodes and AMPs), especially from a large Hadoop cluster. Please note, as discussed previously, one can use a large number of mappers with this export method for better throughput, but not more than the number of AMPs in the Teradata system.
6.4 How to Select Import Methods for Extracting Data from Teradata

TDCH currently supports the following 3 methods for extracting data from Teradata tables:

- **split.by.value**
  
  One should use this import method for extracting data from a non-partitioned PI table, where the PI column(s) has a small number of distinct values. Please note, as discussed previously, one can use only a small number of mappers with this import method.

- **split.by.hash**
  
  One should use this import method for extracting data from a non-partitioned table in situations where split.by.value and split.by.partition are not appropriate. Please note, as discussed previously, one can use only a small number of mappers with this import method.

- **split.by.partition**
  
  - Source table not partitioned:
    
    One should use this import method for extracting a large amount of data from a large Teradata system (i.e., a system with a large number of nodes and AMPs), especially to a large Hadoop cluster.
  
  - Source table already partitioned:
    
    One should use this import method for extracting data from a partitioned table, especially a large amount of data from a large Teradata system. Please note, as discussed previously, one can use only a small number of mappers with this import method. This method is extremely efficient when the partitioning column(s) of the source table are also the PI column(s) of that table, where a large number of mappers can be used.
7 Troubleshooting

7.1 Troubleshooting Requirements

In order to conduct a troubleshooting process, one must have the following ready:

- Command - TeradataIm(Ex)portTool, Sqoop
- Console Output
- Teradata Table DDL
- Hive Table DDL (if Hive or Hcatalog table is involved)

In addition, one should have knowledge of the following to bring the troubleshooting to next level:

- Each mapper’s output on all Hadoop nodes
- Sample data (a few rows/lines)
- Hadoop cluster configuration (under /etc/hadoop/conf)
  - slaves
  - mapred-site.xml
  - hdfs-site.xml
  - scheduler (e.g. capacitiescheduler.xml, fairscheduler.xml, etc.)
7.2 Troubleshooting Overview

The chart below provides an overview of our suggested troubleshooting process:

<table>
<thead>
<tr>
<th>Problem Area</th>
<th>JDBC</th>
<th>Database</th>
<th>Hadoop</th>
<th>Connector</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Export</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Issue Type</th>
<th>Functional (look for exceptions)</th>
<th>Performance (go through checklist)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Check for issues with each job stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup (user error, database configuration, hadoop configuration, database staging)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer Scenario (understand command parameters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import</td>
</tr>
</tbody>
</table>
7.3 Functional: Understand Exceptions

Look in the console output for

- The very last error code
  - 10000: runtime (look for database error code, or JDBC error code, or backtrace)
  - Others: pre-defined (checked) errors by TDCH

- The very first instance of exception messages

Examples:

```
com.teradata.hadoop.exception.TeradataHadoopSQLException:
    com.teradata.jdbc.jdbc_4.util.JDBCException: [Teradata Database] [TeraJDBC 14.00.00.13] [Error 5628]
        [SQLState HY000] Column h3 not found in mydb.import_hive_table.
(omitted)......
13/04/03 09:52:48 INFO tool.TeradataImportTool: job completed with exit code 10000
```

```
com.teradata.hadoop.exception.TeradataHadoopSchemaException: Field data type is invalid
    at com.teradata.hadoop.utils.TeradataSchemaUtils.lookupDataTypeByTypeName(TeradataSchemaUtils.java:1469)
(omitted)......
13/04/03 09:50:02 INFO tool.TeradataImportTool: job completed with exit code 14006
```
7.4 Functional: Data Issues

This category of issues occurs at runtime (most often with the internal.fastload method), and usually it’s not obvious what the root cause is. Our suggestion is that you can check the following:

- Does the schema match the data?
- Is the separator correct?
- Does the table DDL have time or timestamp columns?
  - Check if tnano/tsnano setting has been specified to JDBC URL
- Does the table DDL have Unicode columns?
  - Check if CHARSET setting has been specified to JDBC URL
- Does the table DDL have decimal columns
  - Before release 1.0.6, this may cause issues
- Check Fastload error tables to see what’s inside

7.5 Performance: Back of the Envelope Guide

We are all aware that no throughput is faster than the total I/O or network transfer capacities of the least powerful component in the overall solution. So our methodology is to understand max for the configuration and work it backwards.

Max theoretical throughput <= MIN (  
∑(Ttd-io), ∑(Ttd-transfer),  
∑(Thadoop-io), ∑(Thadoop-transfer), Tnetwork-transfer  
)

Therefore we should:

- Watch out for node-level CPU saturation (including core saturation), because “no CPU = no work can be done”.
- If all-node saturated with either Hadoop or Teradata, consider expanding system footprint and/or lowering concurrency
- If one-node much busier than other nodes with either Hadoop or Teradata, try to balance the workload skew
- If both Hadoop and Teradata are mostly idle, look for obvious user mistakes or configuration issues, and if possible, increase concurrency.
And here is the checklist we could go through in case of slow performance:

- **User Settings**
  - Teradata JDBC URL
    - Connecting to MPP system name? (and not single-node)
    - Connecting through correct (fast) network interface?
      - /etc/hosts
      - ifconfig
  - Using best-performance methods?
  - Using most optimal number of mappers? (small number of mapper sessions can significantly impact performance)
  - Is batch size too small? (increase to 80000)

- **Database**
  - Is database CPU or IO saturated?
    - iostat, mpstat, sar, top
  - Is there any TDWM setting limiting # of concurrent sessions or user’s query priority?
    - tdwmcmd -a
  - DBSCControl settings
    - AWT tasks: maxawttask, maxloadtask, maxloadawt
    - Compression settings
  - Is database almost out of room?
  - Is there high skew to some AMPs (skew on PI column or split-by column)

- **Network**
  - Are Hadoop network interfaces saturated?
    - Could be high replication factor combined with slow network between nodes
  - Are Teradata network interfaces saturated?
    - Could be slow network between systems
Hadoop

- Are hadoop data nodes CPU or IO saturated?
  - iostat, mpstat, sar, top
  - Could be hadoop configuration too small for the job’s size

- Are there settings limiting # of concurrent mappers?
  - mapred-site.xml
  - scheduler configuration

- Are mapper tasks skewed to a few nodes?
  - use ps | grep java on multiple nodes to see if tasks have skew
  - In capacity-scheduler.xml, set maxtasksperheartbeat to force even distribution
7.6 Console Output Structure

13/03/29 11:27:11 INFO tool.TeradataImportTool: TeradataImportTool starts at 136457083127
13/03/29 11:27:16 INFO mapreduce.TeradataInputProcessor: job setup starts at 1364570836879
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: database product is Teradata
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: database version is 13.10
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: jdbc driver version is 14.0
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: input method is split.by.hash
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: input split column is page_name
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: input query is select "page_name", "page_hour", "page_view" from "npv_m" where page_language like '%9sz6n6%' or page_name is not null
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: input database name is
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: input table name is npv_m
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: input conditions are page_language like '%9sz6n6%' or page_name is not null
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: input field names are [page_name, page_hour, page_view]
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: input batch size is 10000
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: input number of mappers are 6
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: job setup ends at 1364570843647
13/03/29 11:27:23 INFO mapreduce.TeradataInputProcessor: job setup time is 6s
13/03/29 11:27:29 INFO mapred.JobClient: map 0% reduce 0%
13/03/29 11:27:54 INFO mapred.JobClient: map 100% reduce 0%
13/03/29 11:27:59 INFO mapred.JobClient: Job complete: job_201303251205_0253
13/03/29 11:27:59 INFO mapred.JobClient: Counters: 19
13/03/29 11:27:59 INFO mapred.JobClient: Job Counters
......
13/03/29 11:27:59 INFO mapred.JobClient: Map output records=4
13/03/29 11:27:59 INFO mapred.JobClient: SPLIT_RAW_BYTES=1326
13/03/29 11:27:59 INFO mapreduce.TeradataInputProcessor: job cleanup starts at 1364570879466
13/03/29 11:28:01 INFO mapreduce.TeradataInputProcessor: job cleanup ends at 1364570881367
13/03/29 11:28:01 INFO mapreduce.TeradataInputProcessor: job cleanup time is 1s
13/03/29 11:28:01 INFO tool.TeradataImportTool: TeradataImportTool ends at 1364570881367
13/03/29 11:28:01 INFO tool.TeradataImportTool: TeradataImportTool time is 50s
13/03/29 11:28:01 INFO tool.TeradataImportTool: job completed with exit code 0

Verify parameter settings
Successful completion of Setup, Run, and Cleanup stages will have a corresponding log entry
Total Elapsed Time & Exit Code
7.7 Troubleshooting Examples

7.7.1 Database doesn’t exist

The error message on top of the error stack trace indicates that the “testdb” database does not exist:

```
com.teradata.hadoop.exception.TeradataHadoopException:
com.teradata.jdbc.jdbc_4.util.JDBCException: [Teradata Database]
[TeraJDBC 14.00.00.13] [Error 3802] [SQLState 42S02] Database ‘testdb’ does not exist.
at com.teradata.jdbc.jdbc_4.util.ErrorFactory.makeDatabaseSQLException(ErrorFactory.java:307)  at
com.teradata.jdbc.jdbc_4.statemachine.ReceiveInitSubState.action(ReceiveInitSubState.java:102)  at
com.teradata.jdbc.jdbc_4.statemachine.StatementReceiveState.subStateMachine(StatementReceiveState.java:302)  at
com.teradata.jdbc.jdbc_4.statemachine.StatementReceiveState.action(StatementReceiveState.java:183)  at
com.teradata.jdbc.jdbc_4.statemachine.StatementController.runBody(StatementController.java:121)  at
com.teradata.jdbc.jdbc_4.statemachine.StatementController.run(StatementController.java:112)  at
com.teradata.jdbc.jdbc_4.TDSession.executeSessionRequest(TDSession.java:624)  at
com.teradata.jdbc.jdbc_4.TDSession.<init>(TDSession.java:288)  at
com.teradata.jdbc.jdk6.JDK6_SQL_Connection.<init>(JDK6_SQL_Connection.java:30)  at
com.teradata.jdbc.jdbc.ConnectionFactory.constructConnection(JConnectionFactory.java:22)  at
com.teradata.jdbc.jdbc.ConnectionFactory.createConnection(ConnectionFactory.java:130)  at
com.teradata.jdbc.jdbc.ConnectionFactory.createConnection(ConnectionFactory.java:120)  at
com.teradata.jdbc.TeraDriver.doConnect(TeraDriver.java:228)  at
com.teradata.jdbc.TeraDriver.connect(TeraDriver.java:154)  at
java.sql.DriverManager.getConnection(DriverManager.java:582)  at
java.sql.DriverManager.getConnection(DriverManager.java:185)  at
com.teradata.hadoop.db.TeradataConnection.connect(TeradataConnection.java:274)
......
(omitted)
```
7.7.2 **Internal fast load server socket time out**

When running export job using the "internal.fastload" method, the following error may occur:

```
Internal fast load socket server time out
```

This error occurs because the number of available map tasks currently is less than the number of map tasks specified in the command line by parameter of "-nummappers". This error can occur in the following conditions:

1. There are some other map/reduce jobs running concurrently in the Hadoop cluster, so there are not enough resources to allocate specified map tasks for the export job.

2. The maximum number of map tasks is smaller than existing map tasks added expected map tasks of the export jobs in the Hadoop cluster.

When the above error occurs, please try to increase the maximum number of map tasks of the Hadoop cluster, or decrease the number of map tasks for the export job.

7.7.3 **Incorrect parameter name or missing parameter value in command line**

All the parameter names specified in the command line should be in lower case. When the parameters names are not correct or the necessary parameter value is missing, the following error will occur:

```
Export (Import) tool parameters is invalid
```

When this error occurs, please double check the input parameters and their values.

7.7.4 **Hive partition column can not appear in the hive table schema**

When running import job with 'hive' job type, the columns defined in the target partition schema cannot appear in the target table schema. Otherwise, the following exception will be thrown:

```
Target table schema should not contain partition schema
```

In this case, please check the provided schemas for Hive table and Hive partition.
7.7.5 Hadoop map task time out when running export or import job

Data distribution skew will result in map tasks having different running time. The running time of some map tasks which process more data will be longer than the value of "mapred.task.timeout". It will cause the exception of map task timeout.

In order to avoid this exception, the value of property "mapred.task.timeout" needs to be configured big enough when data distribution is skew.

7.7.6 String will be truncated if its length exceeds the Teradata String length (VARCHAR or CHAR) when running export job.

When running an export job, if the length of the source string exceeds the maximum length of Teradata’s String type (CHAR or VARCHAR), the source string will be truncated. It will result in data inconsistency.

To prevent that from happening, please carefully set the data schema for source data and target data.

7.7.7 Scaling number of Timestamp data type should be specified correctly in JDBC URL in internal.fastload method

When loading data into Teradata using the internal.fastload method, the following error may occur:

```java
com.teradata.hadoop.exception.TeradataHadoopException: java.io.EOFException
at java.io.DataInputStream.readUnsignedShort(DataInputStream.java:323)
   at java.io.DataInputStream.readUTF(DataInputStream.java:572)
   at java.io.DataInputStream.readUTF(DataInputStream.java:547)
   at com.teradata.hadoop.mapreduce.TeradataInternalFastloadOutputProcessor.beginLoading(TeradataInternalFastloadOutputProcessor.java:889)
   at com.teradata.hadoop.mapreduce.TeradataInternalFastloadOutputProcessor.run(TeradataInternalFastloadOutputProcessor.java:173)
   at com.teradata.hadoop.job.TeradataExportJob.runJob(TeradataExportJob.java:75)
   at com.teradata.hadoop.tool.TeradataJobRunner.runExportJob(TeradataJobRunner.java:192)
   at com.teradata.hadoop.tool.TeradataExportTool.run(TeradataExportTool.java:39)
   at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:65)
   at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:79)
   at com.teradata.hadoop.tool.TeradataExportTool.main(TeradataExportTool.java:395)
   at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```
Usually the error is caused by setting the wrong ‘tsnano’ value in the JDBC URL. In Teradata DDL, the default length of timestamp is 6, which is also the maximum allowed value, but user can specify a lower value.

When ‘tsnano’ is set to

- The same as the specified length of timestamp in the Teradata table: no problem;
- ‘tsnano’ is not set: no problem, it will use the specified length as in the Teradata table
- less than the specified length: an error table will be created in Teradata, but no exception will be shown
- Greater than the specified length: the quoted error message will be received.

7.7.8 Existing Error table error received when exporting to Teradata in internal.fastload method

If the following error occurs when exporting to Teradata using the internal.fastload method:

```
com.teradata.hadoop.exception.TeradataHadoopException: com.teradata.jdbc.jdbc_4.util.JDBCException: [Teradata Database] [TeraJDBC 14.00.00.13] [Error 2634] [SQLState HY000] Existing ERROR table(s) or Incorrect use of export_hdfs_fun1_054815 in Fast Load operation.
```

This is caused by the existence of the Error table. If an export task is interrupted or aborted while running, an error table will be generated and stay in Teradata database. Now when you try to run another export job, the above error will take place.

In this case, simply rerun the export job is OK, as the existed Error table has been deleted automatically by TDCH when the exception is thrown.
7.7.9 No more room in database error received when exporting to Teradata

If the following error occurs when exporting to Teradata:

```java
```

This is caused by the perm space of the database in Teradata being set too low. Please reset it to a higher value to resolve it.

7.7.10 “No more spool space” error received when exporting to Teradata

If the following error occurs when exporting to Teradata:

```java
java.io.IOException: com.teradata.jdbc.jdbc_4.util.JDBCException: [Teradata Database] [TeraJDBC 14.00.00.21] [Error 2646] [SQLState HY000] No more spool space in example_db.
```

This is caused by the spool space of the database in Teradata being set too low. Please reset it to a higher value to resolve it.
7.7.11 Separator is wrong or absent

If the ‘-separator’ parameter is not set or is wrong, you may run into the following error:

```java
java.lang.NumberFormatException: For input string: "12,23.45,101,complex1"
    at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1222)
    at java.lang.Double.valueOf(Double.java:475)
        ... (stack trace)
```

Please make sure the separator parameter’s name and value is specified correctly.

7.7.12 Date / Time / Timestamp format related errors

If you run into one of the following errors:

```java
java.lang.IllegalArgumentException
    at java.sql.Date.valueOf(Date.java:138)
```

```java
java.lang.IllegalArgumentException
    at java.sql.Time.valueOf(Time.java:89)
```

```java
java.lang.IllegalArgumentException: Timestamp format must be yyyy-mm-dd hh:mm:ss[.fffffffff]
```
It is caused by incorrect date / time / timestamp formats:

1) When exporting data with time, date or timestamp type from HDFS text files to Teradata:
   a) Value of date type in text files should follow the format of ‘yyyy-mm-dd’
   b) Value of time type in text files should follow the format of ‘hh:mm:ss’
   c) Value of timestamp type in text files should follow the format of ‘yyyy-mm-dd hh:mm:ss[.f...]’, length of nano should be less than 9.

2) When importing data with time, date or timestamp type from Teradata to HDFS text file:
   a) Value of date type in text files should follow the format of ‘yyyy-mm-dd’
   b) Value of time type in text files should follow the format of ‘hh:mm:ss’
   c) Value of timestamp in text files should follow the format of ‘yyyy-mm-dd hh:mm:ss.fffffffff’, length of nano is 9.

3) When exporting data from Hive text files to Teradata:
   a) Value of timestamp type in Hive text files should follow the format of ‘yyyy-mm-dd hh:mm:ss.fffffffff’ (nano is optional, maximum length is 9)

4) When importing data from Teradata to Hive text files:
   a) Value of timestamp type in Hive text files should follow the format of ‘yyyy-mm-dd hh:mm:ss.fffffffff’ (nano is optional, maximum length is 9)
8 FAQ

8.1 How about the Connector’s Fault Tolerance & Failover Behaviors?

TDCH supports the following fault tolerance behavior with Hadoop task failover for each import and export methods:

8.1.1 Import methods

- split.by.hash method supports failover with map tasks. Hadoop will re-execute the failed tasks and job will continue to run successfully.
- split.by.value method supports failover with map tasks. Hadoop will re-execute the failed tasks and job will continue to run successfully.
- split.by.partition method supports failover with map tasks. Hadoop will re-execute the failed tasks and job will continue to run successfully.

8.1.2 Export methods

- batch.insert method does not support failover with map or reduce tasks. If a task is failed by the Hadoop TaskTracker, the job will exit with an exception code. It is the user's responsibility to manually delete any rows already loaded to the target or the stage table by that point.
- multiple.fastload method supports failover with map or reduce tasks. Hadoop will re-execute the failed tasks and job will continue to run successfully. When a mapper or reducer task fails and a new task attempt is made with the multiple.fastload method, TeradataOutputFormat will examine the context's task attempt ID and attempt to cleanup stage table all data loaded by all previous attempts with the same task ID.
- internal.fastload method does not support failover with map or reduce tasks. If a task is failed by the Hadoop TaskTracker, the job will exit with an exception code. It is the user's responsibility to manually drop the stage table and the Fastload error tables associated with the job.

8.2 How to use User Customized Text Format Parameters?

TDCH provides two parameters, enclosedby and escapeby, for dealing with data containing separator characters and quote characters in the ‘textfile’ of ‘hdfs’. The default values for enclosedby is “ (double quote) and for escapeby is \ (backward slash). If the file format is not ‘textfile’ or when the job type is not ‘hdfs’, these two parameters do not take effect.

When neither parameter is specified, TDCH does not enclose or escape any characters in the data during import or scan for enclose-by or escape-by characters during export. If either or both parameters are provided, then TDCH will process enclose-by and escape-by values as appropriate.
8.3 How to use Unicode character as the separator?

- Using shell to invoke TDCH:

  When user set a Unicode character as the separator, user should input like, -separator “\uxxxx” or –separator \uxxxx where xxxx is the Unicode of this character. Shell will automatically remove double quotes and the first back slash.

- Using other methods to invoke TDCH:

  TDCH accepts a Unicode character as separator with format \uxxxx, user should make sure the separator value passed to TDCH has correct format.

8.4 Why is the actual number of mappers less than the value of -nummappers?

When you specify the number of mappers using the nummappers parameter, but in the execution, you find that the actual number of mappers is less than your specified value. This behavior is due to the fact that we use the getSplits() method of CombineFileInputFormat class of Hadoop to decide partitioned splits number. As a result, the number of mappers for running the job equals to splits number. This explains why the number of mappers is less than or equal to the value specified.

8.5 Why don’t decimal values in Hadoop exactly match the value in Teradata?

When exporting data to Teradata, if the precision of decimal type is more than that of the target Teradata column type, the decimal value will be rounded when stored in Teradata. On the other hand, if the precision of decimal type is less than the definition of the column in the Teradata table, ‘0’s will be appended to the scaling.

8.6 When should charset be specified in JDBC URL?

If the column of the Teradata table is defined as Unicode (UTF-8), then you should specify the same character set in the JDBC URL. Otherwise, it will result in wrong encoding of transmitted data, and there will be no exception thrown. Also if you want to display Unicode data on Shell or other clients correctly, don’t forget to configure your client to display as UTF-8 as well.
8.7 How to configure the dependent jar files of TDCH?

TDCH library depends on some jar files, especially for the job types of ‘hive’ and ‘hcat’. When running it via Hadoop command line, the environment variable HADOOP_CLASSPATH need to be modified to contain these jar files and the parameter of “-libjars” also need to be specified for these jar files.

The dependent jar files are as following:


8.8 How to configure capacity scheduler to prevent task skew?

We can use capacity scheduler configuration to prevent task skew.

Here are steps you should follow:

1. Make sure scheduler you are using is capacity scheduler(check mapred-site.xml and check scheduler)

2. Configure capacity-scheduler.xml(usually in the same location with mapred-site.xml,$HADOOP_HOME/conf):
   - Add this property: mapred.capacity-scheduler.maximum-tasks-per-heartbeat
   - Give a reasonable value of this property, such as 4 for a 28-mapper job.

3. Copy this xml file to each node in Hadoop cluster then restart Hadoop.