ORACLE GOLDENGATE BIG DATA ADAPTER FOR HDFS

Version 1.0

Oracle Corporation

Table of Contents

1. INTRODUCTION

The Oracle GoldenGate Big Data Adapter for HDFS is designed to stream change capture data into the Hadoop Distributed File System (HDFS).

The HDFS Adapter is designed to provide ready-made functionality, while at the same time maintaining simplicity so that the source code is easily understood. Oracle GoldenGate customers can use, modify, and extend the HDFS Adapter to fulfill their specific needs.

The information in this document is not intended to replace the Oracle GoldenGate Java Adapter documentation. This document provides additional information specific to the HDFS Adapter. Customers unfamiliar with the Oracle GoldenGate Java Adapter should start by reviewing the Oracle GoldenGate Java Adapter documentation.

1.1 Functionality

The HDFS Adapter takes unfiltered operations from the source trail file and writes them to files in HDFS.

Directory structure of files
The HDFS Adapter writes to the HDFS directory configured by the user. The default HDFS directory is `/ogg`. The format of file names created in HDFS is the following:

```
<FILE PREFIX><TIMESTAMP><FILE SUFFIX>
```

The default file prefix is `gg_` and the default file suffix is `.txt`. The timestamp is in the format `yyyy-MM-dd_HH-mm-ss.SSS`. Note: HDFS does not allow colons in file names. The prefix and the suffix are optionally configurable by the user.

Change data records
Data for each row operation is written to an HDFS file in the following format:

First is the row metadata:

```
<SCHEMA NAME><FIELD DELIMITER><TABLE NAME><FIELD DELIMITER><OPERATION
TYPE><FIELD DELIMITER><TIMESTAMP><FIELD DELIMITER>
```

Next is the row data:

```
<COLUMN 1 NAME ><FIELD DELIMITER><COLUMN 1 VALUE><FIELD
DELIMITER>...<COLUMN N NAME><FIELD DELIMITER><COLUMN N VALUE><LINE
DELIMITER>
```

The column names can be configured to be omitted.

Data for different tables in the input trail file are interlaced in the output HDFS files. The HDFS Adapter begins writing a new file in HDFS when either of the following events occurs:

1. The Oracle GoldenGate Extract process is started.
2. The configured maximum HDFS file size is reached (default is 1 GB).

1.2 Supported Operations

- Inserts
- Updates including change to primary key(s)
- Deletes

1.3 Unsupported Operations

- Truncate table

2. GETTING STARTED WITH THE HDFS ADAPTER

2.1 Runtime Prerequisites

In order to successfully run the HDFS Adapter, a Hadoop single instance or Hadoop cluster must be installed, running, and network accessible from the machine running the Oracle GoldenGate HDFS Adapter. Apache Hadoop is open source and available for download at http://hadoop.apache.org/. Follow the *Getting Started* links for information on how to install a single-node cluster (also called pseudo-distributed operation mode) or a clustered setup (also called fully-distributed operation mode).

2.2 Accessing the Precompiled HDFS Adapter

The Oracle GoldenGate Big Data Adapter for HDFS installation includes the jar file for the Oracle GoldenGate HDFS Adapter. The precompiled HDFS Adapter was built using version 2.5.1 of the HDFS client. The HDFS Adapter is available in the following location:

GG_ROOT_DIRCTORY/AdapterExamples/big-data/hdfs/bin/ogg-hdfs-adapter-1.0.jar

If the above jar is being used instead of building the HDFS Adapter from source, then ogg-hdfs-adapter-1.0.jar should be added to the `gg.classpath` variable in the properties file. The `gg.classpath` variable must also include the HDFS dependency jars. This is generally resolved by adding *HDFS_ROOT_DIRECTORY*/share/hadoop/common/lib/*:*HDFS_ROOT_DIRECTORY*/share/hadoop/common/* to the `gg.classpath` variable.

2.3 Building with Maven

The Oracle GoldenGate Java Adapter 12 is built and runs with Java 7. It is required that the HDFS Adapter also be built with the Java SE Development Kit 7.

The recommended method of building the Oracle GoldenGate HDFS Adapter is using Apache Maven. Maven is an open source build tool available for download at http://maven.apache.org. Maven is well documented, rich in functionality, and widely used in the Java community. One of the biggest benefits provided by Maven is that it will resolve and download the build and runtime dependencies of the HDFS Adapter from the Maven central repository. Internet connectivity is required to access the Maven central repository. The HDFS example was built and tested with version 3.0.5 of Maven.

The pom.xml file in the *GG_ROOT_DIRECTORY*/AdapterExamples/big-data/hdfs directory instructs Maven what version of the HDFS client to download and to build against. It is important that the HDFS client version match the version of HDFS to which it is connecting. Using

mismatched versions of the HDFS client and HDFS server can result in serious problems. The version of the HDFS client in the pom.xml file is 2.5.1. If using a version of HDFS other than 2.5.1 it is highly recommended that the user modify the pom.xml file so that HDFS Adapter is built against the correct version of the HDFS client, and the correct jar files are downloaded for use at runtime.

WARNING: Many companies employ proxy servers to help protect their network from the hazards of the Internet. Performing a Maven build while connected to the company network or VPN may require that dependency downloads from the Maven central repository go through a proxy server. This may require special configuration of the Maven `settings.xml` file which is generally located in the *`USER HOME`*`/.m2` directory. Consult your network administrator for proxy settings. Consult the Maven documentation on how to correctly configure the `settings.xml` file for your required proxy settings.

1. Navigate to the root build directory
   `GG_ROOT_DIRECTORY``/AdapterExamples/big-data/hdfs`

2. Type the following command to build:
   `mvn clean package`

3. Maven will download all the required dependencies and build the HDFS Adapter. The created jar and the downloaded dependent jars will be placed in the following directory:
   `GG_ROOT_DIRECTORY``/AdapterExamples/big-data/hdfs/target/hdfs-lib`

4. One of the dependencies that Maven does not download is the Oracle GoldenGate Java adapter (`ggjava.jar`). This dependency is located in the following directory:
   `GG_ROOT_DIRECTORY``/ggjava/ggjava.jar`

   This dependency is instead resolved using a relative path inside the `pom.xml` file. If the HDFS Adapter project is copied to a different directory, the relative path to the `ggjava.jar` file will need to be fixed by editing the `pom.xml` file.

2.4 Building with Build Script

A build script, `build.sh`, is provided in the *`GG_ROOT_DIRECTORY`*`/AdapterExamples/big-data/hdfs` directory. This script is an alternative to the Maven build outlined above. The build script performs no downloads of the dependent jars. The build script assumes that HDFS is installed locally and can resolve the build dependencies from the installation. The build requires that the user set the `HDFS_HOME` environment variable to the root HDFS installation directory. None of the dependent jars are copied using the build script. The build script simply builds and packages the HDFS adapter.

2.5 Sample Configuration Files

Sample configuration files for the Oracle GoldenGate Big Data Adapter for HDFS can be found at the following locations:

The HDFS Extract process sample properties file is located at:

GG_ROOT_DIRECTORY/AdapterExamples/big-data/hdfs/dirprm/hdfs.prm

The HDFS Java adapter sample properties file is located at:

GG_ROOT_DIRECTORY/AdapterExamples/big-data/hdfs/dirprm/hdfs.props

These files should be copied to the *GG_ROOT_DIRECTORY*/dirprm directory and renamed and or modified as required.

2.6 Classpath Configuration

The Oracle GoldenGate Big Data Adapter for HDFS obtains connectivity information to HDFS via the Hadoop core-site.xml file. This file is generally located in the *HADOOP_HOME*/etc/hadoop directory. The directory containing the properly configured core-site.xml file must be included in the classpath that is configured in the gg.classpath property in the HDFS Java adapter properties file.

The Oracle GoldenGate HDFS Adapter is dependent upon the HDFS client jars and all of its dependencies. The easiest way to fulfill the runtime dependencies of the HDFS example is by configuring the gg.classpath configuration value in the Java properties file using the wildcard asterisk character (*). The GoldenGate Java properties file is very specific as to how the wildcard character is used.

The following works:

```
gg.classpath=GG_ROOT_DIRECTORY/AdapterExamples/big-data/hdfs/target/hdfs-lib/*
```

The following does NOT work:

```
gg.classpath=GG_ROOT_DIRECTORY/AdapterExamples/big-data/hdfs/target/hdfs-lib/*.jar
```

2.7 Starting the HDFS Adapter

The final step is to create and start the GoldenGate Extract process that invokes the HDFS Adapter.

To create the Extract process from GGSCI:

```
GGSCI>ADD EXTRACT hdfs, EXTTRAILSOURCE dirdat/trail_id
```

To start the Extract process:

```
GGSCI>START hdfs
```

3. Configuration

3.1 GG Adapter Boot Options

The following can be configured using the boot options property.

- Memory allocation for GG Java Adapter JVM (`-Xms` and `-Xmx` set the initial and maximum size respectively).
- Oracle GoldenGate Adapter dependencies (`ggjava.jar`)
- Naming of a properties file which is to contain the log4j configuration. The directory containing this file must be included in the classpath.

javawriter.bootoptions=`-Xmx512m -Xms32m -Djava.class.path=ggjava/ggjava.jar -Dlog4j.configuration=log4j.prop`erties

3.2 HDFS Adapter Properties

The below properties are specific to HDFS Adapter. The `name` attribute represents the handler name configured as part of `gg.handlerlist`.

1. **`gg.handler.`*`name`*`.type`**

   The value of this property should not be changed and always should be `com.goldengate.delivery.handler.hdfs.HDFSHandler`.
   This property is mandatory.

2. **`gg.handler.`*`name`*`.HDFSPrefix`**

   The prefix of all file names created in HDFS. The default is `gg_`.
   This property is optional.

3. **`gg.handler.`*`name`*`.HDFSSuffix`**

   The suffix of all file names created in HDFS. Default is `.txt`.
   This property is optional.

4. **`gg.handler.`*`name`*`.HDFSFilePath`**

   The path location to write files in HDFS. The default is `/ogg`.
   This property is optional.

5. **`gg.handler.`*`name`*`.maxFileSize`**

The maximum file size of created files in HDFS can be configured as raw bytes or using `k`, `m`, or `g` to indicate kilobytes, megabytes, or gigabytes, respectively. Examples of legal values include: 100024, 10k, 10m, 1.5g. The default value is 1g.
This property is optional.

6. **gg.handler.*name*.fieldDelimiter**

The delimiter to be used for field/column value separation. Values like semicolon (;) or comma (,) or any other character can be set. Nonprintable characters like `\u0001`, `\u0002` are also supported. Use the Unicode format in the configuration file when using nonprintable characters as delimiters. The default value is `\u0001`.
This property is optional.

7. **gg.handler.*name*.lineDelimiter**

The delimiter to be used for line/row separation. Printable and nonprintable characters are supported. The default value is `\n`.
This property is optional.

8. **gg.handler.*name*.writeColumnNames**

The default is `true`, meaning column names will be output to the HDFS file before each associated column value. Set to `false` to omit column names.
This property is optional.

4. PERFORMANCE CONSIDERATIONS

By default, data is flushed to the HDFS server at the end of each transaction. This behavior is the same if operating in `op` or `tx` mode, so performance is not likely to be significantly different between modes. If transactions are small (containing few operations), performance may be increased by employing the new transaction grouping functionality. The transaction grouping functionality allows operations which cross multiple transaction boundaries to be grouped together as a single transaction. Because performance is dependent upon many variables, the use of the transaction grouping functionality cannot guarantee increased performance. Transaction grouping is simply a mechanism to help customers tune the Oracle GoldenGate Java Adapter to their specific needs.

4.1 Grouping Configuration

1. **gg.handler.*name*.mode**

   To enable grouping, the value of this property must be set to `tx`.

2. **gg.handler.*name*.maxGroupSize**

   Controls the maximum number of operations that can be held by an operation group – irrespective of whether the operation group holds operations from a single transaction or multiple transactions.
   The operation group will send a transaction commit and end the group as soon as this number of operations is reached. This property leads to splitting of transactions across multiple operation groups

3. **gg.handler.*name*.minGroupSize**

   This is the minimum number of operations that must exist in a group before the group can end. This property helps to avoid groups that are too small by grouping multiple small transactions into one operation group so that it can be more efficiently processed.

NOTE: `maxGroupSize` should always be greater than or equal to `minGroupSize;` i.e. `maxGroupSize >= minGroupSize`.

Example: - Consider a scenario where the source trail has 100 transactions with 10 operations each and the handler is configured in `tx` mode.

Without grouping functionality, transaction commit occurs for every 10 records ultimately flushing a batch of 10 events to HDFS.

With grouping enabled by setting maximum and minimum group size to 1000, transaction commit occurs for 1000 records, ultimately flushing 1000 events to HDFS.

5. LIMITATIONS

The Oracle GoldenGate HDFS Handler does not support truncate table operations. A truncate table operation will cause the Extract process to `ABEND`.

Oracle GoldenGate requires that supplemental logging be at least minimally enabled. How supplemental logging is configured can affect the output of the HDFS Adapter. Primary key updates are supported but may be problematic because that operation must be treated as a delete and subsequent insert because data is only appended to HDFS files.

The new unified update record is not yet supported by the HDFS Adapter and will cause the Extract process to `ABEND`.