Data Storage (Hive and HBase)

Jim Harner

5/30/2020

Initially, bash is used to access Hive. Then an R version is shown using Java and JDBC. The following packages are needed for the R interface to Hive:

```
library(rJava)
library(DBI)
library(RJDBC)
```

4.4 Data Storage

4.4.1 Hive

Hive is a data warehouse infrastructure built on HDFS. It provides data querying and data summarization, and has facilities for reading, writing, and managing large datasets residing in distributed storage using SQL.

Hive was created to make it possible for analysts with strong SQL skills (but meager Java programming skills) to run queries on the huge volumes of data stored in HDFS. SQL isn't a good fit for building complex machine-learning algorithms, but it's great for many analyses, and it has the huge advantage of being very well known in the industry and it is the *lingua franca* in business intelligence tools.

HiveServer1 used Hive CLI (command line interface) to send SQL commands to Hive for execution using MapReduce. Due to deficiencies in HiveServer1, e.g., lack of concurrency and security, HiveServer2 was developed. HiveServer2 has multi-client concurrency and better authentication. It also supports coonections through JDBC and ODBC.

The CLI used in HiveServer2 is `beeline`. Beeline is a JDBC client based on the SQLLine CLI, which communicates with the HiveServer using Thrift APIs. The connection to HiveServer2 is a JDBC URL. The following code can be executed line-by-line in RStudio's Shell.

```
/opt/hive/bin/beeline -u jdbc:hive2://hive:10000
or
# /opt/hive/bin/beeline
# !connect jdbc:hive2://hive:10000
# The user name is `hive` with no password (hit return)
# The prompt becomes: jdbc:hive2://hive:10000>
jdbc:hive2://hive:10000> show databases;
# Use ! to execute beeline commands rather than shell commands
# The following is the same as `show databases;` above
jdbc:hive2://hive:10000> !sql show databases;
# `!quit` or `!q` exits interactive mode
jdbc:hive2://hive:10000> !quit;
```

We can execute an external file with the `-f` option, which in this case creates the `test1` database.

```
/opt/hive/bin/beeline -u jdbc:hive2://hive:10000 -f "hive-test.sql"
```

```
## SLF4J: Class path contains multiple SLF4J bindings.
## SLF4J: Found binding in [jar:file:/opt/apache-hive-2.1.1-bin/jdbc/hive-jdbc-2.1.1-standalone.jar!/org
## SLF4J: Found binding in [jar:file:/opt/apache-hive-2.1.1-bin/lib/log4j-slf4j-impl-2.4.1.jar!/org/slf4
## SLF4J: Found binding in [jar:file:/opt/hadoop-2.9.2/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar
## SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
## SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
## Connecting to jdbc:hive2://hive:10000
## Connected to: Apache Hive (version 2.1.1)
## Driver: Hive JDBC (version 2.1.1)
## 20/06/01 02:04:05 [main]: WARN jdbc.HiveConnection: Request to set autoCommit to false; Hive does no
## Transaction isolation: TRANSACTION_REPEATABLE_READ
## 0: jdbc:hive2://hive:10000> drop table if exists test1 purge;
## No rows affected (0.209 seconds)
## 0: jdbc:hive2://hive:10000> create table test1 (a int, b string);
## No rows affected (0.063 seconds)
## 0: jdbc:hive2://hive:10000> insert into test1 (a,b) values (1, 'foo');
## WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Conside
## No rows affected (1.537 seconds)
## 0: jdbc:hive2://hive:10000> insert into test1 (a,b) values (2, 'bar');
## WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Conside
## No rows affected (1.925 seconds)
## 0: jdbc:hive2://hive:10000>
## Closing: 0: jdbc:hive2://hive:10000
```

We use the `-e` option to specify a query string. Multiple queries are separated by `;`.

```
/opt/hive/bin/beeline -u jdbc:hive2://hive:10000 -e "show tables; select * from test1 LIMIT 10;"
```

```
## SLF4J: Class path contains multiple SLF4J bindings.
## SLF4J: Found binding in [jar:file:/opt/apache-hive-2.1.1-bin/jdbc/hive-jdbc-2.1.1-standalone.jar!/or
## SLF4J: Found binding in [jar:file:/opt/apache-hive-2.1.1-bin/lib/log4j-slf4j-impl-2.4.1.jar!/org/slf
## SLF4J: Found binding in [jar:file:/opt/hadoop-2.9.2/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar
## SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
## SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
## Connecting to jdbc:hive2://hive:10000
## Connected to: Apache Hive (version 2.1.1)
## Driver: Hive JDBC (version 2.1.1)
## 20/06/01 02:04:13 [main]: WARN jdbc.HiveConnection: Request to set autoCommit to false; Hive does no
## Transaction isolation: TRANSACTION_REPEATABLE_READ
## +-----------+--+
## | tab_name  |
## +-----------+--+
## | test1     |
## +-----------+--+
## 1 row selected (0.164 seconds)
## +----------+----------+--+
## | test1.a  | test1.b  |
## +----------+----------+--+
## | 1        | foo      |
## | 2        | bar      |
## +----------+----------+--+
## 2 rows selected (0.114 seconds)
## Beeline version 2.1.1 by Apache Hive
## Closing: 0: jdbc:hive2://hive:10000
```

Notice that the output is messy and surrounds the desired results.

Details of running Hive from the command line can be found here.

The R interface to Hive uses the `DBI` and `RJDBC` packages. We now repeat the above `beeline` commands using R. First, a connection to the database must be established through JDBC. The URL is similar to the one above, but the interface is through the `rstudio`.

```
.jinit(classpath = Sys.getenv("CLASSPATH"))
drv <- JDBC("org.apache.hive.jdbc.HiveDriver",
            classPath = "/opt/hive/jdbc/hive-jdbc-2.1.1-standalone.jar",
            identifier.quote = "'")
con <- dbConnect(drv, "jdbc:hive2://hive:10000/rstudio", "rstudio", "")
```

Next we construct a table in Hive.

```
dbSendUpdate(con, "drop table if exists test1 purge")
dbSendUpdate(con, "create table test1 (a int, b string)")
dbSendUpdate(con, "insert into test1 (a,b) values (1, 'foo')")
dbSendUpdate(con, "insert into test1 (a,b) values (2, 'bar')")
```

We can now show the Hive tables and query a table.

```
show_tables <- dbGetQuery(con, "show tables")
show_tables
```

```
##                  tab_name
## 1                   test1
## 2 values__tmp__table__1
## 3 values__tmp__table__2
```

```
query <- "select * from test1 LIMIT 10"
dbGetQuery(con, query)
```

```
##   test1.a test1.b
## 1       1     foo
## 2       2     bar
```

Hive has many advantages:

1. Ability to handle structured and unstructured data;

2. Real-time streaming of data into Hive from Flume;

3. Extend functionality through user-defined functions;

4. A well-documented, SQL-like interface specifically built for OLAP versus OLTP.

4.4.2 HBase

A scalable, distributed database that supports structured data storage for large tables.

HBase is not a column-oriented database in the RDBMS sense, but utilizes an on-disk column storage format. Although HBase stores data on disk in a column-oriented format, it is distinctly different from traditional columnar databases: whereas columnar databases excel at providing real-time analytical access to data, HBase excels at providing key-based access to a specific cell of data, or a sequential range of cells.

The most common filesystem used with HBase is HDFS, but you are not locked into HDFS because the filesystem used by HBase has a pluggable architecture.

HBase can be used from the command line for basic operations, such as adding, retrieving, and deleting data. To start simply type `hbase` into the shell.

The `rhbase` package in RHadoop provides basic connectivity to the HBASE distributed database, using the Thrift server. R programmers can browse, read, write, and modify tables stored in HBASE from within R.

```
library(rhbase)
hb.init(host="hadoop", port=9000)
```

We have not yet built a container for `hbase`.