Architecture of Enterprise Applications 30
Hive

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents and Objectives

- Hive
  - Basic Concepts
  - Hive DDL and DML
  - Other Issues

- Objectives
  - 能够根据数据仓库的基础要求，设计基于Hive的数据分析方案

Apache Hive

- Hive, a framework for data warehousing on top of Hadoop.
  - Hive grew from a need to manage and learn from the huge volumes of data that Facebook was producing every day from its burgeoning social network.
  - After trying a few different systems, the team chose Hadoop for storage and processing, since it was cost-effective and met their scalability needs.

- 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 that Facebook stored in HDFS.
  - Today, Hive is a successful Apache project used by many organizations as a general-purpose, scalable data processing platform.

- https://hive.apache.org/

Apache Hive

- In normal use, Hive runs on your workstation and converts your SQL query into a series of MapReduce jobs for execution on a Hadoop cluster.
  - Hive organizes data into tables, which provide a means for attaching structure to data stored in HDFS.
  - Metadata— such as table schemas—is stored in a database called the metastore.

Running Hive

REliable, INtelligent & Scalable Systems

- Hive uses Hadoop, so:
  - you must have Hadoop in your path OR
  - `$ export HADOOP_HOME=<hadoop-install-dir>`
- Start NameNode daemon and DataNode daemon:
  - `$ sbin/start-dfs.sh`
- In addition,
  - you must use below HDFS commands to create /tmp and /user/hive/warehouse
  - (aka hive.metastore.warehouse.dir) and set them chmod g+w before you can create a table in Hive.
  - `$ HADOOP_HOME/bin/hadoop fs -mkdir /tmp`
  - `$ HADOOP_HOME/bin/hadoop fs -mkdir /user/hive/warehouse`
  - `$ HADOOP_HOME/bin/hadoop fs -chmod g+w /tmp`
  - `$ HADOOP_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse`

Running Hive

- To use the Hive command line interface (CLI) from the shell:
  - `$ HIVE_HOME/bin/hive`

- **Running HiveServer2 and Beeline**
  - Starting from Hive 2.1, we need to run the schematool command below as an initialization step. For example, we can use "derby" as db type.
  - `$ HIVE_HOME/bin/schematool -dbType <db type> -initSchema`
- HiveServer2 (introduced in Hive 0.11) has its own CLI called Beeline.
  - HiveCLI is now deprecated in favor of Beeline, as it lacks the multi-user, security, and other capabilities of HiveServer2. To run HiveServer2 and Beeline from shell:
  - `$ HIVE_HOME/bin/hiveserver2`
  - `$ HIVE_HOME/bin/beeline -u jdbc:hive2://$HS2_HOST:$HS2_PORT`

Configuration Management Overview

- Hive by default gets its configuration from <install-dir>/conf/hive-default.xml
- Hive configuration is an overlay on top of Hadoop
  - it inherits the Hadoop configuration variables by default.
- Hive configuration can be manipulated by:
  - Editing hive-site.xml and defining any desired variables (including Hadoop variables) in it
  - Using the set command
  - Invoking Hive (deprecated), Beeline or HiveServer2 using the syntax:
    - $ bin/hive --hiveconf x1=y1 --hiveconf x2=y2 //this sets the variables x1 and x2 to y1 and y2 respectively
    - $ bin/hiveserver2 --hiveconf x1=y1 --hiveconf x2=y2 //this sets server-side variables x1 and x2 to y1 and y2 respectively
    - $ bin/beeline --hiveconf x1=y1 --hiveconf x2=y2 //this sets client-side variables x1 and x2 to y1 and y2 respectively.
  - Setting the HIVE_OPTS environment variable to "--hiveconf x1=y1 --hiveconf x2=y2" which does the same as above.

Hive, Map-Reduce and Local-Mode

- Hive compiler generates map-reduce jobs for most queries.
  - These jobs are then submitted to the Map-Reduce cluster indicated by the variable:
  - `mapred.job.tracker`
  - While this usually points to a map-reduce cluster with multiple nodes, Hadoop also offers a nifty option to run map-reduce jobs locally on the user's workstation.
  - This can be very useful to run queries over small data sets – in such cases local mode execution is usually significantly faster than submitting jobs to a large cluster.
  - Data is accessed transparently from HDFS. Conversely, local mode only runs with one reducer and can be very slow processing larger data sets.

Hive, Map-Reduce and Local-Mode

- Starting with release 0.7, Hive fully supports local mode execution.
  - To enable this, the user can enable the following option:
  - `hive> SET mapreduce.framework.name=local;`
- In addition,
  - mapred.local.dir should point to a path that's valid on the local machine (for example /tmp/<username>/mapred/local).
  - (Otherwise, the user will get an exception allocating local disk space.)

Hive, Map-Reduce and Local-Mode

- Starting with release 0.7, Hive also supports a mode to run map-reduce jobs in local-mode automatically.
  - The relevant options are hive.exec.mode.local.auto, hive.exec.mode.local.auto.inputbytes.max, and hive.exec.mode.local.auto.tasks.max:
  - `hive> SET hive.exec.mode.local.auto=false;`
- Note that this feature is *disabled* by default.
  - If enabled, Hive analyzes the size of each map-reduce job in a query and may run it locally if the following thresholds are satisfied:
    - The total input size of the job is lower than: hive.exec.mode.local.auto.inputbytes.max (128MB by default)
    - The total number of map-tasks is less than: hive.exec.mode.local.auto.tasks.max (4 by default)
    - The total number of reduce tasks required is 1 or 0.
  - So for queries over small data sets, or for queries with multiple map-reduce jobs where the input to subsequent jobs is substantially smaller (because of reduction/filtering in the prior job), jobs may be run locally.

DDL Operations

- The Hive DDL operations are documented in Hive Data Definition Language.
- **Creating Hive Tables**
  - `hive> CREATE TABLE pokes (foo INT, bar STRING);`
  - creates a table called pokes with two columns, the first being an integer and the other a string.
  - `hive> CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);`
  - creates a table called invites with two columns and a partition column called ds.
  - The partition column is a virtual column. It is not part of the data itself but is derived from the partition that a particular dataset is loaded into.
  - By default, tables are assumed to be of text input format and the delimiters are assumed to be ^A(ctrl-a).

DDL Operations

REliable, INtelligent & Scalable Systems

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

/user/hive/warehouse Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	chenhaopeng	supergroup	0 B	Dec 24 22:09	0	0 B	invites
drwxr-xr-x	chenhaopeng	supergroup	0 B	Dec 24 22:06	0	0 B	pokes
drwxr-xr-x	chenhaopeng	supergroup	0 B	Dec 24 22:15	0	0 B	u_data
drwxr-xr-x	chenhaopeng	supergroup	0 B	Dec 24 22:27	0	0 B	u_data_new

Showing 1 to 4 of 4 entries Previous 1 Next

Hadoop, 2021.

DDL Operations

- **Browsing through Tables**
  - `hive> SHOW TABLES;`
  - lists all the tables.

  - `hive> SHOW TABLES '.*s';`
  - lists all the table that end with 's'. The pattern matching follows Java regular expressions.

  - `hive> DESCRIBE invites;`
  - `OK`
  - `foo  int`
  - `bar  string`
  - `ds  string`
  -
  - `# Partition Information`
  - `# col_name  data_type  comment`
  - `ds  string`
  - `Time taken: 0.181 seconds, Fetched: 7 row(s)`

DDL Operations

- **Altering and Dropping Tables**
  - Table names can be changed and columns can be added or replaced:
  - `hive> ALTER TABLE events RENAME TO 3koobecaf;`
  - `hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);`
  - `hive> ALTER TABLE invites ADD COLUMNS (new_col2 INT COMMENT 'a comment');`
  - `hive> ALTER TABLE invites REPLACE COLUMNS (foo INT, bar STRING, baz INT COMMENT 'baz replaces new_col2');`
  - Note that REPLACE COLUMNS replaces all existing columns and only changes the table's schema, not the data.
  - The table must use a native SerDe. REPLACE COLUMNS can also be used to drop columns from the table's schema:
  - `hive> ALTER TABLE invites REPLACE COLUMNS (foo INT COMMENT 'only keep the first column');`
  - Dropping tables:
  - `hive> DROP TABLE pokes;`

DDL Operations

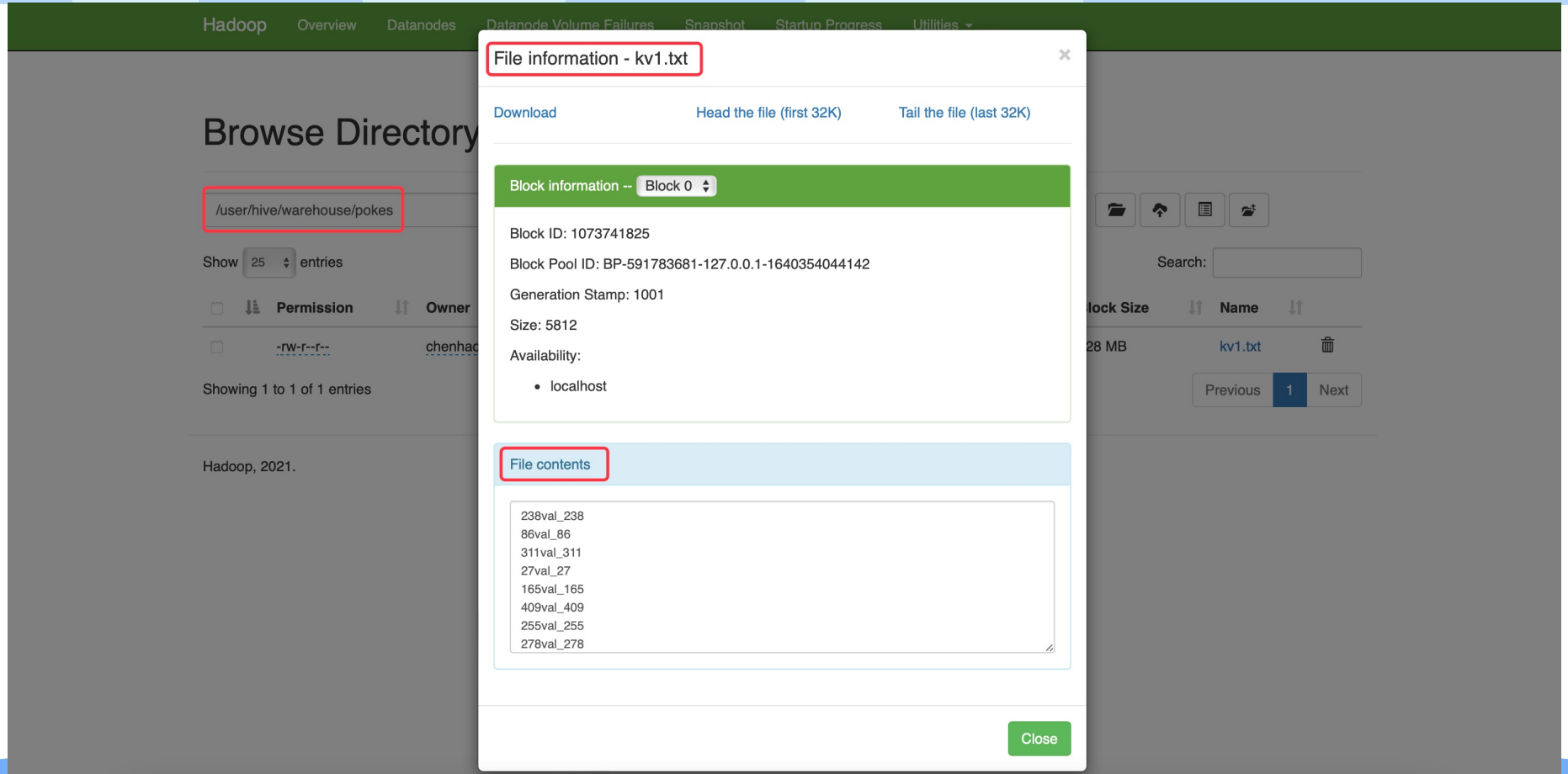
- **Metadata Store**
  - Metadata is in an embedded Derby database whose disk storage location is determined by the Hive configuration variable named javax.jdo.option.ConnectionURL.
    - By default this location is ./metastore_db (see conf/hive-default.xml).
  - Right now, in the default configuration, this metadata can only be seen by one user at a time.
  - Metastore can be stored in any database that is supported by JPOX.
  - The location and the type of the RDBMS can be controlled by the two variables javax.jdo.option.ConnectionURL and javax.jdo.option.ConnectionDriverName.
  - In the future, the metastore itself can be a standalone server.
  - If you want to run the metastore as a network server so it can be accessed from multiple nodes, see Hive Using Derby in Server Mode.

DML Operations

- The Hive DML operations are documented in Hive Data Manipulation Language.
  - Loading data from flat files into Hive:
  - `hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;`
  - Loads a file that contains two columns separated by ctrl-a into pokes table.
  - 'LOCAL' signifies that the input file is on the local file system. If 'LOCAL' is omitted then it looks for the file in HDFS.
  - The keyword 'OVERWRITE' signifies that existing data in the table is deleted. If the 'OVERWRITE' keyword is omitted, data files are appended to existing data sets.
- NOTES:
  - NO verification of data against the schema is performed by the load command.
  - If the file is in hdfs, it is moved into the Hive-controlled file system namespace.
  - The root of the Hive directory is specified by the option hive.metastore.warehouse.dir in hive-default.xml. We advise users to create this directory before trying to create tables via Hive.

DML Operations

REliable, INtelligent & Scalable Systems



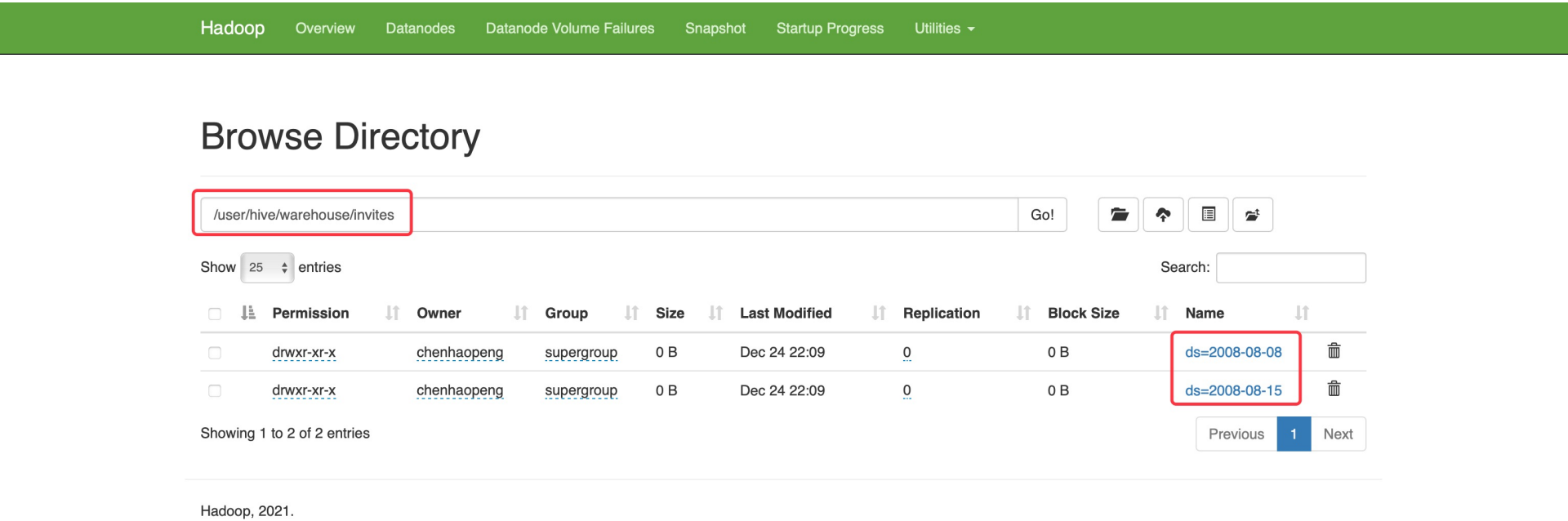
DML Operations

- Data Load
  - `hive> LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');`
  - `hive> LOAD DATA LOCAL INPATH './examples/files/kv3.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-08');`
  - The two LOAD statements above load data into two different partitions of the table invites.
  - Table invites must be created as partitioned by the key ds for this to succeed.

  - `hive> LOAD DATA INPATH '/user/myname/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');`
  - The above command will load data from an HDFS file/directory to the table.
  - Note that loading data from HDFS will result in moving the file/directory. As a result, the operation is almost instantaneous.

DML Operations

REliable, INtelligent & Scalable Systems



SQL Operations

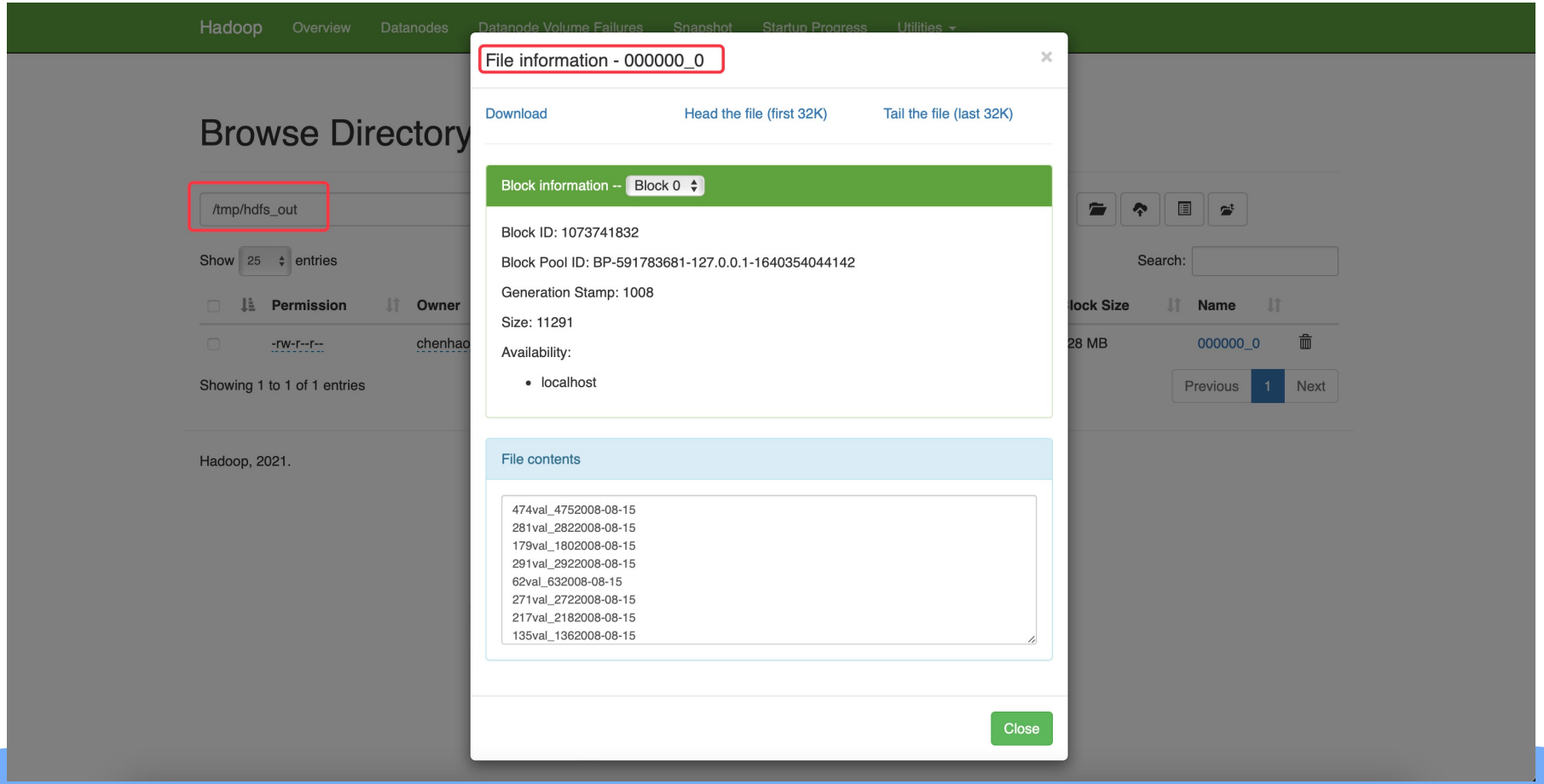
- The Hive query operations are documented in Select.
- **Example Queries**
  - Some example queries are shown below. They are available in build/dist/examples/queries. More are available in the Hive sources at ql/src/test/queries/positive.
- **SELECTS and FILTERS**
  - `hive> SELECT a.foo FROM invites a WHERE a.ds='2008-08-15';`
  - selects column 'foo' from all rows of partition ds=2008-08-15 of the invites table. The results are not stored anywhere, but are displayed on the console.

SQL Operations

- Note that in all the examples that follow, INSERT (into a Hive table, local directory or HDFS directory) is optional.
  - `hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT a.* FROM invites a WHERE a.ds='2008-08-15';`
  - selects all rows from partition ds=2008-08-15 of the invites table into an HDFS directory.
  - The result data is in files (depending on the number of mappers) in that directory.
    NOTE: partition columns if any are selected by the use of *. They can also be specified in the projection clauses.
  - Partitioned tables must always have a partition selected in the WHERE clause of the statement.

SQL Operations

REin
REliable, INtelligent & Scalable Systems



SQL Operations

- **SELECTS and FILTERS**
  - `hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local_out' SELECT a.* FROM pokes a;`
  - selects all rows from pokes table into a local directory.

  - `hive> INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a;`
  - `hive> INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a WHERE a.key < 100;`
  - `hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/reg_3' SELECT a.* FROM events a;`
  - `hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_4' select a.invites, a.pokes FROM profiles a;`
  - `hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT COUNT(*) FROM invites a WHERE a.ds='2008-08-15';`
  - `hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT a.foo, a.bar FROM invites a;`
  - `hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/sum' SELECT SUM(a.pc) FROM pc1 a;`
  - selects the sum of a column. The avg, min, or max can also be used.

SQL Operations

- **GROUP BY**
  - `hive> FROM invites a INSERT OVERWRITE TABLE events SELECT a.bar, count(*) WHERE a.foo > 0 GROUP BY a.bar;`
  - `hive> INSERT OVERWRITE TABLE events SELECT a.bar, count(*) FROM invites a WHERE a.foo > 0 GROUP BY a.bar;`
  - Note that for versions of Hive which don't include HIVE-287, you'll need to use COUNT(1) in place of COUNT(*).
- **JOIN**
  - `hive> FROM pokes t1 JOIN invites t2 ON (t1.bar = t2.bar) INSERT OVERWRITE TABLE events SELECT t1.bar, t1.foo, t2.foo;`

SQL Operations

- **MULTITABLE INSERT**
  - `FROM src`
  - `INSERT OVERWRITE TABLE dest1 SELECT src.* WHERE src.key < 100`
  - `INSERT OVERWRITE TABLE dest2 SELECT src.key, src.value WHERE src.key >= 100 and src.key < 200`
  - `INSERT OVERWRITE TABLE dest3 PARTITION(ds='2008-04-08', hr='12') SELECT src.key WHERE src.key >= 200 and src.key < 300`
  - `INSERT OVERWRITE LOCAL DIRECTORY '/tmp/dest4.out' SELECT src.value WHERE src.key >= 300;`
- **STREAMING**
  - `hive> FROM invites a INSERT OVERWRITE TABLE events SELECT TRANSFORM(a.foo, a.bar) AS (oof, rab) USING '/bin/cat' WHERE a.ds > '2008-08-09';`
  - This streams the data in the map phase through the script /bin/cat (like Hadoop streaming). Similarly – streaming can be used on the reduce side (please see the Hive Tutorial for examples).

Simple Example Use Cases

REliable, INtelligent & Scalable Systems

- **MovieLens User Ratings**
  - First, create a table with tab-delimited text file format:
  - ```
    CREATE TABLE u_data (
     userid INT,
     movieid INT,
     rating INT,
     unixtime STRING)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '\t'
    STORED AS TEXTFILE;
    ```
    ```

Simple Example Use Cases

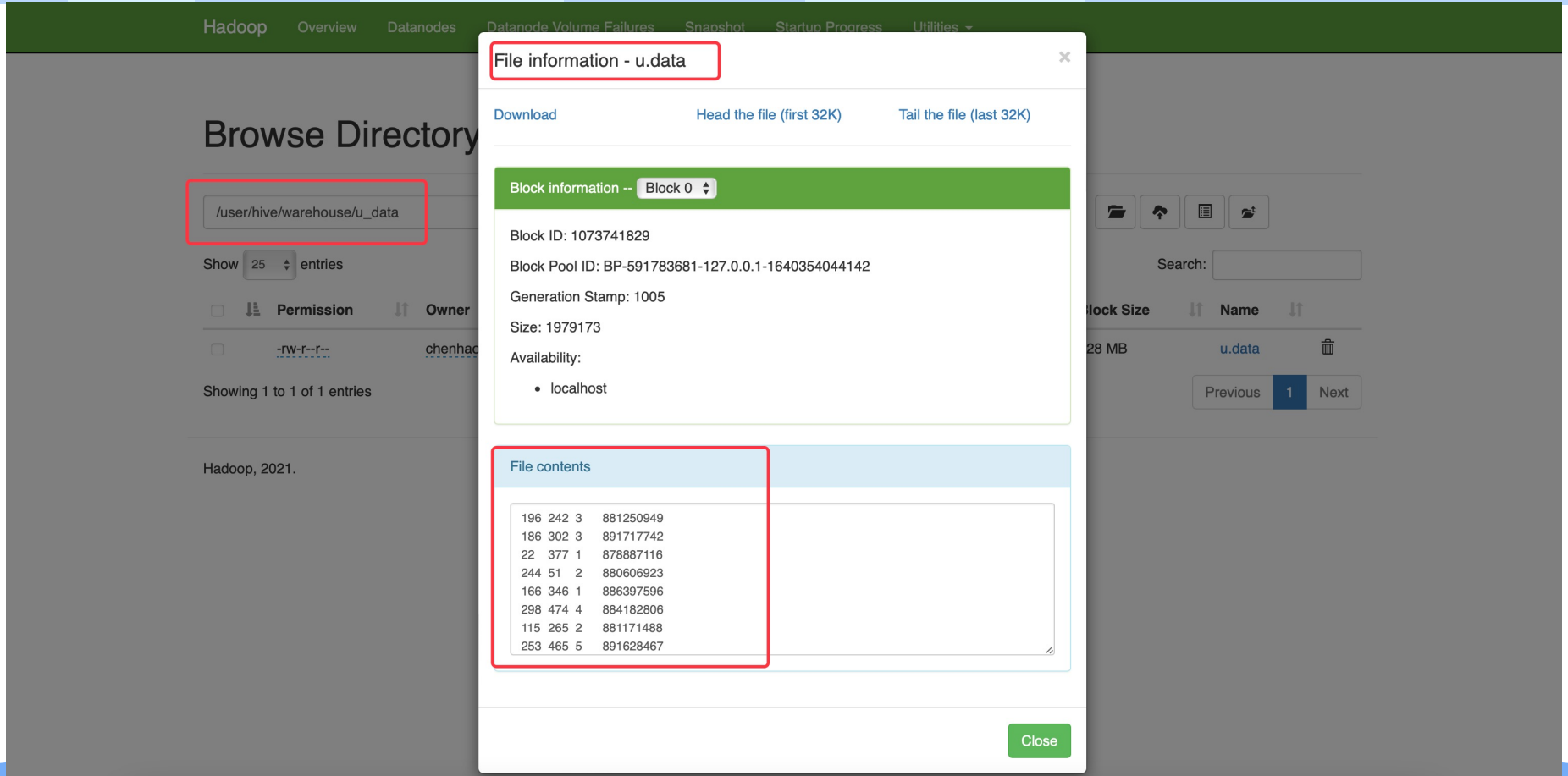
REliable, INtelligent & Scalable Systems

- **MovieLens User Ratings**
  - Then, download the data files from **MovieLens 100k** on the GroupLens datasets page (which also has a README.txt file and index of unzipped files):
  - `wget http://files.grouplens.org/datasets/movielens/ml-100k.zip`
  - or:
  - `curl --remote-name http://files.grouplens.org/datasets/movielens/ml-100k.zip`

  - Unzip the data files:
  - `unzip ml-100k.zip`
  - And load u.data into the table that was just created:
  - `LOAD DATA LOCAL INPATH '<path>/u.data' OVERWRITE INTO TABLE u_data;`

Simple Example Use Cases

REin
REliable, INtelligent & Scalable Systems



Simple Example Use Cases

- Count the number of rows in table u_data:

```
SELECT COUNT(*) FROM u_data;
```

```
hive> SELECT COUNT(*) FROM u_data;
Query ID = chenhaopeng_20211225212223_69a61947-9a84-455e-a009-85a50cbcb817
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2021-12-25 21:22:24,783 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1383216318_0004
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 10286786 HDFS Write: 2381094 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
100000
Time taken: 1.58 seconds, Fetched: 1 row(s)
hive>
```

Simple Example Use Cases

- Now we can do some complex data analysis on the table u_data:
- Create weekday_mapper.py:

```
import sys
import datetime
for line in sys.stdin:
 line = line.strip()
 userid, movieid, rating, unixtime = line.split('\t’)
 weekday = datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()
 print ('\t'.join([userid, movieid, rating, str(weekday)]))
```

Simple Example Use Cases

- Use the mapper script:

```
CREATE TABLE u_data_new (
 userid INT,
 movieid INT,
 rating INT,
 weekday INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';

add FILE weekday_mapper.py;

INSERT OVERWRITE TABLE u_data_new
SELECT
 TRANSFORM (userid, movieid, rating, unixtime)
 USING 'python weekday_mapper.py'
 AS (userid, movieid, rating, weekday)
FROM u_data;

SELECT weekday, COUNT(*)
FROM u_data_new
GROUP BY weekday;
```

Simple Example Use Cases

REliable, INtelligent & Scalable Systems

apache-hive-3.1.2-bin — java -Dproc_jar -Djava.library.path=/Users/chen...

```
hive>
    > SELECT weekday, COUNT(*)
    > FROM u_data_new
    > GROUP BY weekday;
Query ID = chenhaopeng_20211224222723_d9018a35-1b4d-47f9-abd8-e9b333b9ccd0
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2021-12-24 22:27:26,526 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1474112274_0002
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 6316858 HDFS Write: 2358512 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
1       12254
2       13579
3       14430
4       15114
5       14743
6       18229
7       11651
Time taken: 2.97 seconds, Fetched: 7 row(s)
```

An Example

- Let's see how to use Hive to run a query on the weather dataset.

- The first step is to load the data into Hive's managed storage.

- Just like an RDBMS, Hive organizes its data into tables. We create a table to hold the weather data using the CREATE TABLE statement:

```
CREATE TABLE records
        (year STRING, temperature INT, quality INT)
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '\t';
```

An Example

- Next we can populate Hive with the data.
  - This is just a small sample, for exploratory purposes:

```
LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'
OVERWRITE INTO TABLE records;
```

- Running this command tells Hive to put the specified local file in its warehouse directory.
  - There is no attempt, for example, to parse the file and store it in an internal database format, since Hive does not mandate any particular file format.
  - Files are stored verbatim: they are not modified by Hive.

An Example

- In this example, we are storing Hive tables on the local filesystem (fs.default.name is set to its default value of file:///).
  - Tables are stored as directories under Hive's warehouse directory, which is controlled by the hive.metastore.warehouse.dir, and defaults to /user/hive/warehouse.
  - Thus, the files for the records table are found in the /user/hive/warehouse/records directory on the local filesystem:

    % ls /user/hive/warehouse/record/sample.txt

- In this case, there is only one file, sample.txt, but in general there can be more, and Hive will read all of them when querying the table.

An Example

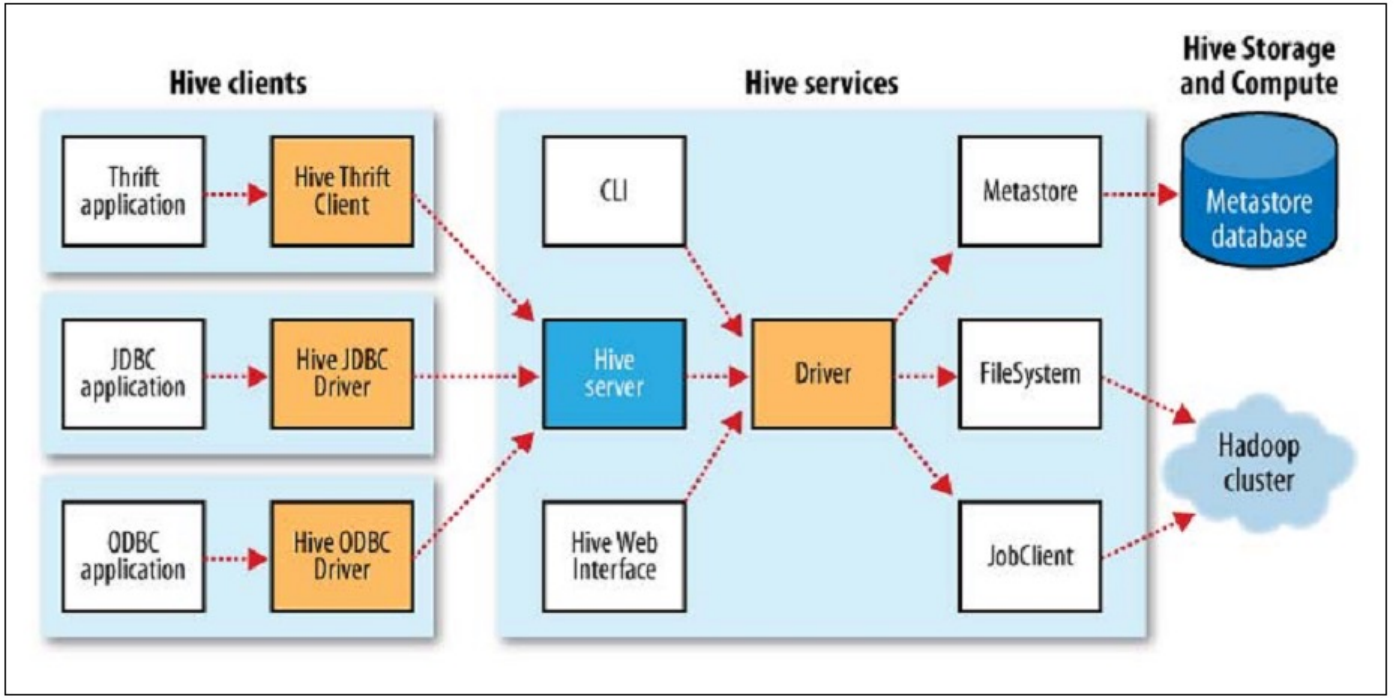
- Now that the data is in Hive, we can run a query against it:

```
hive> SELECT year, MAX(temperature)
    > FROM records
    > WHERE temperature != 9999
    > AND (quality = 0 OR quality = 1 OR quality = 4 OR quality = 5 OR quality = 9)
    > GROUP BY year;
1949 111
1950 22
```

Hive Services

- The Hive shell is only one of several services that you can run using the hive command.
- Type hive –service help to get a list of available service names; the most useful are described below.
  - cli
    - The command line interface to Hive (the shell). This is the default service.
  - hiveserver
    - Runs Hive as a server exposing a Thrift service, enabling access from a range of clients written in different languages.
  - hwi
    - The Hive Web Interface.
  - jar
    - The Hive equivalent to hadoop jar, a convenient way to run Java applications that includes both Hadoop and Hive classes on the classpath.
  - metastore
    - By default, the metastore is run in the same process as the Hive service. Using this service, it is possible to run the metastore as a standalone (remote) process.

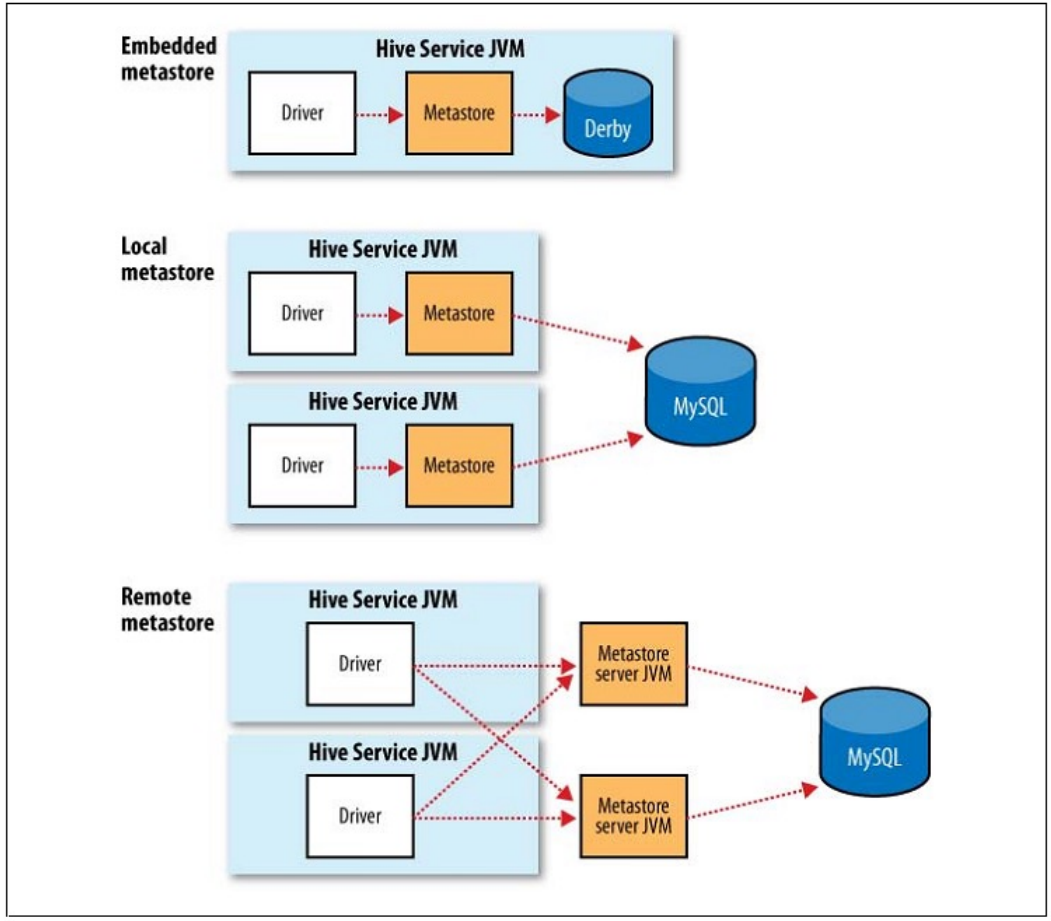
Hive clients



The Metastore

- The metastore is the central repository of Hive metadata.
  - The metastore is divided into two pieces: a service and the backing store for the data.
  - By default, the metastore service runs in the same JVM as the Hive service and contains an embedded Derby database instance backed by the local disk.
  - It also supports multiple sessions (and therefore multiple users) is to use a standalone database. This configuration is referred to as a local metastore, since the metastore service still runs in the same process as the Hive service, but connects to a database running in a separate process, either on the same machine or on a remote machine.
  - Going a step further, there's another metastore configuration called a remote metastore, where one or more metastore servers run in separate processes to the Hive service.

The Metastore



Comparison with Traditional Databases

- Schema on Read Versus Schema on Write
  - In a traditional database, a table's schema is enforced at data load time. If the data being loaded doesn't conform to the schema, then it is rejected.
    - This design is sometimes called schema on write, since the data is checked against the schema when it is written into the database.
  - Hive, on the other hand, doesn't verify the data when it is loaded, but rather when a query is issued.
    - This is called schema on read.
  - There are trade-offs between the two approaches.
    - Schema on read makes for a very fast initial load, since the data does not have to be read, parsed, and serialized to disk in the database's internal format.
    - Schema on write makes query time performance faster, since the database can index columns and perform compression on the data.

- Hive's SQL dialect, called HiveQL, does not support the full SQL-92 specification.

Feature	SQL	HiveQL	References
Updates	UPDATE, INSERT, DELETE	INSERT OVERWRITE TABLE (populates whole table or partition)	"INSERT OVERWRITE TABLE" on page 392, "Updates, Transactions, and Indexes" on page 376
Transactions	Supported	Not supported	
Indexes	Supported	Not supported	
Latency	Sub-second	Minutes	
Data types	Integral, floating point, fixed point, text and binary strings, temporal	Integral, floating point, boolean, string, array, map, struct	"Data Types" on page 378
Functions	Hundreds of built-in functions	Dozens of built-in functions	"Operators and Functions" on page 380
Multitable inserts	Not supported	Supported	"Multitable insert" on page 393
Create table as select	Not valid SQL-92, but found in some databases	Supported	"CREATE TABLE...AS SELECT" on page 394

- Hive's SQL dialect, called HiveQL, does not support the full SQL-92 specification.

Feature	SQL	HiveQL	References
Select	SQL-92	Single table or view in the FROM clause. SORT BY for partial ordering. LIMIT to limit number of rows returned. HAVING not supported.	"Querying Data" on page 395
Joins	SQL-92 or variants (join tables in the FROM clause, join condition in the WHERE clause)	Inner joins, outer joins, semi joins, map joins. SQL-92 syntax, with hinting.	"Joins" on page 397
Subqueries	In any clause. Correlated or noncorrelated.	Only in the FROM clause. Correlated subqueries not supported	"Subqueries" on page 400
Views	Updatable. Materialized or nonmaterialized.	Read-only. Materialized views not supported	"Views" on page 401
Extension points	User-defined functions. Stored procedures.	User-defined functions. MapReduce scripts.	"User-Defined Functions" on page 402, "MapReduce Scripts" on page 396

Tables

- A Hive table is logically made up of the data being stored and the associated metadata describing the layout of the data in the table.
  - The data typically resides in HDFS, although it may reside in any Hadoop filesystem, including the local filesystem or S3.
  - Hive stores the metadata in a relational database—and not in HDFS.

- Managed Tables and External Tables
  - When you create a table in Hive, by default Hive will manage the data, which means that Hive moves the data into its warehouse directory.
  - Alternatively, you may create an external table, which tells Hive to refer to the data that is at an existing location outside the warehouse directory.

- Partitions and Buckets
  - Hive organizes tables into partitions, a way of dividing a table into coarse-grained parts based on the value of a partition column.
  - Partitions are defined at table creation time using the PARTITIONED BY clause, which takes a list of column definitions.
    - For the hypothetical log files example, we might define a table with records comprising a timestamp and the log line itself:

```
CREATE TABLE logs (ts BIGINT, line STRING)
PARTITIONED BY (dt STRING, country STRING);
```

  - When we load data into a partitioned table, the partition values are specified explicitly:

```
LOAD DATA LOCAL INPATH 'input/hive/partitions/file1'
INTO TABLE logs
PARTITION (dt='2001-01-01', country='GB');
```

- Partitions and Buckets
  - Tables or partitions may further be subdivided into buckets, to give extra structure to the data that may be used for more efficient queries.
  - There are two reasons why you might want to organize your tables (or partitions) into buckets.
    - The first is to enable more efficient queries. Bucketing imposes extra structure on the table, which Hive can take advantage of when performing certain queries.
    - The second reason to bucket a table is to make sampling more efficient. When working with large datasets, it is very convenient to try out queries on a fraction of your dataset while you are in the process of developing or refining them.

```
CREATE TABLE bucketed_users (id INT, name STRING)
CLUSTERED BY (id) SORTED BY (id ASC)INTO 4 BUCKETS;
```

- Sorting and Aggregating

```
hive> FROM records2
    > SELECT year, temperature
    > DISTRIBUTE BY year
    > SORT BY year ASC, temperature DESC;
1949 111
1949 78
1950 22
1950 0
1950 -11
```

Querying Data

- MapReduce Scripts
  - Using an approach like Hadoop Streaming, the TRANSFORM, MAP, and REDUCE clauses make it possible to invoke an external script or program from Hive.

```
FROM (
  FROM records2
  MAP year, temperature, quality
  USING 'is_good_quality.py'
  AS year, temperature) map_output
REDUCE year, temperature
USING 'max_temperature_reduce.py'
AS year, temperature;
```

File Formats and Compression

- Hive supports several file formats:
  - Text File
  - SequenceFile
  - RCFile
  - Avro Files
  - ORC Files
  - Parquet
  - Custom INPUTFORMAT and OUTPUTFORMAT
- The hive.default.fileformat configuration parameter
  - determines the format to use if it is not specified in a CREATE TABLE or ALTER TABLE statement.
  - Text file is the parameter's default value.

File Formats and Compression

- RCFile
  - RCFile (Record Columnar File) is a data placement structure designed for MapReduce-based data warehouse systems. Hive added the RCFile format in version 0.6.0.
  - RCFile stores table data in a flat file consisting of binary key/value pairs. It first partitions rows horizontally into row splits, and then it vertically partitions each row split in a columnar way.
  - RCFile stores the metadata of a row split as the key part of a record, and all the data of a row split as the value part.
  - RCFile combines the advantages of both row-store and column-store to satisfy the need for fast data loading and query processing, efficient use of storage space, and adaptability to highly dynamic workload patterns.
  - As row-store, RCFile guarantees that data in the same row are located in the same node.
  - As column-store, RCFile can exploit column-wise data compression and skip unnecessary column reads.

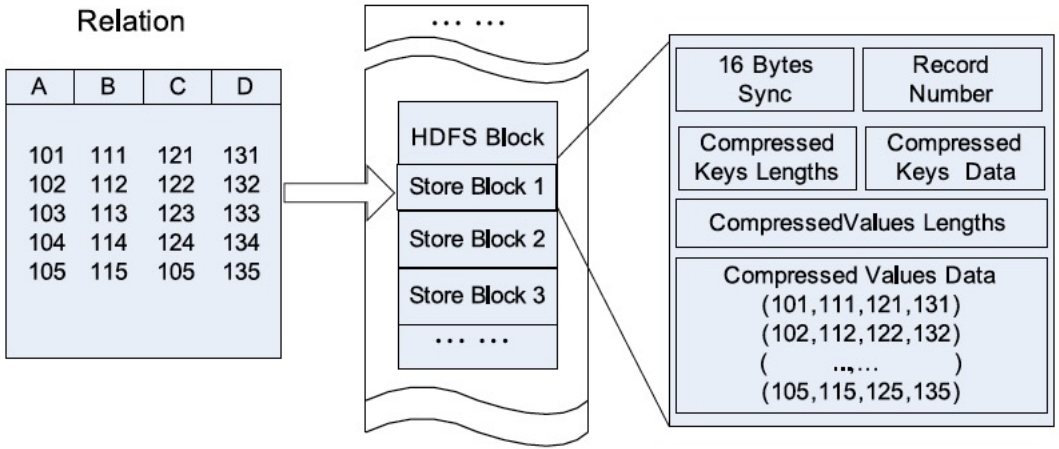


Fig. 1: An example of row-store in an HDFS block.

- 2011 ICDE conference paper "RCFile: A Fast and Space-efficient Data Placement Structure in MapReduce-based Warehouse Systems" by Yongqiang He, Rubao Lee, Yin Huai, Zheng Shao, Namit Jain, Xiaodong Zhang, and Zhiwei Xu

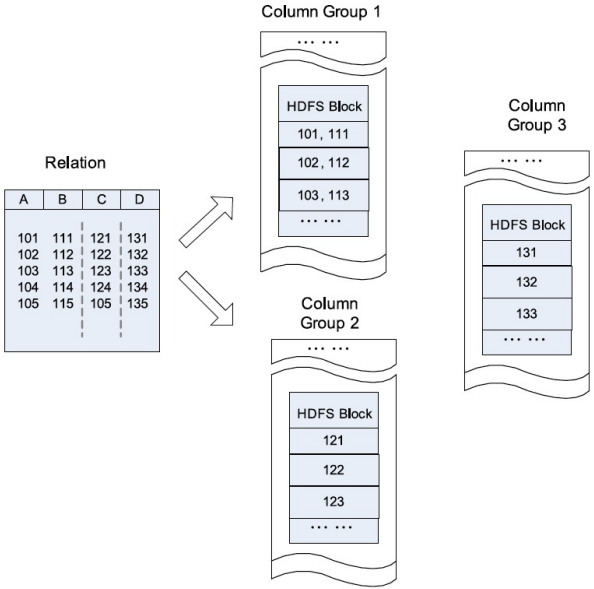


Fig. 2: An example of column-group in an HDFS block. The four columns are stored into three column groups, since column A and B are grouped in the first column group.

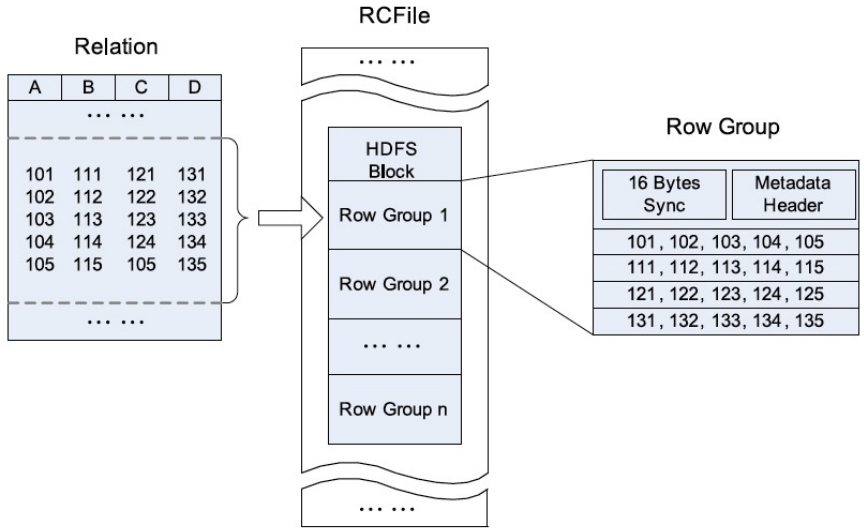


Fig. 3: An example to demonstrate the data layout of RCFile in an HDFS block.

- 2011 ICDE conference paper "RCFile: A Fast and Space-efficient Data Placement Structure in MapReduce-based Warehouse Systems" by Yongqiang He, Rubao Lee, Yin Huai, Zheng Shao, Namit Jain, Xiaodong Zhang, and Zhiwei Xu

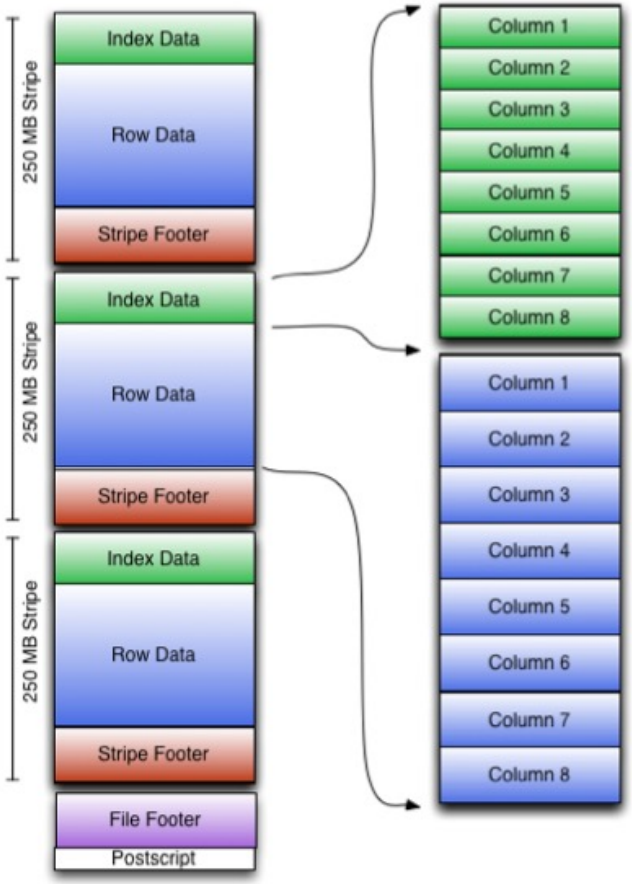
ORCFile

- The *Optimized Row Columnar* (ORC) file format provides a highly efficient way to store Hive data.

- An ORC file contains groups of row data called **stripes**, along with auxiliary information in a **file footer**.
  - At the end of the file a **postscript** holds compression parameters and the size of the compressed footer.
  - The default stripe size is 250 MB. Large stripe sizes enable large, efficient reads from HDFS.
  - The file footer contains a list of stripes in the file, the number of rows per stripe, and each column's data type. It also contains column-level aggregates count, min, max, and sum.

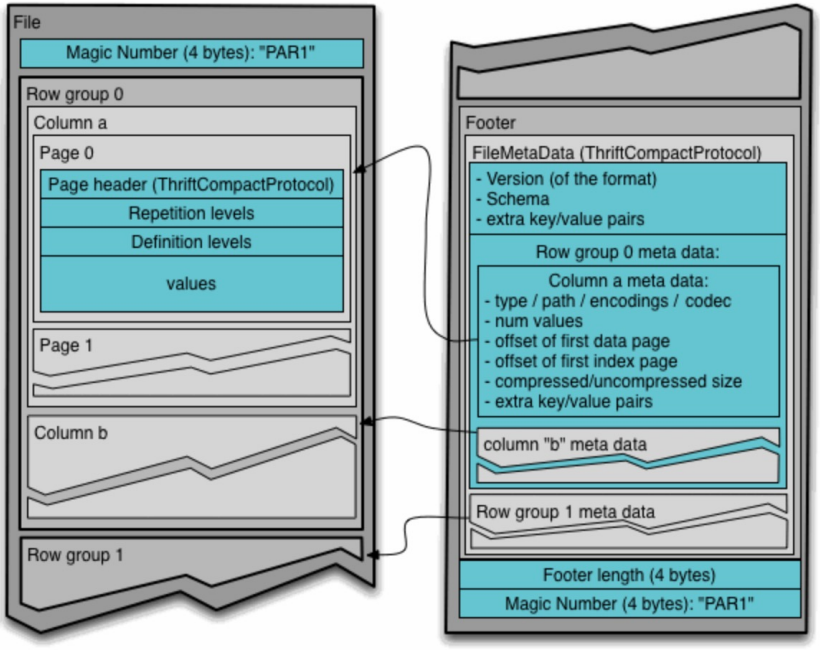
ORCFile

- File Structure



Parquet

- Apache Parquet
  - is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.
  - http://parquet.incubator.apache.org/



File Compression

- Keeping data compressed in Hive tables has, in some cases,
  - been known to give better performance than uncompressed storage; both in terms of disk usage and query performance.
- You can import text files compressed with Gzip or Bzip2 directly into a table stored as TextFile.
  - The compression will be detected automatically and the file will be decompressed on-the-fly during query execution. For example:

```
CREATE TABLE raw (line STRING)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n';

LOAD DATA LOCAL INPATH '/tmp/weblogs/20090603-access.log.gz' INTO TABLE raw;
```

- The table 'raw' is stored as a TextFile, which is the default storage.
  - However, in this case Hadoop will not be able to split your file into chunks/blocks and run multiple maps in parallel. This can cause underutilization of your cluster's 'mapping' power.

File Compression

- The recommended practice is to insert data into another table, which is stored as a SequenceFile.
  - A SequenceFile can be split by Hadoop and distributed across map jobs whereas a GZIP file cannot be. For example:

```
CREATE TABLE raw (line STRING)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n';

CREATE TABLE raw_sequence (line STRING)
   STORED AS SEQUENCEFILE;

LOAD DATA LOCAL INPATH '/tmp/weblogs/20090603-access.log.gz' INTO TABLE raw;

SET hive.exec.compress.output=true;
SET io.seqfile.compression.type=BLOCK; -- NONE/RECORD/BLOCK (see below)
INSERT OVERWRITE TABLE raw_sequence SELECT * FROM raw;
```

  - The value for io.seqfile.compression.type determines how the compression is performed.
  - Record compresses each value individually while BLOCK buffers up 1MB (default) before doing compression.

LZO Compression

- LZO
  - is a lossless data compression library that favors speed over compression ratio.
    See http://www.oberhumer.com/opensource/lzo and http://www.lzop.org for general information about LZO.
- Imagine a simple data file that has three columns
  - id
  - first name
  - last name
- Let's populate a data file containing 4 records:

```
19630001 john lennon
19630002 paul mccartney
19630003 george harrison
19630004 ringo starr
```

LZO Compression

- Let's call the data file /path/to/dir/names.txt.
  - In order to make it into an LZO file, we can use the lzop utility and it will create a names.txt.lzo file.
  - Now copy the file names.txt.lzo to HDFS.

- **Hive Queries**
- **Option 1: Directly Create LZO Files**
  1. Directly create LZO files as the output of the Hive query.
  2. Use lzop command utility or your custom Java to generate .lzo.index for the .lzo files.
- **Hive Query Parameters**

```
SET mapreduce.output.fileoutputformat.compress.codec=com.hadoop.compression.lzo.LzoCodec
SET hive.exec.compress.output=true
SET mapreduce.output.fileoutputformat.compress=true
```

- For example:

```
hive -e "SET
mapreduce.output.fileoutputformat.compress.codec=com.hadoop.compression.lzo.LzoCodec;
SET hive.exec.compress.output=true;SET mapreduce.output.fileoutputformat.compress=true;
<query-string>"
```

LZO Compression

- **Hive Queries**
- **Option 2: Write Custom Java to Create LZO Files**
  1. Create text files as the output of the Hive query.
  2. Write custom Java code to
     a. convert Hive query generated text files to .lzo files
     b. generate .lzo.index files for the .lzo files generated above
- **Hive Query Parameters**

```
SET hive.exec.compress.output=false
SET mapreduce.output.fileoutputformat.compress=false
```

- For example:

```
hive -e "SET hive.exec.compress.output=false;SET
mapreduce.output.fileoutputformat.compress=false;<query-string>"
```

References

- Apache Hive
  - https://hive.apache.org
- hadoop： mkdir: 'input': No such file or directory 相关问题
  - https://blog.csdn.net/ludonqin/article/details/51396187?depth_1-utm_source=distribute.pc_relevant.none-task&utm_source=distribute.pc_relevant.none-task
- 启动hive报错：java.lang.NoSuchMethodError: com.google.common.base.Preconditions.checkArgument(ZLjava/lang/String;Ljava/lang/Object;)V（已解决）
- Hadoop: The Definitive Guide
  - By Tom White
  - O'Reilly Publishing

Thank You!