Big Data Systems (CS4545/CS6545)
Winter 2021

Declarative interface on top of batch processing: HiveQL

Suprio Ray

University of New Brunswick, Fredericton

Acknowledgement

Thanks to Tom White (textbook), Raghav Ayyamani and L. Tang for some of the materials in these slides. Also thanks to various articles and research papers.

Outline

- Hive ⇐
  - Data model
  - System architecture
  - HiveQL
  - File formats

- File-based Data Structures
  - Row-oriented storage formats
  - Column-oriented storage formats

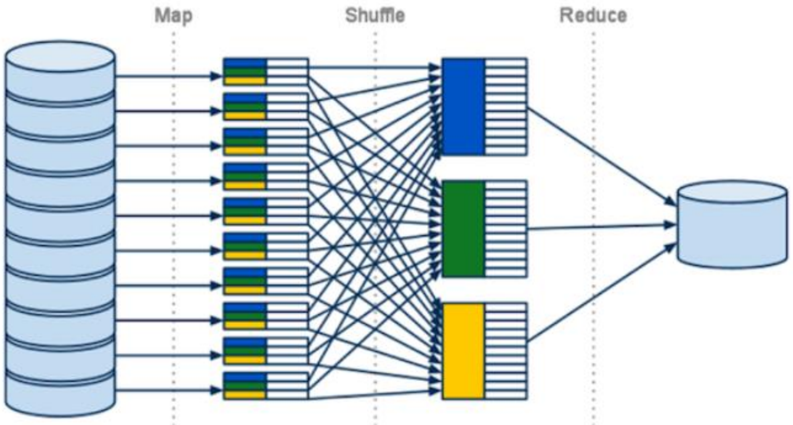
Why Another Data Warehousing System?

- Problem : Data, data and more data
  - Several TBs of data everyday
- The Hadoop Experience:
  - Uses Hadoop File System (HDFS)
  - Scalable/Available

Src: datanami.com

Limitations of MapReduce

- Lacks expressiveness



- Map-Reduce can be somewhat hard to program
- For complex jobs:
  - Multiple stage of Map/Reduce functions
  - Just like asking developer to write specific physical execution plan in the database
- Not Reusable
- Error prone

SQL on MapReduce?

- **Advantages of SQL**
  - SQL has a huge user base
  - SQL is easy to code

- **Solution: Combine SQL and Map-Reduce**
  - Hive on top of Hadoop (open source)

What is HIVE?

- A system for managing and querying unstructured data as if it were structured
  - Uses Map-Reduce for execution
  - HDFS for Storage

- Key Building Principles
  - SQL as a familiar data warehousing tool
  - Extensibility (Pluggable map/reduce scripts in the language of your choice, Rich and User Defined Data Types, User Defined Functions)
  - Interoperability (Extensible Framework to support different file and data formats)
  - Performance

Outline

- Hive
  - Data model
  - System architecture
  - HiveQL

- File-based Data Structures
  - Row-oriented storage formats
  - Column-oriented storage formats

Data Model - Tables

- Tables
  - Analogous to tables in relational DBs.
  - Each table has a corresponding directory in HDFS.
  - Creating a table

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

Data Model - Tables

- Tables
  - Analogous to tables in relational DBs.
  - Each table has a corresponding directory in HDFS.
  - Creating a table

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

  - Loading a table

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

Data Model - Tables

- Location: tables are stored as directories under Hive's warehouse
  - Default /user/hive/warehouse
  - Controlled by the *hive.metastore.warehouse.dir* property

```
$ … ls /user/hive/warehouse/records/
sample.txt
```

- Query: once loaded, we can run a query against it:

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

Managed vs. External Tables

- Managed Tables
  - Default option when a table is created
  - Hive moves the data into its warehouse directory

```
CREATE TABLE managed_table (dummy STRING);

LOAD DATA INPATH '/user/tom/data.txt' INTO table managed_table;
```

  - Drop table semantics: the table, including its metadata *and its data*, is deleted

```
DROP TABLE managed_table;
```

Managed vs. External Tables

- External Tables
  - Point to existing data directories in HDFS
  - Data is assumed to be in Hive-compatible format

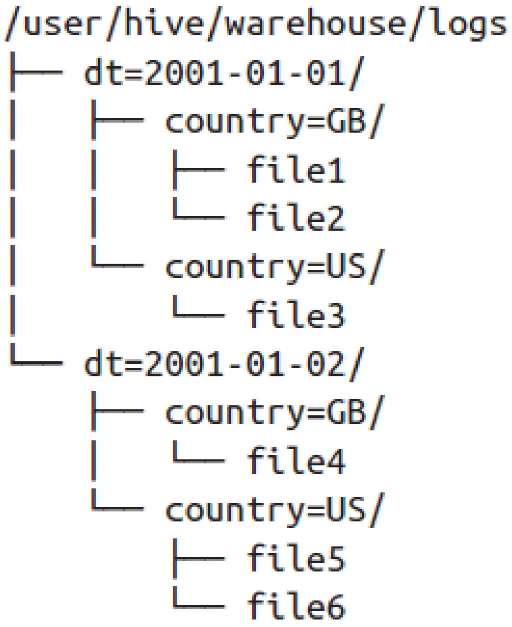
```
CREATE EXTERNAL TABLE external_table (dummy STRING)
LOCATION '/user/tom/external_table';

LOAD DATA INPATH '/user/tom/data.txt' INTO TABLE external_table;
```

  - Drop table semantics: drops only the metadata, leave the data untouched

Table - Partitions

- Partitions
  - A way of dividing a table into coarse-grained parts
  - Can make it faster to do queries on slices of the data
  - A table may be partitioned in multiple dimensions
  - Example
    - Table: a log file; Partition columns: (datestamp, country)



```
/user/hive/warehouse/logs
├── dt=2001-01-01/
│   ├── country=GB/
│   │   ├── file1
│   │   └── file2
│   └── country=US/
│       └── file3
└── dt=2001-01-02/
    ├── country=GB/
    │   └── file4
    └── country=US/
        ├── file5
        └── file6
```

Table - Partitions

- Creating Partitions
  - Defined at table creation time using the PARTITIONED BY clause

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

  - While loading data, the partition values are specified explicitly:

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

Table - Partitions

- Ask Hive to show Partitions

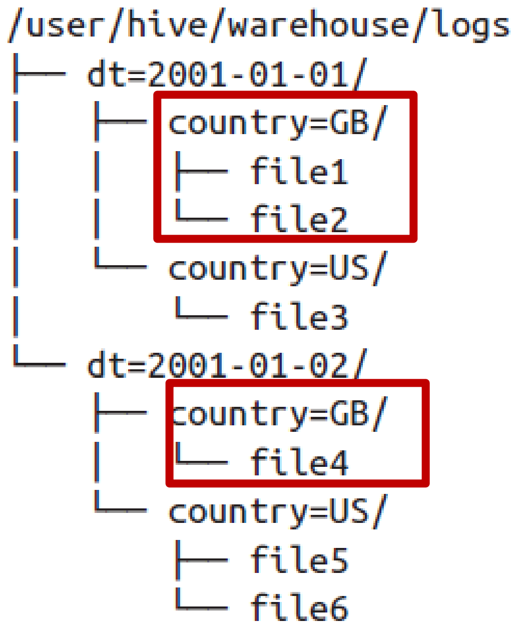
```
hive> SHOW PARTITIONS logs;

dt=2001-01-01/country=GB
dt=2001-01-01/country=US
dt=2001-01-02/country=GB
dt=2001-01-02/country=US
```

- Query on specific Partitions

```
SELECT ts, dt, line  FROM logs
 WHERE country='GB';
```

will only scan *file1*, *file2*, and *file4*



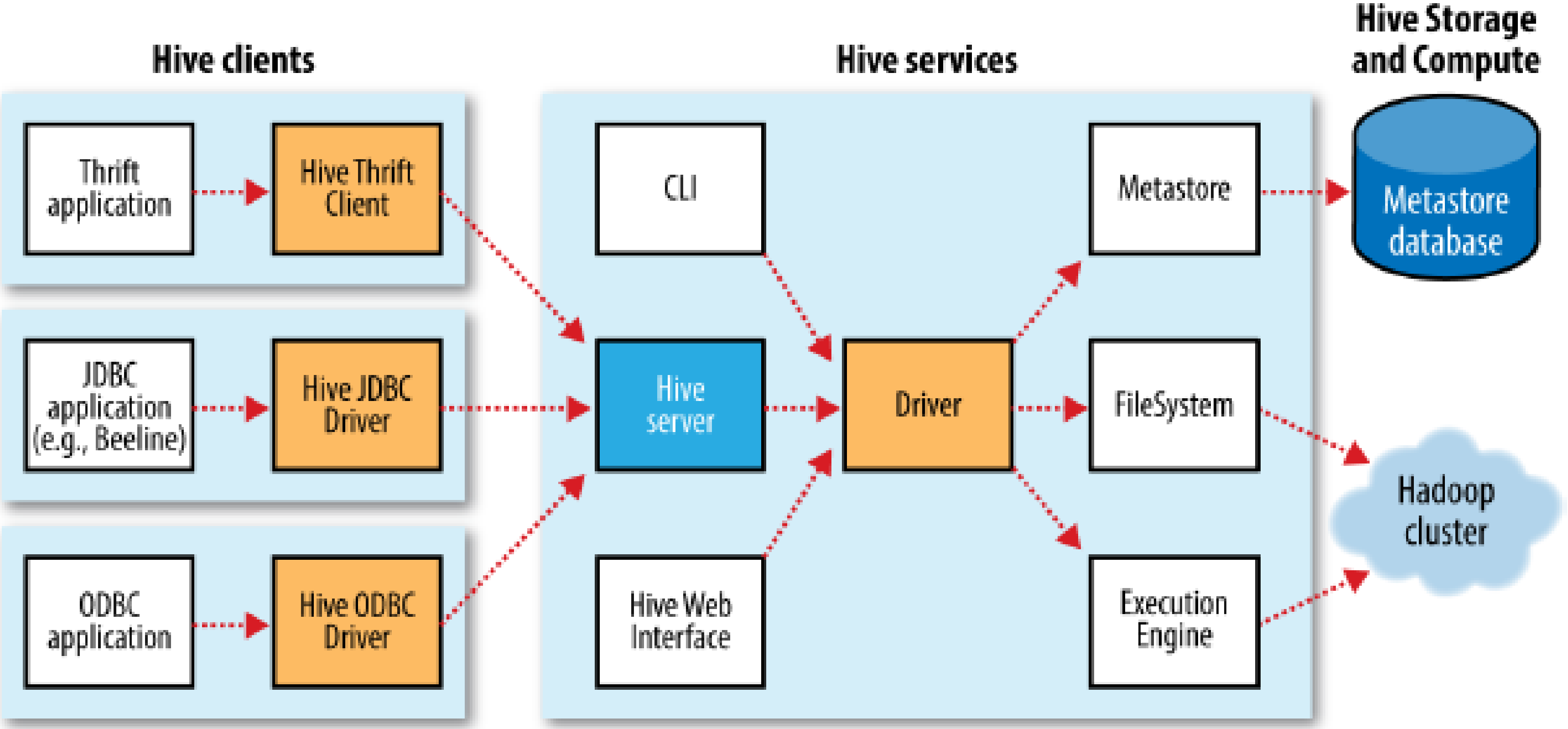
```
/user/hive/warehouse/logs
├── dt=2001-01-01/
│   ├── country=GB/
│   │   ├── file1
│   │   └── file2
│   └── country=US/
│       └── file3
└── dt=2001-01-02/
    ├── country=GB/
    │   └── file4
    └── country=US/
        ├── file5
        └── file6
```

Outline

- Hive
  - Data model
  - System architecture ⬅
  - HiveQL
  - File formats

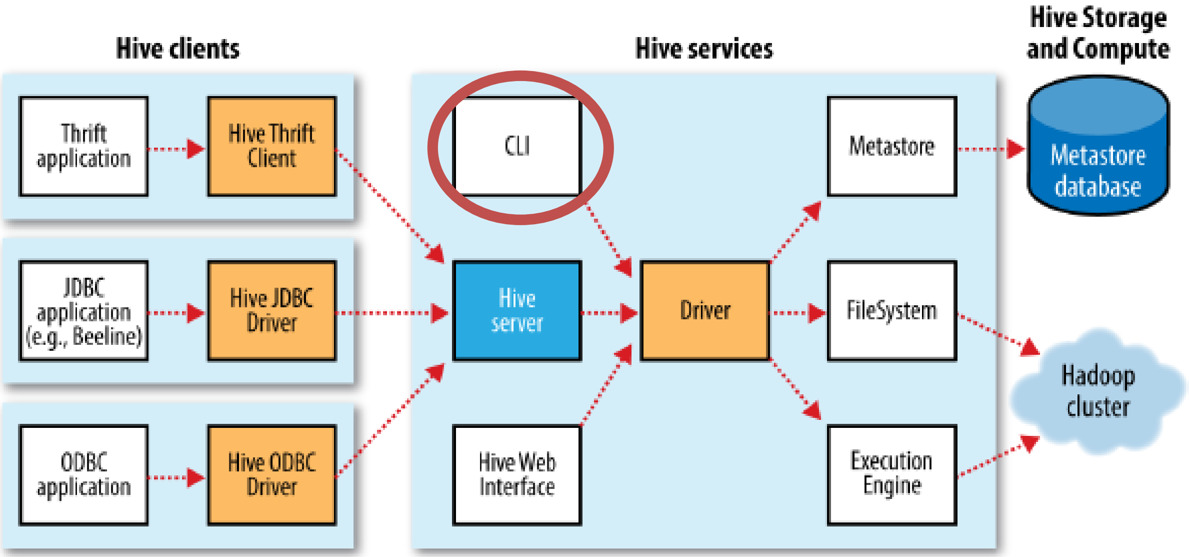
- File-based Data Structures
  - Row-oriented storage formats
  - Column-oriented storage formats

Hive Architecture



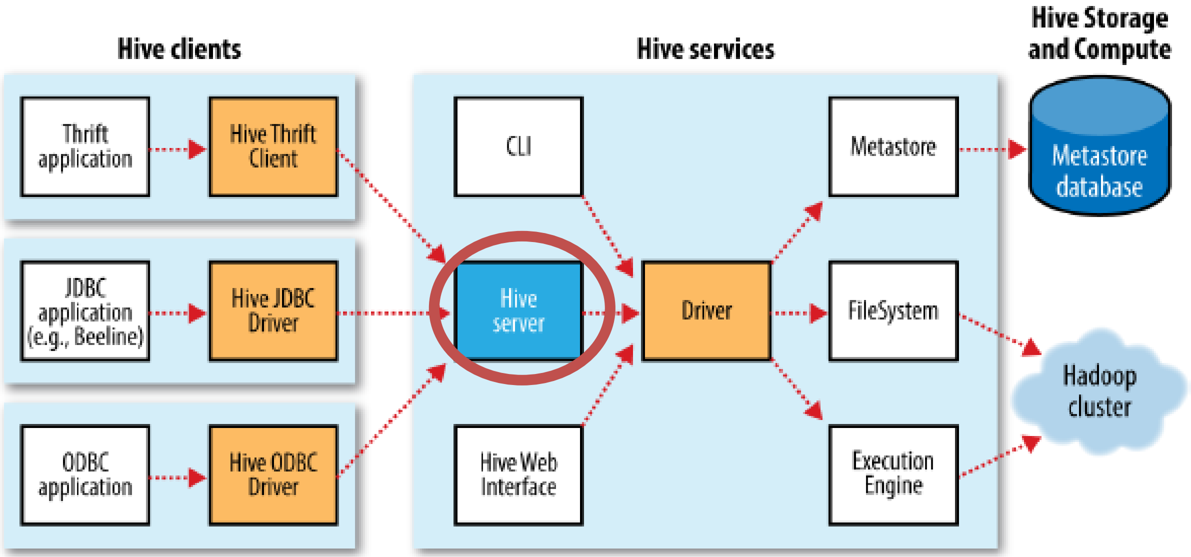
Hive Services

- cli
  - The command-line interface to Hive (the shell). This is the default service.



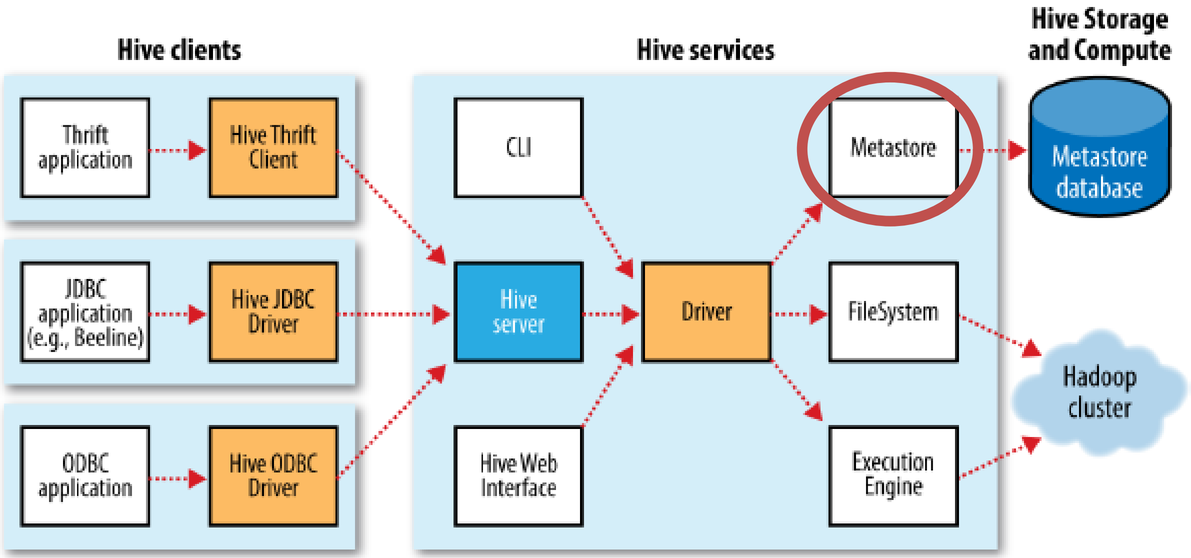
Hive Services

- hiveserver2
  - Runs Hive as a server, enabling access from a range of clients
  - Applications using the Thrift, JDBC, and ODBC connectors need to run a Hive server to communicate with Hive



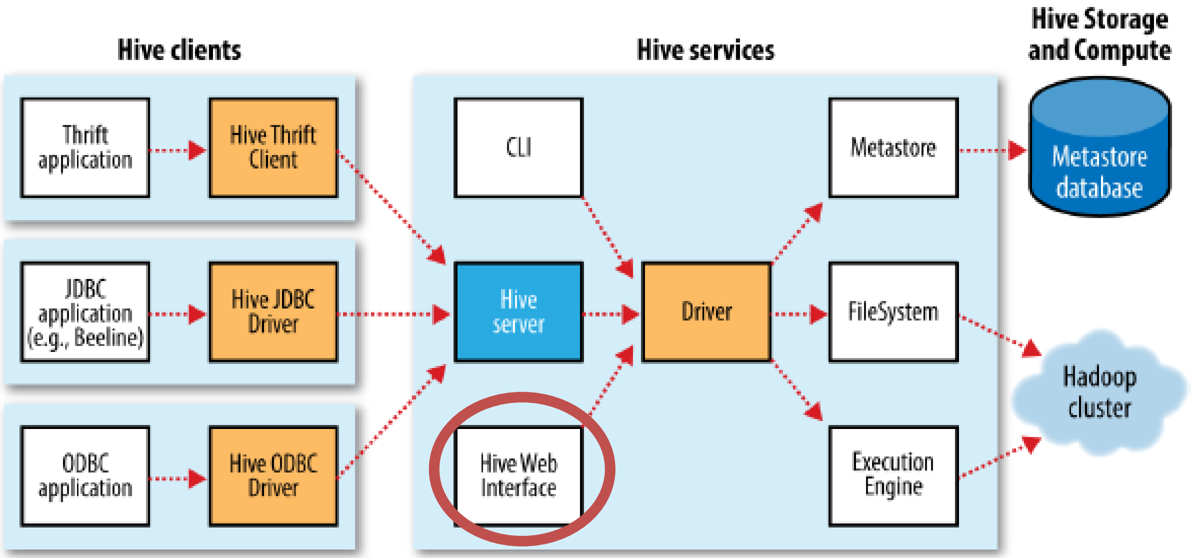
Hive Services

- metastore
  - The central repository of Hive metadata



Hive Services

- hwi
  - A simple web interface that can be used as an alternative to the CLI without having to install any client software

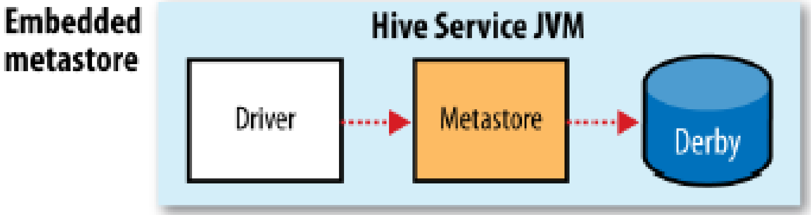


Hive Services

- beeline
  - A command-line interface to Hive that works in embedded mode, or by connecting to a hiveserver2 process using JDBC

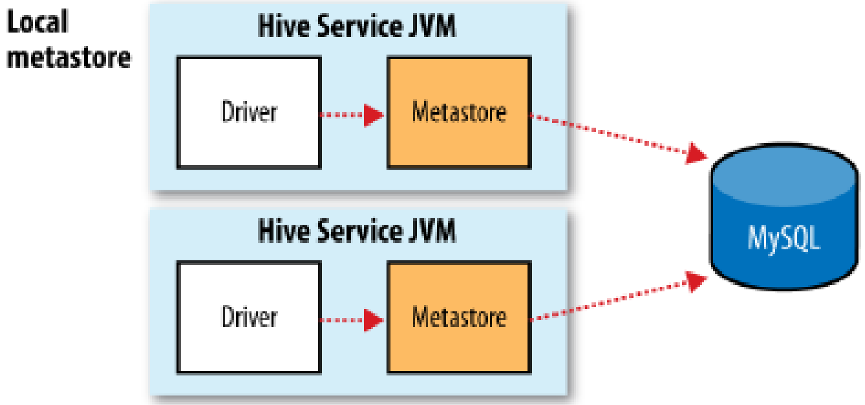
Metastore configurations

- Embedded metastore
  - The metastore service runs in the same JVM as the Hive service and contains an embedded Derby database instance backed by the local disk.
  - Can have only one Hive session open at a time that accesses the same metastore



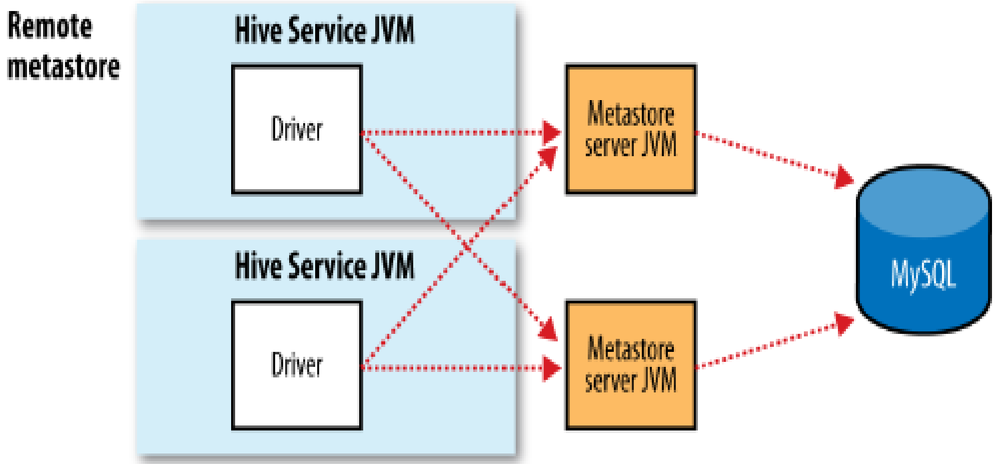
Metastore configurations

- Local metastore
  - Supports multiple sessions (and therefore multiple users)
  - Metastore service still runs in the same process as the Hive service, but connects to a database running in a separate process
  - MySQL is a popular choice



Metastore configurations

- Remote metastore
  - One or more metastore servers run in separate processes to the Hive service
  - Brings better manageability and security because the database tier can be completely firewalled off



Outline

- Hive
  - Data model
  - System architecture
  - HiveQL ⬅
  - File formats

- File-based Data Structures
  - Row-oriented storage formats
  - Column-oriented storage formats

Hive vs. traditional relational databases

	Hive	Relational databases
Schema on Read Versus Schema on Write	**Schema on read.** Does not verify the schema when the data is loaded, but rather when the a query is issued	**Schema on write.** 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.
In-place updates	**Not supported**. Changes resulting from inserts, updates, and deletes are stored in small delta files. They are periodically merged into the base table files by MapReduce jobs that are run in the background	**Supported**. Updates are reflected in real-time (mostly).
Index	There are currently two index types: *compact* and *bitmap*. (no longer supported since v3)	Various sophisticated indexes supported

Hive Query Language (HiveQL)

- Basic SQL
  - From clause sub-query
  - ANSI JOIN   (theta join supported since 0.14)
  - Multi group-by
  - Sampling
  - Objects Traversal
- Extensibility
  - Pluggable Map-reduce scripts using TRANSFORM

Type System

- Primitive types
  - Integers: TINYINT, SMALLINT, INT, BIGINT.
  - Boolean: BOOLEAN.
  - Floating point numbers: FLOAT, DOUBLE .
  - String: STRING.

- Complex types
  - Structs: {a INT; b INT}.
  - Maps: M['group'].
  - Arrays: ['a', 'b', 'c'], A[1] returns 'b'.

Complex type example

- Create table

```
create table tab11
 (id int,
  name string,
  sal bigint,
  sub array<string>,
  dud map<string,int>,
  addr struct<city:string,state:string,pin:bigint>
 )
 row format delimited
 fields terminated by ','
 collection items terminated by '$'
 map keys terminated by '#';
```

- Data file

```
1,abc,40000,a$b$c,pf#500$epf#200,hyd$ap$500001
2,def,3000,d$f,pf#500,bang$kar$600038
```

Hive Query Language (HiveQL) - contd.

- Insertion

```
INSERT OVERWRITE TABLE sample1 '/tmp/hdfs_out' SELECT * FROM sample
    WHERE ds='2012-02-24';
```

```
INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT * FROM sample
    WHERE ds='2012-02-24';
```

```
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/hive-sample-out' SELECT *
    FROM sample;
```

HiveQL – Join

- Hive equi-join

page_view

pageid	**userid**	time
1	**111**	9:08:01
2	**111**	9:08:13
1	**222**	9:08:14

X

user

userid	age	gender
111	25	female
222	32	male

=

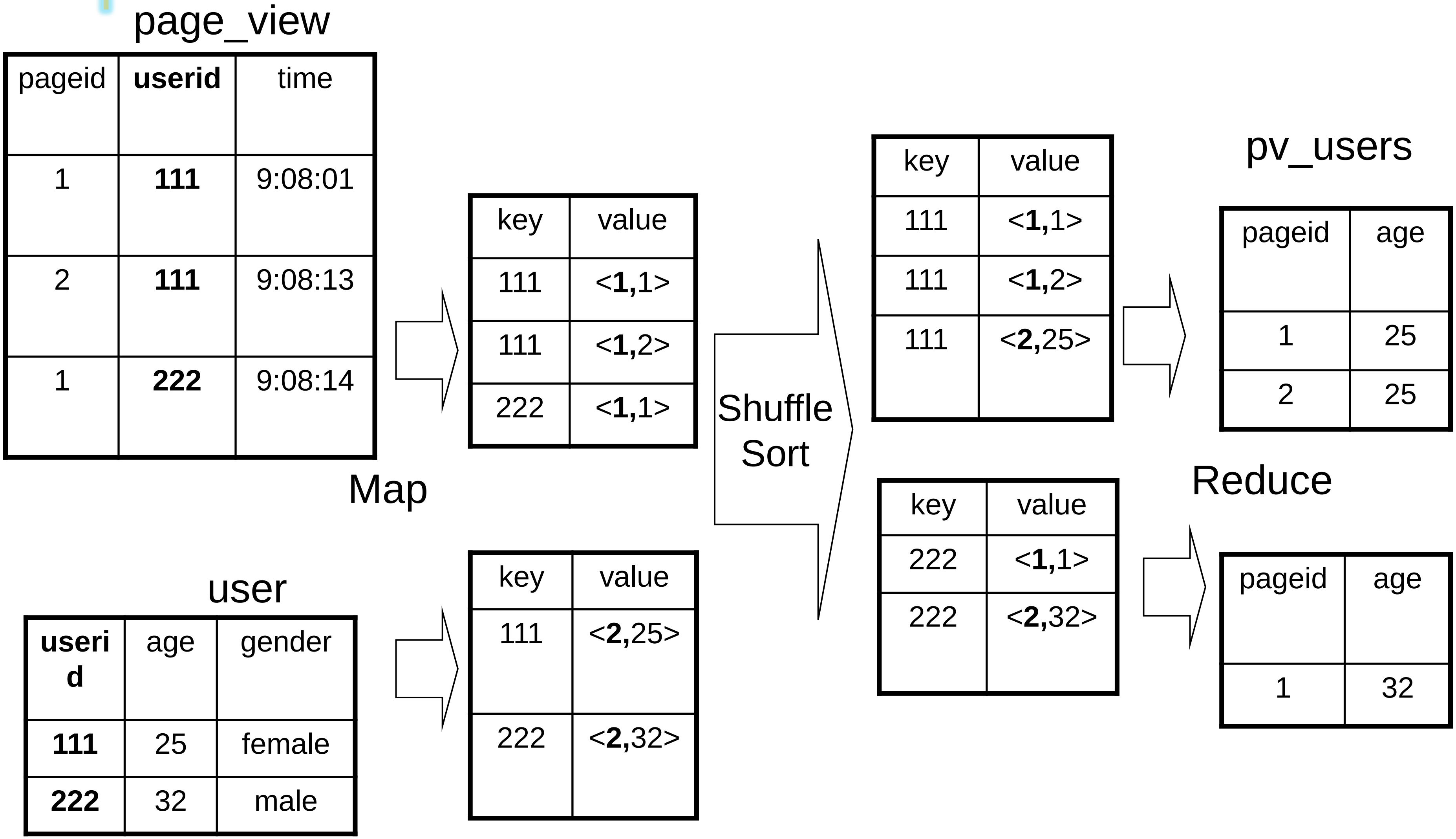
pv_users

pageid	age
1	25
2	25
1	32

- SQL:

```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv JOIN user u ON (pv.userid =
   u.userid);
```

HiveQL – Join in Map Reduce



page_view

pageid	**userid**	time
1	**111**	9:08:01
2	**111**	9:08:13
1	**222**	9:08:14

user

userid	age	gender
111	25	female
222	32	male

Map

key	value
111	<**1**,1>
111	<**1**,2>
222	<**1**,1>

key	value
111	<**2**,25>
222	<**2**,32>

Shuffle Sort

key	value
111	<**1**,1>
111	<**1**,2>
111	<**2**,25>

key	value
222	<**1**,1>
222	<**2**,32>

Reduce

pv_users

pageid	age
1	25
2	25

pageid	age
1	32

HiveQL – Group By

pv_users

pageid	age
1	25
2	25
1	32
2	25

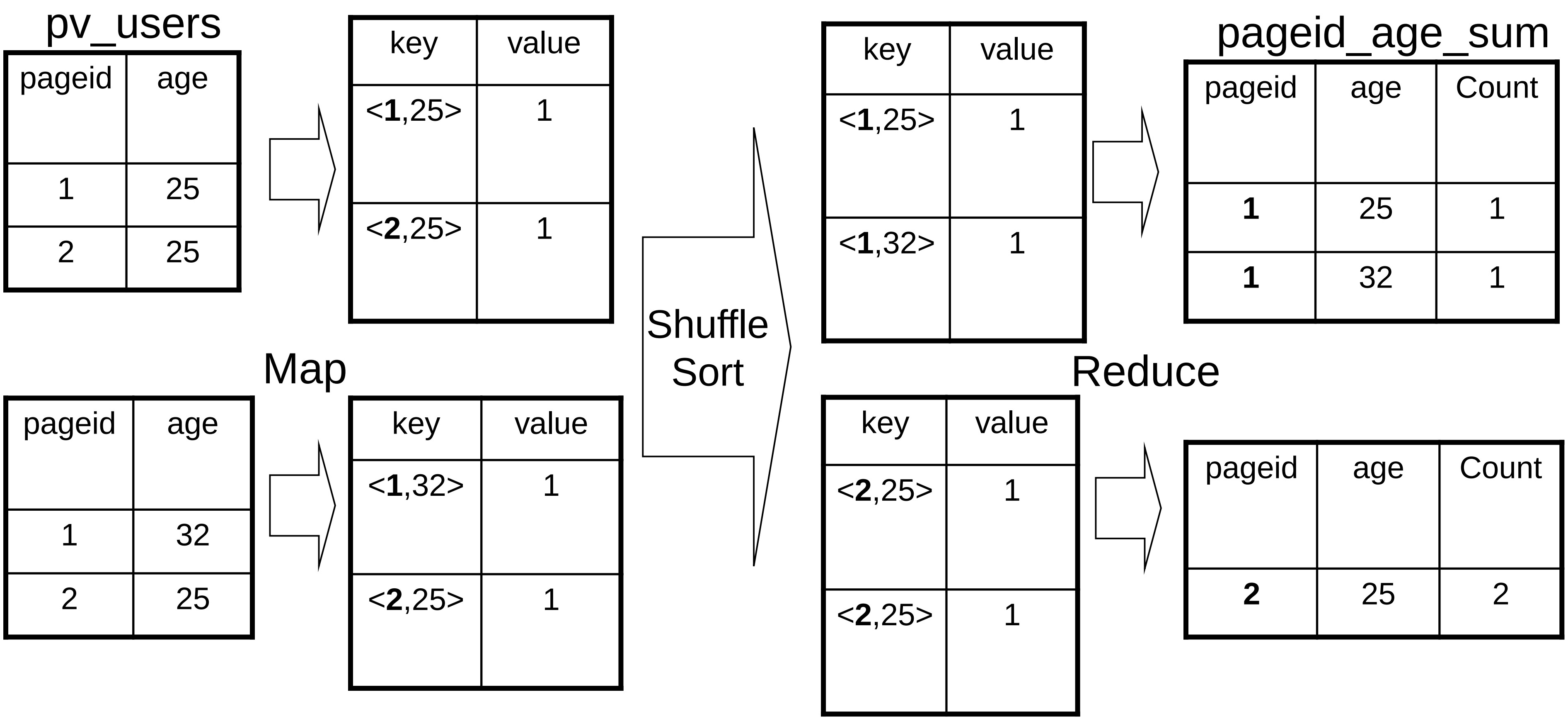
pageid_age_sum

pageid	age	Count
1	25	1
1	32	1
2	25	2

- SQL:

```
INSERT INTO TABLE pageid_age_sum
SELECT pageid, age, count(1)
FROM pv_users
GROUP BY pageid, age;
```

HiveQL – Group By in Map Reduce



pv_users

pageid	age
1	25
2	25

pageid	age
1	32
2	25

Map

key	value
<**1**,25>	1
<**2**,25>	1

key	value
<**1**,32>	1
<**2**,25>	1

Shuffle Sort

key	value
<**1**,25>	1
<**1**,32>	1

key	value
<**2**,25>	1
<**2**,25>	1

Reduce

pageid_age_sum

pageid	age	Count
1	25	1
1	32	1

pageid	age	Count
2	25	2