Hive User Group Meeting

August 2009

Hive Overview

Why Another Data Warehousing System?

Data, data and more data

200GB per day in March 2008

5+TB(compressed) raw data per day today

What is HIVE?

- A system for managing and querying structured data built on top of Hadoop
  - Map-Reduce for execution
  - HDFS for storage
  - Metadata on raw files

- Key Building Principles:
  - SQL as a familiar data warehousing tool
  - Extensibility – Types, Functions, Formats, Scripts
  - Scalability and Performance

Why SQL on Hadoop?

```
hive> select key, count(1) from kv1 where key > 100 group by
   key;
```

vs.

```
$ cat > /tmp/reducer.sh
uniq -c | awk '{print $2"\t"$1}`
$ cat > /tmp/map.sh
awk -F '\001' '{if($1 > 100) print $1}`
$ bin/hadoop jar contrib/hadoop-0.19.2-dev-streaming.jar -input
   /user/hive/warehouse/kv1 -mapper map.sh -file
   /tmp/reducer.sh -file /tmp/map.sh -reducer reducer.sh -
   output /tmp/largekey -numReduceTasks 1
$ bin/hadoop dfs -cat /tmp/largekey/part*
```

Data Model

- **Tables***
  - **Analogous to tables in relational DBs**
  - **Each table has corresponding directory in HDFS**
  - **Example**
    - **Page views table name: pvs**
    - **HDFS directory**
      - **`/wh/pvs`**

* **Databases** – supported in metastore, but not yet in query language

Data Model

- **Partitions**
  - **Analogous to dense indexes on partition columns**
  - **Nested sub-directories in HDFS for each combination of partition column values**
  - **Example**
    - **Partition columns: ds, ctry**
    - **HDFS subdirectory for ds = 20090801, ctry = US**
      - **`/wh/pvs/ds=20090801/ctry=US`**
    - **HDFS subdirectory for ds = 20090801, ctry = CA**
      - **`/wh/pvs/ds=20090801/ctry=CA`**

Data Model

- **Buckets**
  - **Split data based on hash of a column - mainly for parallelism**
  - **One HDFS file per bucket within partition sub-directory**
  - **Example**
    - **Bucket column: user into 32 buckets**
    - **HDFS file for user hash 0**
      - **`/wh/pvs/ds=20090801/ctry=US/part-00000`**
    - **HDFS file for user hash bucket 20**
      - **`/wh/pvs/ds=20090801/ctry=US/part-00020`**

Data Model

- » **External Tables**
  - › **Point to existing data directories in HDFS**
  - › **Can create tables and partitions – partition columns just become annotations to external directories**
  - › **Example: create external table with partitions**

```
CREATE EXTERNAL TABLE pvs(userid int, pageid int,
                          ds string, ctry string)
PARTITIONED ON (ds string, ctry string)
STORED AS textfile
LOCATION ‘/path/to/existing/table’
```

  - › **Example: add a partition to external table**

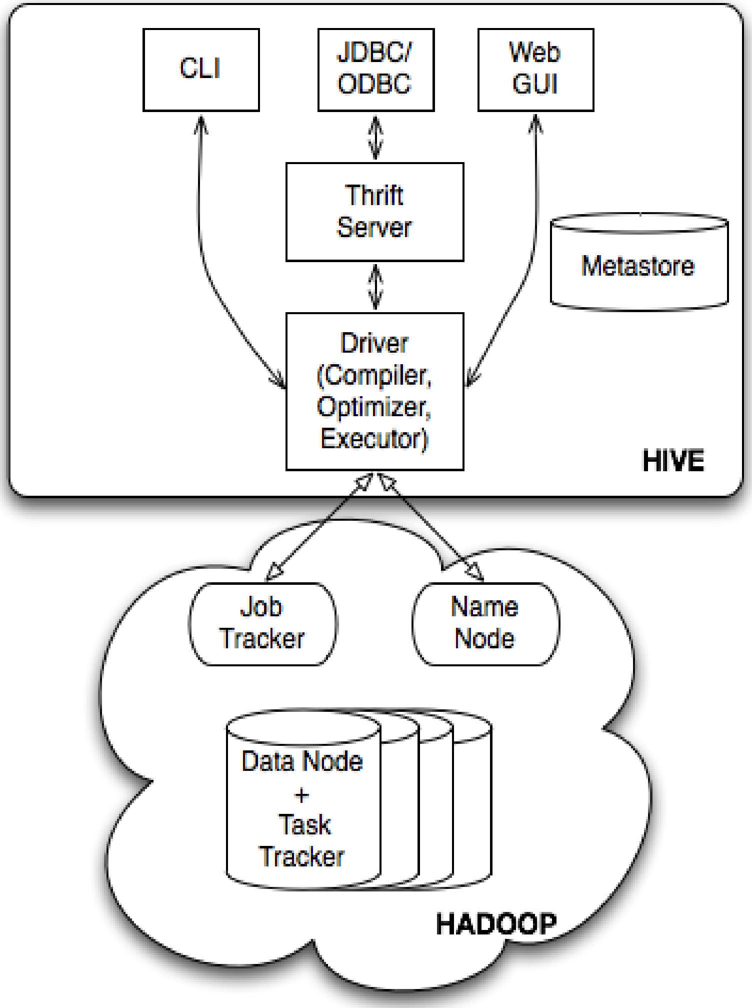
```
ALTER TABLE pvs
ADD PARTITION (ds=‘20090801’, ctry=‘US’)
LOCATION ‘/path/to/existing/partition’
```

Data Types

- **Primitive Types**
  - **integer types, float, string, date, boolean**
- **Nestable Collections**
  - **array<any-type>**
  - **map<primitive-type, any-type>**
- **User-defined types***
  - **Structures with attributes which can be of any-type**

* More details about user defined types in extensibility section

Hive Architecture



Hive Query Language

- SQL
  - Sub-queries in from clause
  - Equi-joins
    - Inner
    - Left, Right, full Outer
  - Multi-table Insert
  - Multi-group-by
- Sampling

Hive Query Language

- **Extensibility**
  - **Pluggable Map-reduce scripts**
  - **Pluggable User Defined Functions**
  - **Pluggable User Defined Types**
    - **Complex object types: List of Maps**
  - **Pluggable Data Formats**
    - **Apache Log Format**
    - **Columnar Storage Format**

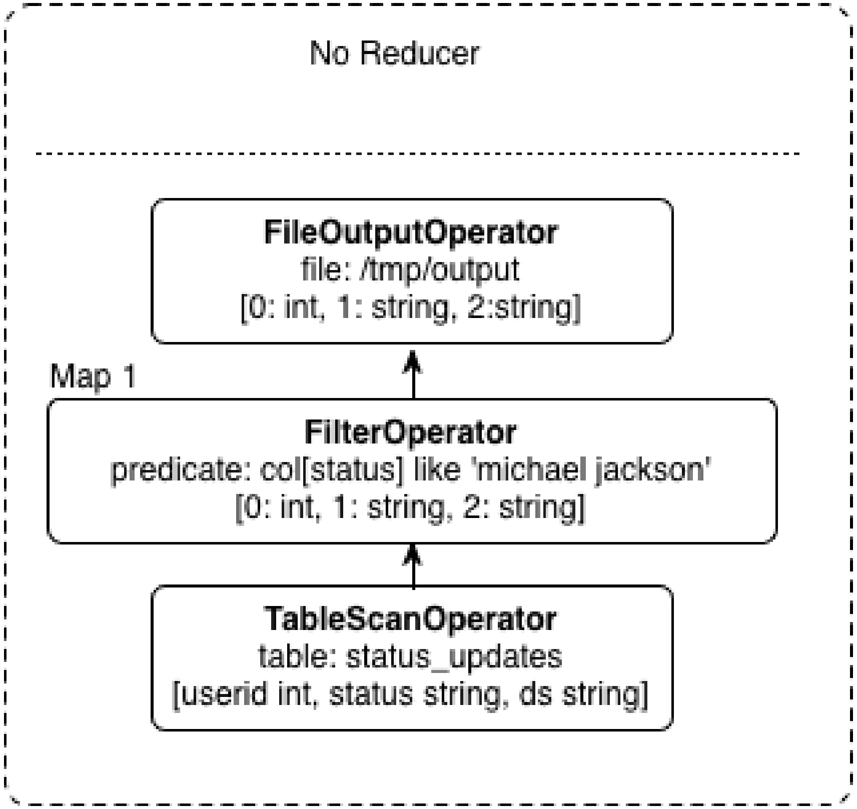
Example Application

- **Status updates table:**
  - `status_updates(userid int, status string, ds string)`
- **Load the data from log files:**
  - `LOAD DATA LOCAL INPATH ‘/logs/status_updates’ INTO TABLE status_updates PARTITION (ds=’2009-03-20’)`
- **User profile table**
  - `profiles(userid int, school string, gender int)`
- **Load the data from MySQL udb potentially using sqoop***

* sqoop - http://www.cloudera.com/hadoop-sqoop

Example Query (Filter)

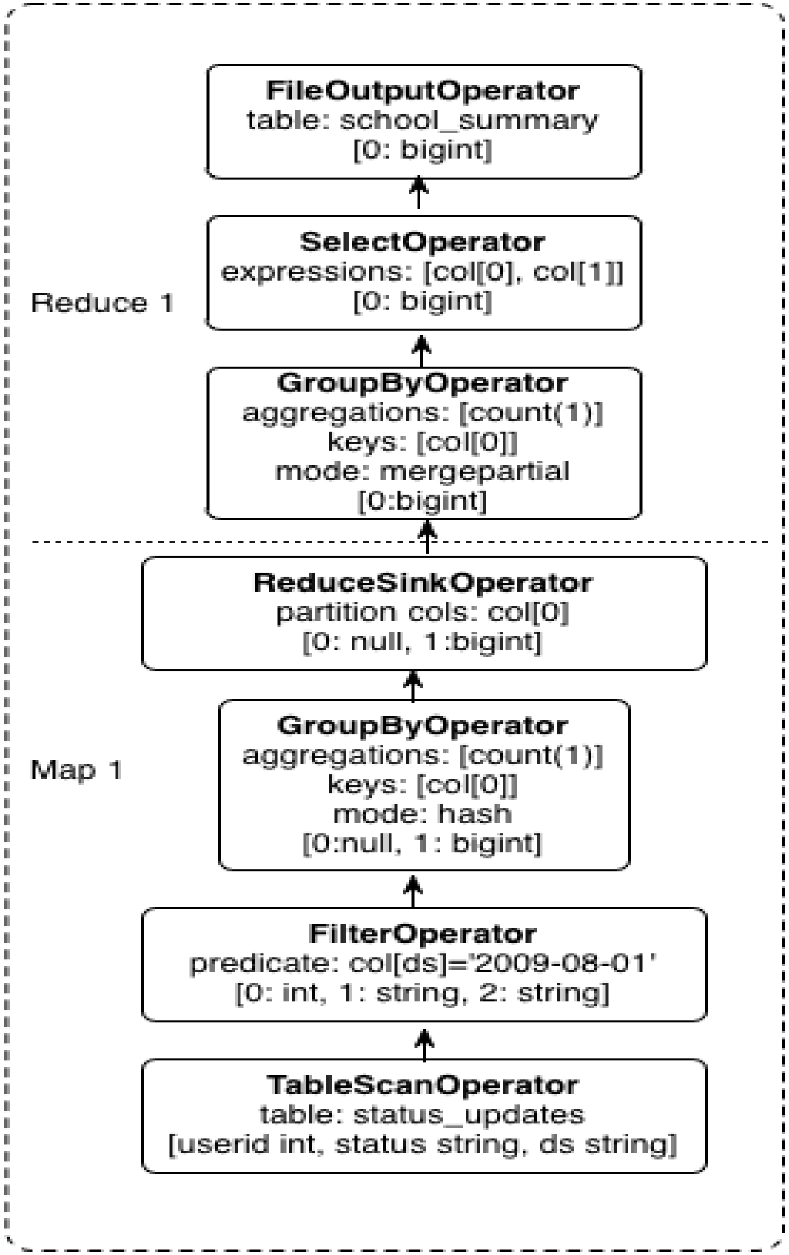
- Filter status updates containing ‘michael jackson’
  - `SELECT * FROM status_updates WHERE status LIKE ‘michael jackson’`


No Reducer
Map 1
FileOutputOperator
file: /tmp/output
[0: int, 1: string, 2:string]
FilterOperator
predicate: col[status] like 'michael jackson'
[0: int, 1: string, 2: string]
TableScanOperator
table: status_updates
[userid int, status string, ds string]

Example Query (Aggregation)

- **Figure out total number of status_updates in a given day**

  -
    ```
    SELECT COUNT(1) FROM status_updates WHERE
    ds = '2009-08-01'
    ```



Reduce 1

FileOutputOperator
table: school_summary
[0: bigint]

SelectOperator
expressions: [col[0], col[1]]
[0: bigint]

GroupByOperator
aggregations: [count(1)]
keys: [col[0]]
mode: mergepartial
[0:bigint]

Map 1

ReduceSinkOperator
partition cols: col[0]
[0: null, 1 :bigint]

GroupByOperator
aggregations: [count(1)]
keys: [col[0]]
mode: hash
[0:null, 1: bigint]

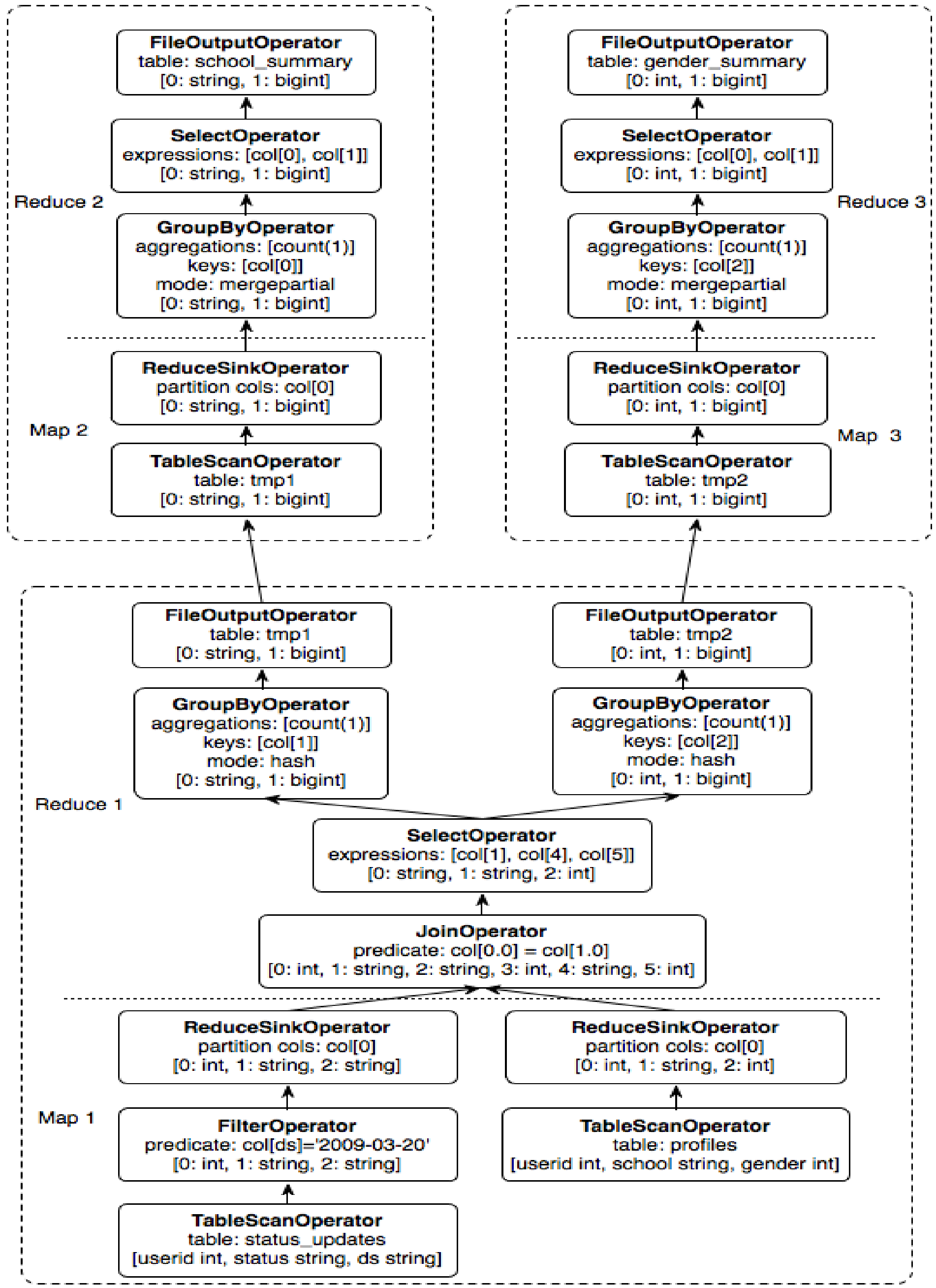
FilterOperator
predicate: col[ds]='2009-08-01'
[0: int, 1: string, 2: string]

TableScanOperator
table: status_updates
[userid int, status string, ds string]

- **Next example query combines group-by, joins and multi-table inserts.**

Example Query (multi-group-by)

```
FROM (SELECT a.status, b.school, b.gender
      FROM status_updates a JOIN profiles b
           ON (a.userid = b.userid and
               a.ds='2009-03-20' )
      ) subq1
INSERT OVERWRITE TABLE gender_summary
                      PARTITION(ds='2009-03-20')
SELECT subq1.gender, COUNT(1)
GROUP BY subq1.gender
INSERT OVERWRITE TABLE school_summary
                           PARTITION(ds='2009-03-
  20')
SELECT subq1.school, COUNT(1)
GROUP BY subq1.school
```



Reduce 2

FileOutputOperator
table: school_summary
[0: string, 1: bigint]

SelectOperator
expressions: [col[0], col[1]]
[0: string, 1: bigint]

GroupByOperator
aggregations: [count(1)]
keys: [col[0]]
mode: mergepartial
[0: string, 1: bigint]

Map 2

ReduceSinkOperator
partition cols: col[0]
[0: string, 1: bigint]

TableScanOperator
table: tmp1
[0: string, 1: bigint]

Reduce 3

FileOutputOperator
table: gender_summary
[0: int, 1: bigint]

SelectOperator
expressions: [col[0], col[1]]
[0: int, 1: bigint]

GroupByOperator
aggregations: [count(1)]
keys: [col[2]]
mode: mergepartial
[0: int, 1: bigint]

Map 3

ReduceSinkOperator
partition cols: col[0]
[0: int, 1: bigint]

TableScanOperator
table: tmp2
[0: int, 1: bigint]

Reduce 1

FileOutputOperator
table: tmp1
[0: string, 1: bigint]

FileOutputOperator
table: tmp2
[0: int, 1: bigint]

GroupByOperator
aggregations: [count(1)]
keys: [col[1]]
mode: hash
[0: string, 1: bigint]

GroupByOperator
aggregations: [count(1)]
keys: [col[2]]
mode: hash
[0: int, 1: bigint]

SelectOperator
expressions: [col[1], col[4], col[5]]
[0: string, 1: string, 2: int]

JoinOperator
predicate: col[0.0] = col[1.0]
[0: int, 1: string, 2: string, 3: int, 4: string, 5: int]

Map 1

ReduceSinkOperator
partition cols: col[0]
[0: int, 1: string, 2: string]

ReduceSinkOperator
partition cols: col[0]
[0: int, 1: string, 2: int]

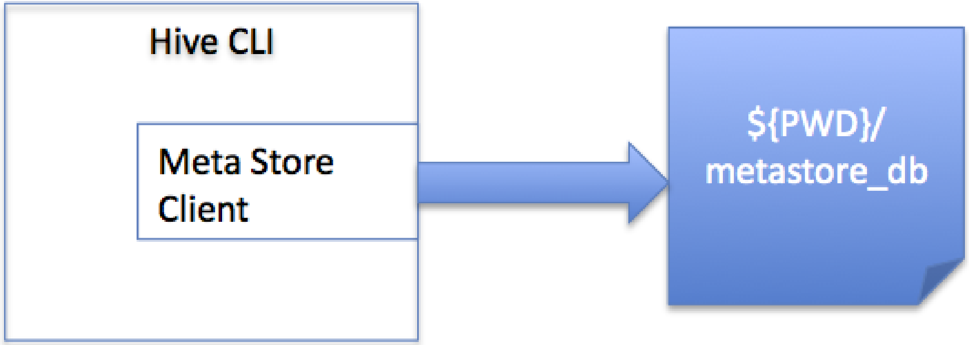
FilterOperator
predicate: col[ds]='2009-03-20'
[0: int, 1: string, 2: string]

TableScanOperator
table: profiles
[userid int, school string, gender int]

TableScanOperator
table: status_updates
[userid int, status string, ds string]

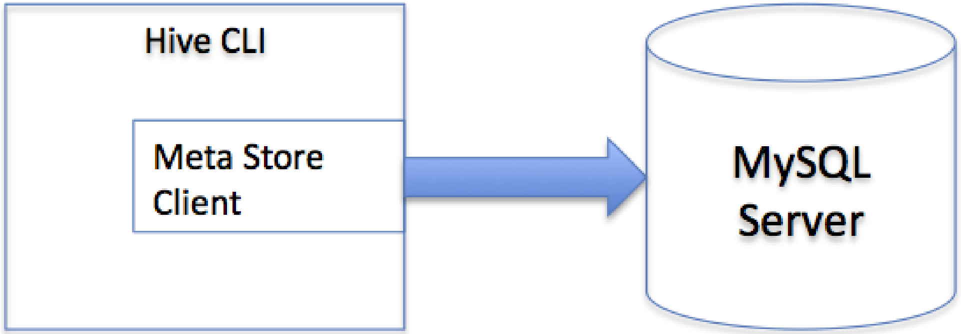
Hive Metastore

Single User Mode (Default)



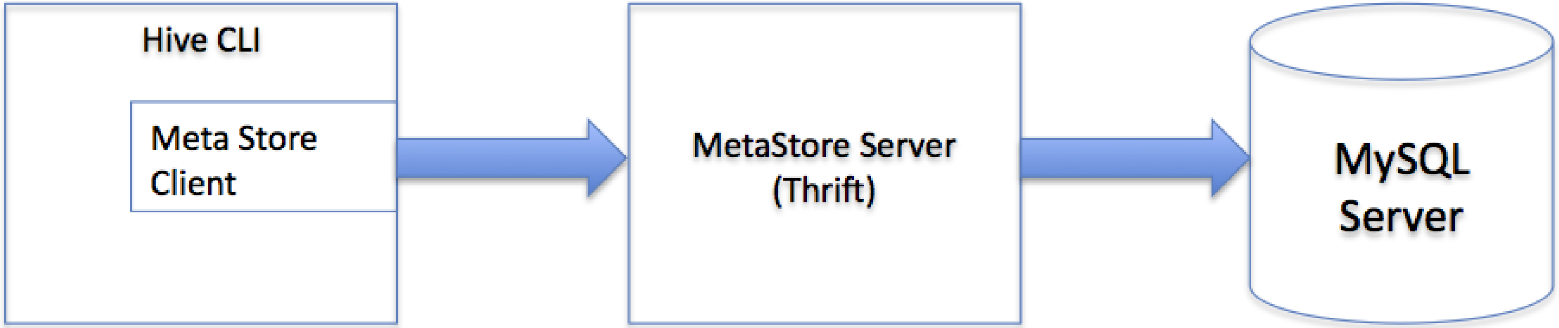
Parameter	Description	Example
javax.jdo.option.ConnectionURL	JDBC connection URL along with database name containing metadata	jdbc:derby:;databaseName=metastore_db;create=true
javax.jdo.option.ConnectionDriverName	JDBC driver name. Embedded Derby for Single user mode.	org.apache.derby.jdbc.EmbeddedDriver
javax.jdo.option.ConnectionUserName	User name for Derby database	APP
javax.jdo.option.ConnectionPassword	Password	mine

Multi User Mode



Parameter	Description	Example
javax.jdo.option.ConnectionURL	JDBC connection URL along with database name containing metadata	jdbc:mysql://<host name>/<database name>?createDatabaseIfNotExist=true
javax.jdo.option.ConnectionDriverName	Any JDO supported JDBC driver.	com.mysql.jdbc.Driver
javax.jdo.option.ConnectionUserName	User name	
javax.jdo.option.ConnectionPassword	Password	

Remote Server



- Server Configuration same as multi user mode client config (prev slide). To run server

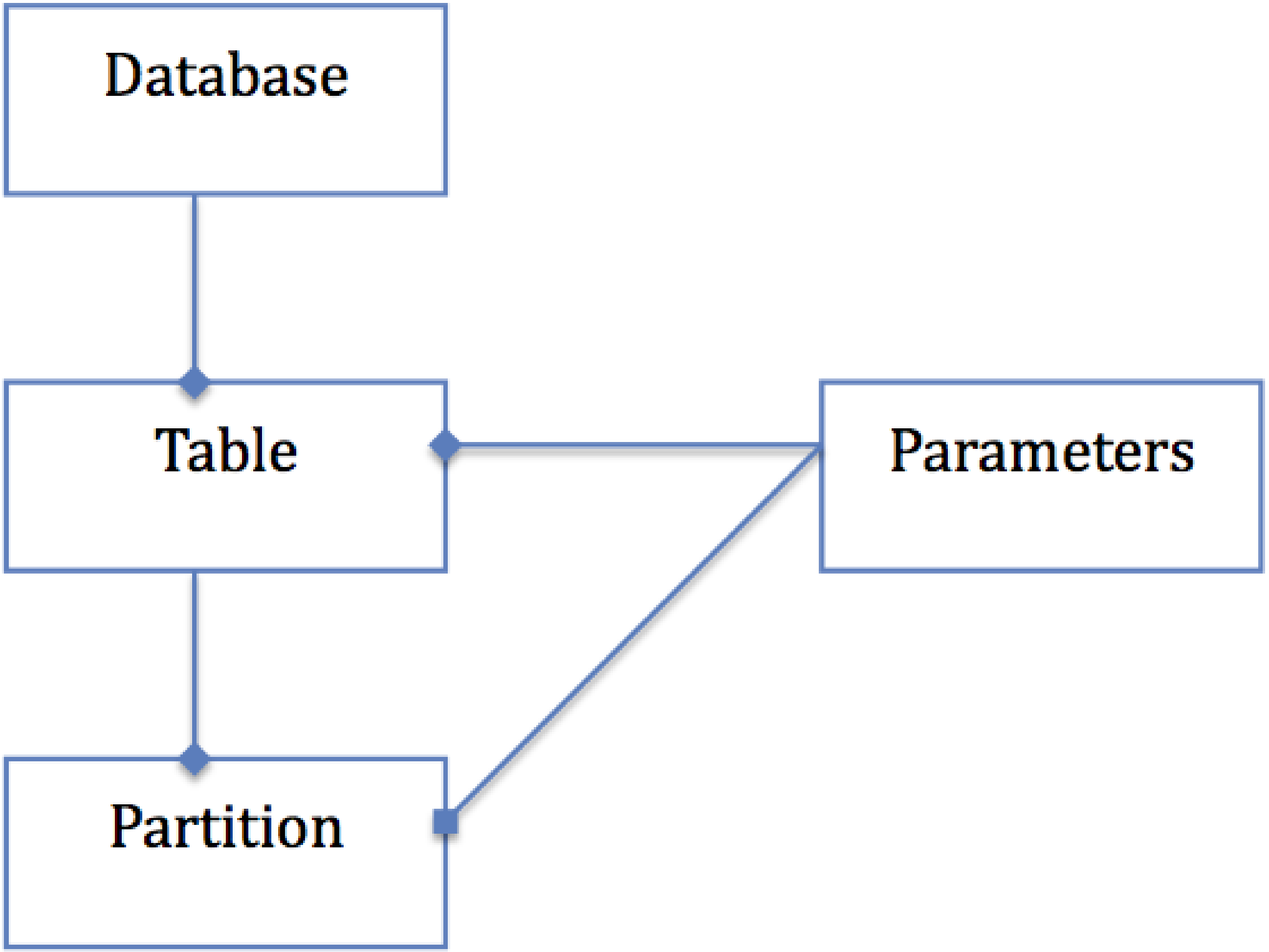
```
$JAVA_HOME/bin/java -Xmx1024m -Dlog4j.configuration=file://$HIVE_HOME/conf/hms-log4j.properties
    -Djava.library.path=$HADOOP_HOME/lib/native/Linux-amd64-64/ -cp $CLASSPATH
    org.apache.hadoop.hive.metastore.HiveMetaStore
```

- Client Configuration

Parameter	Description	Example
hive.metastore.uris	Location of the metastore server	thrift://<host_name>:9083
hive.metastore.local		false

- **Single User Mode**
  - **Unit tests**
  - **Evaluation**
- **Multi User Mode**
  - **Any significant Hive project**
- **Remote Server**
  - **For non-Java metastore clients**

Meta Data Model



Hive Optimizations

Optimizations

- **Column Pruning**
- **Predicate Pushdown**
- **Partition Pruning**
- **Join**
- **Group By**
- **Merging of small files**

Column Pruning

- **As name suggests – discard columns which are not needed**
  - `SELECT a,b FROM T WHERE e < 10;`
  - `T contains 5 columns (a,b,c,d,e)`
- **Columns c,d are discarded**
- **Select only the relevant columns**
- **Enabled by default**
  - `hive.optimize.cp = true`

Predicate Pushdown

- **Move predicate closer to the table scan only.**
- **Enabled by default:**
  - `hive.optimize.ppd = true`

- **Predicates moved up across joins.**

  -
    ```
    SELECT * FROM T1 JOIN T2 ON (T1.c1=T2.c2 AND T1.c1
    < 10)
    ```

  -
    ```
    SELECT * FROM T1 JOIN T2 ON (T1.c1=T2.c2) WHERE
    T1.c1 < 10
    ```
- **Special needs for outer joins:**
  - **Left outer join: predicates on the left side aliases are pushed**
  - **Right outer join: predicates on the right side aliases are pushed**
  - **Full outer join: none of the predicates are pushed**

- **Non-deterministic functions (eg. rand()) not pushed.**
- **Use annotation:**
  - `@UDFType(deterministic=false)`
- **The entire expression containing non-deterministic function is not pushed up**
  - `C1 > 10 and c2 < rand()`

Partition Pruning

- **Reduce list of partitions to be scanned**
- **Works on parse tree currently – some known bugs**

Partition Pruning

- Reduce list of partitions to be scanned
- Works on parse tree currently – some known bugs

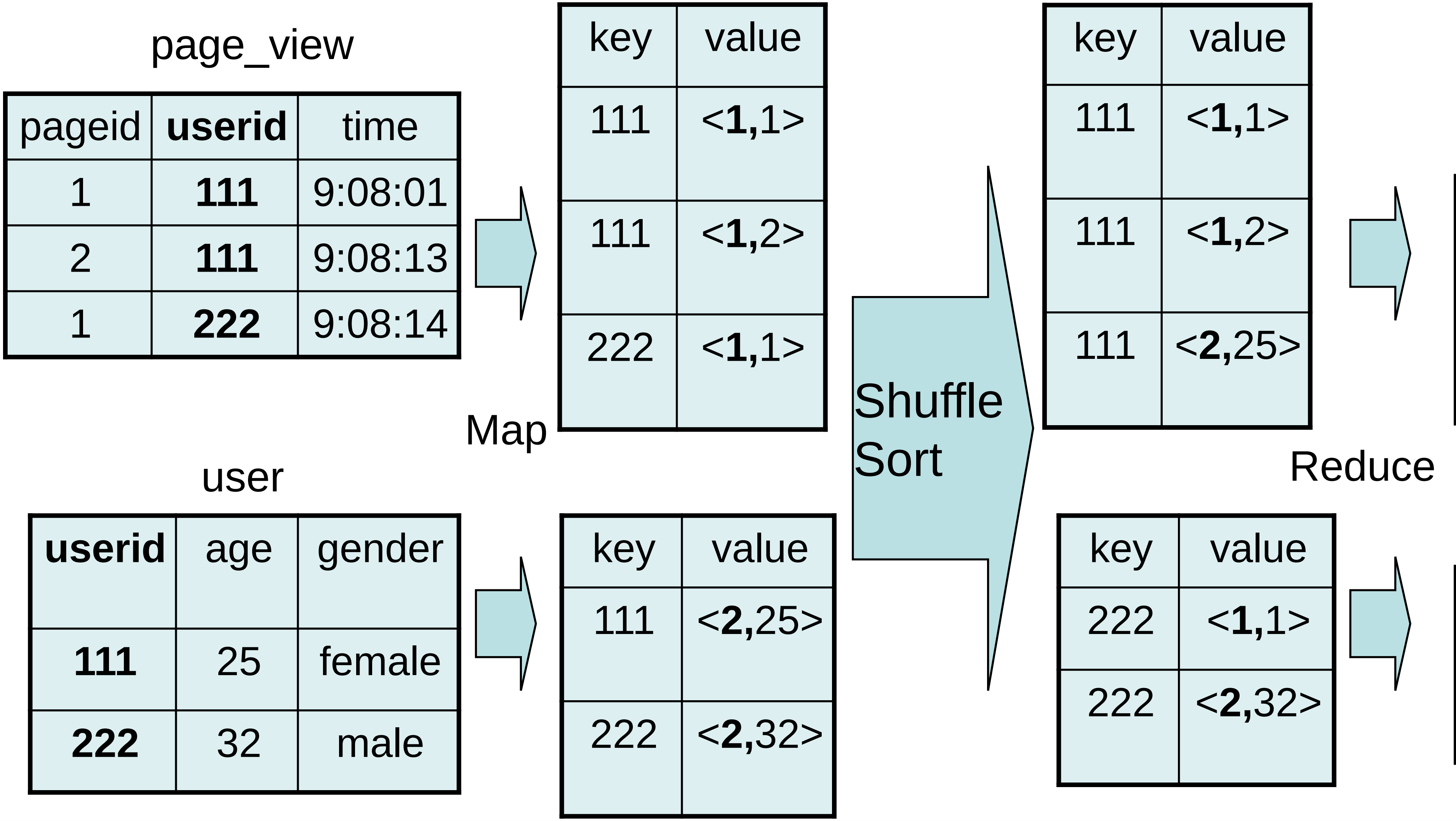
```
SELECT * FROM
  (SELECT c1, COUNT(1) FROM T GROUP BY c1) subq
WHERE subq.prtn = 100;

SELECT * FROM T1 JOIN
  (SELECT * FROM T2) subq ON (T1.c1=subq.c2)
WHERE subq.prtn = 100;
```

- `hive.mapred.mode = nonstrict`
- Strict mode, scan of a complete partitioned table fails

Hive QL – Join

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

Hive QL – 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

Hive QL – Join

- **Rightmost table streamed – whereas inner tables data is kept in memory for a given key. Use largest table as the right most table.**

- **`hive.mapred.mode = nonstrict`**
- **In strict mode, Cartesian product not allowed**

Hive QL – Join

```
INSERT OVERWRITE TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view p JOIN user u
  ON (pv.userid = u.userid)
  JOIN newuser x on (u.userid = x.userid);
```

Hive QL – Join

- Same join key – merge into 1 map-reduce job – true for any number of tables with the same join key.
- 1 map-reduce job instead of ‘n’
- The merging happens for OUTER joins also

Hive QL – Join

```
INSERT OVERWRITE TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view p JOIN user u
  ON (pv.userid = u.userid)
  JOIN newuser x on (u.age = x.age);
```

Hive QL – Join

- **Different join keys – 2 map-reduce jobs**
- **Same as:**

```
INSERT OVERWRITE TABLE tmptable SELECT *
FROM page_view p JOIN user u
  ON (pv.userid = u.userid;

INSERT OVERWRITE TABLE pv_users
SELECT x.pageid, x.age
FROM tmptable x JOIN newuser y on (x.age = y.age);
```

-

Join Optimizations

- Map Joins
  - User specified small tables stored in hash tables on the mapper backed by jdbm
  - No reducer needed

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

Hive QL – Map Join

page_view

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

Hash table

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

user

userid	age	gender
111	25	female
222	32	male

pv_users

Pageid	age
1	25
2	25
1	32

Map Join

- Optimization phase
- n-way map-join if (n-1) tables are map side readable
- Mapper reads all (n-1) tables before processing the main table under consideration
- Map-side readable tables are cached in memory and backed by JDBM persistent hash tables

Parameters

- `hive.join.emit.interval = 1000`
- `hive.mapjoin.size.key    = 10000`
- `hive.mapjoin.cache.numrows = 10000`

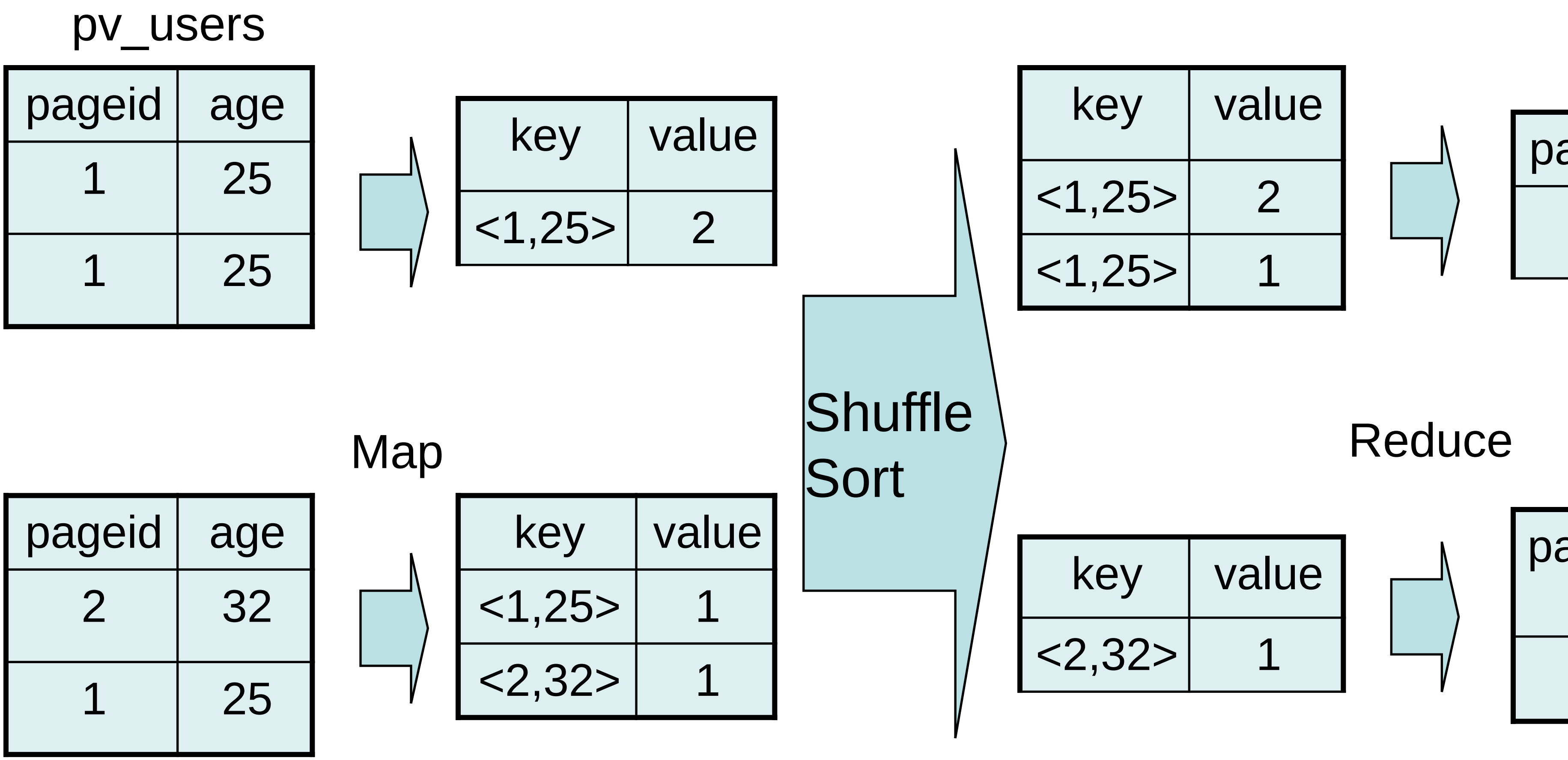
Future

- Sizes/statistics to determine join order
- Sizes to enforce map-join
- Better techniques for handling skews for a given key
- Using sorted properties of the table
- Fragmented joins
- n-way joins for different join keys by replicating data

Hive QL – Group By

```
SELECT pageid, age, count(1)
FROM pv_users
GROUP BY pageid, age;
```

Hive QL – Group By in Map Reduce



pv_users

pageid	age
1	25
1	25

key	value
<1,25>	2

pageid	age
2	32
1	25

Map

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

Shuffle Sort

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

key	value
<2,32>	1

Reduce

pa

pa

Group by Optimizations

- **Map side partial aggregations**
  - **Hash-based aggregates**
  - **Serialized key/values in hash tables**
  - **90% speed improvement on Query**
    - `SELECT count(1) FROM t;`

- **Load balancing for data skew**

Parameters

- hive.map.aggr = true
- hive.groupby.skewindata = false
- hive.groupby.mapaggr.checkinterval = 100000
- hive.map.aggr.hash.percentmemory = 0.5
- hive.map.aggr.hash.min.reduction = 0.5

Multi GroupBy

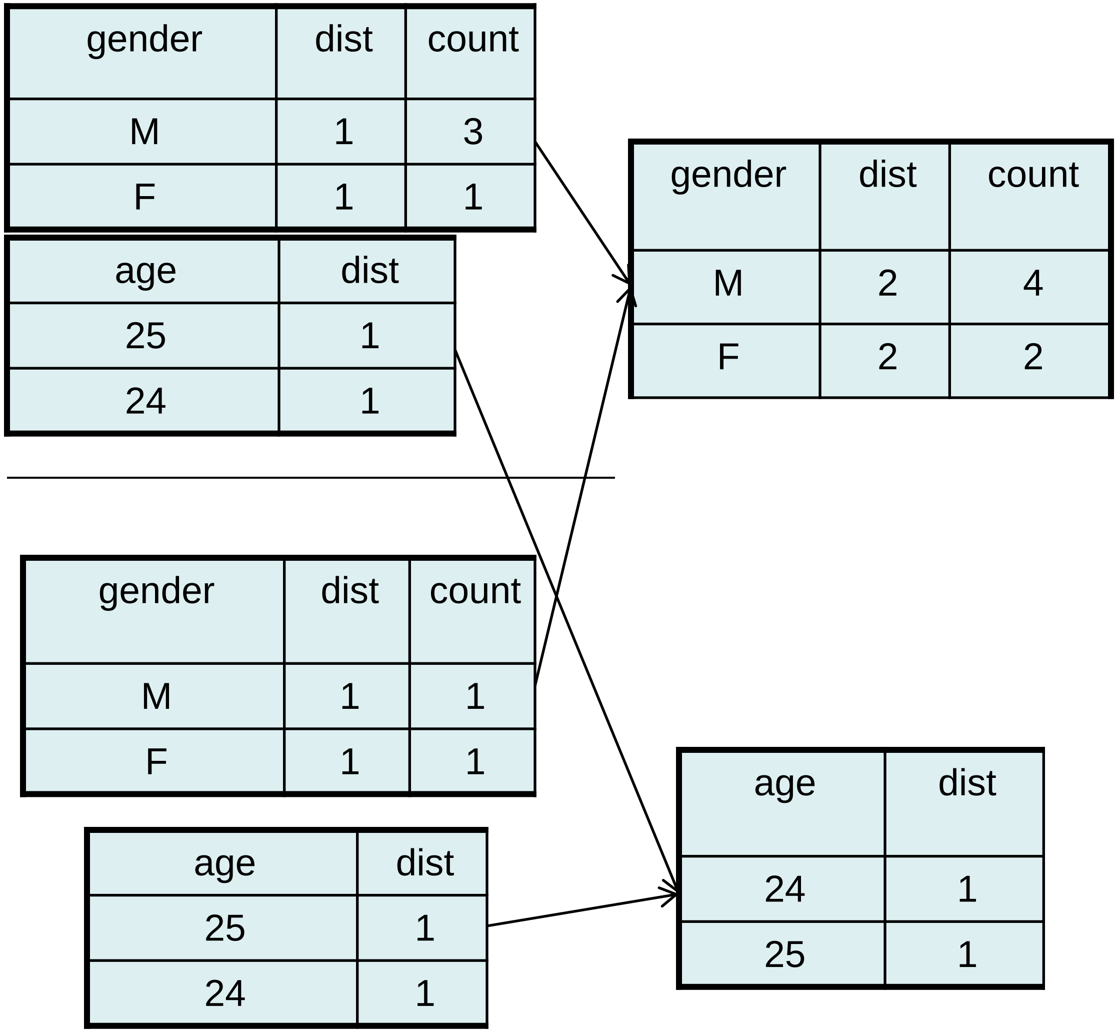
```
FROM pv_users
  INSERT OVERWRITE TABLE pv_gender_sum
   SELECT gender, count(DISTINCT userid), count(userid)
     GROUP BY gender
  INSERT OVERWRITE TABLE pv_age_sum
   SELECT age, count(DISTINCT userid)
     GROUP BY age
  □
```

Hive QL – Group By in Map Reduce

pv_users

gender	age	userid
M	25	1
M	25	2
M	25	1
M	24	1
F	24	2
F	24	1

Key: userid
Value: gender, age



gender	dist	count
M	1	3
F	1	1

age	dist
25	1
24	1

gender	dist	count
M	1	1
F	1	1

age	dist
25	1
24	1

gender	dist	count
M	2	4
F	2	2

age	dist
24	1
25	1

- **n+1 map-reduce jobs instead of 2n**
- **Single scan of input table**
- **Same distinct key across all groupbys**
- **Always use multi-groupby**

Merging of small files

- Lots of small files creates problems for downstream jobs

  -
    ```
    SELECT * from T where x < 10;
    ```

```
hive.merge.mapfiles      = true
hive.merge.mapredfiles   = false
hive.merge.size.per.task = 256*1000*1000
```

- Increases time for current query

Hive Extensibility Features

Agenda

- Introduction

- File Format
- SerDe
- Map/Reduce Scripts (Transform)
- UDF
- UDAF

- How to contribute the work

» Introduction

Hive is an open system

- Different on-disk data formats
  - Text File, Sequence File, …
- Different in-memory data formats
  - Java Integer/String, Hadoop IntWritable/Text …
- User-provided map/reduce scripts
  - In any language, use stdin/stdout to transfer data …
- User-defined Functions
  - Substr, Trim, From_unixtime …
- User-defined Aggregation Functions
  - Sum, Average …

» File Format

File Format Example

-
  ```
  CREATE TABLE mylog (
     user_id BIGINT,
     page_url STRING,
     unix_time INT)
  STORED AS TEXTFILE;
  ```

-
  ```
  LOAD DATA INPATH '/user/myname/log.txt'
  INTO TABLE mylog;
  ```

Existing File Formats

	TEXTFILE	SEQUENCEFILE	RCFILE
Data type	text only	text/binary	text/binary
Internal Storage order	Row-based	Row-based	Column-based
Compression	File-based	Block-based	Block-based
Splitable*	YES	YES	YES
Splitable* after compression	NO	YES	YES

*** Splitable: Capable of splitting the file so that a single huge file can be processed by multiple mappers in parallel.**

When to add a new File Format

- User has files with special file formats not supported by Hive yet, and users don’t want to convert the files before loading into Hive.

- User has a more efficient way of storing data on disk.

How to add a new File Format

- **Follow the example in contrib/src/java/org/apache/hadoop/hive/contrib/fileformat/base64**

- **Base64TextFileFormat supports storing of binary data into text files, by doing base64 encoding/decoding on the fly.**

-
```
CREATE TABLE base64_test(col1 STRING, col2 STRING)
  STORED AS
    INPUTFORMAT
'org.apache.hadoop.hive.contrib.fileformat.base64.Base64TextInputFormat'
    OUTPUTFORMAT
'org.apache.hadoop.hive.contrib.fileformat.base64.Base64TextOutputFormat';
```

» SerDe

SerDe Examples

-
  ```
  CREATE TABLE mylog (
    user_id BIGINT,
    page_url STRING,
    unix_time INT)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
  ```

-
  ```
  CREATE table mylog_rc (
    user_id BIGINT,
    page_url STRING,
    unix_time INT)
  ROW FORMAT SERDE
    'org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe'
  STORED AS RCFILE;
  ```

SerDe

- SerDe is short for serialization/deserialization. It controls the format of a row.
- Serialized format:
  - Delimited format (tab, comma, ctrl-a …)
  - Thrift Protocols
  - ProtocolBuffer*
- Deserialized (in-memory) format:
  - Java Integer/String/ArrayList/HashMap
  - Hadoop Writable classes
  - User-defined Java Classes (Thrift, ProtocolBuffer*)

- * ProtocolBuffer support not available yet.

Existing SerDes

	LazySimpleSerDe	LazyBinarySerDe (HIVE-640)	BinarySortable SerDe
serialized format	delimited	proprietary binary	proprietary binary sortable*
deserialized format	LazyObjects*	LazyBinaryObjects*	Writable
	ThriftSerDe (HIVE-706)	**RegexSerDe**	**ColumnarSerDe**
serialized format	Depends on the Thrift Protocol	Regex formatted	proprietary column-based
deserialized format	User-defined Classes, Java Primitive Objects	ArrayList<String>	LazyObjects*

*** LazyObjects: deserialize the columns only when accessed.**

*** Binary Sortable: binary format preserving the sort order.**

When to add a new SerDe

- User has data with special serialized format not supported by Hive yet, and users don’t want to convert the data before loading into Hive.
- User has a more efficient way of serializing the data on disk.

How to add a new SerDe for text data

- Follow the example in contrib/src/java/org/apache/hadoop/hive/contrib/serde2/RegexSerDe.java

- RegexSerDe uses a user-provided regular expression to deserialize data.

-
```
CREATE TABLE apache_log(host STRING,
   identity STRING, user STRING, time STRING, request STRING,
   status STRING, size STRING, referer STRING, agent STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
   "input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) (-|\\[[^\\]]*\\]) ([^
 \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*)(?: ([^ \"]*|\"[^\"]*\") ([^
 \"]*|\"[^\"]*\"))?",
   "output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s
 %9$s”)
STORED AS TEXTFILE;
```

How to add a new SerDe for binary data

- Follow the example in
  contrib/src/java/org/apache/hadoop/hive/contrib/serde2/thrift (HIVE-706)
  serde/src/java/org/apache/hadoop/hive/serde2/binarysortable

-
```
CREATE TABLE mythrift_table
ROW FORMAT SERDE
 'org.apache.hadoop.hive.contrib.serde2.thrift.ThriftSerDe'
WITH SERDEPROPERTIES (
 "serialization.class" = "com.facebook.serde.tprofiles.full",
 "serialization.format" =
 "com.facebook.thrift.protocol.TBinaryProtocol“);
```

- NOTE: Column information is provided by the SerDe class.

- **Map/Reduce Scripts (Transform)**

Map/Reduce Scripts Examples

```
» add file page_url_to_id.py;
» add file my_python_session_cutter.py;
» FROM
    (SELECT TRANSFORM(user_id, page_url, unix_time)
       USING 'page_url_to_id.py'
       AS (user_id, page_id, unix_time)
     FROM mylog
     DISTRIBUTE BY user_id
     SORT BY user_id, unix_time) mylog2
   SELECT TRANSFORM(user_id, page_id, unix_time)
     USING 'my_python_session_cutter.py'
     AS (user_id, session_info);
```

Map/Reduce Scripts

- Read/Write data through stdin and stdout.
- Print debug messages to stderr.

- Data Format is based on:
  - Text File Format
  - Delimited Row Format (using "\t")

- We can override the row format with HIVE-708.

- **UDF (User-defined Functions)**

UDF Example

- `add jar build/ql/test/test-udfs.jar;`
- `CREATE TEMPORARY FUNCTION testlength AS 'org.apache.hadoop.hive.ql.udf.UDFTestLength';`
- `SELECT testlength(src.value) FROM src;`
- `DROP TEMPORARY FUNCTION testlength;`

- UDFTestLength.java:

```
package org.apache.hadoop.hive.ql.udf;
public class UDFTestLength extends UDF {
  public Integer evaluate(String s) {
    if (s == null) {
      return null;
    }
    return s.length();
  }
}
```

More efficient UDF

- Avoid creating new objects
- Avoid UTF-8 encoding/decoding

- UDFTestLength.java:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
public class UDFTestLength extends UDF {
  IntWritable result = new IntWritable();
  public IntWritable evaluate(Text s) {
    if (s == null) {
      return null;
    }
    result.set(countUDF8Characters(s));
    return result;
  }
}
```

Overloaded UDF

» add jar build/contrib/hive_contrib.jar;

» CREATE TEMPORARY FUNCTION example_add AS 'org.apache.hadoop.hive.contrib.udf.example.UDFExampleAdd';

» SELECT example_add(1, 2) FROM src;

» SELECT example_add(1.1, 2.2) FROM src;

» UDFExampleAdd.java:

```
public class UDFExampleAdd extends UDF {
  public Integer evaluate(Integer a, Integer b) {
    if (a == null || b == null) return null;
    return a + b;
  }
  public Double evaluate(Double a, Double b) {
    if (a == null || b == null) return null;
    return a + b;
  }
}
```

Implicit Type Conversions for UDF

- **`SELECT example_add(1, 2.1) FROM src;`**
- **`Answer = 3.1`**

- **Hive does implicit type conversion for UDF.**
- **1 (int) is converted to 1.0 (double), and then passed to the UDF.**
- **That's why the answer is 3.1.**

- *** Implicit type conversion is controlled by UDFResolver and can be different for each UDF.**

Variable-length Arguments (HIVE-699)

» `SELECT example_add(1, 2) FROM src;`

» `SELECT example_add(1, 2, 3) FROM src;`

» `SELECT example_add(1, 2, 3, 4.1) FROM src;`

» **UDFExampleAdd.java:**

```
public class UDFExampleAdd extends UDF {
  public Integer evaluate(Integer... a) {
    int total = 0;
    for (int i=0; i<a.length; i++)
      if (a[i] != null) total += a[i];
    return total;
  }
  // the same for Double
  public Double evaluate(Double... a)
}
```

Summary for UDFs

- **Writing UDF in Java is simple.**
- **Hadoop Writables/Text provides better efficiency.**
- **UDF can be overloaded.**
- **Hive supports implicit type conversions.**
- **UDF can take variable-length arguments, just as in Java.**

- ***GenericUDF provides the best performance (avoiding Java reflection, allows short-circuit evaluation, etc).**

- UDAF (User-defined Aggregation Functions)

UDAF Example

-
  ```
  SELECT page_url, count(1), count(DISTINCT user_id)
  FROM mylog;
  ```

-
  ```
  public class UDAFCount extends UDAF {
    public static class Evaluator implements UDAFEvaluator {
     private int mCount;
    public void init() {mcount = 0;}
    public boolean iterate(Object o) {
      if (o!=null) mCount++; return true;}
    public Integer terminatePartial() {return mCount;}
    public boolean merge(Integer o) {mCount += o; return true;}
    public Integer terminate() {return mCount;}
  }
  ```

Overloaded UDAF

```
public class UDAFSum extends UDAF {
   public static class IntEvaluator implements UDAFEvaluator {
    private int mSum;
    public void init() {mSum = 0;}
    public boolean iterate(Integer o) {mSum += o; return true;}
    public Integer terminatePartial() {return mSum;}
    public boolean merge(Integer o) {mSum += o; return true;}
    public Integer terminate() {return mSum;}
   }
   public static class DblEvaluator implements UDAFEvaluator {
    private double mSum;
    public void init() {mSum = 0;}
    public boolean iterate(Double o) {mSum += o; return true;}
    public Double terminatePartial() {return mSum;}
    public boolean merge(Double o) {mSum += o; return true;}
    public Double terminate() {return mSum;}
   }
}
```

UDAF with Complex Intermediate Result

```
public class UDAFExampleAvg extends UDAF {
  public static class State {
    private long cnt;
    private double sum;
  }
  public static class Evaluator implements UDAFEvaluator {
    State s;
    public void init() {s.cnt = 0; s.sum = 0;}
    public boolean iterate(Double o) {s.cnt++; s.sum += o;}
    public State terminatePartial() {return this;}
    public boolean merge(State o)
      {s.cnt += o.s.cnt; s.sum += o.s.mSum;}
    public Double terminate()
      {return s.cnt == 0 ? null : s.sum/s.cnt;}
  }
}
```

UDAF without Partial Aggregations

- Implement dummy `terminatePartial` and `merge` functions that throw a RunTimeException.

- Do the following before running the query:
  - `set hive.map.aggr=false;`
  - `set hive.groupby.skewindata=false;`

Summary for UDAFs

- Writing UDAF is similar to writing UDF.
- UDAFs are overloaded via multiple static inner classes.
- UDAFs (as well as UDF) can return complex objects.
- We can disable partial aggregations for UDAFs.

- *GenericUDAF provides the best performance (avoiding Java reflection, etc).

Comparison of UDF/UDAF v.s. M/R scripts

	UDF/UDAF	M/R scripts
language	Java	any language
data format	in-memory objects	serialized streams
1/1 input/output	supported via UDF	supported
n/1 input/output	supported via UDAF	supported
1/n input/output	not supported yet (UDTF)	supported
Speed	faster	Slower

*** UDTF: User-defined Table Generation Function HIVE-655**

- **How to Contribute the Work**

How to contribute the work

- contrib is the directory for contribution of general-purpose file format / serde / udf / udaf.

- Write the code.
- Add a test case (a .q file). Verify the code works.
- "svn add" the new files. Create a JIRA and a patch.

- Reference: http://wiki.apache.org/hadoop/Hive/HowToContribute

Q & A

- File Format
- SerDe
- Map/Reduce Scripts (Transform)
- UDF
- UDAF

Hive – Roadmap

Branches and Releases

- Branch out Hive-0.4 by end of the week (8/7/2009)
- Release Hive-0.4 and Hive-0.3.1 by (8/29/2009)
- Drop support for Hadoop-0.17 in trunk?

Features available in 0.4

- ODBC driver
- Performance Optimizations
  - Map side joins
  - Columnar Storage
  - Lazy SerDe
  - Predicate pushdown
- Features
  - Multi group by
  - New UDFs and UDAFs

Future stuff

- Views and data variables
- Dumbo integration
- Create table as select
- Indexes
- Inserts without listing partitions
- Use sort properties to optimize query
- Insert appends
- IN, exists and correlated subqueries
- Types – timestamp, enums
- HAVING clause
- Better error reporting

More Future stuff

- Statistics
- More join optimizations
- Persistent UDFs and UDAFs
- Help on the CLI