facebook

Hive – A Petabyte Scale Data Warehouse Using Hadoop

Ashish Thusoo, **Raghotham Murthy**, Joydeep Sen Sarma, Zheng Shao, **Namit Jain,** Prasad Chakka, Suresh Anthony, Hao Liu, Ning Zhang

Facebook Data Infrastructure

ICDE 2010

Agenda

- Motivation for Hive
- Usage in Facebook
- Hive Details
  - System Architecture
  - Data Model
  - Query Language
- Hive Optimizations
  - Group By and Join Optimizations
- Extensibility Features
- Future Work

facebook

Motivation for Hive

Why Another Data Warehousing System?

Data, data and more data

200GB per day in March 2008

15+TB(compressed) raw data per day today

Existing solutions

- Not available
- Not scalable
- Expensive
- Proprietary

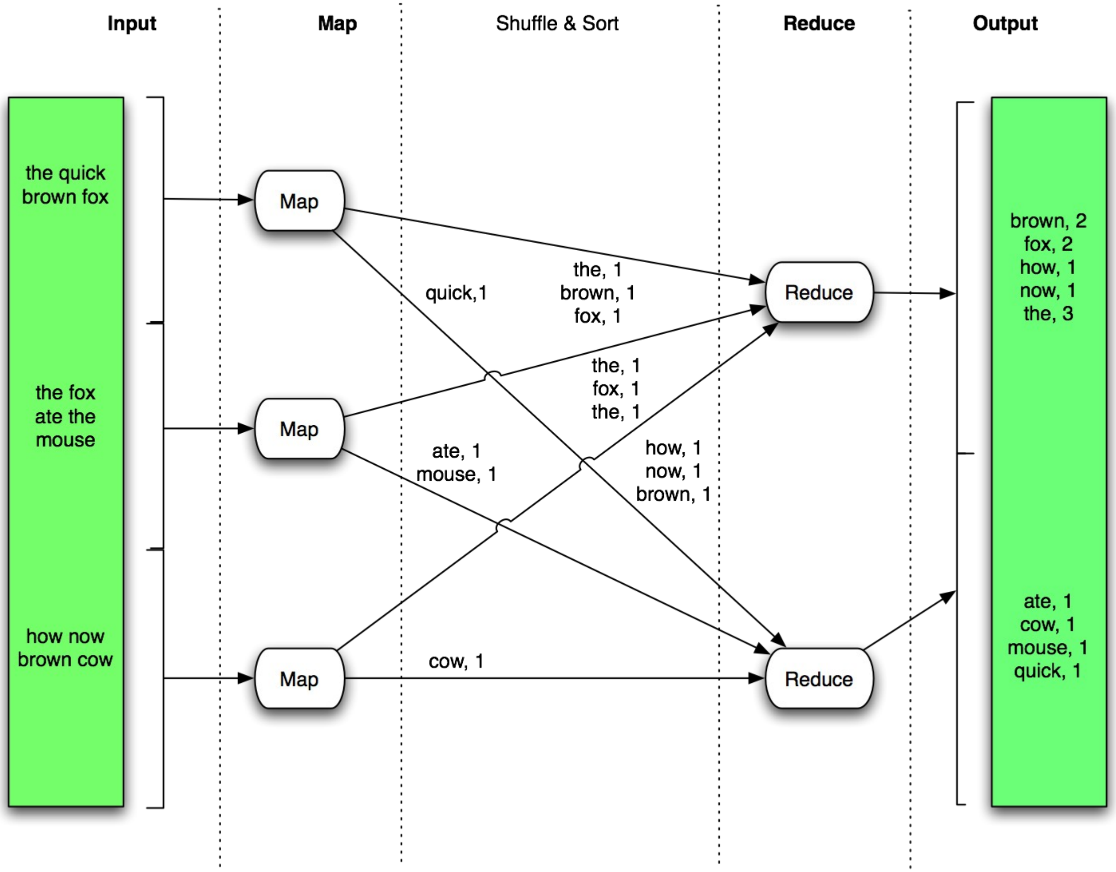
Lets try Hadoop…

- Pros
  - Superior in availability/scalability/manageability

- Cons: Programmability and Metadata
  - Efficiency not that great, but throw more hardware
  - Map-reduce hard to program (users know sql/bash/python)
  - No schema

- Solution: HIVE

Word Count using Map Reduce



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 hdfs files

- Key Building Principles:
  - SQL is a familiar language
  - Extensibility – Types, Functions, Formats, Scripts
  - Performance

Hive: Simplifying Hadoop – New Technology Familiar Interfaces

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

vs.

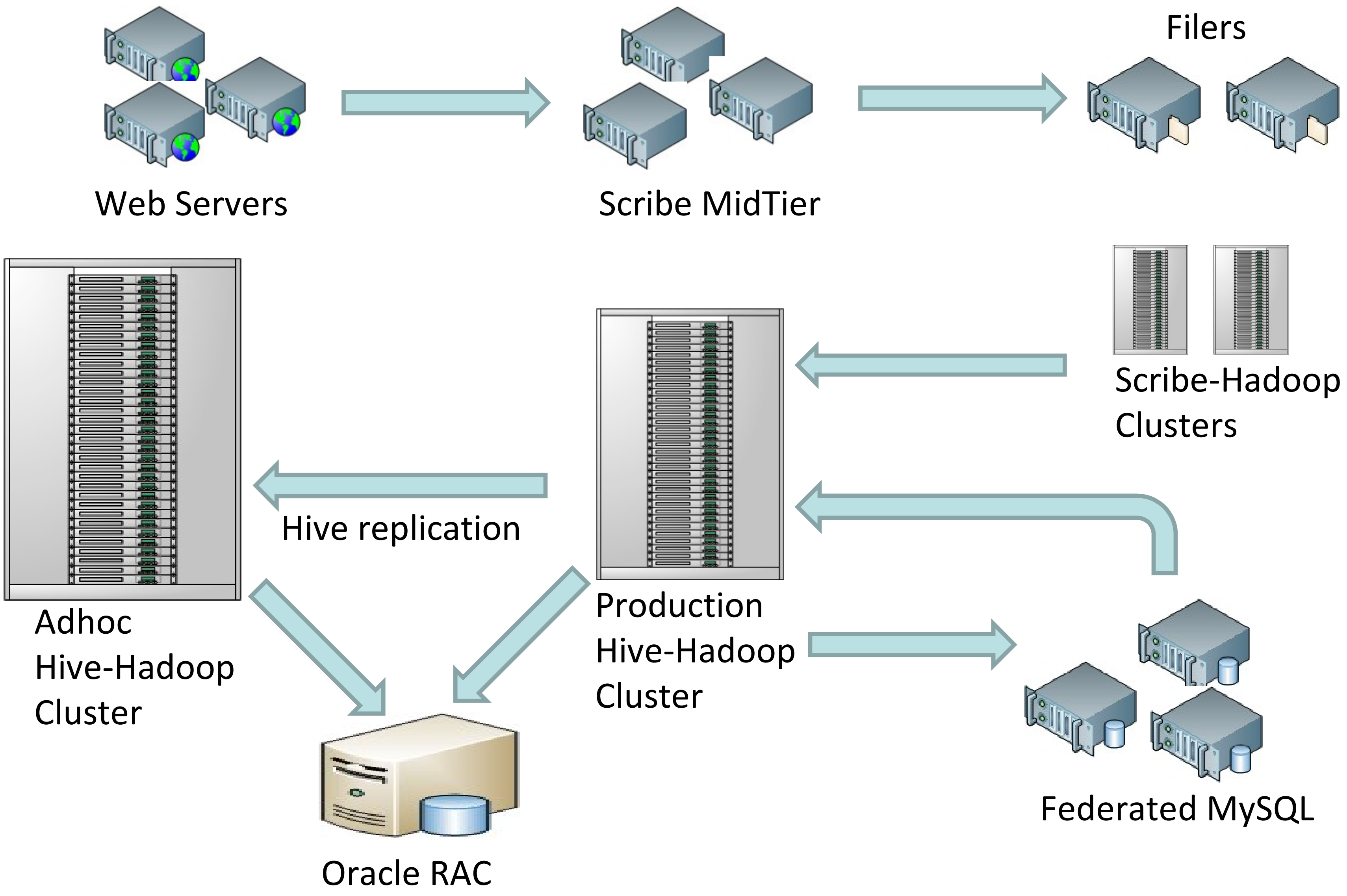
```
$ cat > /tmp/reducer.sh
uniq -c | awk '{print $2"\t"$1}`
$ cat > /tmp/map.sh
awk -F '\001' '{if($1 > 100) print $1}`
$ bin/hadoop jar contrib/hadoop-0.19.2-dev-streaming.jar
   -input /user/hive/warehouse/kv1 -mapper map.sh -file
   /tmp/reducer.sh -file /tmp/map.sh -reducer reducer.sh
   -output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

facebook

Usage in Facebook

facebook

Data Flow Architecture at Facebook



Warehousing at Facebook

- Instrumentation
- Automatic ETL
  - Realtime (work in progress)
- Metadata Discovery (CoHive)
- Query (HIVE)
- Workflow specification and execution (Chronos)
- Reporting tools
- Monitoring and alerting

Hadoop & Hive Cluster @ Facebook

- **Production Cluster**
  - 300 nodes/2400 cores
  - 3PB of raw storage
- **Adhoc Cluster**
  - 1200 nodes/9600 cores
  - 12PB of raw storage
- **Node (DataNode + TaskTracker) configuration**
  - 2CPU, 4 core per cpu
  - 12 x 1TB disk (900GB usable per disk)

Hive & Hadoop Usage @ Facebook

- **Statistics per day:**
  - 10TB of compressed new data added per day
  - 135TB of compressed data scanned per day
  - 7500+ Hive jobs per day
  - 80K compute hours per day

- **Hive simplifies Hadoop:**
  - New engineers go though a Hive training session
  - ~200 people/month run jobs on Hadoop/Hive
  - Analysts (non-engineers) use Hadoop through Hive
  - 95% of hadoop jobs are Hive Jobs

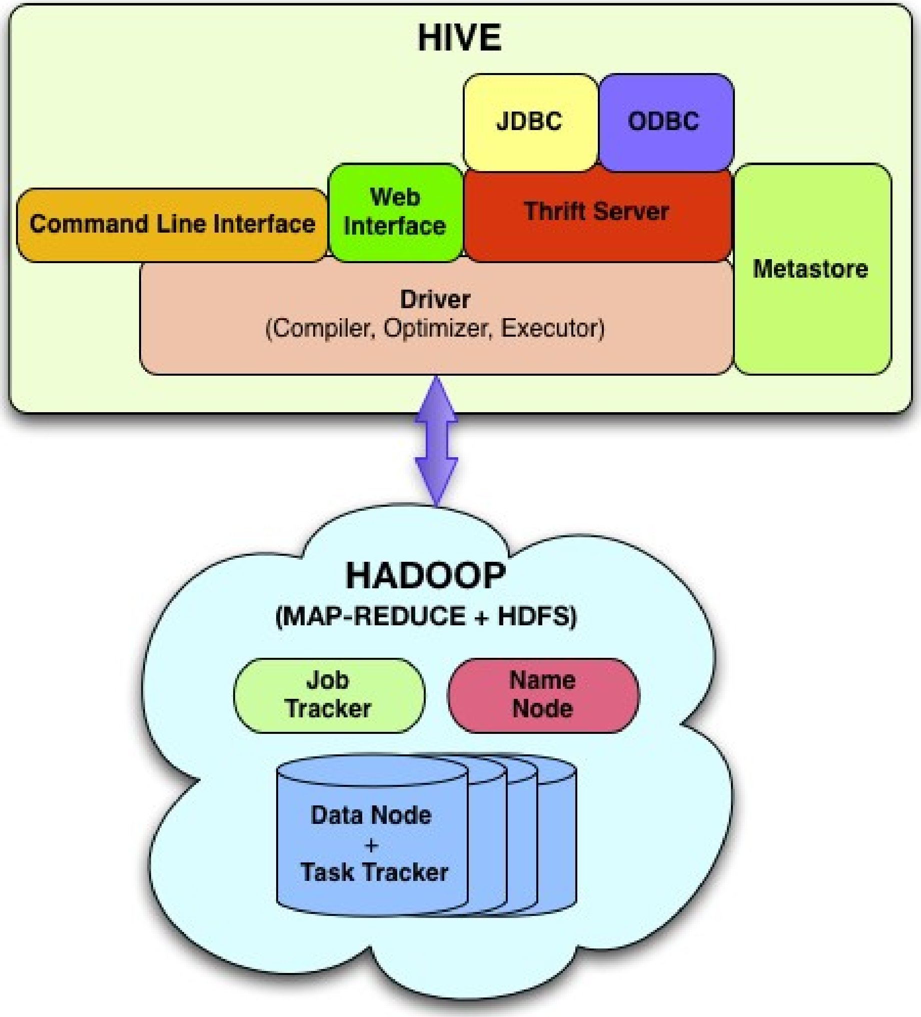
Hive & Hadoop Usage @ Facebook

- Types of Applications:
  - Reporting
    - Eg: Daily/Weekly aggregations of impression/click counts
    - Measures of user engagement
    - Microstrategy dashboards
  - Ad hoc Analysis
    - Eg: how many group admins broken down by state/country
  - Machine Learning (Assembling training data)
    - Ad Optimization
    - Eg: User Engagement as a function of user attributes
  - Many others

facebook

HIVE DETAILS

System Architecture



Data Model

Hive Entity	Sample Metastore Entity	Sample HDFS Location
Table	T	/wh/T
Partition	date=d1	/wh/T/date=d1
Bucketing column	userid	/wh/T/date=d1/part-0000 … /wh/T/date=d1/part-1000 (hashed on userid)
External Table	extT	/wh2/existing/dir (arbitrary location)

Column Data Types

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

Hive Query Language

- DDL
  - {create/alter/drop} {table/view/partition}
  - create table as select
- DML
  - Insert overwrite
- QL
  - Sub-queries in from clause
  - Equi-joins (including Outer joins)
  - Multi-table Insert
  - Sampling
  - Lateral Views
- Interfaces
  - JDBC/ODBC/Thrift

Example Application

- Status updates table:

  -
    ```
    status_updates(userid int, status string, ds
    string)
    ```
- Load the data from log files:

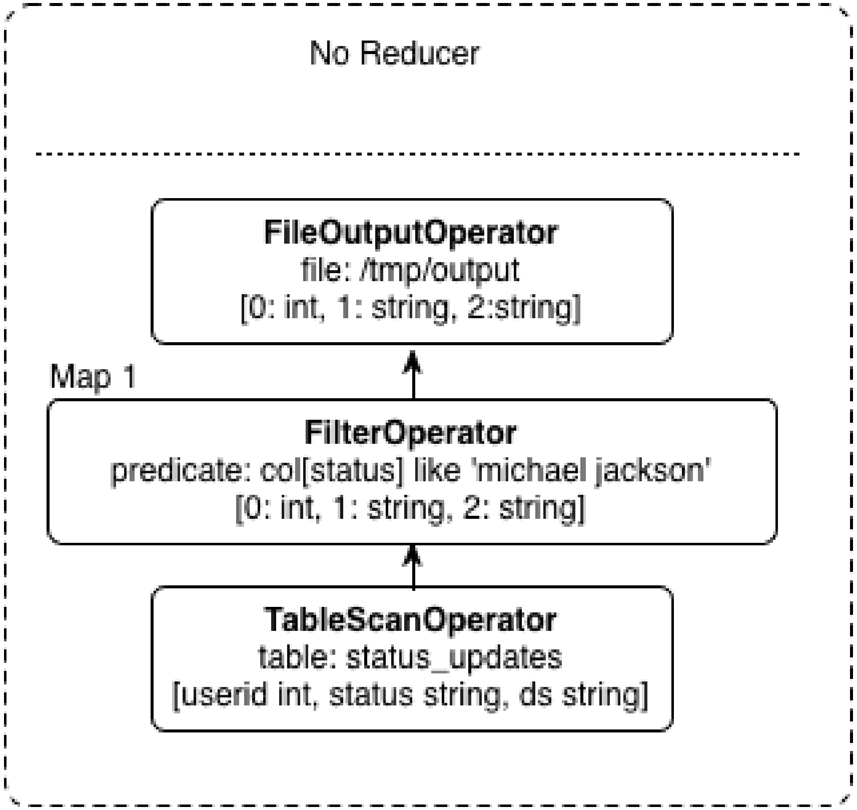
  -
    ```
    LOAD DATA LOCAL INPATH ‘/logs/status_updates’ INTO
    TABLE status_updates PARTITION (ds=’2009-03-20’)
    ```
- User profile table

  -
    ```
    profiles(userid int, school string, gender int)
    ```

facebook

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]

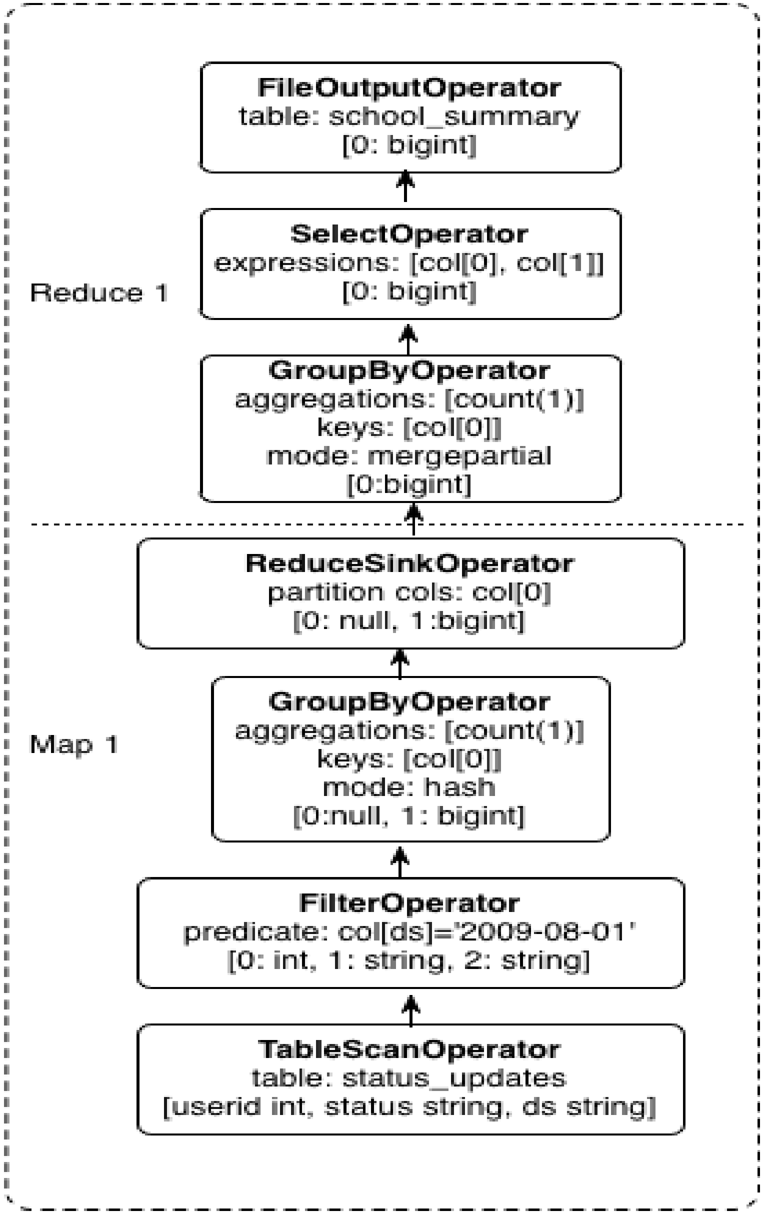
FilterOperator
predicate: col[status] like 'michael jackson'
[0: int, 1: string, 2: string]

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

facebook

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’



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
```

Hive Optimizations

Optimizations

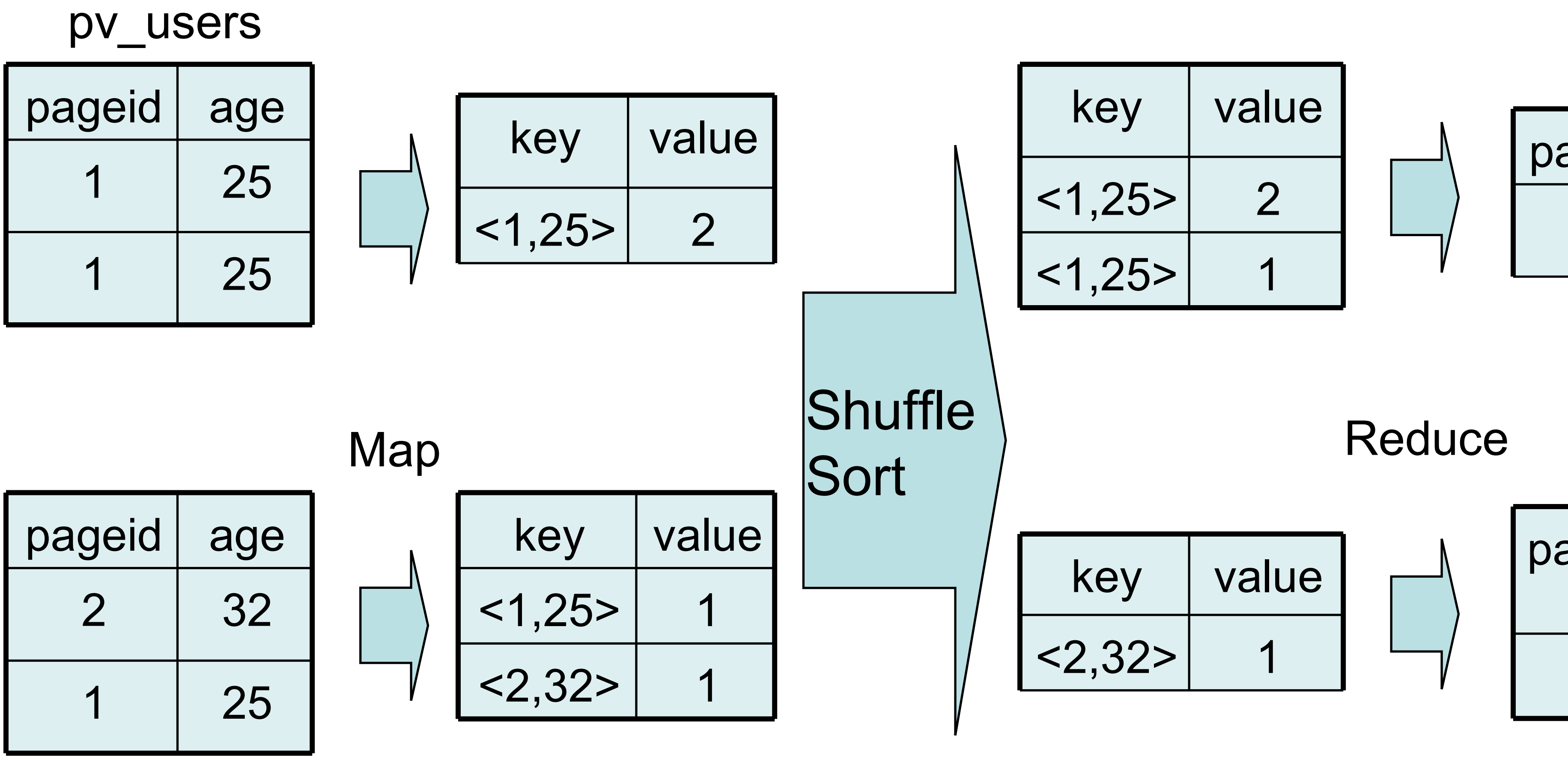
- Column Pruning
  - Also pushed down to scan in columnar storage (RCFILE)
- Predicate Pushdown
  - Not pushed below Non-deterministic functions (eg. rand())
- Partition Pruning
- Sample Pruning
- Handle small files
  - Merge while writing
  - CombinedHiveInputFormat while reading
- Small Jobs
  - SELECT * with partition predicates in the client
- Restartability (Work In Progress)

Group By Optimizations

Group By Implementation

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

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

Group By Optimizations

- Map side partial aggregations
  - Hash-based aggregates
  - Sort-based aggregates
- Load balancing for data skew
  - Using multiple map-reduce jobs

Multi Group By

- Same distinct key across n group-by queries

```
FROM pv_users
  INSERT OVERWRITE TABLE pv_gender_sum
          SELECT gender, count(DISTINCT userid), count(1)
     GROUP BY gender
   INSERT OVERWRITE TABLE pv_age_sum
          SELECT age, count(DISTINCT userid), count(1)
     GROUP BY age
```

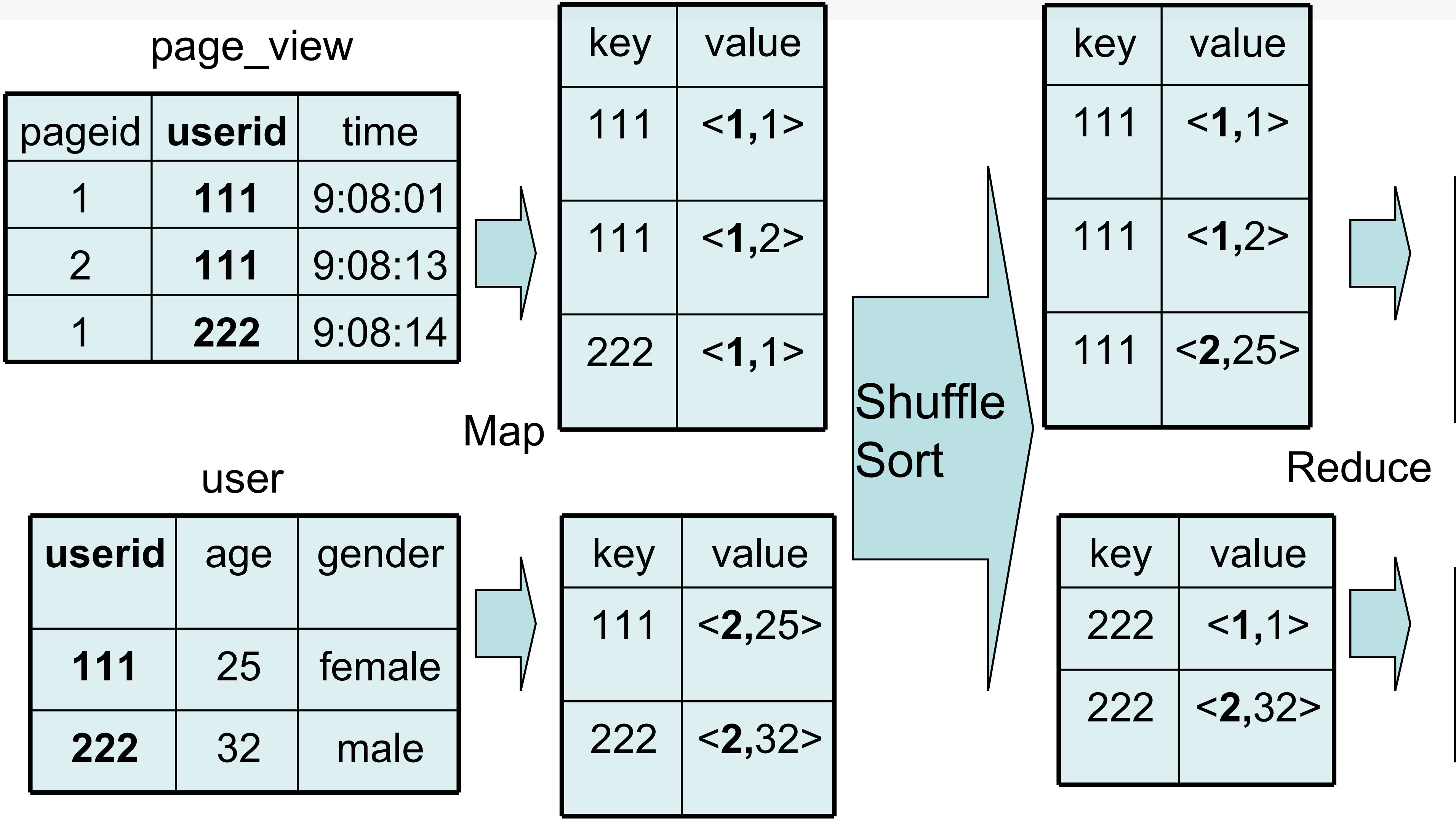
- Single scan of input table, n+1 map-reduce jobs
  - Spray/sort by userid
  - Compute partial aggregates on common reducer
  - Spray by gender and age separately

Join Optimizations

Join Implementation

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

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

Multi-way Join Optimization

-way join with same join key across all joins

```
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);
```

erge into 1 map-reduce job (OUTER joins also)

- Use n tags

map-reduce job instead of n map-reduce jobs

Bag-Join Optimizations

- By default, stream rightmost table
  - Allow hint to specify largest table (which will get streamed)
- Slight skews handled by spilling to local disk on reducer
- Large skews handled by
  - Spilling to hdfs
  - follow-up map-reduce job where skewed table is streamed

Map-Side Join Optimizations

- No reducer needed
- User specified small tables stored in hash tables (backed by local disk) on the mapper
- Bucketized join
  - If two tables are bucketized similarly on join key
- Sort-Merge join (work in progress)

Hive Extensibility Features

Hive is an open system

- Different on-disk data formats
  - Text File, Sequence File, RCFile
- 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 …
  - Configurable serialization formats to transfer data
- User-defined Functions
  - Substr, Trim, From_unixtime …
- User-defined Aggregation Functions
  - Sum, Average …
- User-define Table Functions
  - Explode …

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.**

SerDe

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

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;
  ```

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.**

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);
```

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();
  }
}
```

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;}
  }
  ```

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	supported via UDTF	supported
Speed	Faster (in same process)	Slower (spawns new process)

Powered by Hive

facebook

cnet

digg

Chitika
Turning page views into profits

bizo

Grooveshark

hi5

HubSpot

last.fm

Open Source Community

- Release Candidate Hive-0.5 out on 02/23/2010
- 50+ contributors and growing
- 11 committers
  - 3 external to Facebook
- Available as a sub project in Hadoop
  - http://wiki.apache.org/hadoop/Hive (wiki)
  - http://hadoop.apache.org/hive (home page)
  - http://svn.apache.org/repos/asf/hadoop/hive (SVN repo)
  - ##hive (IRC)
  - Works with hadoop-0.17, 0.18, 0.19, 0.20
- Mailing Lists:
  - hive-{user,dev,commits}@hadoop.apache.org

Future

- Dynamic Inserts into multiple partitions
- More join optimizations
- Persistent UDFs, UDAFs and UDTFs
- Benchmarks for monitoring performance
- IN, exists and correlated sub-queries
- Statistics
- Materialized Views