Big Data
- Apache Hive

Dr. Qing "Matt" Zhang
ITU

Outline

- Motivations
- Hive Introduction
- Hadoop & Hive Deployment and Usage
- Technical Details

Motivations

- Data, data and more data
  - E.g., 200 GB/day in March 2008 → 500+ TB/day in 2012 at Facebook
  - More than 300 PB data in storage
- Fast, faster and real-time
  - Users expect faster response time on fresher data
  - Data used to be available for query in next day now available in minutes.

Motivations

- Queries, queries and more queries
  - More than 200 unique users query on the data warehouse every day
  - 7500+ queries on production cluster/day, mixture of ad-hoc queries and ETL/reporting queries.

Why not Existing Data Warehousing Systems?

- **Cost** of analysis and storage on proprietary systems does not support trends towards more data.
  - Cost based on data size (15 PB costs a lot!)
    - $40-50,000 per terabyte of data stored.
  - Expensive hardware and supports
- **Closed** and proprietary systems

Why not Use Existing Data Warehousing Systems?

- Limited **Scalability** does not support trends towards more data
  - Product designed decades ago (not suitable for petabyte DW)
  - ETL is a big bottleneck
- **Long** product development & release cycle
  - Users requirements changes frequently (agile programming practice)

Lets try Hadoop

- Pros
  - Superior in availability/scalability/ manageability
  - Efficiency not that great, but throw more hardware
  - Partial Availability/resilience/scale more important than ACID

Lets try Hadoop

- Cons: Programmability and Metadata
  - Map-reduce hard to program (users know SQL/bash/python)
  - Need to publish data in well known schemas
- Solution: HIVE

What is HIVE?

- A system for managing and querying structured data built on top of Hadoop
  - Map-Reduce for execution
  - HDFS for storage
  - Metadata in an RDBMS
- A SQL SELECT statement =>MapReduce translator
  - The Hive interpreter runs on a user's machine
  - Takes Hive queries and turns them into Java MapReduce code
  - Submits the code to the cluster
  - Displays the results back to the user

What is HIVE?

- Key Building Principles:
  - SQL as a familiar data warehousing tool
    - HiveQL is based on standard SQL
  - Extensibility – Types, Functions, Formats, Scripts
  - Scalability and Performance
  - Interoperability

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*

Hive is not RDBMS

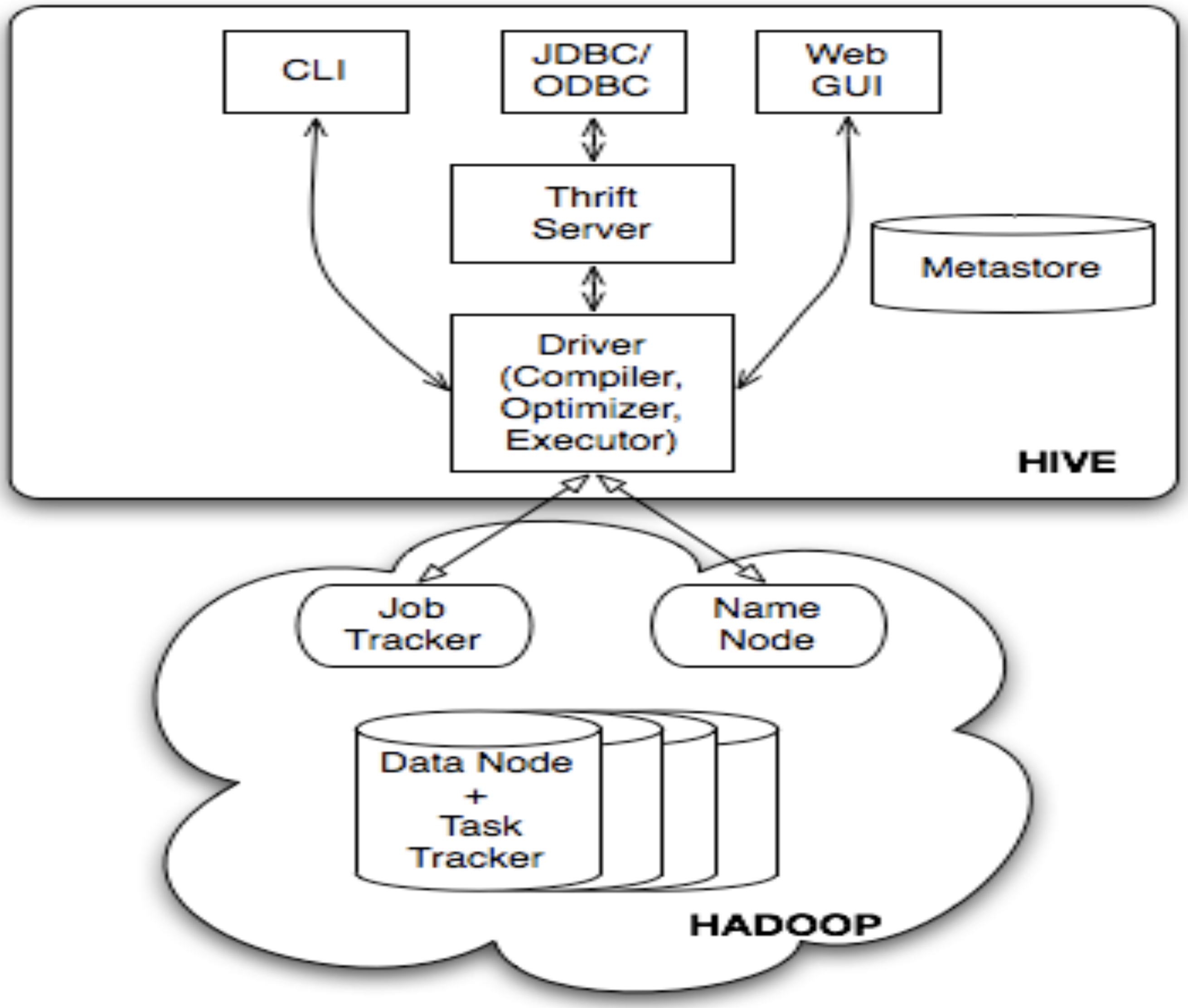
- RDBMSs have many strengths
  - Thousands of simultaneous clients
  - Very fast response time
  - Support for transactions
  - Support for modifying existing records
- It is important to realize that Hive does not somehow turn a Hadoop cluster into an RDBMS
  - The Hive interpreter simply converts HiveQL queries into MapReduce code
  - Even a simple query will usually take many seconds or more to produce a result
  - In very rare cases (e.g. *SELECT * FROM table*), Hive will fetch data from Hadoop without MapReduce

Hive vs RDBMS

	RDBMS	Hive
Language	SQL-92 standard (maybe)	Subset of SQL-92 plus Hive-specific extensions
Update Capabilities	INSERT, UPDATE, and DELETE	INSERT OVERWRITE and INSERT INTO; no UPDATE or DELETE
Transactions	Yes	No
Latency	Sub-second	Minutes or more
Indexes	Any number of indexes, very important for performance	No indexes, data is always scanned (in parallel)*
Data size	TBs	PBs

Hive Architecture

Hive Architecture



Accessing Hive

- Use Hive Shell
  - Invoke hive shell from command line by typing “hive”, will enter its command prompt
  - HiveQL terminates with semicolon (;)

```
hive>
```

- Execute a HiveQL script file
  - *$ hive -f myquery.hql*
- Execute from command line directly
  - *$ hive -e 'select * from users’*
- Access via Hue

Hive Data Store

- Tables are stored in Hive's Warehouse HDFS directory

```
hive> CREATE TABLE employees (name string, age int);
hive> SELECT * FROM employees;
OK
Steve  32
John   28
Brian  35

$ hadoop fs -ls /user/hive/warehouse/employees
drwxr-xr-r . . . /user/hive/warehouse/employees/data1.txt

$ hadoop fs -cat /user/hive/warehouse/employees/data1.txt
Steve  32
John   28
Brian  35
```

Data Interpretation

- Hive 'layers' a table definition on to a directory
  - Schema on read
- The table definition describes the layout of the data files
  - Typically they are delimited (using commas, tabs, or other characters)
  - Hive's default delimiter is the ^A character
  - This does not have to be the case if you use a custom Serializer/Deserializer (SerDe)
- This table definition is saved in Hive's Metastore

Hive Metastore

- The Hive Metastore is held in a set of tables stored in an RDBMS
  - Typically either Derby (the default) or MySQL
- The data held in the Metastore includes
  - Table definitions
    - Table name, column names, column data types, etc.
  - Information on where the table data is stored in HDFS
  - Row format of files in the table
  - Storage format of the files in the table
    - Determines which InputFormat and OutputFormat the MapReduce jobs will use

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
  - Load/Insert/Update/Delete

Hive Query Language

- QL
  - Sub-queries in from clause
  - Equi-joins (including Outer joins)
    - Non-equality conditions are very hard to turn into MR jobs
  - Multi-table Insert/dynamic partition insert
  - Sampling
  - Lateral Views
- Interfaces
  - JDBC/ODBC/Thrift

More on Lateral Views

- Convert list into separate rows

Jsing complex data types

```
• CREATE TABLE t
  (id INT, letters ARRAY<STRING>)…
• SELECT * FROM t;
```

id	letters
1	["a","b","c"]
2	["d","e"]
3	["f","g","h","i"]

LATERAL VIEW

```
• SELECT id, l FROM t LATERAL VIEW
  explode(letters) lets AS l;
```

Original table:

id	letters
1	["a","b","c"]
2	["d","e"]
3	["f","g","h","i"]

id	l
1	a
1	b
1	c
2	d
2	e
3	f
3	g
3	h
3	i

Hive: Making Optimizations Transparent

- Joins:
  - Joins try to reduce the number of map/reduce jobs needed.
  - Memory efficient joins by streaming largest tables.
  - Map Joins
    - User specified small tables stored in hash tables on the mapper
    - No reducer needed

Hive: Making Optimizations Transparent

- Aggregations:
  - 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

Hive: Making Optimizations Transparent

- Storage:
  - Column oriented data formats
  - Column and Partition pruning to reduce scanned data
  - Lazy de-serialization of data
- Plan Execution
  - Parallel Execution of Parts of the Plan

Optimizations

- Column Pruning
  - Discard column which are irrelevant (c, d), select only relevant columns (a, b, e)

```
SELECT a,b FROM T WHERE e < 10;
T contains 5 columns (a,b,c,d,e)
```

  - Enabled by default
    - *hive.optimize.cp = true*

Optimizations

- Predicate Pushdown
  - Filtering rows early in the processing, by pushing down predicates to the scan (if possible)
  - Not pushed below Non-deterministic functions (eg. rand())
- Partition Pruning
  - Reduce list of partitions to be scanned

Optimizations

- Map-side joins
  - The small tables are replicated in all the mappers and joined with other tables
  - No reducer needed
- Join reordering
  - Only materialized and kept small tables in memory
  - This ensures that the join operation does not exceed memory limits on the reducer side

Optimizations

- Handle small files
  - Merge while writing
  - CombinedHiveInputFormat while reading
- Small Jobs
  - SELECT * with partition predicates in the client
  - Local mode execution

Hive: Open & Extensible

- Different on-disk storage(file) formats
  - Text File, Sequence File, etc
- Different serialization formats and data types
  - LazySimpleSerDe, ThriftSerDe
- User-provided map/reduce scripts
  - In any language, use stdin/stdout to transfer data

Hive: Open & Extensible

- User-defined Functions
  - Substr, Trim, From_unixtime
- User-defined Aggregation Functions
  - Sum, Average etc
- User-define Table Functions
  - takes input and writes out multiple rows of data

MapReduce Scripts Examples

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

Hive Usage

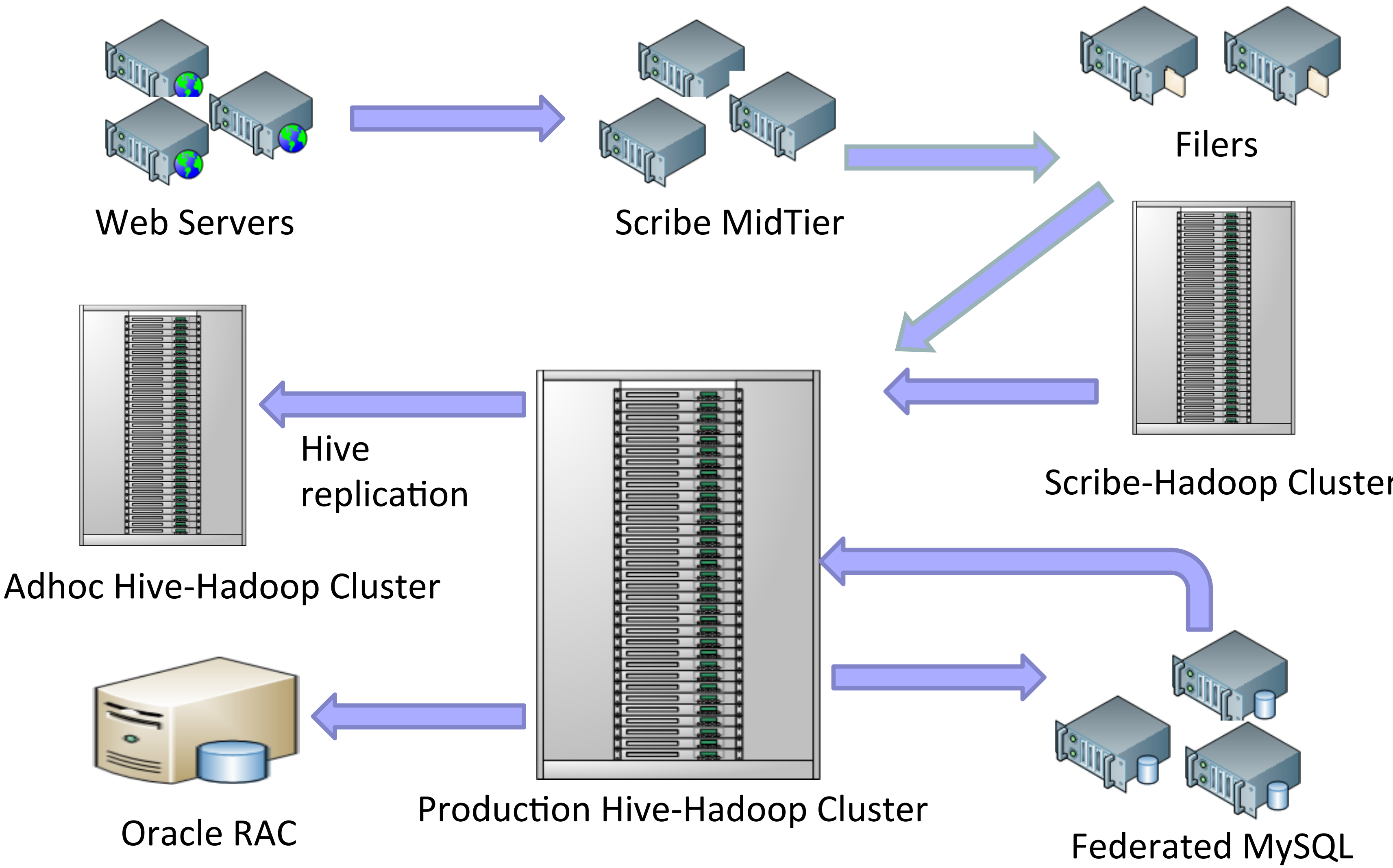
Hive & Hadoop Usage

- Types of Applications:
  - Reporting
    - Eg: Daily/Weekly aggregations of impression/click counts
    - Measures of user engagement
    - Microstrategy reports
  - Ad hoc Analysis
    - Eg: how many group admins broken down by state/country

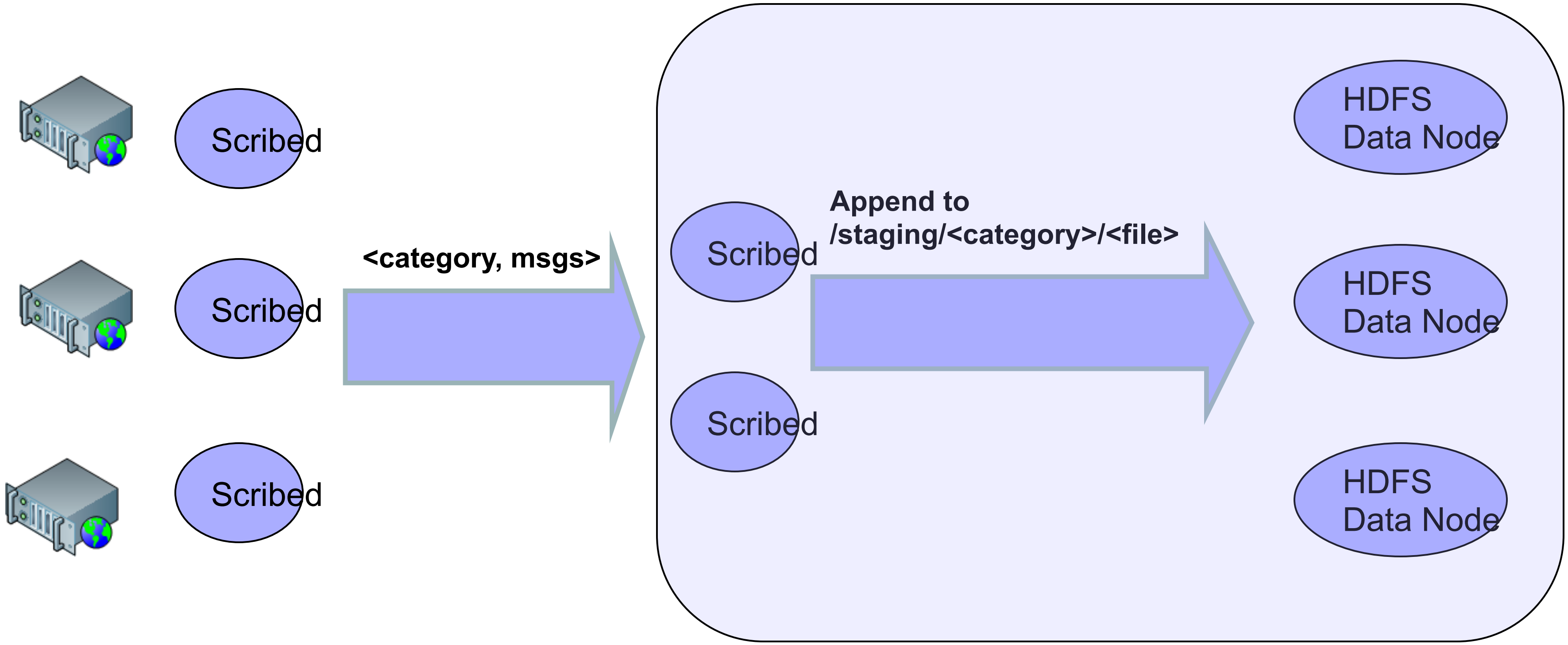
Hive & Hadoop Usage

- Types of Applications:
  - Machine Learning (Assembling training data)
    - Ad Optimization
    - Eg: User Engagement as a function of user attributes
  - Many others

Data Flow Architecture



Scribe-HDFS



Scribe-HDFS

Scribe-HDFS: Near real time Hadoop

- Clusters collocated with the web servers
- Network is the biggest bottleneck
- Typical cluster has about 50 nodes.
- Stats:
  - 50TB/day of raw data logged
  - 99% of the time data is available within 20 seconds

Warehousing at Facebook

- Instrumentation (PHP/Python etc.)
- Automatic ETL
  - Continuous copy data to Hive tables
- Metadata Discovery (CoHive)
- Query (Hive)
- Workflow specification and execution (Chronos)
- Reporting tools
- Monitoring and alerting

More Real-World Use Cases

- Bizo: We use Hive for reporting and ad hoc queries.
- Chitika: for data mining and analysis
- CNET: for data mining, log analysis and ad hoc queries
- Digg: data mining, log analysis, R&D, reporting/analytics
- Grooveshark: user analytics, dataset cleaning, machine learning R&D.

More Real-World Use Cases

- Hi5: analytics, machine learning, social graph analysis.
- HubSpot: to serve near real-time web analytics.
- Last.fm: for various ad hoc queries.
- Trending Topics: for log data normalization and building sample data sets for trend detection R&D.
- VideoEgg: analyze all the usage data

Technical Details

Data Model

- External Tables
  - Point to existing data directories in HDFS
    - Does not use the default location for the table
    - Use existing data in the file
    - Data is NOT deleted from HDFS when the external table is deleted
  - Can create tables and partitions
  - Partition columns just become annotations to external directories

Data Model

- Example: create external table with partitions

```
CREATE EXTERNAL TABLE pvs(uhash 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'
```

Hive QL – Join in Map Reduce

page_view

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

user

uhash	age_bkt	gender
111	B3	female
222	B4	male

Map

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

key	value
111	<**2**,B3>
222	<**2**,B4>

Shuffle Sort

key	value
111	<**1**,1>
111	<**1**,2>
111	<**2**,B3>

key	value
222	<**1**,1>
222	<**2**,B4>

Reduce

pv_users

Pageid	age_bkt
1	B3
2	B3

pageid	age_bkt
1	B4

Hive QL – Group By

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

Hive QL – Group By in Map Reduce

pv_users

pageid	age_bkt
1	B3
1	B3

pageid	age_bkt
2	B4
1	B3

Map

key	value
<1,B3>	2

key	value
<1,B3>	1
<2,B4>	1

Shuffle Sort

key	value
<1,B3>	2
<1,B3>	1

key	value
<2,B4>	1

Reduce

pa

pa

Hive Extensibility Features

Hive is an open system

- Different on-disk storage(file) formats
- Different serialization formats and data types
- User-provided map/reduce scripts
- User-defined Functions
- User-defined Aggregation Functions
- User-define Table Functions

Storage Format Example

- ```
  CREATE TABLE mylog (
      uhash 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.**

Serialization Formats

- SerDe is short for serialization/de-serialization. It controls the format of a row.
- Serialized format:
  - Delimited format (tab, comma, ctrl-a …)
  - Thrift Protocols

Serialization Formats

- De-serialized (in-memory) format:
  - Java Integer/String/ArrayList/HashMap
  - Hadoop Writable classes
  - User-defined Java Classes (Thrift)

SerDe Examples

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

- ```
  CREATE table mylog_rc (
    uhash    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.**

UDF Example

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

Hive Interoperability

Interoperability: Interfaces

- JDBC
  - Enables integration with JDBC based SQL clients
- ODBC
  - Enables integration with Microstrategy
- Thrift
  - Enables writing cross language clients
  - Main form of integration with PHP based Web UI

More

- Use sort properties to optimize query
- IN, exists and correlated sub-queries
- Statistics
- Indexes
- More join optimizations
- Better techniques for handling skews for a given key

Summary

- We identified challenges for big data queries
- We covered the architecture and optimization in Hive
- We discussed the Usages of Hive
- We also covered some technical details of Hive

END