Cloud and Cluster Data Management

HIVE

THANKS TO M. GROSSNIKLAUS

Facebook: Motivation for Hive

- Growth of the Facebook data warehouse
  - 2007: 15TB of net data
  - 2010: 700TB of net data
  - 2011: >30PB of net data
  - 2012: >100PB of net data
- Scalable data analysis used across the company
  - ad hoc analysis
  - business intelligence
  - Insights for the Facebook Ad Network
  - analytics for page owners
  - system monitoring

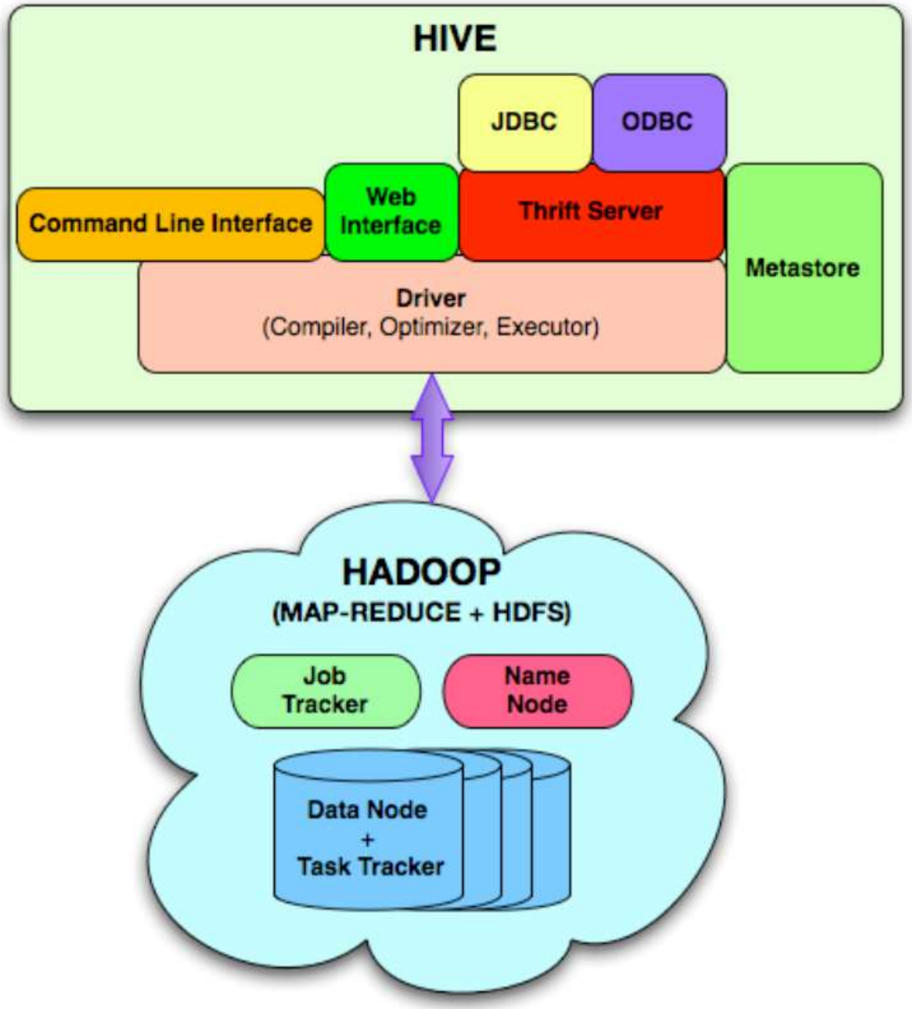
Motivation for Hive (continued)

- Original Facebook data processing infrastructure
  - built using a commercial RDBMS prior to 2008
  - became inadequate as daily data processing jobs took longer than a day
- Hadoop was selected as a replacement
  - pros: petabyte scale and use of commodity hardware
  - cons: using it was not easy for end user not familiar with map-reduce
  - "Hadoop lacked the expressiveness of [..] query languages like SQL and users ended up spending hours (if not days) to write programs for even simple analysis."

Motivation for Hive (continued)

- Hive is intended to address this problem by bridging the gap between RDBMS and Hadoop
  - "Our vision was to bring the familiar concepts of tables, columns, partitions and a subset of SQL to the unstructured world of Hadoop"
- Hive provides:
  - tools to enable easy data extract/transform/load (ETL)
  - a mechanism to impose structure on a variety of data formats
  - access to files stored either directly in HDFS or in other data storage systems such as Hbase, Cassandra, MongoDB, and Google Spreadsheets
  - a simple SQL-like query language
  - query execution via MapReduce

Hive Architecture



- **Clients** use command line interface (CLI), Web UI, or JDBC/ODBC driver
- **HiveServer** provides Thrift and JDBC/ODBC interfaces
- **Metastore** stores system catalogue and metadata about tables, columns, partitions etc.
- **Driver** manages lifecycle of HiveQL statement as it moves through Hive

Figure Credit: "Hive – A Petabyte Scale Data Warehouse Using Hadoop" by A. Thusoo et al., 2010

Data Model

- Schemas are **required** in Hive
- Hive structures data into well-understood database concepts like tables, columns, rows, and partitions
- Primitive types
  - **Integers**: bigint (8 bytes), int (4 bytes), smallint (2 bytes), tinyint (1 byte)
  - **Floating point**: float (single precision), double (double precision)
  - **String**

Complex Types

- Complex types
  - **Associative arrays**: map<key-type, value-type>
  - **Lists**: list<element-type>
  - **Structs**: struct<field-name: field-type, …>
- Complex types are templated and can be composed to create types of arbitrary complexity
  - `li list<map<string, struct<p1:int, p2:int>>>`

Complex Types

- Complex types
  - **Associative arrays**: map<key-type, value-type>
  - **Lists**: list<element-type>
  - **Structs**: struct<field-name: field-type, …>
- Accessors
  - **Associative arrays:** m['key']
  - **Lists:** li[0]
  - **Structs:** s.field-name
- Example:
  - **`li list<map<string, struct<p1:int, p2:int>>`**
  - **`t1.li[0]['key'].p1`** gives the **`p1`** field of the struct associated with the **`key`** of the first array of the list **`li`**

Participation Question 1

- From the list of data models below, what the most similar data model to hive? Briefly explain your answer.
  - Relational
  - Key-value
  - Document
  - Column family
  - Graph

Query Language

- HiveQL is a subset of SQL plus some extensions
  - **`FROM`** clause sub-queries
  - various types of joins: inner, left outer, right outer and outer joins
  - Cartesian products
  - group by and aggregation
  - union all
  - create table as select
- Limitations
  - only equality joins
  - joins need to be written using ANSI join syntax (not in **`WHERE`** clause)
  - no support for inserts in existing table or data partition
  - all inserts overwrite existing data

Query Language

- Hive supports user defined functions written in java
- Three types of UDFs
  - UDF: user defined function
    - Input: single row
    - Output: single row
  - UDAF: user defined aggregate function
    - Input: multiple rows
    - Output: single row
  - UDTF: user defined table function
    - Input: single row
    - Output: multiple rows (table)

Creating Tables

- Tables are created using the **CREATE TABLE** DDL statement
- Example:

```
CREATE TABLE t1(
    st string,
    fl float,
    li list<map<string, struct<p1:int, p2:int>>
);
```

- Tables may be partitioned or non-partitioned (we'll see more about this later)
- Partitioned tables are created using the **PARTITIONED BY** statement

```
CREATE TABLE test_part(c1 string, c2 string)
PARTITIONED BY (ds string, hr int);
```

Inserting Data

- Example

```
INSERT OVERWRITE TABLE t2
SELECT t3.c2, COUNT(1)
FROM t3
WHERE t3.c1 <= 20
GROUP BY t3.c2;
```

  – **OVERWRITE** (instead of **INTO**) keyword to make semantics of insert statement explicit

- Lack of **INSERT INTO**, **UPDATE**, and **DELETE** enable simple mechanisms to deal with reader and writer concurrency
- At Facebook, these restrictions have not been a problem
  – data is loaded into warehouse daily or hourly
  – each batch is loaded into a new partition of the table that corresponds to that day or hour

Inserting Data

- Hive supports inserting data into HDFS, local directories, or directly into partitions (more on that later)
- Inserting into HDFS

```
INSERT OVERWRITE DIRECTORY '/output_dir'
SELECT t3.c2, AVG(t3.c1)
FROM t3
WHERE t3.c1 > 20 AND t3.c1 <= 30
GROUP BY t3.c2;
```

- Inserting into local directory

```
INSERT OVERWRITE LOCAL DIRECTORY '/home/dir'
SELECT t3.c2, SUM(t3.c1)
FROM t3
WHERE t3.c1 > 30
GROUP BY t3.c2;
```

Inserting Data

- Hive supports inserting data into multiple tables/files from a single source given multiple transformations
- Example (corrected from paper):

```
FROM t1
    INSERT OVERWRITE TABLE t2
    SELECT t1.c2, count(1)
    WHERE t1.c1 <= 20
    GROUP BY t1.c2;

    INSERT OVERWRITE DIRECTORY '/output_dir'
    SELECT t1.c2, AVG(t1.c1)
    WHERE t1.c1 > 20 AND t1.c1 <= 30
    GROUP BY t1.c2;

    INSERT OVERWRITE LOCAL DIRECTORY '/home/dir'
    SELECT t1.c2, SUM(t1.c1)
    WHERE t1.c1 > 30
    GROUP BY t1.c2;
```

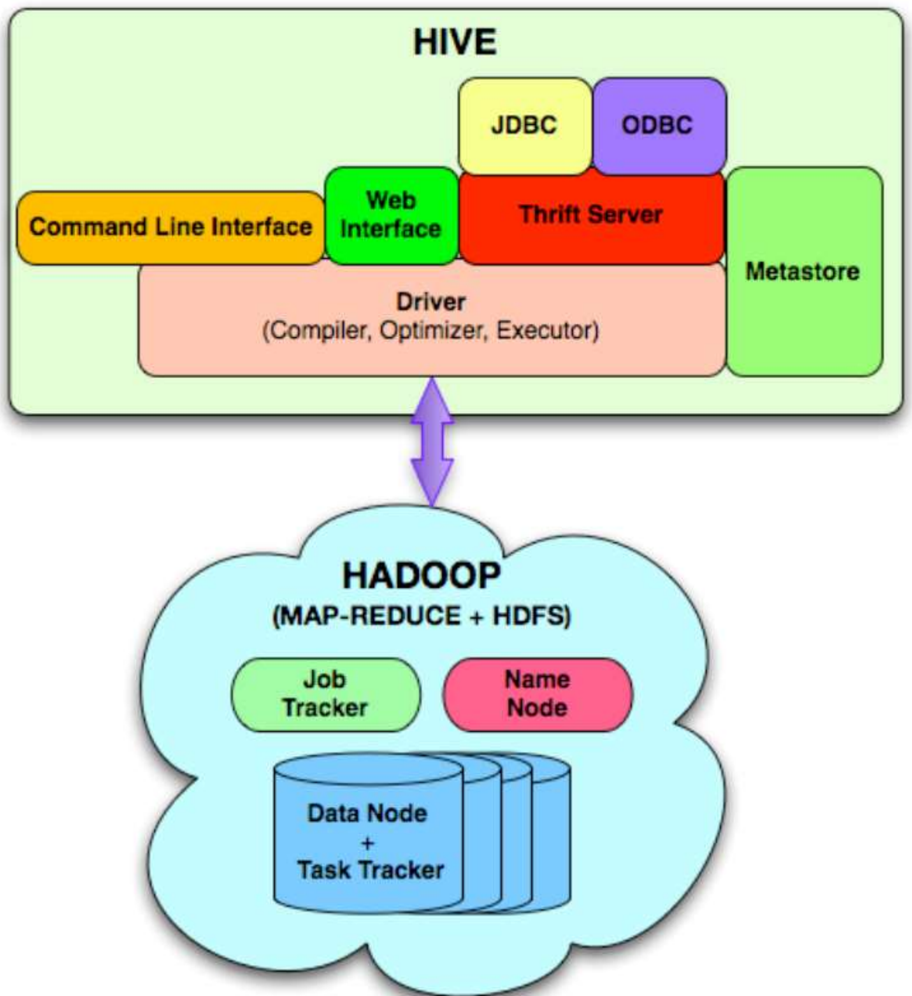
We Gotta Have Map-Reduce!

- HiveQL has extensions to express map-reduce programs
- Example

```
FROM (
  MAP doctext USING 'python wc_mapper.py'
              AS (word, cnt)
  FROM docs CLUSTER BY word
) a
REDUCE word, cnt USING 'python wc_reduce.py';
```

  - **MAP** clause indicates how the input columns are transformed by the mapper UDF (and supplies schema)
  - **CLUSTER BY** clause specifies output columns that are hashed and distributed to reducers
  - **REDUCE** clause specifies the UDF to be used by the reducers

Hive Architecture



- **Clients** use command line interface, Web UI, or JDBC/ODBC driver
- **HiveServer** provides Thrift and JDBC/ODBC interfaces
- **Metastore** stores system catalogue and metadata about tables, columns, partitions etc.
- **Driver** manages lifecycle of HiveQL statement as it moves through Hive

Figure Credit: "Hive – A Petabyte Scale Data Warehouse Using Hadoop" by A. Thusoo et al., 2010

Metastore

- Stores system catalog and metadata about tables, columns, partitions, etc.
- Uses a traditional RDBMS “as this information needs to be served fast”
- Backed up regularly (since everything depends on this)
- Needs to be able to scale with the number of submitted queries (we don’t want thousands of Hadoop workers hitting this DB for every task)
- Only Query Compiler talks to Metastore (metadata is then sent to Hadoop workers in XML plans at runtime)

Data Storage

- Table metadata associates data in a table to HDFS directories
  - **tables**: represented by a top-level directory in HDFS
  - **table partitions**: stored as a sub-directory of the table directory
  - **buckets**: stores the actual data and resides in the sub-directory that corresponds to the bucket's partition, or in the top-level directory if there are no partitions

- Tables are stored under the Hive root directory

```
CREATE TABLE test_table (…);
```

  - Creates a directory like

    <warehouse_root_directory>/test_table

    where <warehouse_root_directory> is determined by the Hive configuration

Partitions

- Partitioned tables are created using the **PARTITIONED BY** clause in the **CREATE TABLE** statement

```
CREATE TABLE test_part(c1 string, c2 int)
PARTITIONED BY (ds string, hr int);
```

- Note that partitioning columns are not part of the table data
- New partitions can be created through an **INSERT** statement or an **ALTER** statement that adds a partition to a table

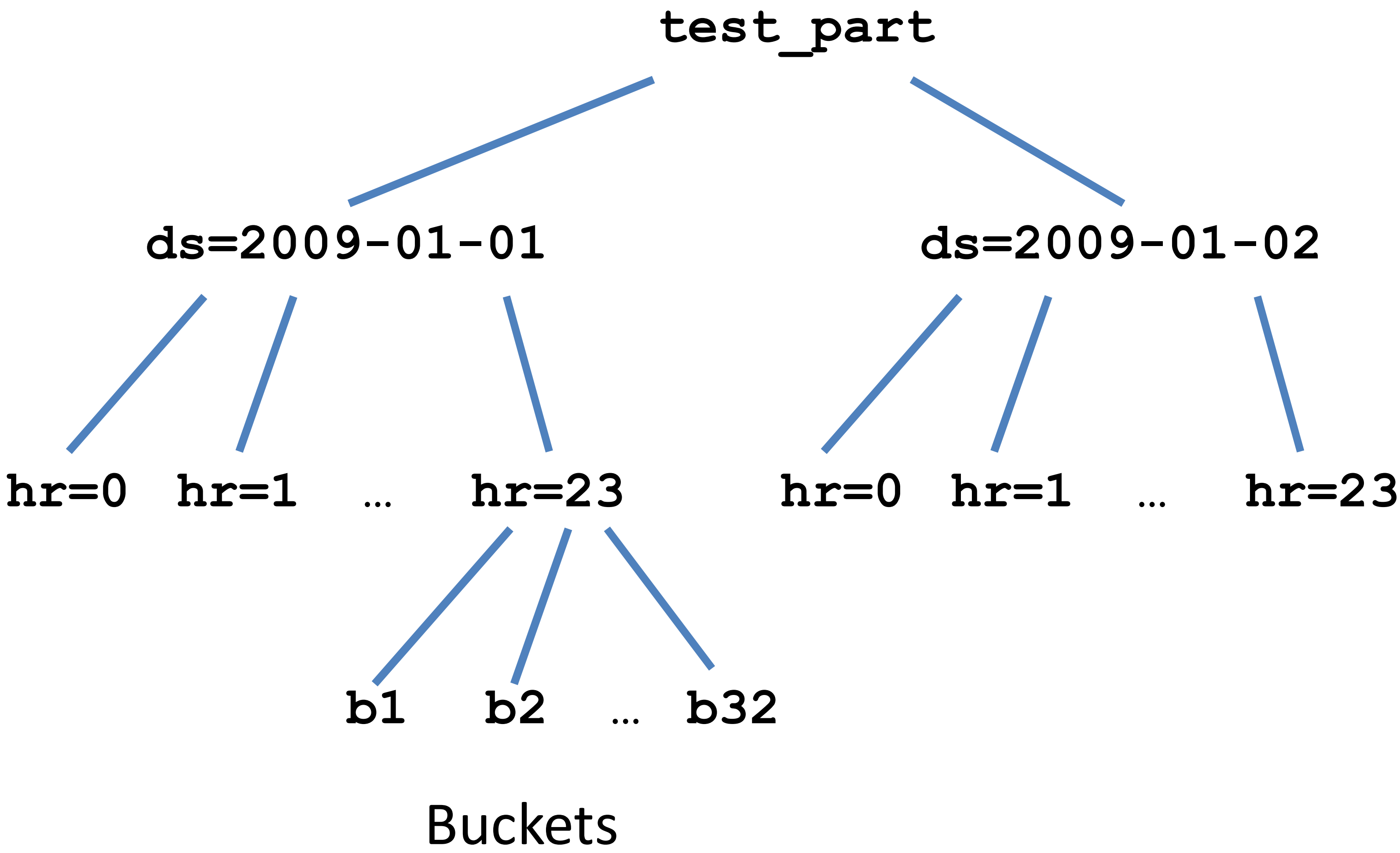
Partition Example

```
INSERT OVERWRITE TABLE test_part
PARTITION(ds='2009-01-01', hr=12)
SELECT * FROM t;
ALTER TABLE test_part
ADD PARTITION(ds='2009-02-02', hr=11);
```

- Each of these statements creates a new directory
  - /…/test_part/ds=2009-01-01/hr=12
  - /…/test_part/ds=2009-02-02/hr=11
- HiveQL compiler uses this information to prune directories that need to be scanned to evaluate a query

```
SELECT * FROM test_part WHERE ds='2009-01-01';
SELECT * FROM test_part
WHERE ds='2009-02-02' AND hr=11;
```

Directory Structure



Participation Question 2

How is partitioning in Hive different from indexing in a DBMS?

Buckets

- A bucket is a file in the leaf level directory of a table or partition
- Users specify number of buckets and column on which to bucket data using the **`CLUSTERED BY`** clause

```
CREATE TABLE test_part(c1 string, c2 int)
PARTITIONED BY (ds string, hr int)
CLUSTERED BY (c1) INTO 32 BUCKETS;
```

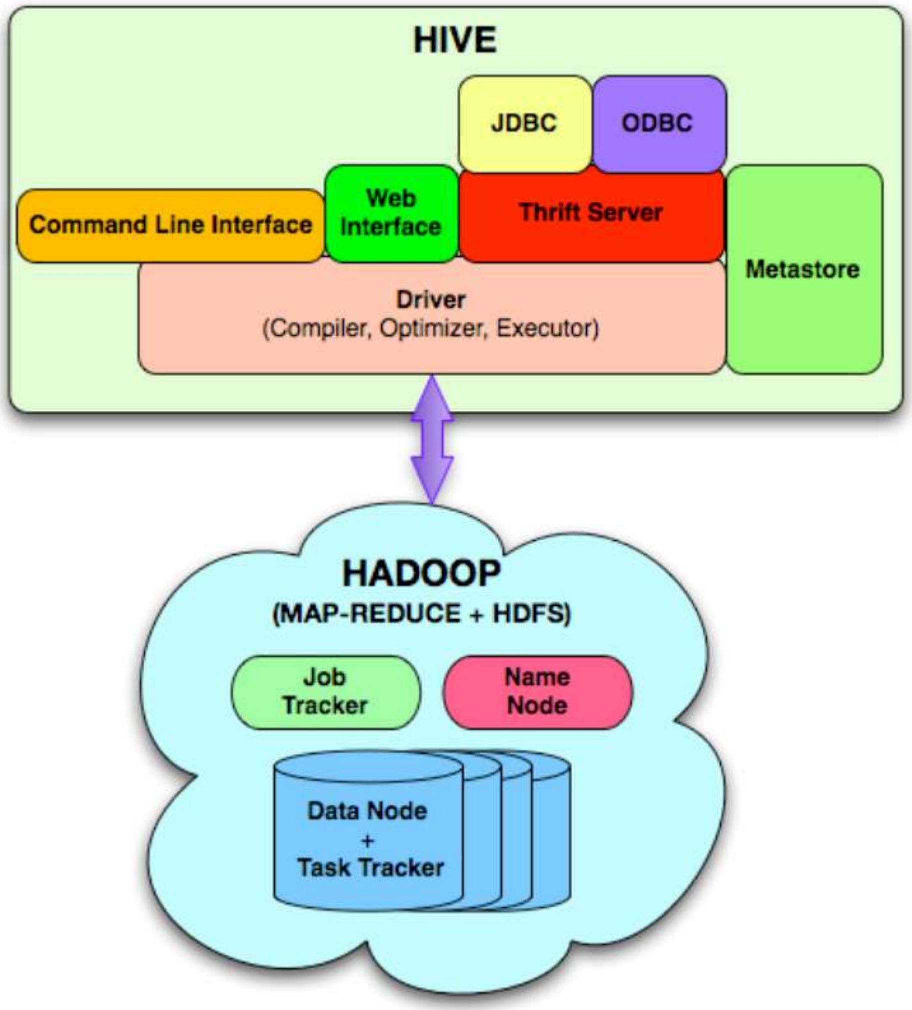
Buckets

- Bucket information is then used to prune data in the case the user runs queries on a sample of data
- Example:

```
SELECT * FROM test_part TABLESAMPLE (2 OUT OF 32);
```

  – This query will only use 1/32 of the data as a sample from the second bucket in each partition

Hive Architecture



- **Clients** use command line interface, Web UI, or JDBC/ODBC driver
- **HiveServer** provides Thrift and JDBC/ODBC interfaces
- **Metastore** stores system catalogue and metadata about tables, columns, partitions etc.
- **Driver** manages lifecycle of HiveQL statement as it moves through Hive

Figure Credit: "Hive – A Petabyte Scale Data Warehouse Using Hadoop" by A. Thusoo et al., 2010

Query Compiler

- Parses HiveQL using Antlr to generate an abstract syntax tree
- Type checks and performs semantic analysis based on Metastore information
- Naïve rule-based optimizations
- Compiles HiveQL into a directed acyclic graph of MapReduce tasks

Optimizations

- Column Pruning
  - Ensures that only columns needed in query expressions are deserialized and used by the execution plan
- Predicate Pushdown
  - Filters out rows in the first scan if possible
- Partition Pruning
  - Ensures that only partitions needed by the query plan are used

Optimizations

- Map side joins
  - If one table in a join is very small it can be replicated in all of the mappers and joined with other tables
  - User must know ahead of time which are the small tables and provide hints to Hive

```
SELECT /*+ MAPJOIN(t2) */ t1.c1, t2.c1
FROM t1 JOIN t2 ON(t1.c2 = t2.c2);
```

- Join reordering
  - Smaller tables are kept in memory and larger tables are streamed in reducers ensuring that the join does not exceed memory limits

Optimizations

- GROUP BY repartitioning
  - If data is skewed in GROUP BY columns the user can specify hints as with MAPJOIN

```
set hive.groupby.skewindata=true;
SELECT t1.c1, sum(t1.c2)
FROM t1
GROUP BY t1;
```

- Hashed based partial aggregations in mappers
  - Hive enables users to control the amount of memory used on mappers to hold rows in a hash table
  - As soon as that amount of memory is used, partial aggregates are sent to reducers.

Execution Engine

- MapReduce tasks are executed in the order of their dependencies
- Independent tasks can be executed in parallel

Hive Usage at Facebook

- Data processing task
  - more than 50% of the workload are ad-hoc queries
  - remaining workload produces data for reporting dashboards
  - range from simple summarization tasks to generate rollups and cubes to more advanced machine learning algorithms
- Hive is used by novice and expert users
- Types of Applications:
  - Summarization
    - Eg: Daily/Weekly aggregations of impression/click counts
  - Ad hoc Analysis
    - Eg: how many group admins broken down by state/country
  - Data Mining (Assembling training data)
    - Eg: User Engagement as a function of user attributes

Hive Usage Elsewhere

- CNET
  - Hive used for data mining, internal log analysis and ad hoc queries
- eHarmony
  - Hive used as a source for reporting/analytics and machine learning
- Grooveshark
  - Hive used for user analytics, dataset cleaning, and machine learning R&D
- Last.fm
  - Hive used for various ad hoc queries
- Scribd
  - Hive used for machine learning, data mining, ad-hoc querying, and both internal and user-facing analytics
- Also: Netflix, Tata Consultancy, Hortonworks, Quoble

References

- C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins: **Pig Latin: A Not-So-Foreign Language for Data Processing.** *Proc. Intl. Conf. on Management of Data (SIGMOD), pp. 1099-1110, 2008.*
- A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, R. Murthy: **Hive – A Petabyte Scale Data Warehouse Using Hadoop.** *Proc. Intl. Conf. on Data Engineering (ICDE), pp. 996-1005, 2010.*