PES UNIVERSITY ONLINE

BIG DATA

Big Data Algorithms – Relational Operations

K V Subramaniam
Computer Science and Engineering

Class Overview

- Relational algebra overview
- Select and Project with MR
- Set Operations
- Join
- Grouping and Aggregation
- Case Study: HIVE

Relational Operations

Summary of Relational Algebra

- Relations
  - Tables; columns = attributes
  - Rows = tuples; $R(A_{1,} A_{2,} \ldots, A_n)$
- Relational Operators
  - Selection: σ_C *(R)* select from *R* according to condition *C*
  - Projection σ_S *(R)* select from *R* subset of attributes *S*
  - Union, intersection, difference
  - Natural join
  - Grouping: partition *R* according to attributes *G*
  - Aggregation: *SUM, COUNT, AVG, MAX, MIN*

Simple Problem

id	Name	Role	Team
1	Virat Kohli	Captain	RCB
2	Gautham Gambhir	Captain	KKR
3	Anil Kumble	Coach	MI
4	Virender Sehwag	Coach	KXIP

1. Write an SQL query to list
   a) All the details for the Coaches.
   b) Only the names of the coaches
   c) Total #coaches.
2. What type of a relational operation are we using?

Simple Problem

- Data in a database is well structured.
- Instead assume that data is stored in a file in HDFS as shown below.

```
1, Virat Kohli, Captain, RCB
2, Gautham Gambhir, Captain, KKR
3, Anil Kumble, Coach, MI
4, Virender Sehwag, Coach, KXIP
```

- Now we need to perform the query.

Select and Project in MapReduce

Selection

- map
  - Read each row t of table
  - Check if it satisfies condition *C*
  - If so, output (*t*,*t*)
- Reduce
  - do nothing

How will the map output look for the query 1a ?

Map

<“3, Anil Kumble, Coach, MI”,”3, Anil Kumble, Coach, MI”>
<“4, Virender Sehwag, Coach,KXIP”,“4, Virender Sehwag, Coach,KXIP”>

Reduce

3, Anil Kumble, Coach, MI
4, Virender Sehwag, Coach,KXIP

Projection

- map
  - Read each row t of table
  - Calculate subset of attributes *t′*
  - Output (*t′*,*t′*)
- reduce
  - Eliminate duplicates
    - (*t′*,[*t′*,*t′*,*t′*]) -> (*t′*,*t′*)

Map

<“Anil Kumble”,” Anil Kumble”>

<“Virender Sehwag”,“Virender Sehwag”>

Reduce

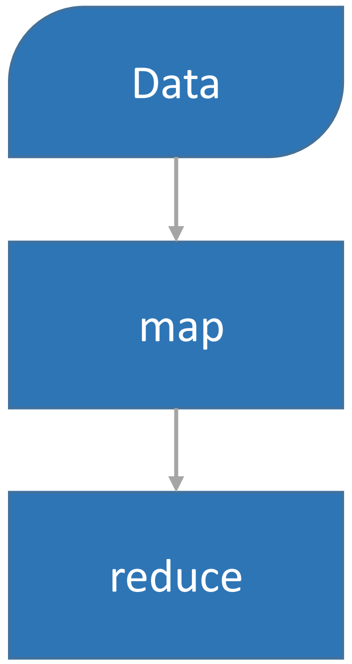
Anil Kumble

Virender Sehwag

Relational Operations requiring two input files – Set operations

Class Exercise - Union…

- Union *R U S*
  - *R* and *S* have the same structure
  - Output records in *R* or *S*
  - *We need to solve a problem here*
- The basic problem :
  - Need to read in 2 input files
  - produce 1 output file
- Reading multiple input files
  - Same problem in Matrix-vector multiplication
  - Need to read in matrix ***M*** and vector ***v***

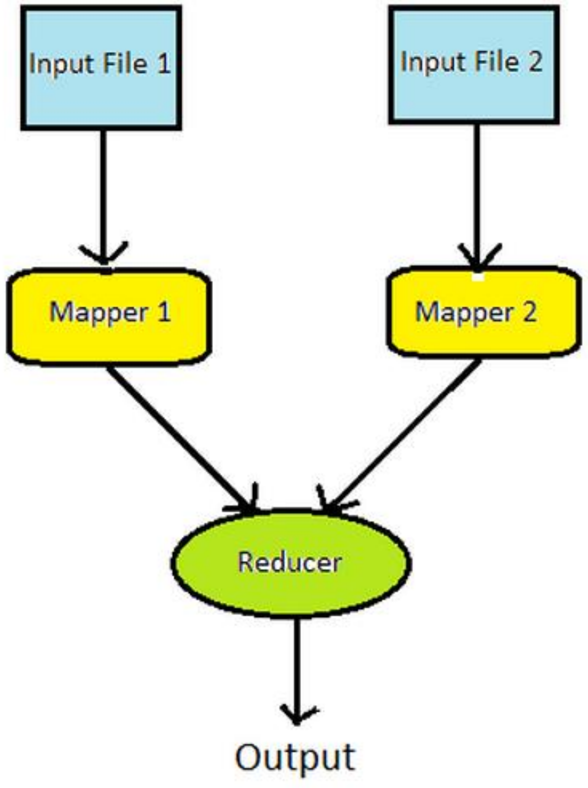


One Solution

- map reads in one file
- If we have multiple input files, we can try to combine them into one file
  - This can be used as input to map
- Problem: multiple input files can be combined in many ways
  - We can have all the records of one file followed by all the records of another file
  - Or merge the two files and sort the records
  - Files may not have the same structure

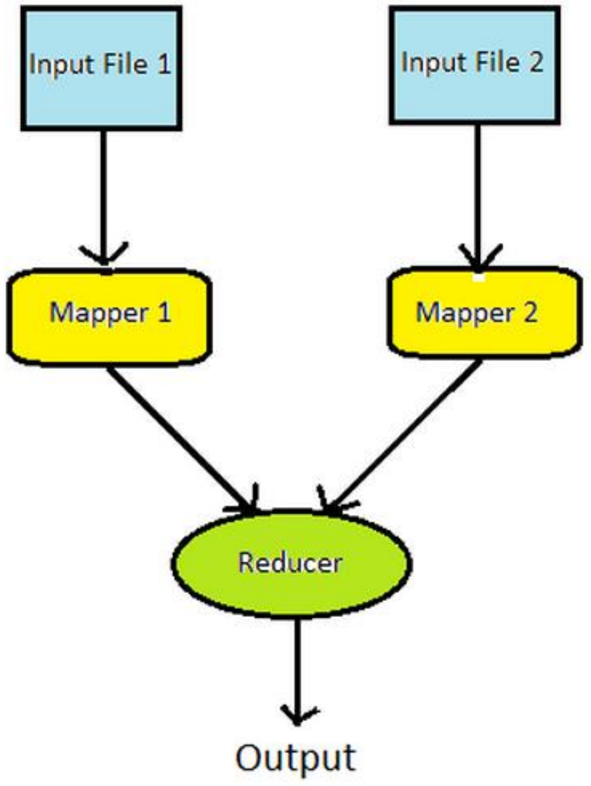
The Hadoop Solution

- Use map reduce itself to merge the files
- Example
  - We have two mappers
  - Each mapper reads in one file
  - Each mapper writes a key-value pair
  - The reducer uses the keys to merge the files
  - This output can be used as input to subsequent map reduce passes



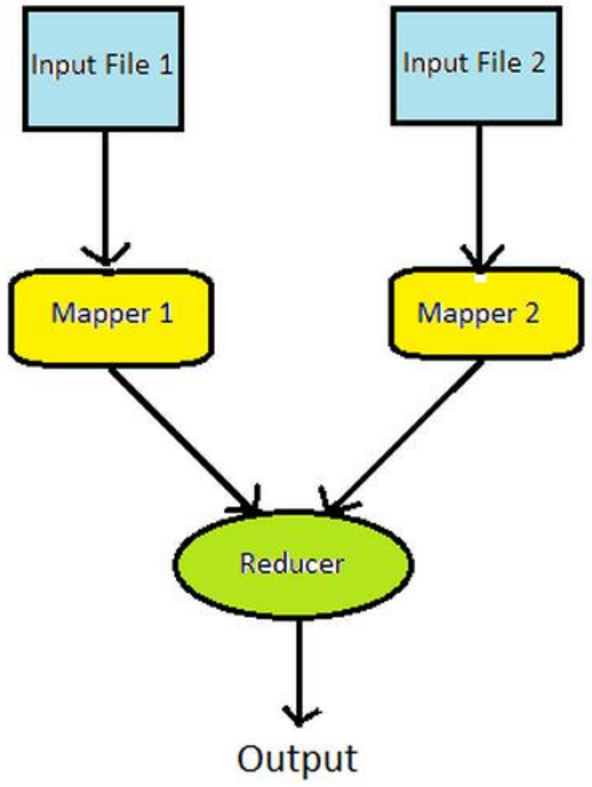
Union…

- Union *R U S*
  - Need to read in 2 input files and produce 1 output file
  - Need to have 2 mappers reading different input files
  - Can be done using *MultipleInput* option in Hadoop
    - MultipleInputs.
      - addInputPath (job,
      - new Path (args[0]),
      - TextInputFormat.class,
      - Mapper1.class);



...Union

- mapper 1
  - Read each row of table *R*
  - Output (*t*,*t*)
- mapper 2
  - Read each row of table *S*
  - Output (*t*,*t*)
- reducer
  - Eliminate duplicates if any
    - (*t*,[*t*,*t*]) -> (*t*,*t*)



Sample Problem

- Given the following input for two files
  - File 1
    - A
    - B
    - C
    - D
  - File 2
    - A
    - E
    - F
    - C

- Show (for the Union algorithm)
  - Input and output of mapper 1
  - Input and output of mapper 2
  - Input and output of reducer

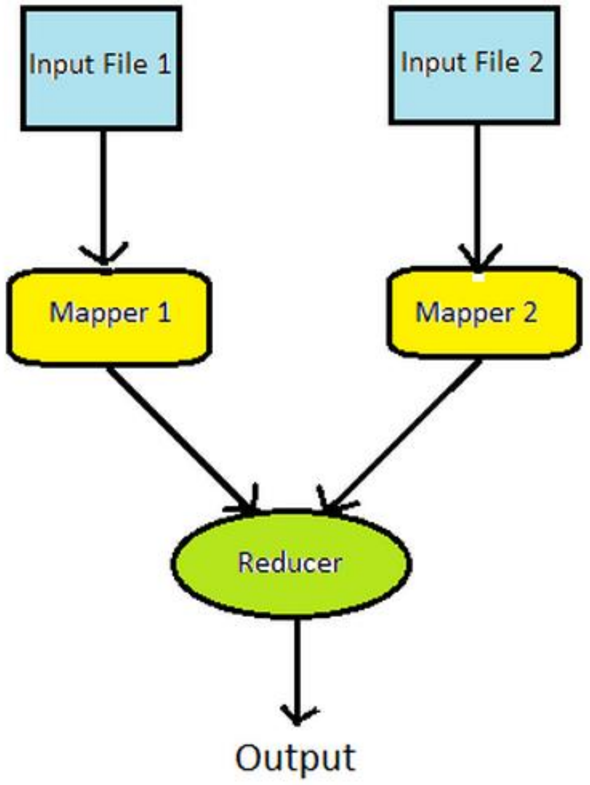
Solution

- Mapper 1 output
  - A , A
  - B, B
  - C, C
  - D, D
- Mapper 2 output
  - A, A
  - E, E
  - F, F
  - C, C

- Reducer Input
  - A, [A, A]
  - B, B
  - C, [C, C]
  - D, D
  - E, E
  - F, F
- Reducer Output
  - A
  - B
  - C
  - D
  - E
  - F

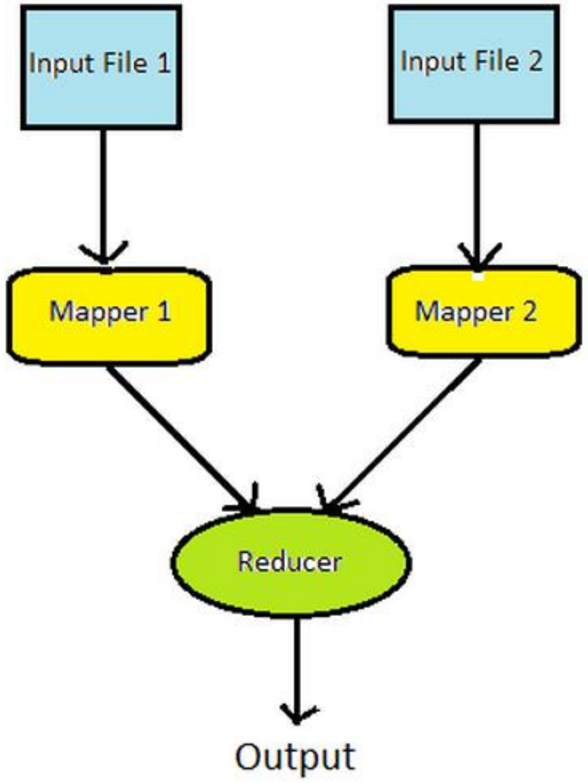
Intersection

- Compute $R \cap S$
- mapper 1
  - Read each row of table R
  - Output (t,t)
- mapper 2
  - Read each row of table S
  - Output (t,t)
- reducer
  - Output only duplicates
    - $(t,[t,t])$ -> (t,t)



Difference

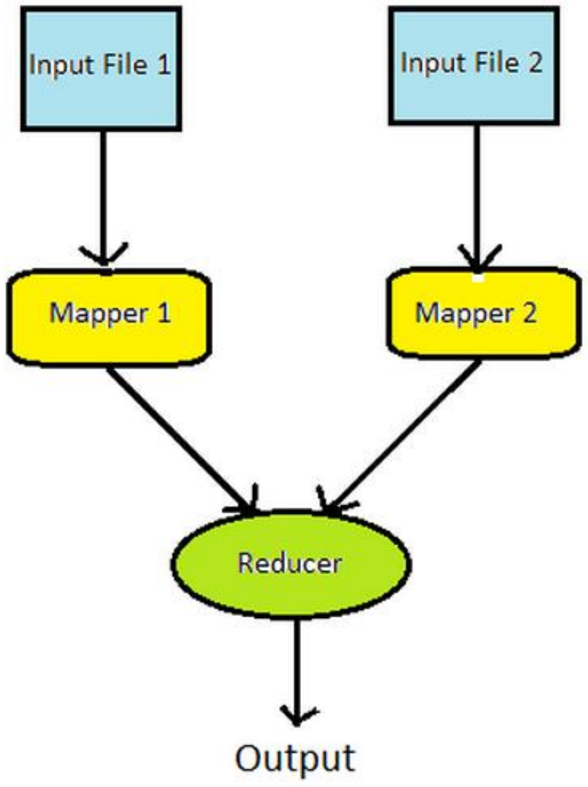
- Compute *R* – S
    - All rows in *R,* not in *S*
- mapper 1
    - Read each row of table *R*
    - Output (*t*,*R*)
- mapper 2
    - Read each row of table *S*
    - Output (*t*,*S*)
- reducer
    - Output only
        - (*t*,[R]) -> (*t*,*t*)



Join

Natural Join

- Join R and S on attributes B
  - A, C are the other attributes in R,S
- mapper 1
  - Read (a,b) of R, output (b,(R,a))
- mapper 2
  - Read (b,c) of S, output (b,(S,c))
- reducer
  - for each pair b,(R,a)) and (b,(S,c)), output (a,b.c)



Problem

- Given the following input for two files
  - Table Employee E(Name, age)
    - Gabbar 35
    - Viru 37
    - Jai 33
    - Baldev 44
    - Basanti 31
  - Table Dept D(Name, Dept)
    - Gabbar Bandit
    - Viru Hero
    - Jai Hero
    - Baldev Police
    - Basanti Heroine

- Show (for the Natural Join algorithm)
  - Input and output of mapper 1
  - Input and output of mapper 2
  - Input and output of reducer

Solution

- Mapper 1 Output
  - Gabbar, (E, 35)
  - Viru, (E, 37)
  - Jai, (E, 33)
  - Baldev, (E, 44)
  - Basanti, (E, 31)
- Mapper 2 Output
  - Gabbar, (D, Bandit)
  - Viru, (D, Hero)
  - Jai, (D, Hero)
  - Baldev (D, Police)
  - Basanti (D, Heroine

- Reducer Input
  - Gabbar, (E, 35), (D, Bandit)
  - Viru, (E, 37), (D, Hero)
  - Jai, (E, 33), (D, Hero)
  - Baldev, (E, 44), (D, Police)
  - Basanti, (E, 31), (D, Heroine)
- Reducer Output
  - Gabbar, 35, Bandit
  - Viru, 37, Hero
  - Jai, 33, Hero
  - Baldev, 44, Police
  - Basanti, 31, Heroine)
-

Grouping and Aggregation

Grouping and Aggregation...

- For relation *R*(*A*,*B*,*C*) group by A and aggregate by function *f(*B)
  - Social networking site has a relationship *Friends* (*User, Friend, Date of friendship*)
  - Grouping by *User* and aggregating by *COUNT* (*Friend*) produces a table
    - First column is the *User*
    - Second column is the count of *Friends* for the *User*

...Grouping and Aggregation

- map
  - for each line (a,b,c) (e.g., User, Friend, Date)
  - Output (a,b) (e.g., User, Friend)
- reduce
  - Aggregate (a, [b1 , b2 , b3 , ...]) into (a, f(b1 , b2 , b3 , ...))

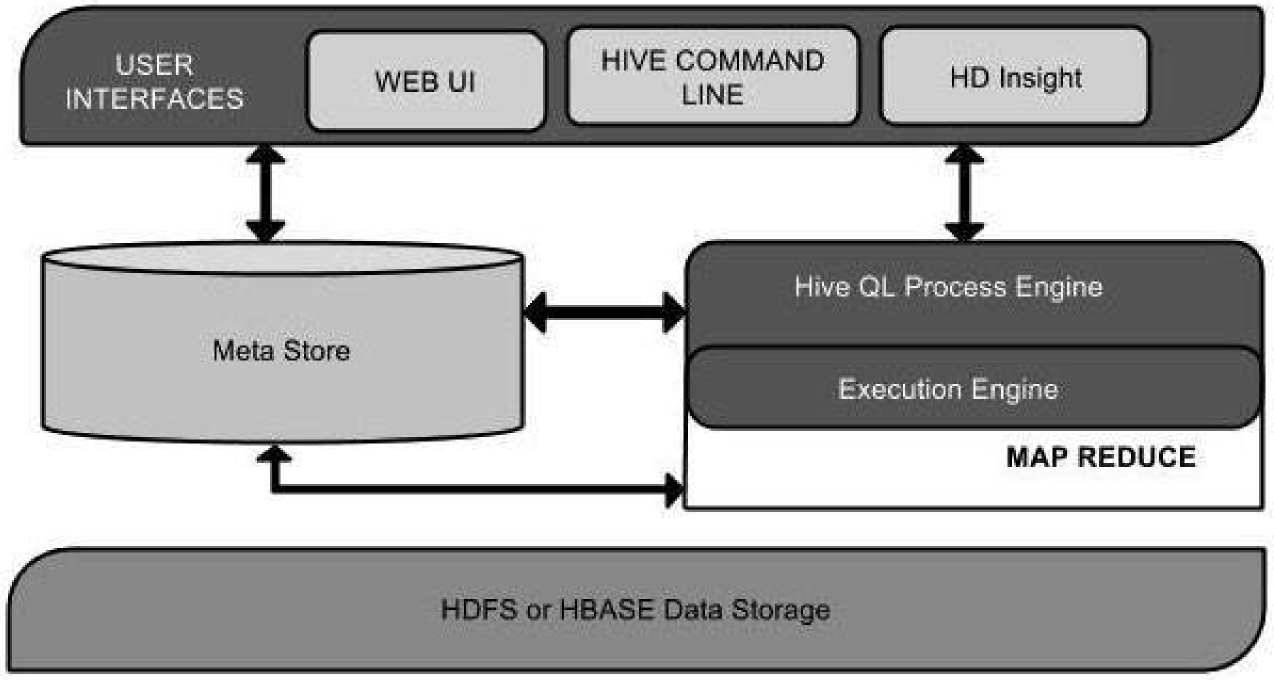
Case Study : HIVE

What is HIVE

- A system for querying and managing structured data built on top of Map/Reduce and Hadoop
- Facebook data
  - Structured logs with rich data types (structs, lists and maps)
  - A user base wanting to access this data in the language of their choice
  - A lot of traditional SQL workloads on this data (filters, joins and aggregations)
  - Other non SQL workloads

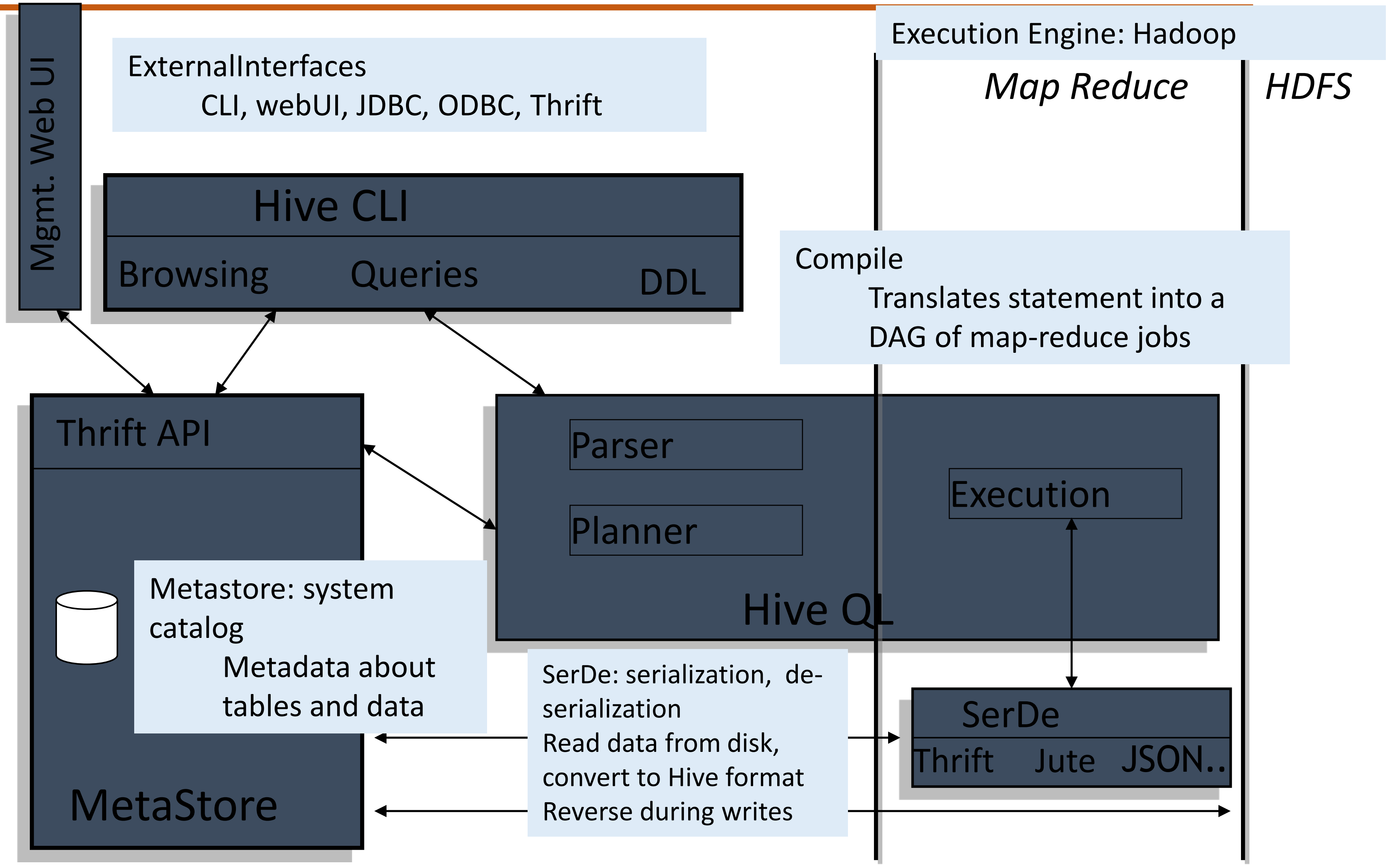
What is HIVE

- Data is stored in HDFS
- Meta Store – for schema
- User submits SQL query
  - Converted to MR jobs



http://ewh.ieee.org/r6/scv/computer/nfic/2008/Facebook%20Hive%20by%20Ashish%20Thusoo.ppt

HIVE Components


Execution Engine: Hadoop
Map Reduce
HDFS
Mgmt. Web UI
ExternalInterfaces
CLI, webUI, JDBC, ODBC, Thrift
Hive CLI
Browsing
Queries
DDL
Compile
Translates statement into a DAG of map-reduce jobs
Thrift API
Parser
Planner
Execution
Hive QL
Metastore: system catalog
Metadata about tables and data
MetaStore
SerDe: serialization, de-serialization
Read data from disk, convert to Hive format
Reverse during writes
SerDe
Thrift
Jute
JSON..

Data Model

1. Hive data stored as HDFS files

/hive/clicks

/hive/clicks/ds=2008-03-25

/hive/clicks/ds=2008-03-25/0

2. Table – similar to table in relational database
   - Mapped to HDFS directory
   - Data for table clicks is in the directory /hive/clicks
   - For scalability, tables divided into multiple files and directories

3. Partition
   - Part of table partitioned on values of columns
   - Implemented as a n HDFS subdirectory
   - If clicks is partitioned on column ds Data with a particular ds value 2008-03-25 will be stored in /hive/clicks/ds=2008-03-25
   - If further divided on ctry value US will be stored in the directory /hive/clicks/ds=2008-03-25/ctry=US.

4. Bucket
   - Subdivision of partition
   - Divided based on hash of a specified column

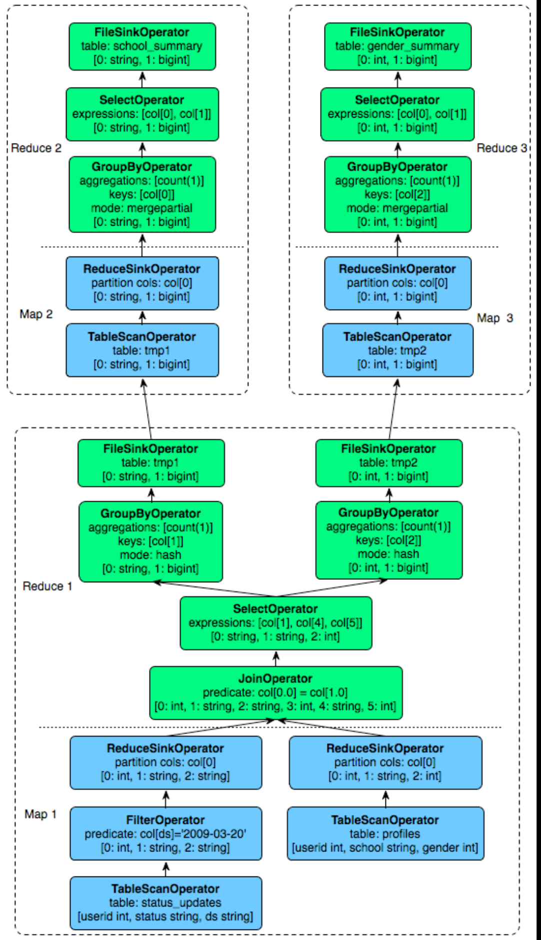
Thusoo, Ashish, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. "Hive: a warehousing solution over a map-reduce framework." *Proceedings of the VLDB Endowment* 2, no. 2 (2009): 1626-1629

HIVE compilation example

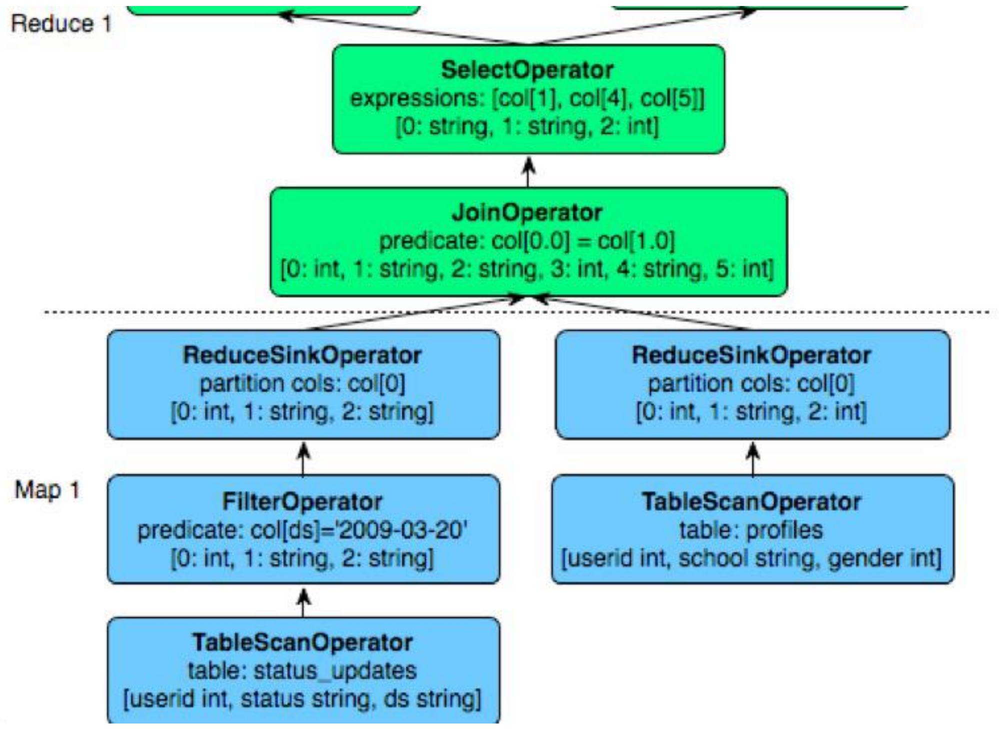
```
FROM (SELECT a.status, b.school, b.gender
      FROM status_updates a JOIN profiles b
           ON (a.userid = b.userid and
               a.ds='2009-03-20' )
      ) subq1
INSERT OVERWRITE TABLE gender_summary
                       PARTITION(ds='2009-03-20')
SELECT subq1.gender, COUNT(1) GROUP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

Thusoo, Ashish, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. "Hive: a warehousing solution over a map-reduce framework." *Proceedings of the VLDB Endowment* 2, no. 2 (2009): 1626-1629

HIVE compilation example



```
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
```


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

Thusoo, Ashish, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. "Hive: a warehousing solution over a map-reduce framework." *Proceedings of the VLDB Endowment* 2, no. 2 (2009): 1626-1629

THANK YOU

K V Subramaniam
Dept. of Computer Science and Engineering
subramaniamkv@pes.edu