Hive

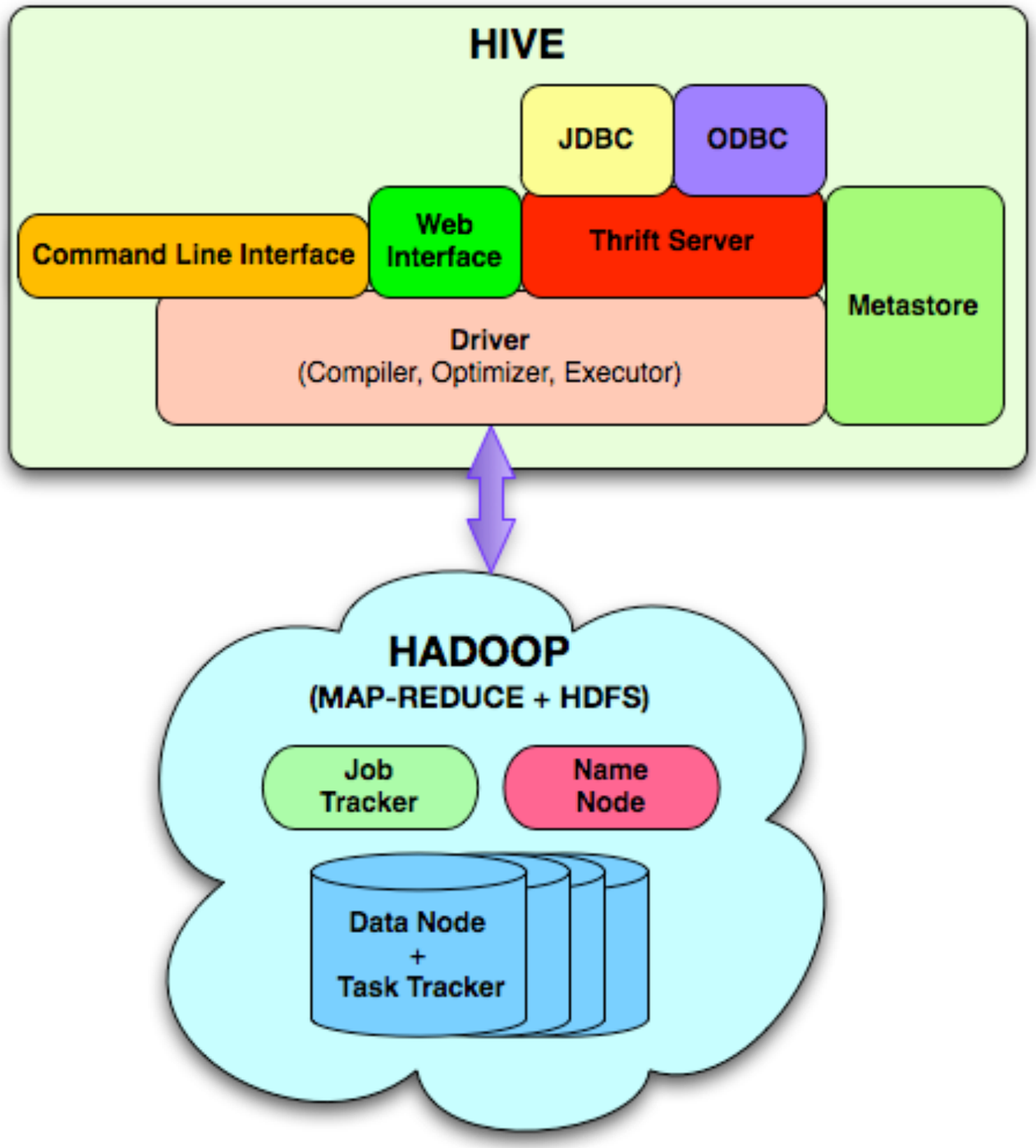
Arend Hintze

Hive

- Programming framework build on top of Hadoop
- Created to make it possible for analysis with strong SQL skills (and little to no Java programming) to run queries on large volumes of data
- Developed for Facebook

https://cwiki.apache.org/confluence/display/Hive/Tutorial

Hive architecture



BASICS

- Separation of data from the table schema:
    - Data is stored on the HDFS (raw files, not a table)
    - Schema is stored on you local directory
- Unlike relational databases (SQL), you can’t simply update or delete an individual record
    - HADOOP is meant to do batch processing and thus does not allow “transactions” (update, remove, add data)

Example File Movies

- id (int)
- name (string)
- year (int)
- rating (float)
- length (int)

atomic data types in HIVE

Category	Type	Description	Literal examples
Primitive	`TINYINT`	1-byte (8-bit) signed integer, from -128 to 127	`1`
	`SMALLINT`	2-byte (16-bit) signed integer, from -32,768 to 32,767	`1`
	`INT`	4-byte (32-bit) signed integer, from -2,147,483,648 to 2,147,483,647	`1`
	`BIGINT`	8-byte (64-bit) signed integer, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	`1`
	`FLOAT`	4-byte (32-bit) single-precision floating-point number	`1.0`
	`DOUBLE`	8-byte (64-bit) double-precision floating-point number	`1.0`
	`BOOLEAN`	true/false value	`TRUE`
	`STRING`	Character string	`'a', "a"`

complex data types HIVE

Category	Type	Description	Literal examples
Complex	ARRAY	An ordered collection of fields. The fields must all be of the same type.	`array(1, 2)`
	MAP	An unordered collection of key-value pairs. Keys must be primitives; values may be any type. For a particular map, the keys must be the same type, and the values must be the same type.	`map('a', 1, 'b', 2)`
	STRUCT	A collection of named fields. The fields may be of different types.	`struct('a', 1, 1.0)`

```
CREATE TABLE complex (
  col1 ARRAY<INT>,
  col2 MAP<STRING, INT>,
  col3 STRUCT<a:STRING, b:INT, c:DOUBLE>
);
```

Creating Tables

```
CREATE TABLE movies (
   >id BIGINT,
   >name STRING,
   >year BIGINT,
   >rating FLOAT,
   >length BIGINT)
   >ROW FORMAT DELIMITED FIELDS
   >TERMINATED BY ‘,’
   >STORED AS TEXTFILE;
```

this will only make the table!

Getting data into tables

```
LOAD DATA INPATH
  >‘/user/cloudera/movies/movies_data.csv’
  > OVERWRITE INTO TABLE movies;
```

warning: Will remove the file from the original location in the HDFS

create and load at the same time

```
CREATE TABLE movies (
   >id BIGINT,
   >name STRING,
   >year BIGINT,
   >rating FLOAT,
   >length BIGINT)
   >ROW FORMAT DELIMITED FIELDS
   >TERMINATED BY ‘,’
   >STORED AS TEXTFILE;
   >location ‘/user/cloudera/movies’;
```

checking on tables

```
SHOW tables;
```

```
DESCRIBE movies;
```

getting rid of tables

```
DROP TABLE tableName;
```

ALTERING TABLES

```
ALTER TABLE moveis RENAME movies;
ALTER TABLE movies ADD COLUMNS (comment
STRING);
```

simple queries:

```
SELECT * FROM movies WHERE year=1990;
```

the usual logic applies

```
SELECT * FROM movies WHERE rating>3.0;
```

GROUP

```
SELECT COUNT(1) FROM movies GROUP BY year;
```

COUNT

```
SELECT COUNT(1) FROM movies WHERE rating>3.0;
```

storing results in tables

```
CREATE TABLE watchable AS
 SELECT * FROM movies WHERE rating>4.5;

INSERT OVERWRITE TABLE watchable
SELECT * FROM movies WHERE rating>4.0;
```

Partitioned Tables

- instead of using indexes on columns to speed up execution -> partitions

- HIVE organizes tables into partitions to create course grained parts

- using partitions can make it faster to answer queries on slices of data

Partitions

- create table by using PARTITIONED BY clause
- a table can have one of more partition columns
- don’t mix partitioned and unpartitioned tables!

Partitioning Example

```
<147 hadoop1:~ >cat states/MI.txt
A 1
A 2
A 3
B 10
B 20
B 30
B 40
C 5
C 6
C 7
<148 hadoop1:~ >cat states/IL.txt
F 4
F 6
G 6
G 7
G 9
H 1
H 2
H 1
```

creating partitioned table

```
hive> create table users (
    >    name STRING,
    >    value INT)
    > partitioned by (state STRING)
    > row format delimited
    > fields terminated by ' '
    > stored as textfile;
```

```
hive> load data local inpath 'states/MI.txt' into table users
    > partition (state = 'MI');

hive> load data local inpath 'states/OH.txt' into table users
    > partition (state = 'OH');

hive> load data local inpath 'states/IL.txt' into table users
    > partition (state = 'IL');
```

the partitioned table:

```
hive> show partitions users;
OK
state=IL
state=MI
state=OH
Time taken: 0.183 seconds
hive> describe users;
OK
name    string
value   int
state   string
```

```
hive> select * from users;
OK
F       4       IL
F       6       IL
G       6       IL
G       7       IL
G       9       IL
H       1       IL
H       2       IL
H       1       IL
A       1       MI
A       2       MI
A       3       MI
B       10      MI
B       20      MI
B       30      MI
B       40      MI
C       5       MI
C       6       MI
C       7       MI
D       4       OH
D       5       OH
E       3       OH
E       5       OH
E       6       OH
E       7       OH
```

using partitions in a query:

```
hive> select count(*) from users where state = 'MI';
10
```

```
hive> select state, count(distinct name)
    > from users
    > group by state;

IL      3
MI      3
OH      2
```

inner Joins

```
hive> SELECT * FROM sales;
Joe     2
Hank    4
Ali     0
Eve     3
Hank    2
hive> SELECT * FROM things;
2       Tie
4       Coat
3       Hat
1       Scarf
```

We can perform an inner join on the two tables as follows:

```
hive> SELECT sales.*, things.*
    > FROM sales JOIN things ON (sales.id = things.id);
Joe     2       2       Tie
Hank    2       2       Tie
Eve     3       3       Hat
Hank    4       4       Coat
```

Hive can integrate your own mapper and reducer

```
FROM (
      FROM pv_users
      MAP pv_users.userid, pv_users.date
      USING 'map_script'
      AS dt, uid
      CLUSTER BY dt) map_output

   INSERT OVERWRITE TABLE pv_users_reduced
       REDUCE map_output.dt, map_output.uid
       USING 'reduce_script'
       AS date, count;
```

tableName

key, value

mapper file name

The key value tuple has to be tab-delimited - remember the streaming API?

mapper script example

```
import sys
import datetime

for line in sys.stdin:
line = line.strip()
userid, unixtime = line.split('\t')
weekday = datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()
print ','.join([userid, str(weekday)])
```

Hive allows you to customize the merge/shuffle

```
FROM (
    FROM pv_users
    MAP pv_users.userid, pv_users.date
    USING 'map_script'
    AS c1, c2, c3
    DISTRIBUTE BY c2
    SORT BY c2, c1) map_output

INSERT OVERWRITE TABLE pv_users_reduced

    REDUCE map_output.c1, map_output.c2, map_output.c3
    USING 'reduce_script'
    AS date, count;
```