Big data: architectures and data analytics

HIVE

Based on "Hive" by prof. Torlone
http://torlone.dia.uniroma3.it/bigdata/L5-Hive.pdf

Motivation

- Analysis of data made by both engineering and non-engineering people
- The data are growing fast
- Current RDBMS can NOT handle it
- Traditional solutions are often not scalable, expensive and proprietary

Motivation

- Hadoop supports data-intensive distributed applications.
- But…
  - You have to use MapReduce model
    - Hard to program
    - Not Reusable
    - Error prone
  - For complex jobs: multiple stage of MapReduce jobs
  - Alternative and more efficient tools exist today (e.g., Spark) but they are not easy to use
  - Most users know Java/SQL/Bash

Possible solution

- Make the unstructured data looks like tables regardless how it really lay out
- SQL-like lased queries can be executed directly against these tables
- Generate specify execution plan for these queries
- Hive!
- A big data management system storing structured data on Hadoop file system
- Provide an easy query these data by executing Hadoop-based plans
- Today just a part of a large category of solutions called "SQL over Hadoop"

What is Hive?

- An infrastructure built on top of Hadoop for providing data summarization, query, and analysis.
  - Structure
  - Access to different storage
  - HiveQL (very close to a subset of SQL)
  - Query execution via MapReduce,Tez, and Spark
- Key Building Principles:
  - SQL is a familiar language
  - Extensibility – Types, Functions, Formats, Scripts
  - Performance

Application scenario

- No real-time queries (high latency)!
- No row level updates!
- Not designed for online transaction processing!
- Best use: batch jobs over large sets of append-only data
  - Log processing
  - Data/Text mining
  - Business intelligence
- However: current version allows a form of ACID transaction at the row level
  - One application can add rows while another reads from the same partition without interfering with each other

7

Data Units

- Databases
  - Containers of tables and other data units
- Tables
  - Homogeneous units of data which have the same schema.
  - Basic type columns (Int, Float, Boolean)
  - Complex type: Lists / Maps / Arrays

8

Data Units

- Partitions
  - Each Table can have one or more partition columns (or partition keys).
  - Each unique value of the partition keys defines a partition of the Table.
  - Queries can run on the relevant partition thereby speeding up the analysis significantly.
  - Partition columns are virtual columns, they are not part of the data itself

9

Type System

- Primitive types
  - Integers: TINYINT, SMALLINT, INT, BIGINT
  - Boolean: BOOLEAN
  - Floating point numbers: FLOAT, DOUBLE
  - String: STRING
  - Date string: TIMESTAMP
- Complex types
  - Structs: {a INT; b INT}
  - Maps: M['group']
  - Arrays: ['a', 'b', 'c'], A[1] returns 'b'

10

Examples

```
CREATE TABLE demo1(id INT, name STRING);

CREATE TABLE employees (
   name STRING,
   salary FLOAT,
   subordinates ARRAY<STRING>,
   deductions MAP<STRING, FLOAT>,
   address STRUCT<street:STRING, city:STRING,
                  state:STRING, zip:INT>
);
```

Terminators

'\n'	Between rows (records)
^A ('\001')	Between fields (columns)
^B ('\002')	Between ARRAY and STRUCT elements and MAP key-value pairs
^C ('\003')	Between each MAP key and value

12

The actual file format

```
CREATE TABLE employees (
   name STRING,
   salary FLOAT,
   subordinates ARRAY<STRING>,
   deductions MAP<STRING, FLOAT>,
   address STRUCT<street:STRING, city:STRING,
                  state:STRING, zip:INT>
);
```

13

The actual file format

- John Doe^A100000.0^AMary Smith^BTodd Jones^AFederal Taxes^C0.2^BState Taxes^C0.05^BInsurance^C.1^A1 Michigan Ave.^BChicago^BIL^B60600\n

14

Partitioning

- CREATE TABLE message_log (
  status STRING,
  msg STRING,
  hms STRING) PARTITIONED BY (year INT, month INT, day INT);

Partitioning

- The partition column is virtual
- Separate directories for each partition column
- On disk:
  - message_log/year=2015/month=01/day=01/
  - …
  - message_log/year=2015/month=01/day=31/
    message_log/year=2015/month=02/day=01/
  - …
  - message_log/year=2015/month=12/day=31/

16

Advantages of partitioning

- Speed queries by limiting scans to the correct partitions specified in the WHERE clause:
- SELECT * FROM message_log
  WHERE year = 2015 AND month = 01 AND day = 31;

17

Other DDL Operations

- CREATE TABLE sample (foo INT, bar STRING) PARTITIONED BY (ds STRING);
- SHOW TABLES 's*';
  - Returns the list of tables starting with 's'
- DESCRIBE sample;
  - Returns the schema and characteristics of sample
- ALTER TABLE sample ADD COLUMNS (new_col INT);
  - Adds columns

18

Other DDL Operations

- DROP TABLE sample;
  - Drops table sample

External tables

- You can create an (external) table from data that are already in HDFS
- CREATE EXTERNAL TABLE employees (name STRING,…) LOCATION '/data/employees/input';
  - LOCATION is a directory
    - Hive will read all the files it contains and "load" their content into the defined table
- The table data are not deleted when you drop the table
- The table metadata are deleted from the Metastore

Locations

- The locations can be local, in HDFS, or in S3
  - LOCATION 'file://path/to/data';
  - LOCATION 'hdfs://server:port/path/to/data';
  - LOCATION 's3n://mybucket/path/to/data';
- Joins can join table data from any such sources

Loading data: examples

- Loading from a local file that contains two columns separated by ctrl-a into sample table
  - LOAD DATA LOCAL INPATH './sample.txt' OVERWRITE INTO TABLE sample PARTITION (ds='2015-02-24');
- Loading from HDFS
  - LOAD DATA INPATH '/user/hive/sample.txt' OVERWRITE INTO TABLE sample PARTITION (ds='2015-02-24');

Loading data: examples

- Loading from CSV
  - LOAD DATA LOCAL INPATH './sample.txt' OVERWRITE INTO TABLE sample PARTITION (ds='2015-02-24') ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

Create and import

- CREATE LOCAL TABLE sample (foo INT, bar STRING) PARTITIONED BY (ds STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE location './sample.txt';

Select statements: examples

- SELECT ymd, symbol FROM stocks WHERE exchange = 'NASDAQ' AND symbol = 'AAPL' ;
  - Queries involving projection require a MR job
- SELECT * FROM stocks WHERE exchange = 'NASDAQ' AND symbol = 'AAPL' ;
  - If a * query is over partitions: no MR job is required!
- A * query without the WHERE clause does not require MR as well

25

Storing the results

- Select column 'foo' from all rows of partition ds=2015-02-24
  - SELECT foo FROM sample WHERE ds='2015-02-24';
- Store the result into a local directory
  - INSERT OVERWRITE LOCAL DIRECTORY '/tmp/hdfs_out' SELECT * FROM sample WHERE ds='2015-02-24';
- Store the result in HDFS:
  - INSERT OVERWRITE DIRECTORY '/tmp/hive-sample-out' SELECT * FROM sample;

26

Aggregations and groups

- SELECT count(*) FROM stocks WHERE exchange = 'NASDAQ' AND symbol = 'AAPL' ;
- SELECT avg(price_close) FROM stocks WHERE exchange = 'NASDAQ' AND symbol = 'AAPL' ;
- SELECT year(ymd), avg(price_close) FROM stocks WHERE exchange = 'NASDAQ' AND symbol = 'AAPL` GROUP BY year(ymd);

27

Aggregations and Groups

- SELECT MAX(foo) FROM sample;
- SELECT ds, COUNT(*), SUM(foo) FROM sample GROUP BY ds;

28

Joins

- SELECT s.ymd, s.symbol, s.price_close, d.dividend FROM stocks s JOIN dividends d ON s.ymd = d.ymd AND s.symbol = d.symbol WHERE s.ymd > '2015-01-01';
- Only equality (x = y) conditions are allowed
- Put the biggest table last!
- Reducer will stream the last table and buffer (put in the cache) the others

29

Join examples

- CREATE TABLE customer (id INT, name STRING, address STRING);
- CREATE TABLE order_cust (id INT, cus_id INT, prod_id INT, price INT);
- SELECT * FROM customer c JOIN order_cust o ON (c.id=o.cus_id);
- SELECT c.id, c.name, c.address, ce.exp FROM customer c JOIN (SELECT cus_id, sum(price) AS exp FROM order_cust GROUP BY cus_id) ce ON (c.id=ce.cus_id);

30

Types of Join

- Four kinds supported:
  - Inner Joins
  - Outer Joins (left, right, full)
  - Semi Joins (not discussed here)
  - Map-side Joins (an optimization of others)

An example of outer join

- SELECT s.ymd, s.symbol, s.price_close, d.dividend FROM (SELECT ymd, symbol, price_close FROM stocks WHERE exchange = 'NASDAQ' AND symbol = 'AAPL') s LEFT OUTER JOIN (SELECT ymd, symbol, dividend FROM dividends WHERE exchange = 'NASDAQ' AND symbol = 'AAPL') d ON s.ymd = d.ymd AND s.symbol = d.symbol;

Map-side Joins

- Join tables in the mapper
- Optimization that eliminates the reduce step
- Useful if all but one table is small
- SELECT s.ymd, s.symbol, s.price_close, d.dividend FROM dividends d JOIN stocks s ON s.ymd = d.ymd AND s.symbol = d.symbol;
- If all but one table is small enough, the mapper can load the small tables in memory (through distributed cache) and do the joins there, rather than invoking an expensive reduce step.
- It can't be used with RIGHT/FULL OUTER joins

Built-in Functions

- Work on a single row
- Mathematical
  - round, floor, ceil, rand, exp…
- Collection
  - size, map_keys, map_values, array_contains
- Type Conversion
  - cast
- Date
  - from_unixtime, to_date, year, datediff…

Built-in Functions

- Conditional
  - if, case, coalesce
- String
  - length, reverse, upper, trim…
- To retrieve the complete list of functions
  - hive> SHOW FUNCTIONS;
    - !
    - !=
    - … abs acos
    - …

Built-in Functions

- hive> DESCRIBE FUNCTION year;
  - year(date) - Returns the year of date
- hive> DESCRIBE FUNCTION EXTENDED year;
  - year(date) - Returns the year of date
  - date is a string in the format of 'yyyy-MM-dd HH:mm:ss' or 'yyyy-MM-dd'.
  - Example:
    - SELECT year('2009-03-07') FROM src LIMIT 1;
    - 2009

Examples

- SELECT year(ymd) FROM stocks;
- SELECT year(ymd), avg(price_close) FROM stocks WHERE symbol = 'AAPL' GROUP BY year(ymd);

Table Generating Function

- SELECT explode(subordinates) AS subs FROM employees;
  - Generates zero or more output rows for each input row.
  - Takes an array (or a map) as an input and outputs the elements of the array (map) as separate rows.
  - Effectively a new table
- More flexible way to use TGFs:
  - SELECT name, sub FROM employees LATERAL VIEW explode(subordinates) subView AS sub;

Example

pageAds

pageid (string)	adid_list(Array<int>)
"front_page"	[1, 2, 3]
"contact_page"	[3, 4, 5]

SELECT pageid, adid
FROM pageAds LATERAL VIEW explode(adid_list) subA AS adid;

subA

pageid (string)	adid (int)
"front_page"	1
"front_page"	2
...	...

User-defined function (UDF)

```
// Java
import org.apache.hadoop.hive.ql.exec.UDF;
public class NowUDF extends UDF {
  public long evaluate() {
      return System.currentTimeMillis();
  }
}
```

- You compile this Java code and build a jar file…

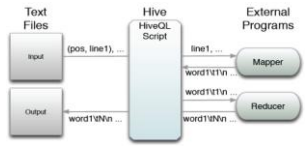
UDF usage

- Include the jar in the HIVE_CLASSPATH using ADD JAR
- Create a TEMPORARY FUNCTION
- -- HQL

```
ADD JAR path_to_jar;
…
CREATE TEMPORARY FUNCTION now AS
  'com…NowUDF';
SELECT epoch_millis FROM … WHERE epoch_millis <
  now() …;
```


Calling external Map/Reduce Scripts

- Hive provides a technique for calling external programs to perform map and reduce operations



Pros

- Pros
  - A easy way to process large scale data
  - Support SQL-based queries
  - Provide more user defined interfaces to extend
  - Programmability
  - Efficient execution plans for performance
  - Interoperability with other database tools

43

Cons

- Cons
  - Potential inefficiency
  - No easy way to append data
    - Updates are available starting in Hive 0.14
  - Files in HDFS are rarely updated
- Future work
  - Views / Variables
  - More operator
    - In/Exists semantic

44