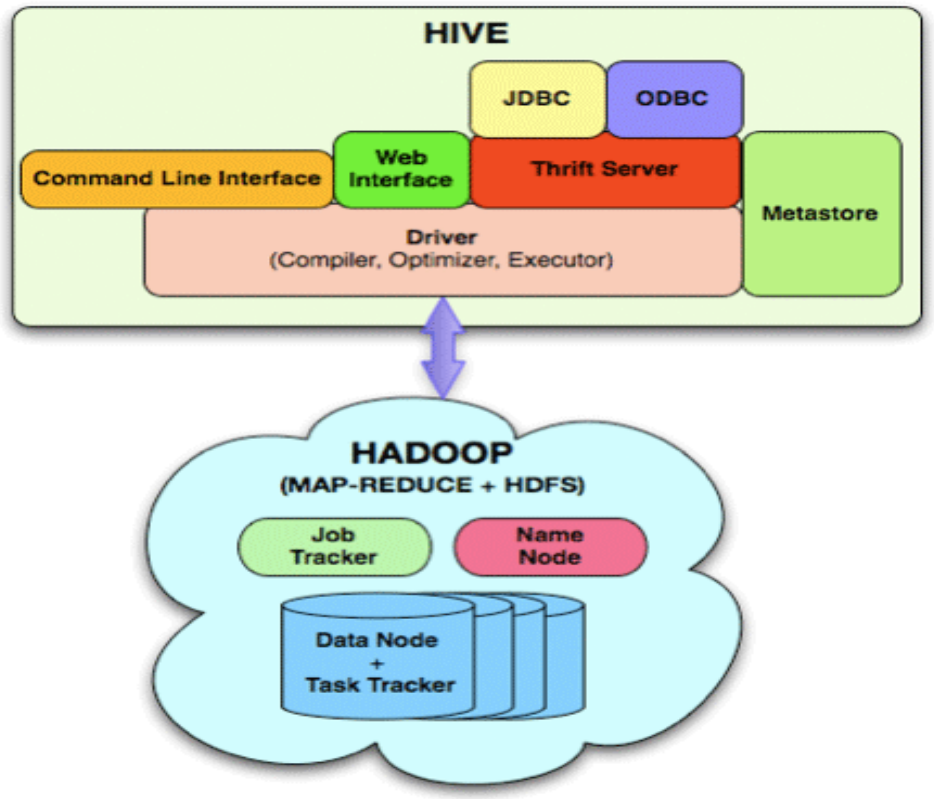
Contents

Hive Documentation

Hive Introduction

The Apache Hive ™ data warehouse software facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time this language also allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL.

Architecture



The above diagram shows the basic Hadoop Hive architecture. Primarily The diagram represents CLI (Command Line Interface), JDBC/ODBC and Web GUI (Web Graphical User Interface).This represents when user comes with CLI (Hive Terminal) it directly connected to Hive Drivers, When User comes with JDBC/ODBC(JDBC Program) at that time by using API(Thrift Server) it connected to Hive driver and when the user comes with Web GUI(Ambari server) it directly connected to Hive Driver.

The hive driver receives the tasks (Queries) from user and send to Hadoop architecture. The Hadoop architecture uses name node, data node, job tracker and task tracker for receiving and dividing the work what Hive sends to Hadoop (Mapreduce Architecture).

Features of Hive

Apache Hive supports analysis of large datasets stored in Hadoop's HDFS and compatible file systems such as Amazon S3 filesystem. It provides an SQL-like language called HiveQL with schema on read and transparently converts queries to map/reduce, Apache Tez and Spark jobs. All three execution engines can run in Hadoop YARN. To accelerate queries, it provides indexes, including bitmap indexes.

By default, Hive stores metadata in an embedded Apache Derby database and other client/server databases like MySQL can optionally be used.

Currently, there are four file formats supported in Hive, which are TEXTFILE, SEQUENCEFILE, ORC and RCFILE. Apache Parquet can be read via plugin in versions later than 0.10 and natively starting at 0.13.

Other features of Hive include:

- Indexing to provide acceleration, index type including compaction and Bitmap index as of 0.10, more index types are planned.
- Different storage types such as plain text, RCFile, HBase, ORC, and others.
- Metadata storage in an RDBMS, significantly reducing the time to perform semantic checks during query execution.
- Operating on compressed data stored into the Hadoop ecosystem using algorithms including DEFLATE, BWT, snappy, etc.
- Built-in user defined functions (UDFs) to manipulate dates, strings, and other data-mining tools.

Advantages over RDBMS

- The biggest advantage is developer productivity, though this can come at the expense of execution speed (mostly latency) and efficiency (high throughput via brute force).
- Hive offers custom UDF packages and makes it easy to contribute new UDFs.
- You can explicitly control the map and reduce transform stages, and even route them through simple scripts written quickly in languages like Python or Perl.
- You can also work with many different serialization formats, making it easy to ingest nearly any kind of data.
- You can connect your Hive queries to several Hadoop packages for statistical processing, including Apache Mahout, RHipe, RHive or RHadoop. For these reasons Hive can improve developer productivity when working with challenging data formats or complex analytical tasks.

References

- http://www.quora.com/What-are-the-advantages-of-Hive-over-SQL
- http://www.dummies.com/how-to/content/the-architecture-of-apache-hive.html
- http://www.hadooptpoint.com/hadoop-hive-architecture/

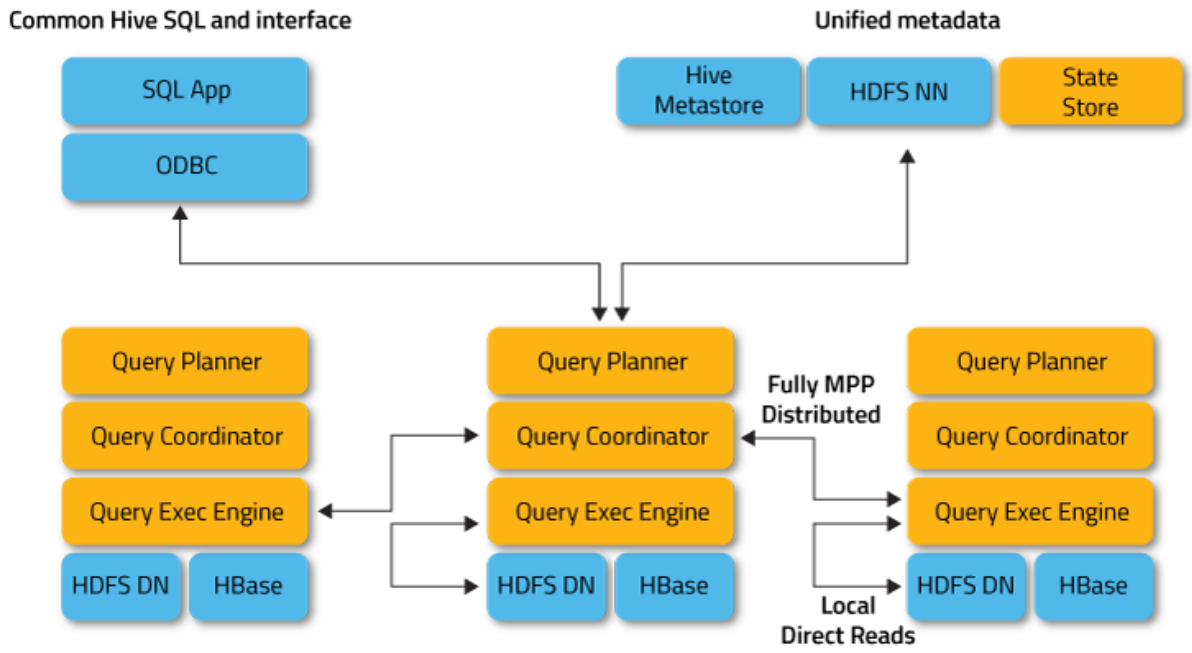
- http://www.quora.com/What-is-the-criteria-to-chose-Pig-Hive-Hbase-Storm-Solr-Spark-to-analyze-your-data-in-Hadoop
- http://en.wikipedia.org/wiki/Apache_Hive

Impala Documentation

Impala raises the bar for SQL query performance on Apache Hadoop while retaining a familiar user experience. With Impala, you can query data, whether stored in HDFS or Apache HBase – including SELECT, JOIN, and aggregate functions – in real time. Furthermore, Impala uses the same metadata, SQL syntax (Hive SQL), ODBC driver, and user interface (Hue Beeswax) as Apache Hive, providing a familiar and unified platform for batch-oriented or real-time queries. (For that reason, Hive users can utilize Impala with little setup overhead.)

Architecture

To avoid latency, Impala circumvents MapReduce to directly access the data through a specialized distributed query engine that is very similar to those found in commercial parallel RDBMSs. The result is order-of-magnitude faster performance than Hive, depending on the type of query and configuration.



How Impala Fits Into the Hadoop Ecosystem

Impala makes use of many familiar components within the Hadoop ecosystem. Impala can interchange data with other Hadoop components, as both a consumer and a producer, so it can fit in flexible ways into your ETL and ELT pipelines.

How Impala Works with Hive

A major Impala goal is to make SQL-on-Hadoop operations fast and efficient enough to appeal to new categories of users and open up Hadoop to new types of use cases. Where practical, it makes use of existing Apache Hive infrastructure that many Hadoop users already have in place to perform long-running, batch-oriented SQL queries.

In particular, Impala keeps its table definitions in a traditional MySQL or PostgreSQL database known as the **metastore**, the same database where Hive keeps this type of data. Thus, Impala can access tables defined or loaded by Hive, as long as all columns use Impala-supported data types, file formats, and compression codecs.

How Impala Uses HDFS

Impala uses the distributed filesystem HDFS as its primary data storage medium. Impala relies on the redundancy provided by HDFS to guard against hardware or network outages on individual nodes. Impala table data is physically represented as data files in HDFS, using familiar HDFS file formats and compression codecs. When data files are present in the directory for a new table, Impala reads them all, regardless of file name. New data is added in files with names controlled by Impala.

Advantages

There are many advantages to this approach over alternative approaches for querying Hadoop data, including:

- Thanks to local processing on data nodes, network bottlenecks are avoided.
- A single, open, and unified metadata store can be utilized.
- Costly data format conversion is unnecessary and thus no overhead is incurred.
- All data is immediately query-able, with no delays for ETL.
- All hardware is utilized for Impala queries as well as for MapReduce.
- Only a single machine pool is needed to scale

HiveQL Features not Available in Impala

The current release of Impala does not support the following SQL features that you might be familiar with from HiveQL:

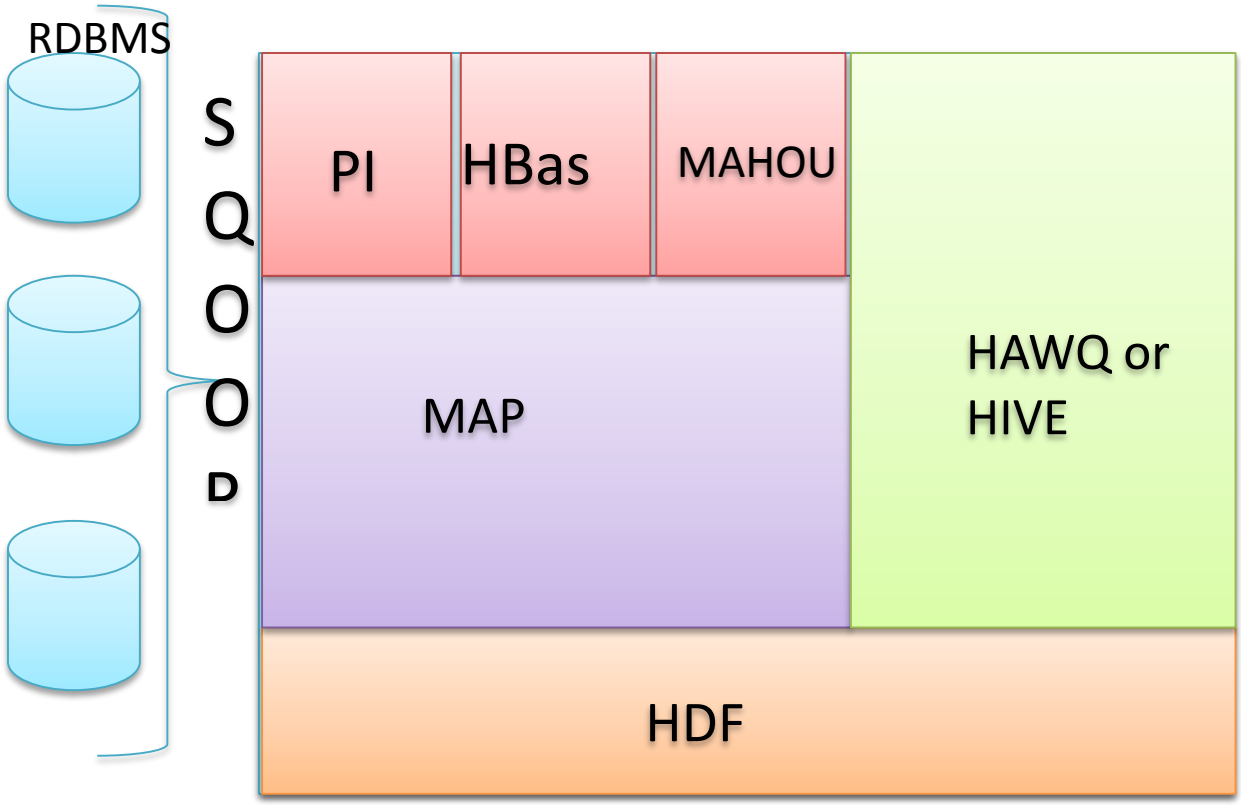
- Non-scalar data types such as maps, arrays, structs.
- Extensibility mechanisms such as TRANSFORM, custom file formats, or custom SerDes.
- XML and JSON functions.
- Certain aggregate functions from HiveQL: covar_pop, covar_samp, corr, percentile, percentile_approx, histogram_numeric, collect_set; Impala supports the set of aggregate functions listed in Impala Aggregate Functions and analytic functions listed in Impala Analytic Functions.
- Sampling.
- Lateral views.
- Multiple DISTINCT clauses per query, although Impala includes some workarounds for this limitation.

References

- The Impala Cook book http://www.slideshare.net/cloudera/the-impala-cookbook-42530186?qid=3dae99ed-f6cb-431c-a3b6-106318b3f571&v=qf1&b=&from_search=2

PIG Documentation

Architecture


RDBMS
S
Q
O
O
PI
HBas
MAHOU
MAP
HAWQ or
HIVE
HDF

Pig Overview

Pig is a simple-to-understand data flow language used in the analysis of large data sets

Pig scripts are automatically converted into MapReduce jobs by the Pig interpreter, so you can analyze the data in a Hadoop cluster even if you aren't familiar with MapReduce

How Pig Script works

Users Of Pig

Yahoo is one of the heaviest user of hadoop, runs 40% of all its hadoop jobs using PIG

How to access Pig

Grunt, Pig shell

Submit the PIG scripts directly

Pig server JAVA class, a JDBC like interface

Pig Pen, an eclipse plugin

Pig Data Types

Scalar Types:

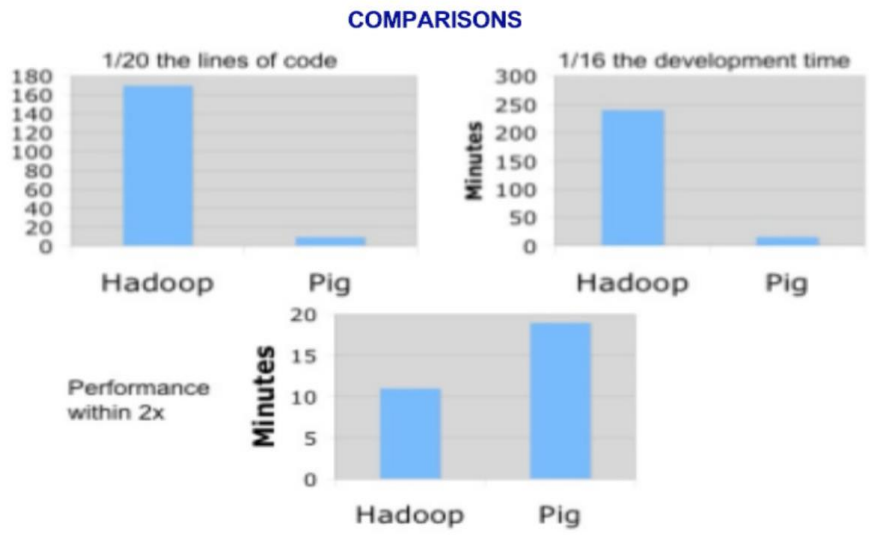
- Int
- Long
- Chararray
- Bytearray

Complex Types:

- Tuple
- Bag

Why do we use Pig

- PIG is a data flow scripting language, used for performing map reduce task



How to load data in Pig

- We have few data loaders available in PIG:

Pig Storage: load/stores relations using field-delimited text format

JSON loader: load/stores relation which are in JSON format

Bin Storage: load/stores relations from binary files

How to start the Pig shell

We can run PIG shell in two modes:

Local mode

MapReduce mode (on the HDFS)

Command to dive into PIG

./pig –x local (for the local mode)

./pig –x mapreduce (for the distributed mode)

Some of the operations in PIG

- LOAD

- LIMIT
- DUMP
- STORE
- DESCRIBE
- AVG
- CONCAT
- COUNT
- MIN
- MAX
- FILTER
- JOIN
- GROUP BY
- ORDER BY

Example

```
A = load '/Users/prateekgangwal/Desktop/pivotal/CustomerSet_1' USING PigStorage('|') AS
(CommunityCount,CommunityIndex,CommunitySize,CustomerIndex:int,CustomerMarket,Operator
Code,OperatorName:chararray,ISDN,IMSI,IMEI,MobileBrand:chararray,MobileModel:chararray);

B = GROUP A BY MobileBrand;

C = FOREACH B GENERATE group, COUNT(A) as count;

D = ORDER C BY count DESC;

STORE D INTO '/usr/local/bin/pig/bin/HighestMobileBrand';
```

Output

```
Nokia Mobile Phones Ltd 20
Motorola Inc    13
Ericsson        10
Sagem   6
Samsung Electronics     5
```

Nokia 5
Samsung 5
Motorola 4
Acer Peripherals Inc 3
Sony-Ericsson 3
Siemens AG 3
Bosch Telecom 2
Longcheer Telecommunication 2
Xiamen Chabridge Telecom Equipment 2
Panasonic Mobile Communications Co Ltd 1
Amoisonic Electronics Co. Ltd 1
TCL Mobile Communication 1
Maxon Cellular Systems 1
Lenovo Legend Xoceco 1
Wonu Telecom Co Ltd 1
LG Electronics 1
LG Electronics 1
Telsda Mobile 1

AVRO Documentation

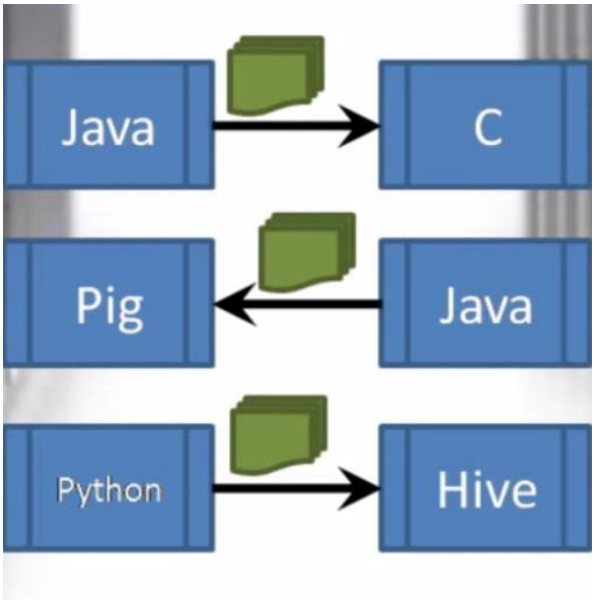
Avro is an apache open source project that provides two services for Hadoop:

Data Serialization

Data Deserialization

We can use this services together or independently

- Data stored using Avro can easily be passed from a program written in one language to a program written in another language, even from a complied language like C to a scripting language like PIG



- Avro stores the data into very compact and efficient format. It stores both the data definition and the data together in one message or file
- The data definition is stored in JSON format and the data is stored in binary format

Schema and Code generation

```
{"namespace": "example.avro",

"type": "record",

"name": "User",

"fields": [

{"name": "name", "type": "string"},     {"name": "favorite_number",  "type": ["int", "null"]},

{"name": "favorite_color", "type": ["string", "null"]}  ] }
```

- Code generation allows us to automatically create classes based on our previously-defined schema
- First let's create some Users and set their fields.

```
User user1 = new User();

user1.setName("Alyssa");

user1.setFavoriteNumber(256);

// Leave favorite color null
```

- Now once the user is created and the fields are set we can perform the serialization and the deserialization for the user objects created

Avro MapReduce API

- Unlike the normal hadoop MapReduce the Avro API is slightly different
- public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {}
- Arvo MapReduce

  map(IN, Collector<Pair<K,V>>)

So the key value is generated at the intermediate step i.e. the output of map and the input of the reducer