# Data input

## Hive

Data warehouse infrastructure built on top of Hadoop

Hive does not currently support update statements. Additionally, since it runs batch processing on Hadoop, it can take minutes or even hours to get back results for queries. Hive must also be provided with a predefined schema to map files and directories into columns and it is not ACID compliant.

Hive should be used for analytical querying of data collected over a period of time - for instance, to calculate trends or website logs. Hive should not be used for real-time querying since it could take a while before any results are returned.

## Hbase

Hbase is a NoSQL key/value store which runs on top of HDFS.

Unlike Hive, HBase operations run in real-time on its database rather than MapReduce jobs

In order to run HBase, ZooKeeper is required - a server for distributed coordination such as configuration, maintenance, and naming.

HBase is perfect for real-time querying of Big Data. Facebook use it for messaging and real-time analytics. They may even be using it to count Facebook likes.

## Import structured data into Hive using Sqoop

Apache Sqoop, The nice thing about Sqoop is that we can automatically load our relational data from MySQL into HDFS, while preserving the structure.

[cloudera@quickstart ~]$ sqoop import-all-tables \

-m 1 \

--connect jdbc:mysql://quickstart:3306/retail\_db \

--username=retail\_dba \

--password=cloudera \

--compression-codec=snappy \

--as-parquetfile \

--warehouse-dir=/user/hive/warehouse \

--hive-import

To confirm the data is imported into hdfs

[cloudera@quickstart ~]$ hadoop fs -ls /user/hive/warehouse/

[cloudera@quickstart ~]$ hadoop fs -ls /user/hive/warehouse/categories/

Query structured data through Imapla

-- Most popular product categories

select c.category\_name, count(order\_item\_quantity) as count

from order\_items oi

inner join products p on oi.order\_item\_product\_id = p.product\_id

inner join categories c on c.category\_id = p.product\_category\_id

group by c.category\_name

order by count desc

limit 10;

## Bulk upload using hdfs (hadoop)

[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -mkdir /user/hive/warehouse/original\_access\_logs

[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -copyFromLocal /opt/examples/log\_files/access.log.2 /user/hive/warehouse/original\_access\_logs

[cloudera@quickstart ~]$ hadoop fs -ls /user/hive/warehouse/original\_access\_logs

## Build Hive table

First, you'll take advantage of Hive's flexible SerDes (serializers / deserializers) to parse the logs into individual fields using a regular expression. Second, you'll transfer the data from this intermediate table to one that does not require any special SerDe

CREATE EXTERNAL TABLE intermediate\_access\_logs (

ip STRING,

date STRING,

method STRING,

url STRING,

http\_version STRING,

code1 STRING,

code2 STRING,

dash STRING,

user\_agent STRING)

ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'

WITH SERDEPROPERTIES (

'input.regex' = '([^ ]\*) - - \\[([^\\]]\*)\\] "([^\ ]\*) ([^\ ]\*) ([^\ ]\*)" (\\d\*) (\\d\*) "([^"]\*)" "([^"]\*)"',

'output.format.string' = "%1$$s %2$$s %3$$s %4$$s %5$$s %6$$s %7$$s %8$$s %9$$s")

LOCATION '/user/hive/warehouse/original\_access\_logs';

CREATE EXTERNAL TABLE tokenized\_access\_logs (

ip STRING,

date STRING,

method STRING,

url STRING,

http\_version STRING,

code1 STRING,

code2 STRING,

dash STRING,

user\_agent STRING)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

LOCATION '/user/hive/warehouse/tokenized\_access\_logs';

ADD JAR /usr/lib/hive/lib/hive-contrib.jar;

INSERT OVERWRITE TABLE tokenized\_access\_logs SELECT \* FROM intermediate\_access\_logs;

The final query will take a minute to run. It is using a MapReduce job, just like our Sqoop import did, to transfer the data from one table to the other in parallel

## Query data in Impala

select count(\*),url from tokenized\_access\_logs

where url like '%\/product\/%'

group by url order by count(\*) desc;

# Spark

Relationship strength analytics

Spark-shell –master yan-client

Will start a scala interactive cosole

RDD is Spark data structure for working with distributed datasets

# Apache Solr

To break up documents into fields and store them and search them

Solrctrl – command to create solr config to define fields for solr to index

Streaming tool Flume and Morphline (a simple way to accomplish on-the-fly ETL)