# Data input

## HDFS

File copy to HDFS, Ambari File Viewer

## File commands

### Copy files

* Hdfs dfs –put /etc/passwd /user/cloudera

Similar to hdfs dfs -copyFromLocal

### List files

* Hdfs dfs –ls /user/cloudera

## 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.

Hive has three main functions: data summarization, query and analysis. Hive provides tools that enable easy data extraction, transformation and loading (ETL).

Hive View (Hortonworks Ambari Hive View)/ Cloudera Impala/ beeline (command line interactive)

Basically the steps are create table (stored as ), load unstructured data into table, then query from table

### Create table

CREATE TABLE geolocation\_stage (truckid string, driverid string, event string, latitude DOUBLE, longitude DOUBLE, city string, state string, velocity BIGINT, event\_ind BIGINT, idling\_ind BIGINT)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

TBLPROPERTIES ("skip.header.line.count"="1");

The ROW FORMAT clause specifies each row is terminated by the new line character.

The FIELDS TERMINATED BY clause specifies that the fields associated with the table (in our case, the two csv files) are to be delimited by a comma.

The STORED AS clause specifies that the table will be stored in the TEXTFILE format.

By default, when you create a table in Hive, a directory with the same name gets created in the /apps/hive/warehouse folder in HDFS. Using the Ambari Files User View, navigate to the /apps/hive/warehouse folder. You should see both a geolocation\_stage and trucks\_stage directory:

### Basic queries

show tables;

SELECT \* FROM geolocation\_stage LIMIT 100;

describe geolocation\_stage;

show create table geolocation\_stage;

### Load data into Hive

Using Ambari File copy and paste

Or

LOAD DATA INPATH '/tmp/maria\_dev/data/trucks.csv' OVERWRITE INTO TABLE trucks\_stage;

The original file will be deleted

### Create table using ORC (Optimized Row Columnar)

CREATE TABLE geolocation STORED AS ORC AS SELECT \* FROM geolocation\_stage;

To verify :

describe formatted geolocation;

### LATERALView and Stack function

To reformat table

### CreateTableAsSelect(CTAS)

Persist these results into a table, this is a fairly common pattern in Hive and it is called Create Table As Select (CTAS ).

### command shell

sudo hive

ready to enter commands and sql

hive

quit

## Pig

Pig is a high-level scripting language used with Apache Hadoop. Pig enables data workers to write complex data transformations without knowing Java.

Through the User Defined Functions(UDF) facility in Pig, Pig can invoke code in many languages like JRuby, Jython and Java. You can also embed Pig scripts in other languages. The result is that you can use Pig as a component to build larger and more complex applications that tackle real business problems.

Pig works with data from many sources, including structured and unstructured data, and store the results into the Hadoop Data File System.

Pig scripts are translated into a series of MapReduce jobs that are run on the Apache Hadoop cluster.

### HCatLoader

HCatalog allows us to share schema across tools and users within our Hadoop environment. It also allows us to factor out schema and location information from our queries and scripts and centralize them in a common repository. Since it is in HCatalog we can use the HCatLoader() function. Pig allows us to give the table a name or alias and not have to worry about allocating space and defining the structure. We just have to worry about how we are processing the table.

a = LOAD 'geolocation' using org.apache.hive.hcatalog.pig.HCatLoader();

### Load from file

batting = LOAD '/user/maria\_dev/Batting.csv' USING PigStorage(',');

### Relational Operators:Filter

b = filter a by event != 'normal';

### Relational Operators:FOREACH

c = FOREACH b GENERATE driverid, event, (int) '1' as occurance;

add ‘1’ to every row for 1 column

### Relational Operators:Join

h = join e by driverid, g by driverid;

### Store data

store final\_data into 'riskfactor' using org.apache.hive.hcatalog.pig.HCatStorer();

store data into file

STORE movies\_greater\_than\_three\_point\_five INTO '/user/hadoop/movies\_greater\_than\_three\_point\_five' USING PigStorage (',');

### add arguments

-useHCatalog, this will allow to use HCatLoader and HCatStorer

### Compare Pig with Hive

Pig is pipe lined, while Hive is more like SQL, decalarative

### Using Tez

Pig scripts can be executed just using MapReduce engine or using Tez which is much faster

2016-06-07 02:35:17,813 [main] INFO org.apache.pig.Main - Pig script completed in 1 minute, 58 seconds and 465 milliseconds (118465 ms)

Under Tez

2016-06-07 02:38:56,312 [main] INFO org.apache.pig.Main - Pig script completed in 46 seconds and 608 milliseconds (46608 ms)

### Pig Grunt Shell

You can enter pig scripts into Gig Grunt Shell, every command must end with semi colon

Enter Pig or pig –x mapreduce under command shell you will get grunt>

grunt> A = load '/user/cloudera/passwd' using PigStorage(':');

grunt> B= foreach A generate $0,$4,$5;

grunt> dump B;

grunt> store B into ‘userinfo.out’;

grunt> quit;

in this case pig reads a file from hdfs and process it and store it.

## 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.

Hbase shell/ Hbase MapREduce/ Hbase API/Hbase external API

### Create table

hbase(main):006:0> create 'userinfotable',{NAME=>'username'},{NAME=>'fullname'},{NAME=>'homedir'}

### create records

hbase(main):007:0> put 'userinfotable', 'r1','username','vcsa'

0 row(s) in 0.2020 seconds

hbase(main):008:0> put 'userinfotable', 'r2','username','sauser'

0 row(s) in 0.0170 seconds

hbase(main):009:0> put 'userinfotable', 'r1','fullname','VirtualMachine Admin'

0 row(s) in 0.0110 seconds

hbase(main):010:0> put 'userinfotable', 'r2','fullname','SAS Admin'

0 row(s) in 0.0130 seconds

### Scan table

hbase(main):001:0> scan 'userinfotable'

ROW COLUMN+CELL

r1 column=fullname:, timestamp=1465964853974, value=VirtualMa

chine Admin

r1 column=username:, timestamp=1465964800703, value=vcsa

r2 column=fullname:, timestamp=1465964872347, value=SAS Admin

r2 column=username:, timestamp=1465964822484, value=sauser

2 row(s) in 0.5130 seconds

hbase(main):002:0> scan 'userinfotable',{COLUMNS=>'fullname'}

ROW COLUMN+CELL

r1 column=fullname:, timestamp=1465964853974, value=VirtualMa

chine Admin

r2 column=fullname:, timestamp=1465964872347, value=SAS Admin

2 row(s) in 0.7090 seconds

## 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)

# Hadoop2 Cluster To AWS EC2

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud.

<http://hortonworks.com/blog/deploying-hadoop-cluster-amazon-ec2-hortonworks/>

## Set up aws vm

RHEL 64bit

M1. Large/2cpus/8G/ 2 \*SSD or EBS 1D/hour

Save the private key

Create an image from this instance sharing the same private keys

Launch -select the size of nodes to 6

DNS information for all 6 instances

## Install Ambari Server on Install one

Webget … ambari.repo

Yum install ambary-server

Ambari-server setup

Then start ambari-server

## Using ambari-server to deploy the cluster

Follow the wizard

Select stack: HDP 2.0.6

Put all DNS names into Target Hosts

And shared key

Select all the services that you need:

Ambari-server will tell you which services run on which instances

Config credentials for some of the services