Intro to Hadoop and MapReduce

[redacted] more and more data is being generated

[redacted]

Big Data

"Data too big to be processed on a single machine"

[redacted]

3 Vs: volume, variety, velocity

- volume: size of the data
  - price to store data has dropped incredibly (storing data reliably is more expensive)
  - Some companies throw away data because it's too expensive to store it reliably, like logs
  - On Hadoop you can store data in its raw format, and manipulate it and reformat it later → All data is useful at some point
- variety: data coming from a lot of different sources and formats
  - With databases data must fit in pre-defined tables, but most data is unstructured or semi-structured
  - Store data in original format so you're not throwing anything away
- velocity: speed at which data [redacted] arrives, ready to be processed
  - Store data at a rate of also TB/day
  - Predict what the user wants for example

[redacted] Get hadoop!

Hadoop consists of a way to store data, known as the Hadoop Distributed file system, and a way to process data with MapReduce.

Data is split and stored across the collection of machines, or cluster, when we want to process the data we process it where it is actually stored.
Machines can be added to the cluster as the amount of data grows

The machines don't need to be high end

Hadoop ecosystem:

[redacted]

to make hadoop easier to use

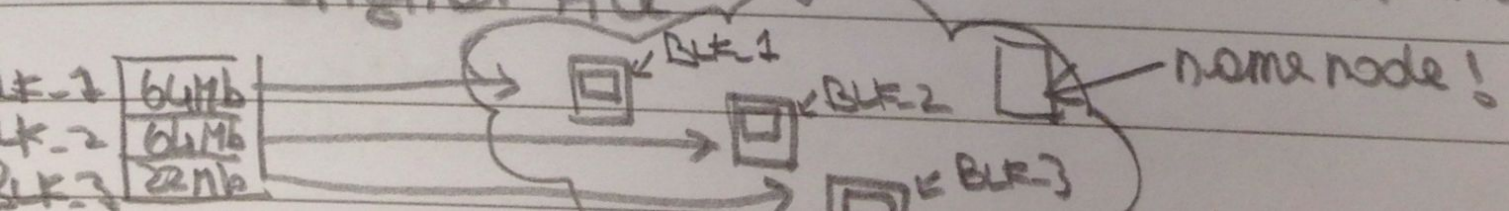
- Hive helps querying data without knowing how to code ~~(map reduce is to)~~
  SQL entered is then interpreted in mapreduce code and run in the cluster
- Pig, analyse data in a fairly simple scripting language rather than mapreduce, also here the code is turned into mapreduce
- Impala, queries data in SQL, but directly accesses it from HDFS, instead of running mapreduce jobs. It is optimized for low-latency queries (while hive is optimized for running long batch processing jobs)
- Sqoop, takes data from a relational database and puts it in HDFS

- Flume ingests data as generated by external systems, and puts it into the cluster
- HBase is a realtime database, built on top of HDFS
- Hue, graphical front end to the cluster
- Oozie, workflow management TOOL
- Mahout, machine learning library

Making all these talk to one another and work well can be tricky, to make installing and maintaining a cluster like this easier, Cloudera, has put together a distribution of Hadoop called CDH
It takes all the ecosystem project and Hadoop, and packages them together so that installation is easy and they compatible together
= makes sure

HDFS

When a file is loaded into HDFS, it is splitted into chunks, which we call blocks.
Each block is 64Mb by default, and has a unique name (BLK_some large number).
As the File is uploaded to HDFS, each block will get stored on one node in the cluster.
A daemon is running on each of the machines in the cluster ~~Each machine is called data node~~
A separate machine called datanodes runs a Daemon called NameNode, it contains the Metadata that says which block make up the original File

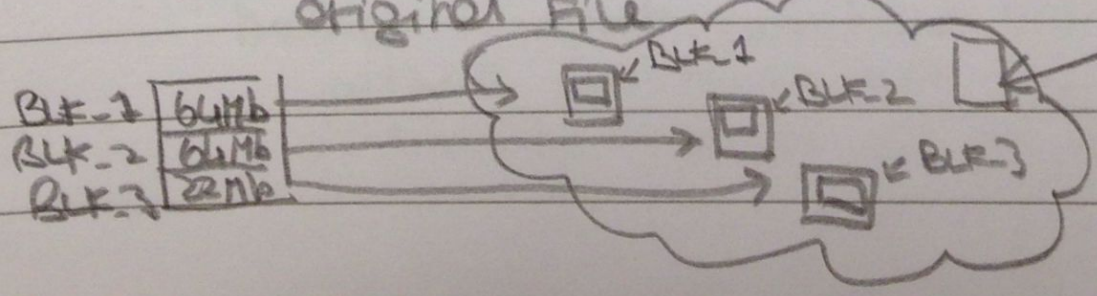


- Flume ingests data as generated by external systems, and puts it into the cluster
- HBase is a realtime database, built on top of HDFS
- Hue, graphical front end to the cluster
- Oozie, workflow management tool
- Mahout, machine learning library

Making all these talk to one another and work well can be tricky, to make installing and maintaining a cluster like this easier, Cloudera has put together a distribution of Hadoop called CDH
It takes all the ecosystem project and Hadoop, and packages them together so that installation is easy and they compatible together
makes sure

HDFS

When a file is loaded into HDFS, it is splitted into chunks, which we call blocks.
Each block is 64Mb by default, and has a unique name (BLK_some large number).
As the File is uploaded to HDFS, each block will get stored on one node in the cluster.
A daemon is running on each of the machines in the cluster called datanodes
A separate machine runs a Daemon called NameNode, it contains the Metadata that says which block make up the original File



Data redundancy!

* IF one node fails, the file has missing data. For that reason, Hadoop replicates each block three times as it is stored in HDFS (picks three nodes at random and puts one copy of the block on each of the three)

IF a node fails, the namenode sees that a block is under-replicated and it will arrange to have it re-replicated so that there are three copies

[redacted]

Namenode is configured to store the metadata not only on its own hard drive but also somewhere on the network file system

IF the namenode fails, the entire cluster is unaccessible, IF the metadata is lost, the entire cluster's data is lost
to avoid that, most clusters have configured two namenodes.

- The active namenode works as before
- The standby can be configured to take over IF the active fails

All of the commands that interact with the hadoop file system, start with hadoop fs

hadoop fs -ls <list of what's in home directory, in the hadoop cluster

hadoop fs -put purchases.txt
upload purchases.txt file
destination filename can be specified

hadoop fs -tail purchases.txt (-cat for the entire content)
shows the last few lines of the file

other hadoop -fs commands closely mirror UNIX filesystem commands

```
hadoop fs -mv purchases.txt purch.txt
hadoop fs -rm purch.txt
hadoop fs -mkdir myinput
hadoop fs -put purchases.txt myinput
```

=

Map Reduce

processing a large file from top to bottom can take a long time

MapReduce is designed to process data in parallel. The file is broke into chunks and processed in parallel

~~1 Each chunk is assigned to a mapper that will at the same time each one a different portion of the data.~~

~~2 Reducers collect results that belong in the same element but were computed in different mappers.~~

~~3 Parts each deal with a small~~

Mappers are programs, each deals with a small amount of data and work in parallel their output is called the intermediate records

Hadoop deals with all data in the form of key value pairs. That's what intermediate records are

Shuffle and sort, shuffle is the movement of int records from mappers to reducers, the sort is the fact that the reducers will organize the INT. records.

Each reducer works on one set of records at a time, it gets a key and a list of all the values that are processed in some way, then it writes out the final result

=

You can have the final result sorted by adding an extra step that merges the result

[redacted] =

Daemons of mapreduce

↓

piece of code running all the time [redacted]

=

A daemon runs on each machine

When running a mapreduce job, you submit the job to the jobtracker, which splits the work into mappers and reducers that will run on other cluster nodes.

Running map and reduce tasks is handled by a daemon called the task tracker, which runs on each datanode

Since the task tracker runs on the same machine as the datanodes, the hadoop framework will have the map tasks work directly on the pieces of data that are stored on that machine

By default hadoop will use a HDFS block as the input split for each mapper and it'll make sure that a mapper works on data on the same machine.

IF a block needs processing, the task tracker on the machine where the block is will literally be the one chosen to process it. But that's not always possible as TaskTrackers could be busy

... in which case a different node will be chosen to process the block and it will be streamed over the network

=

to submit the job

hadoop jar /usr/lib/hadoop-.../streaming/...
-mapper m.sole -reducer r.sole
-file m.sole -file r.sole -input dirin
output dirout ← the output directory must not already exist!

=

hadoop produces _SUCCESS, _logs, part-0000 or more depending on reducers

I can get the result by doing
hadoop fs -get part0000 localFile.txt

A simplified command for the one above is
hs m.sole r.sole dirin dirout

=