What is Hadoop?

an open source **software platform** for **distributed storage** and **distributed processing** of very large data sets on computer clusters built from **commodity hardware** - Hortonworks

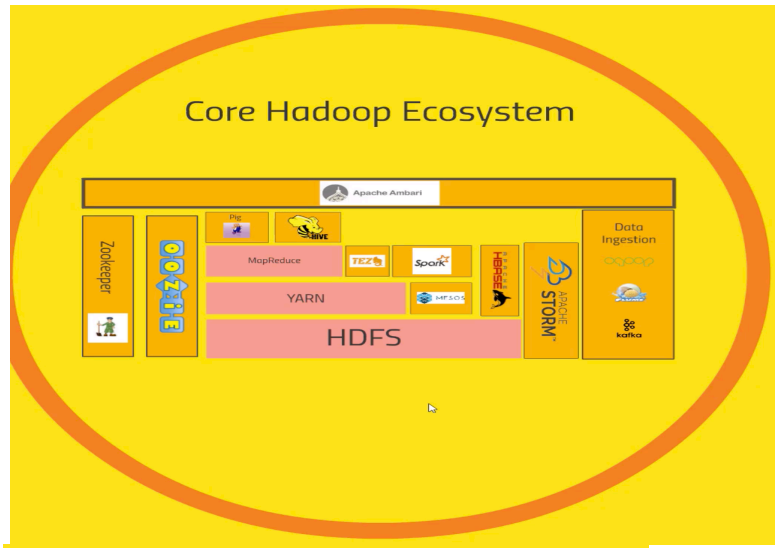
Software platform is a bunch of software that runs on a cluster of computers, Hadoop grabs the power of multiple pcs instead of running software on a single pc.

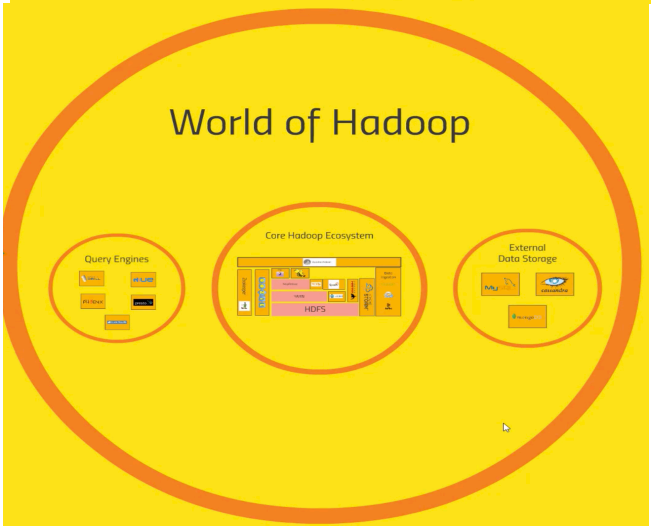
bigdata refers to getting terabytes of information every day or even more.
So, inorder to store the terabyte sof information we use **distributed storage**. it is like terabytes of data distributed across all of the hard drives in your cluster as one single file system.

if one of the harddrives present in the computer crashes into the flames. hadoop gonna able to retrieve it for you because it keeps backup copies of all your data on other places in your cluster of computers and its very reliable.

Not only hadoop stores your data in a distrubuted storage it can also **distribute the processing of data** as well. In this way, you can retreive the data you want very quickly instead of waiting for days to retrieve data from a single computer. it is using divide and conquer problem across all the PCs in a cluster.

clusters built from **commodity hardware**, that means aws, google cloud platform or microsoft azure that sell cloud services so that you cant rent off hardware which is used in a hadoop cluster.





- HDFS - Hadoop Distributed File System, distributed data storage of hadoop
- YARN - Yet Another Resource Negotiator, manages the resources on your computing cluster
- MapReduce - programming model that allows you to process your data across
-
- Pig - High Level programming API, SQL style syntax, transforms the script into something that will run on map reduce.
- Hive - its just like a database, similar to pig but it really looks like a sql database

- Ambari - gives a view of your cluster. sits on top of everything. ambari belongs to hortonworks
- Mesos - alternative to yarn
- Spark - sitting at the some level of map reduce, handle sql queries, streaming data in real time, it can do machine learning.
- Tez - uses some techniques of spark, produces optimal plans for executing queries (uses directed acyclic graph, its a leg up on what map reduce does) , hive through tez can be faster than hive through map reduce
- Hbase - NoSql DATABASE, meant for very large transaction rates, fast way of exposing data , so that mapreduce/ spark can use it to transform it
- Storm - way of processing streaming data in real time, update ml models, transform data in all real time
- Ooozie - scheduling jobs on your cluster, so when you have more complicated operations that require loading data into hive and then integrating with the pig and may be querying with SPARK and then transforming the results into Hbase OOozie can manaze that all for you.
- ZooKeeper - Coordinating everthing on your cluster, keeping track of which nodes are up and down
- Data Ingestion
- Sqoop - RDMS TO HDFS
- Flume - Collects Web Logs in Real time, ingest them into your cluster.
- Kafka - collect general data in real time and ingest them into your hadoop cluster.
- External Data Storage
- MySQL - RDMS, IMPORT AND EXPORT the data from sqoop into your cluster and write queries.
- cassandra - columnar data, exposing data real time
- mangoDB - columndar data, exposing data real time

- mysql, cassandra, mangoDB can all integrate with your cluster
- Apache Drill - interactive query engine, works with cassandra, mangodb, habse, sql.
- Hue - Interactive query engine, works well with hive and hbase.
- Apache Phoenix - Interactive query engine, similar to apache drill , gives you ACID guarentees.
- Presto - query engine, execute queries across your entire cluster
- zeppeline - notebook type approach to the UI, query engine

Hadoop Distributed File System / Why Hadoop?

Data is large now a days, whether the data is stock market or contains the dna information.

Scientists say all the world's data can fit on a DNA hard drive the size of a teaspoon - https://qz.com/345640/scientists-say-all-the-worlds-data-can-fit-on-a-dna-hard-drive-the-size-of-a-teaspoon/

HDFS Handles big files, it is really optimized.

it handles them by breaking the big files into the blocks. each block is 128mb by default so we can understand even a block is large

store the blocks across several commodity hardwares. it will actually store than one copy of each block inorder to handle failure.

HDFS Architecture

it consists of a single name node and several data nodes

Name Node - it keeps tracks of where all blocks live. it keep tracks of edit log. edit log stores whats being modified, moved, created. basically, it keep tracks of everthying that is present in datanode.

Data Node : it stores each block of each file, they also talk to each other maintain the copies of blocks.

In HDFS to read a file, the client node first talks to the name node which directs the client node to the appropriate data node. Name node also determines the most efficient data nodes for you to reach from your client.

To write a file, the client tells the name node, name node creates a new entry. name node acknowledges client that go create your file. client node then talks to a single data node about its file. then the data nodes talk to each other to replicate the blocks of that file across multiple nodes. sends the acknowledgements saying ok to client node and finally back to name node. then name node records the fact that this new file exists and it is stored in a replicated manner across data nodes.

What if namenode crashes ?

Back up your metadata constantly. most simplest thing to do. disadvantage is because of the lag there will be some loss of information.

maintain a merged copy of edit log (secondary node) from your primary name node.

HDFS Federation - each namenode manages a sepcific namespace volume. so we will loose only some information if one of the nodes fail. it is better than entire hdfs cluster going down.

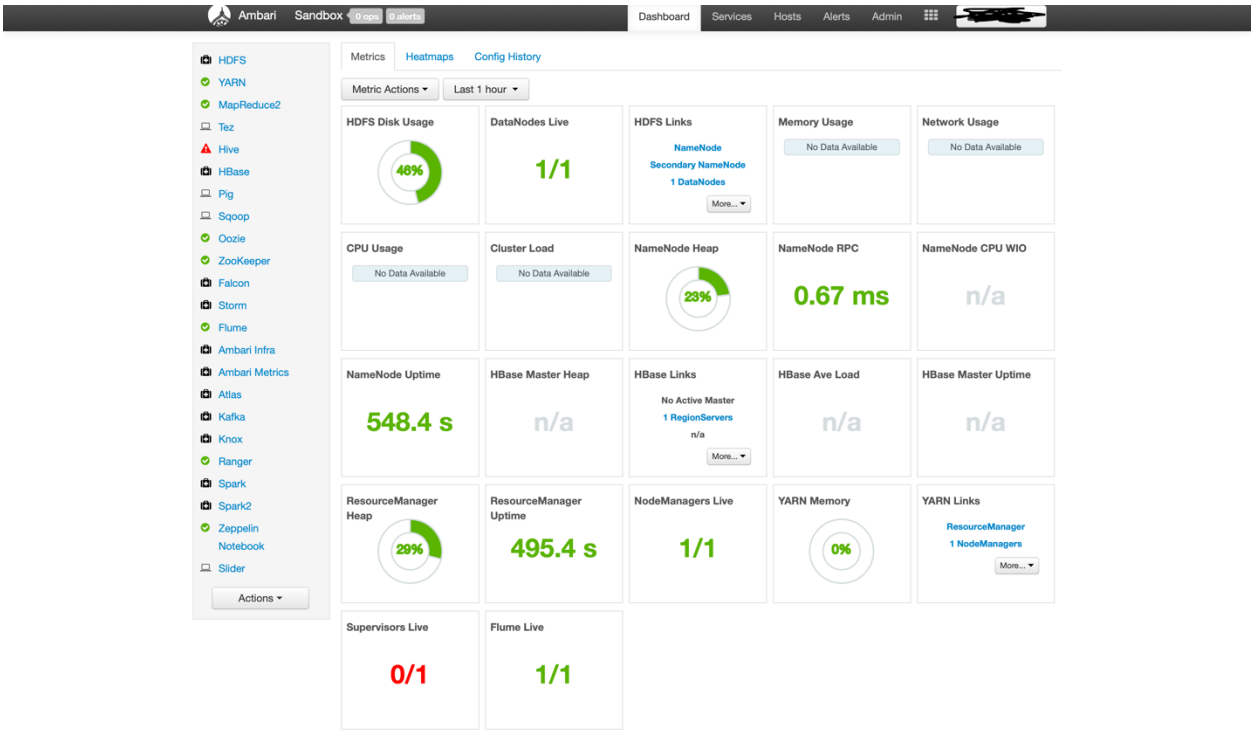
HDFS High Availability

Hot standby namenode using shared edit log

Zookeeper tracks active namenode

uses extreme measures to ensure only one namenode is used at a time.

Ambari Interface



- Dashboard gives an overview of your cluster is doing and its overall health

- Ambari is also primarly used for installing hadoop. how?
    - start up with a blank empty cluster
    - install ambari on your master node
    - use that to actually set the rest with the specific services you want

- To change the configurations of a particular services
    - click a service that you want to change in the services tab
    - under the config tab you can interactively change configuragtions
    - its better to not mess with them unless you know what you are doing

- you can view which host is running which service under the hosts tab

- under the alerts tab you can set up automatic alerts that will actually email you when something goes wrong

    - you can set things up to really minimize the false positives there.

- if you are a admin you can start or stop or restart a service
    - ambari -admin -password -reset
    - this is executed via super user permissions on your terminal which in turn can be executing the su root command and giving the password

Click files view present in the grid options to view the entire HDFS file system thats running on our hadoop cluster. this file view allows us to view and manipulate all the files in our HDFS file system.

2) Using command line:

On putty/terminal connecting to the sandbox server can use commands like

- hadoop fs -mkdir directory - to create a directory on hdfs
- hadoop fs -ls to see the list of directories
- wget http://..... link to get data from an online source
- hadoop fs -copyFromLocal file directory/file to copy from one place to other
- hadoop fs -rm file directory/file to remove a file
- hadoop fs -rmdir directory to remove a directory
- hadoop fs to get list of commands

MapReduce

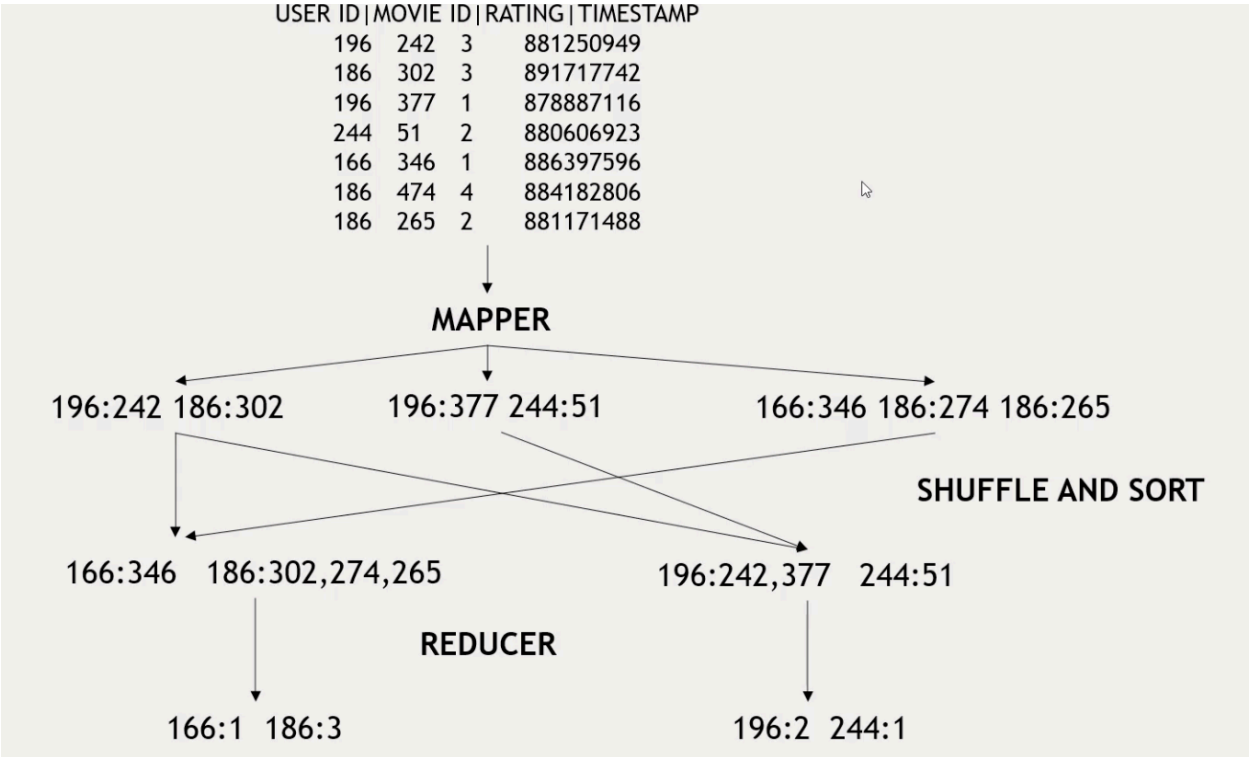
Distributes the processing of data on your cluster.

there are two things that mapreduce do?

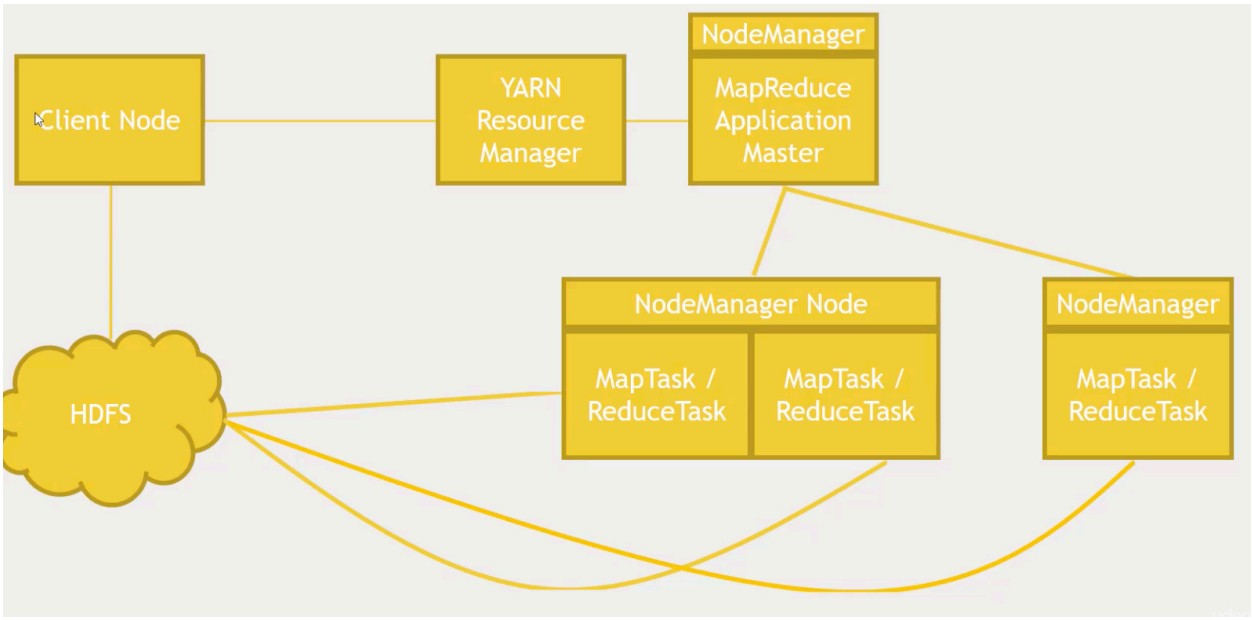
1)map - it maps data. -> key value (transform your data)
2)reduce - it reduces data. (aggregate your data)

it also deals with node failure, application master monitors your mappers and reducers on each partition

A simple example below illustrates the concept of mapreduce on a cluster.



- Dataset gets chopped up into three parts. first two lines sent to one node, second two lines sent to second node and remaining lines sent to remaining node.
- each node here can be a computer, multiple machines working together takes the advantage of hadoop's distributed storage and distributed processing.
- mapper transforms each individual line of the data into a key value pair.
- Here, key is the user id and value is the movie id for each individual rating.
- shuffle and sort will merge all the duplicate keys into a single unique key and sort those keys. then reducer is giving out the length for each key i.e number of movies rated by the each user.



So when you kick things off from a client node,

resource manager keep tracks of status of different computers, application master keeps tracks of all of your tasks and node manager keep tracks of individual pcs as a whole. they all talk to data present in the HDFS.

Node closer to the data will be given high priority for a task by the resource manager.

Natively map reduce code is written in java but we can write them in any language such as python with help of concept of streaming.

How to handle failure :

Application master monitors worker tasks for errors or hanging
restarts as needed
preferably on a different node
what if the application master goes down?
YARN can try to restart it
what if an entire node goes down?
this could be the appication master
the resource manager will try to restart it on different pc
what if the resource manager goes down?
can set up "high availability" using zookeeper to have a hot standby (second backup resource manager)

Map reduce, its been largely superseded by technologies like Spark or higher level tools that actually let you issue sql style queries such as hive to your cluster

Run Locally

python script.py data-filename

Run with Hadoop

python script.py -r hadoop --hadoop-streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar filename

if the file is not in local system but on the hdfs cluster then

python script.py -r hadoop --hadoop streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar hdfs://.....path

Main problem with the mapreduce it takes a lot of development cycle time.

Apache Pig

its a scripting language called PigLatin built on top of Hadoop and mapreduce that lets you create mapreduce jobs without having to write mappers and reducers.

SQL- like syntax, step by step language to set up different relations on your data using a very simple script format.

Pig runs on top of tez in addition to mapreduce so there is no performance drawback infact very faster analysis than map reduce.

it is highly extensible with **user defined fucntions** (custom function)

Running Pig

Grunt - command line repl intrepreter
Script - save pig script to a file, run that command line using pig
Ambari / Hue - web browser

Find the oldest 5-star movies

create a relation with a given schema :

relation = LOAD 'data path name' As (column1 : int, column2: int, ...);

Functions :

Use PigStorage() if you need a different delimiter.

relation2 = LOAD 'data path name' USING PigStorage('delimiter') AS (column1 : int, column2: int, ...);

Use STORE to write a relation to disk.
-STORE relation INTO 'newrelationname' USING PigStorage('');

Use DUMP relationname to view whats in that relation, useful for debugging things

Creating a relation from another relation : FOREACH / GENERATE

relation3 = FOREACH metadata GENERATE column1, column2;

GROUP BY

it actually creates a bag that contains tuples of individual rows associated with a given specific column tuple. its is a reduce operation.

Average (AVG)/ COUNT / MAX/ MIN on groupby relation to get some insight.

DESCRIBE relation; to describe the schema of a relation

FILTER

FILTER relation BY (some column condition)

DISTINCT, gives you back the unique values within a relation.

MAPREDUCE, which lets you call explicit mappers and reducers on a relation

STREAM, to stream the results of Pig to a process and just use stdin and stdout like map reduce streaming.

JOIN to join the data by some common column present in the two relations, DESCRIBE relation will be helpful here.

COGROUP, is just a variation of join, join creates row together into a single tuple whereas cogrouo creates a separate tuple for each key, so its little bit more organized

CROSS, cartesian product, which lets you try all the combinations b/w two relations

RANK , is like order but it actually assigns a rank number to each row.
ORDER BY

ORDER the data by a particular column,

ORDER relation BY relation::column ;

UNION, takes two relations and squishes them together

SPLIT, takes a relation and split it up more than one relation

Diagnostics

DESCRIBE

EXPLAIN

ILLUSTRATE

Both explain and illustrate tells about what going on under the hood with PIG

USER Defined Functions:

REGISTER - import jar file that contains user defined functions

DEFINE - assign names to those functions so you can actually use them with your Pig scripts and

IMPORT - importing macros for Pig File, so that resuable bits of Pig code can be used in other scripts

AVG , CONCAT, COUNT, MAX, MIN , SIZE, SUM

Storage classes - PigStorage, TextLoader, JsonLoader, AvroStorage, ParquetLoader, OrcStorage, HBaseStorage

SPARK

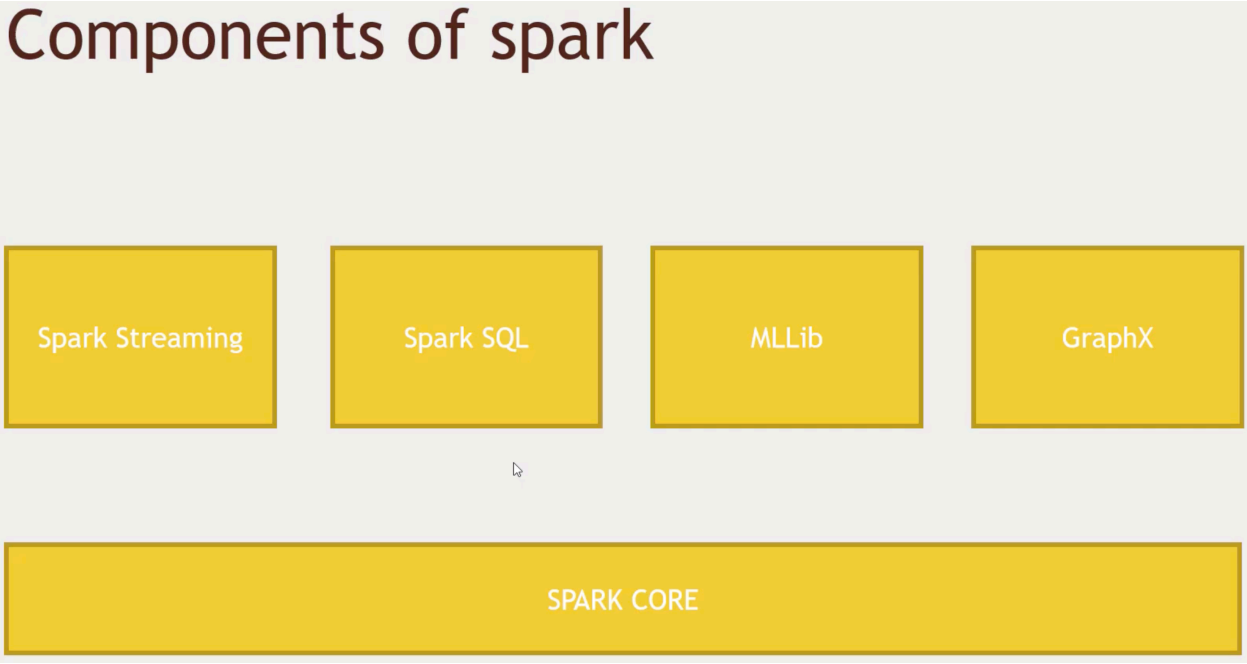
Spark is very much faster than MAPREDUCE because it uses the concept of directed acyclic graph. it helps is to find the result in a very optimal way from a set of operations.

it is memory based this is also one of the reason why disc access or storage is way lot faster than using the map reduce becuase map reduce is disc based.

SPARK uses RDD (resilient distributed dataset), which is used to transform, reduce the data in a very quick way.

so basically, you are writing a script in python/scala/java that takes an RDD of your input data and transforms it in whatever the way you want

it is having a rich eco system surrounding Spark that lets you do a wide variety of tasks on big data across the cluster.



Scala code in Spark looks a lot like python code

Python code to square numbers in a data set:

```
nums = sc.parallelize([1,2,3,4])
squared = numps.map(lambda x:x*x).collect()
```

Scala code to square numbers in a data set:

```
val nums = sc.parallelize(List(1,2,3,4))
val squared = nums.map(x=>x*x).collect()
```

At the end of the day, I am a python programmer but it is not hard for me to move from python to scala

RDD

SPARK context is sort of the environment that your driver program and it is what it creates **RDDS**

Creating Rdds

Below are the number of ways to create an RDD

- nums = parallelize([1,2,3,4])

- sc.textFile("file://pathname")
- or aws service called s3n:// or hdfs://

- hiveCtx = HiveContext(sc)

- rows = hiveCtx.sql('SELECT name, age FROM users")

- can also create from
- JDBC, Cassandra, HBase, Elasticsearch.
- JSON, CSV, squence files, object files, various compressed formats

Transforming RDDS

map - one to one relationship between the input and the output
ex : rdd = parallelize([1,2,3,4] , squaredx= rdd.map(lambdax:x*x)

flatmap - where input lines may or may not result in one or more output lines in the resulting RDD

Filter

distinct

sample

union, intersection, subtract, cartesian

lot of functions look similar to pig but more powerful

RDD actions

crunch the numbers down, so that is RDD actions do

collect - what ever it is present in RDD and return the python object

countByValue

take - take top 10 results from the RDD

top - same idea as take

reduce - define a function to combine all the values associated with a key

Main point in SPARK is

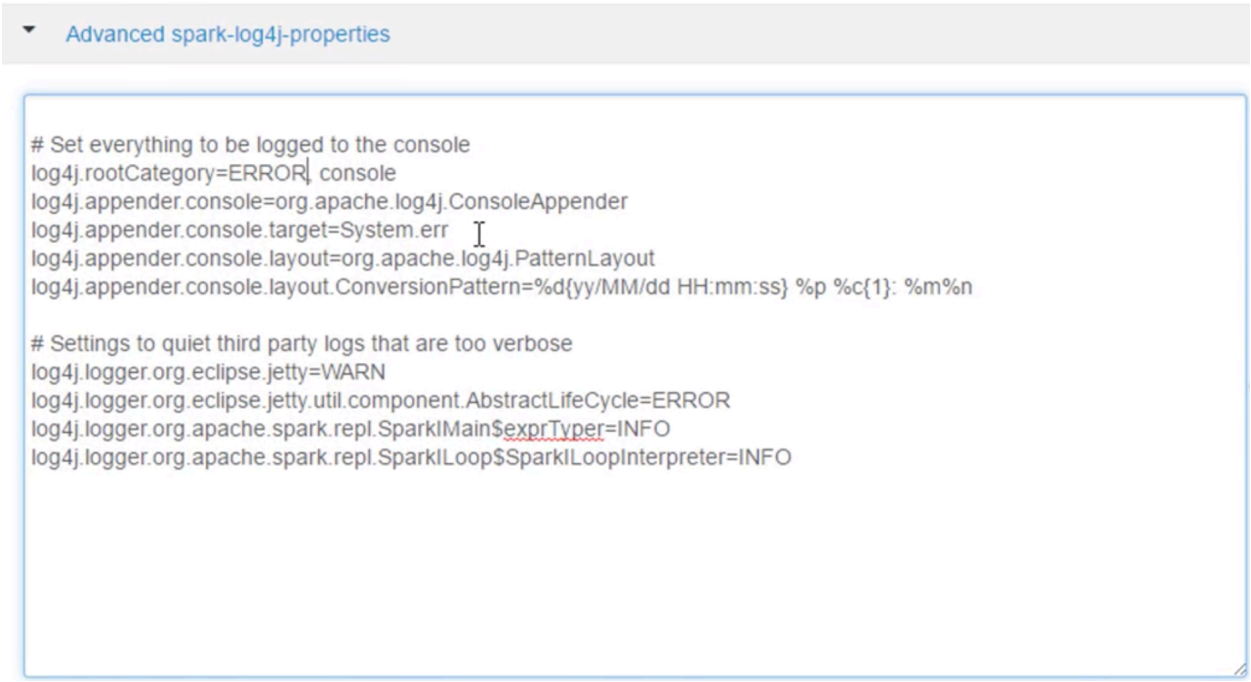
- Nothing will actually kickoff in your driver program until an action is called - Laze evaluation.
- it wants to find the result in a very optimal way. so when you are transforming spark actually creates a graph dependencies
- then it will find a optimal way from these dependencies to get the desired result

debugging is hard maybe

Change the Log level of spark by default. By default its very verbose. its hard to see results when you are actually running the script as results

How to change the log level?

Go to Ambari
click Spark and then configs
search for Advanced spark-log4j-properties,
change log4j.rootCategory = ERROR, console



Running Spark (PySpark) on real cluster :

spark-submit filename.py

spark-submit can actually take a bunch of parameters as well such as which cluser to run on, how much memory you want to allocate to each executor.

Usually cluster is preconfigured.

SPARK SQL

- Extends RDD to a "DataFrame" Object
What is a dataframe?:
contains row objects
can run sql queries

has a schema (leading to more efficient storage)

Read and write to JSON, Hive, parquet

communicates with JDBC/ODBC , Tableau

from pyspark.sql import SQLContext, ROW
Functions like filter, select, groupby can be performed on DataFrame objects
As Dataframes are built on top of RDD, RDD can be got back anytime

In Spark 2.0
Dataframe is a Dataset of Row objects.

DataSets can wrap known, typed data too.

Spark SQL exposes a JDBC/ODBC server

you can create new tables or query existing ones that were catched using
hiveCtx.cacheTable("tableName")

UDF's

it supports user defined functions

```
from pyspark.sql.types import IntegerType
hiveCtx.registerFunction("square", lambda x: x*x, IntegerType())
df = hiveCtx.sql("SELECT square('someNumericFiled') FROM tableName)
```