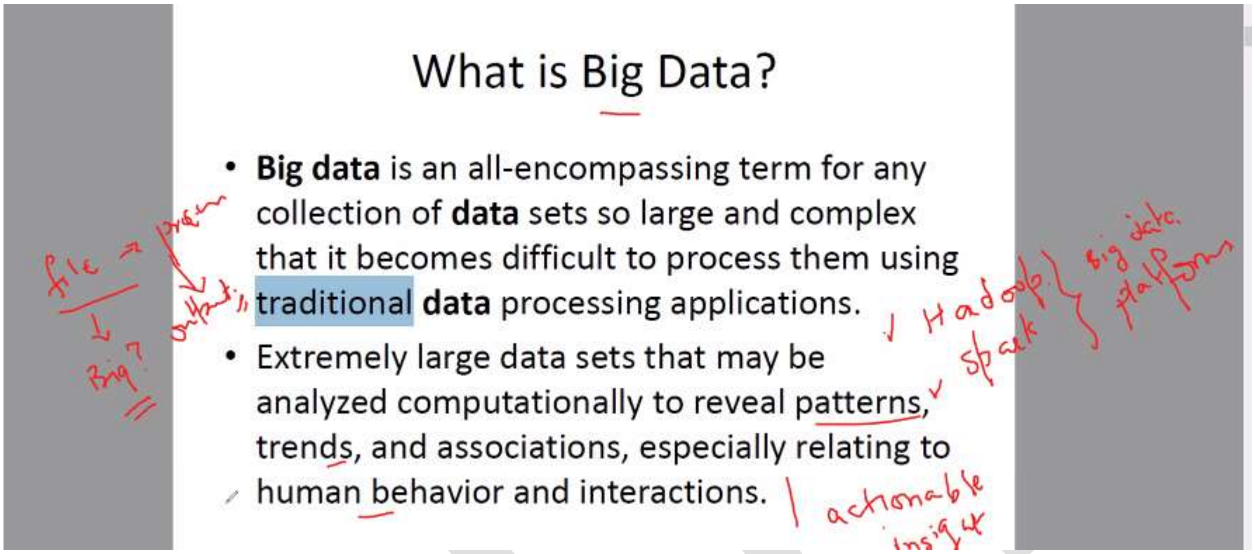
1) BIG DATA:

1. Any data which is complex and big and which cannot be get handled in single system called big data
2. The data which is generated by mostly social media
3. Complex in term of type of data and its structure
4. Actionable insight is to find pattern from raw data
5. DATA WHICH CAN NOT BE HANDLED BY TRADITIONAL SYSTEM



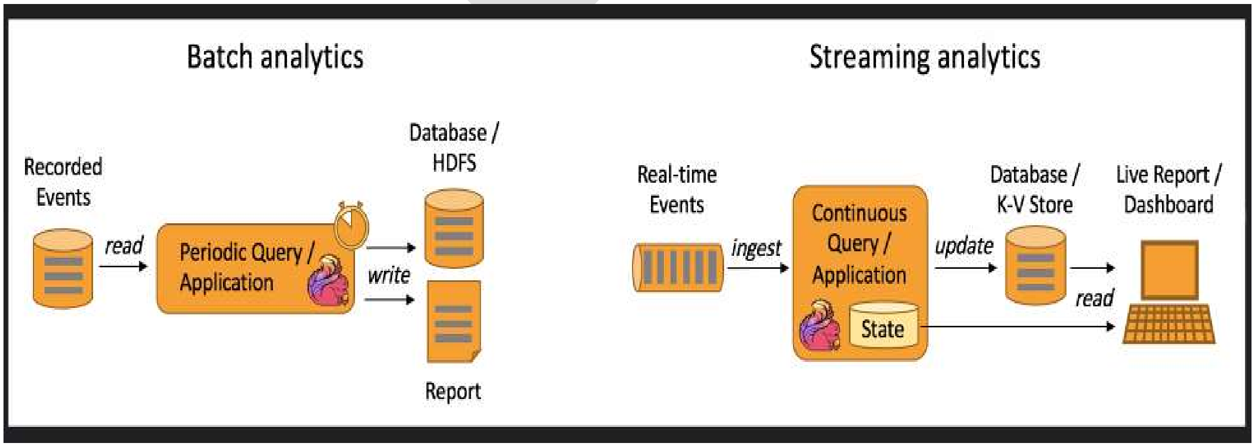
What is Big Data?

- **Big data** is an all-encompassing term for any collection of **data** sets so large and complex that it becomes difficult to process them using traditional **data** processing applications.
- Extremely large data sets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions.

2) 5V OFF BIG DATA :

1. **Volume**
2. **Variety**
3. **Velocity**
4. **Veracity**
5. **Value**

3) BATCH VS STREAMING ANALYTICS:



Example of Batch Processing:

1. Banking Institute
2. Medical Sector
3. Service Industries
4. Mostly **HADOOP** is used in Batch process

Example of streaming data:

5. Share market
6. Social media
7. Facebook and Instagram
8. Mostly **APACHE SPARKE** is used in streaming data

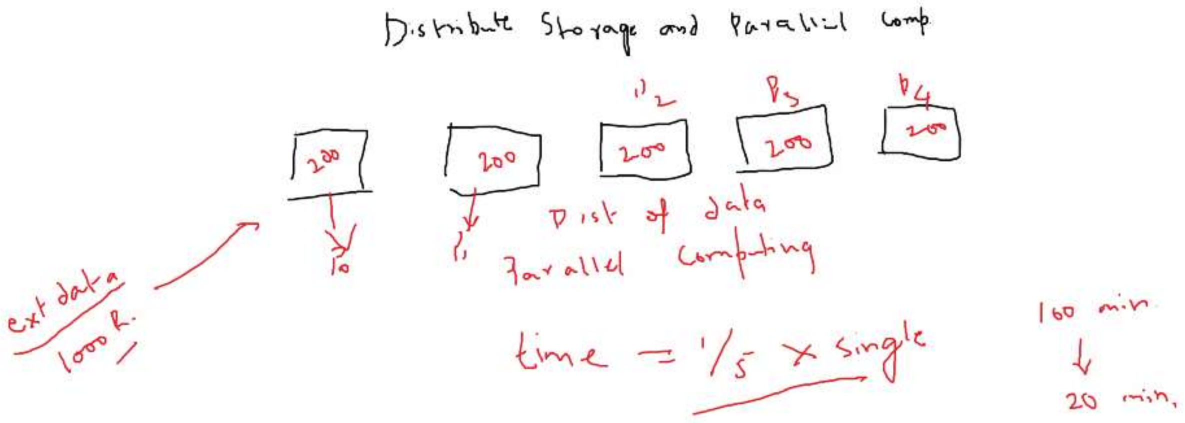
(Batch process google)

4) HADOOP:

1. In HADOOP cluster we can able to store any amount of data
2. We can extends memory space storage in Hadoop.
3. We can store any type of data whether it structured or unstructured.
4. It used to store and process data with high speed.
5. Objective of Hadoop is to process the data very fast.

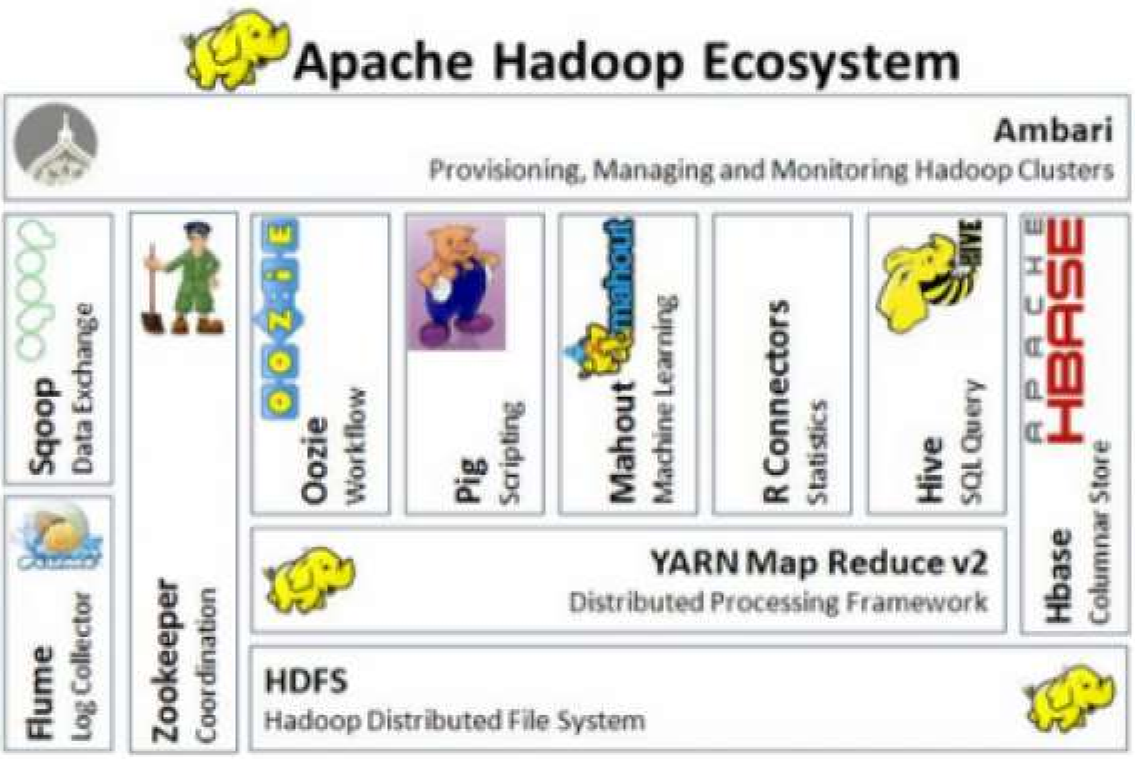
4) A. DISTRIBUTED STORAGE AND PARRALLEL COMPUTING:

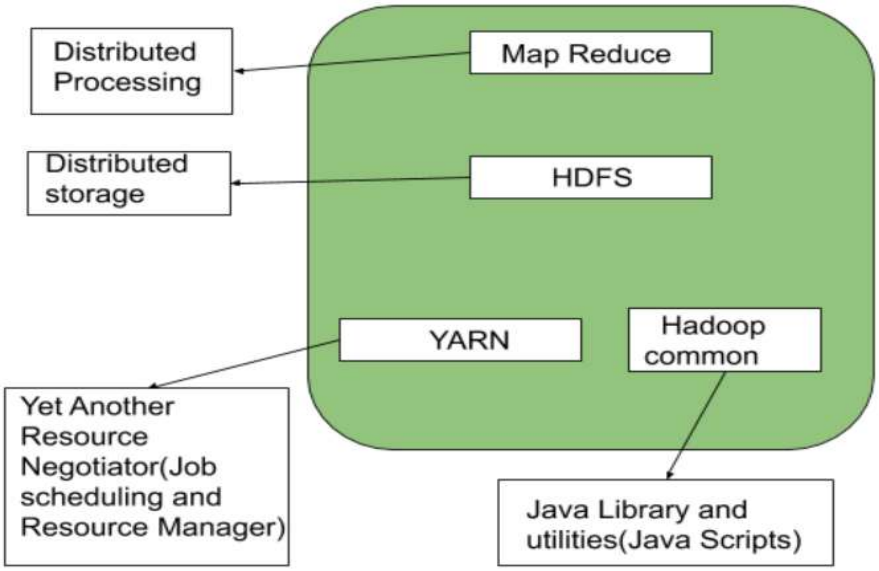
1. Data is divided and distributed and saved in multiple nodes
2. This is done for parallel computing



3. HDFS is HADOOP distributed file system which store data

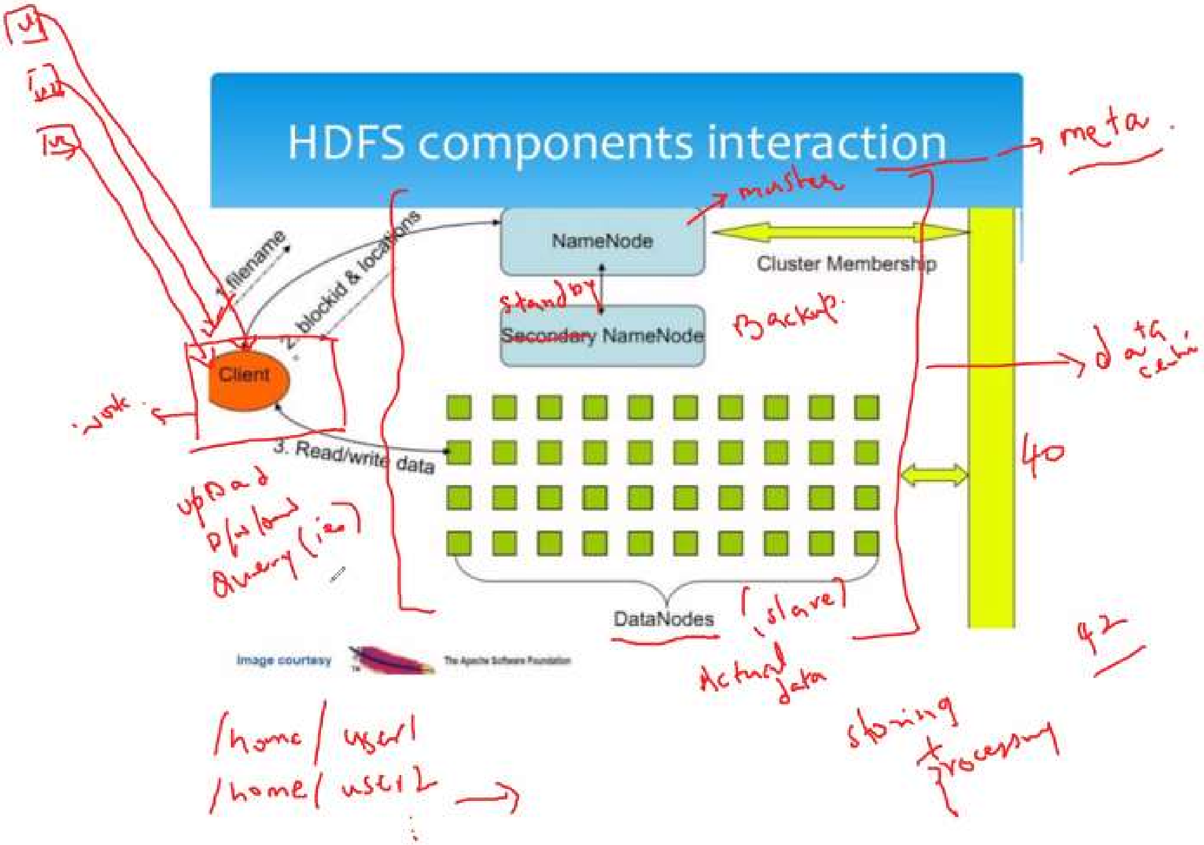
5) HADOOP ECOSYSTEM:




Distributed Processing
Map Reduce
Distributed storage
HDFS
YARN
Hadoop common
Yet Another Resource Negotiator(Job scheduling and Resource Manager)
Java Library and utilities(Java Scripts)

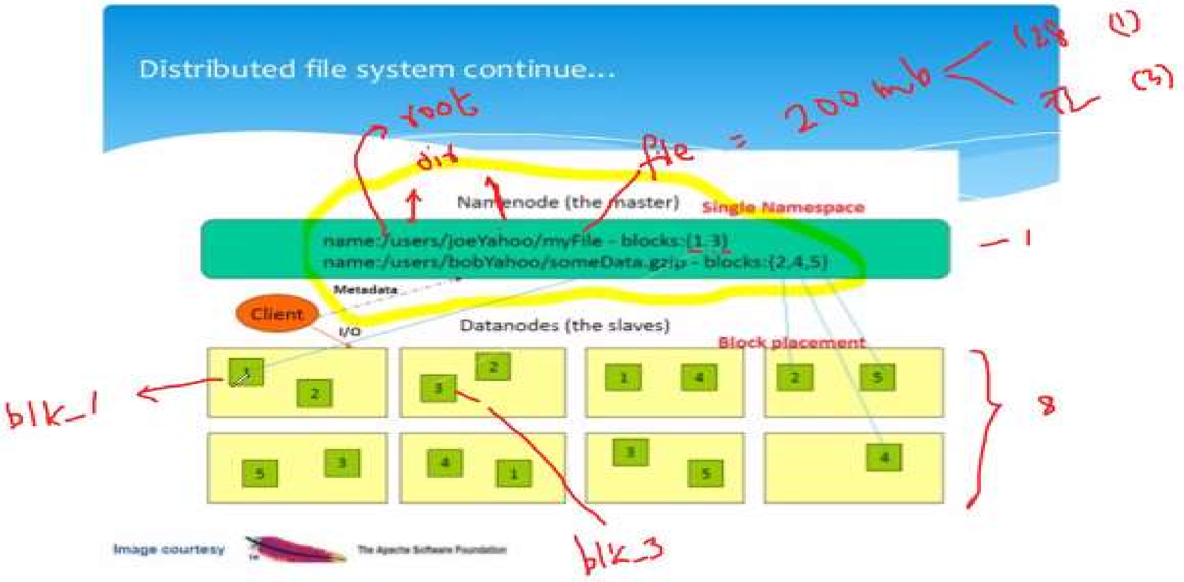
6) HADOOP ECOSYSTEM:

1. **It consist of two node NAME NODE and DATA NODE**



2. **Block size property rule:**
   a. Every memory is divided into 128 mb
   b. Which is called as blocks
3. **Replication**
   a. There are chances that node failure so than we take file backup using replication
   b. Reification factor default value is 3 and 20% space for **I/O** operation
   c. We are replicating data and accessing data called as **fault tolerance**
   d. Signal send by data node to name node called as Heart bit signal.
   e. If node is failure then name node should create new data node
   f. **Why we are using RF3?**

      2 copies are used for fault tolerance and another copy is used for disaster recovery and this is called as ***RACK AWARENESS***

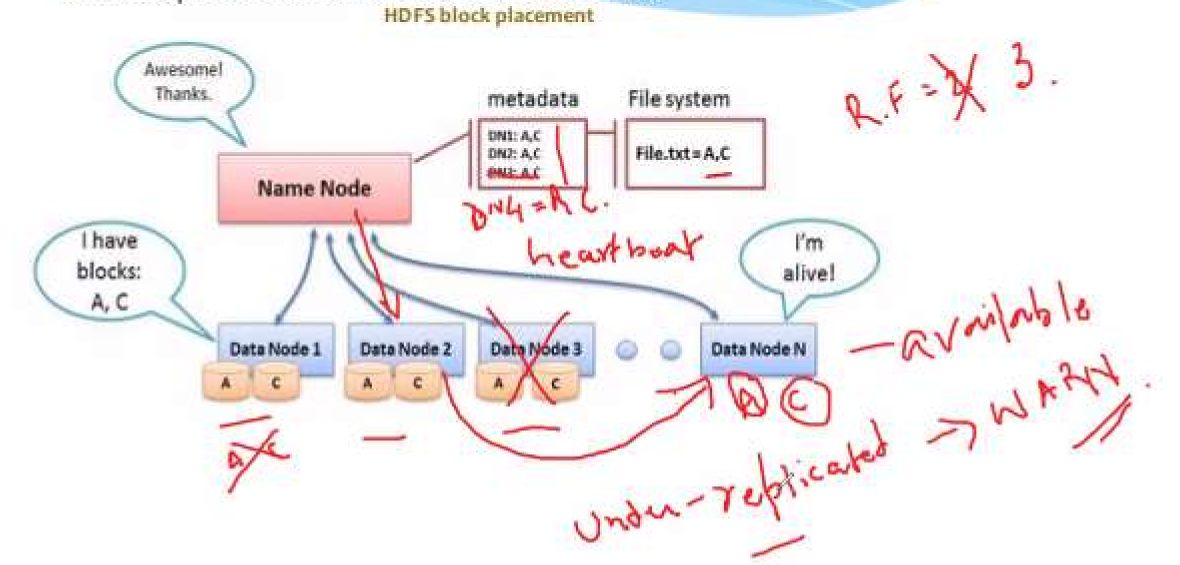


Distributed file system continue...

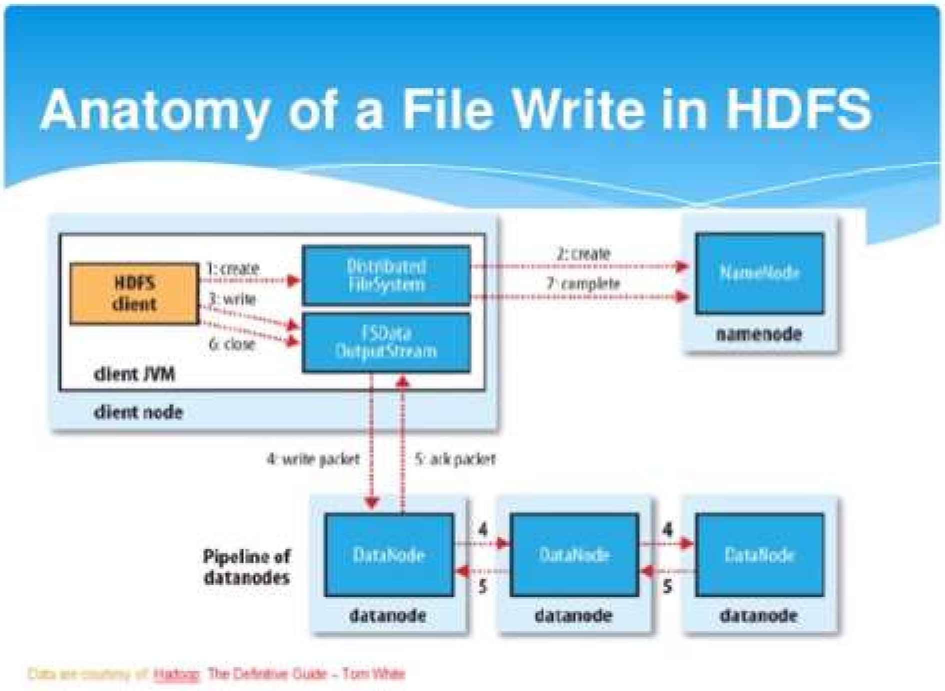
Image courtesy The Apache Software Foundation

HDFS Blocks continues...

- ❖HDFS creates several replication of the data blocks.
- ❖**Each and every data block is replicated to multiple nodes** across the cluster.
- ❖BlockReport contains all the blocks on a Datanode.



HDFS block placement



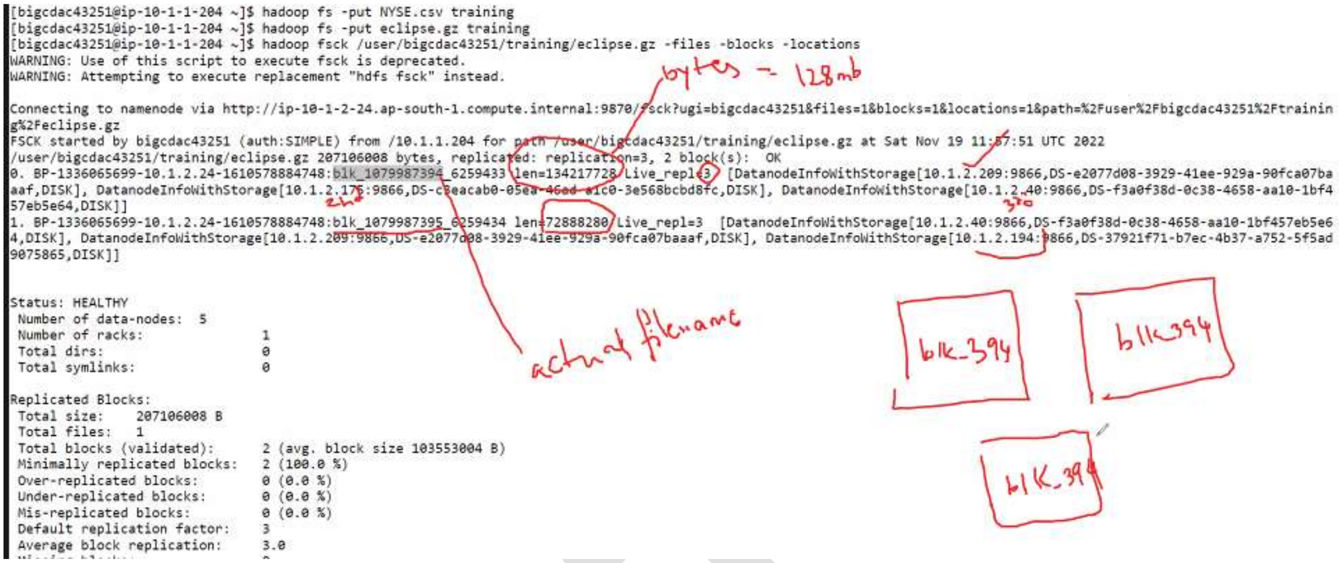
Anatomy of a File Write in HDFS

Data are courtesy of Hadoop: The Definitive Guide – Tom White

4. **Zookeeper:**
    a. Zookeeper checks the name node if name node breakdown it switches to backup name node.
    b. It checks the name node by taking heartbeat signal from name node.

8) HADOOP COMMOND:

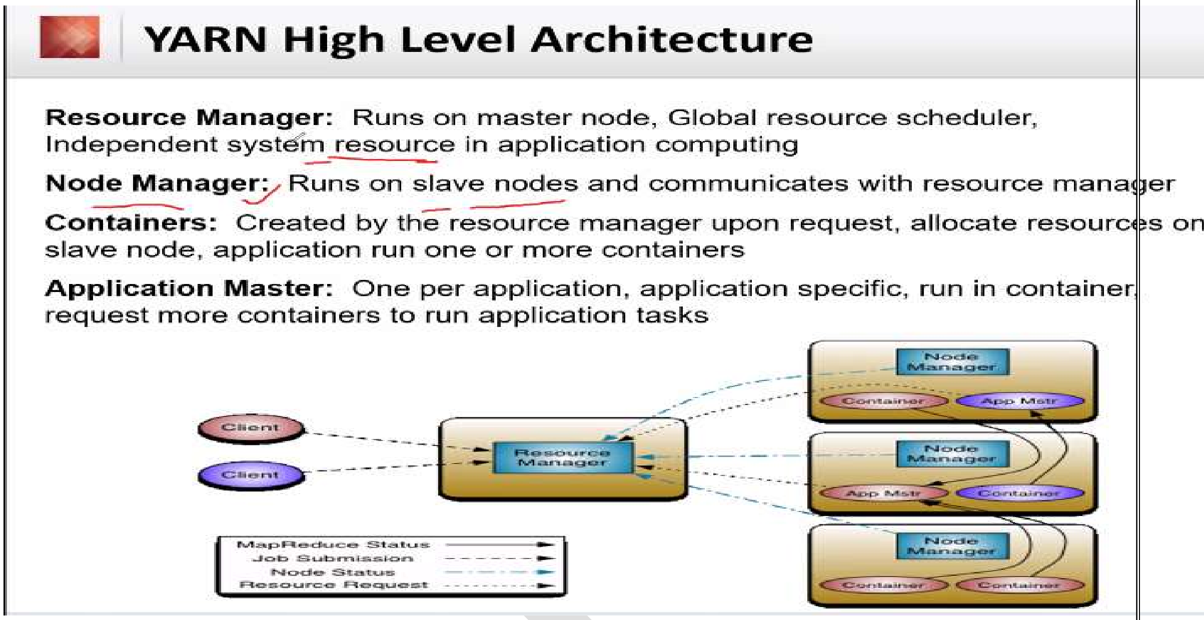
1. To use hadoop server we have to mention hadoop fs it is compulsory.
2. E.g. hadoop fs –ls /
3. java –version
4. hadoop version
5. `hadoop fs -mkdir /user/bigcdac432571/training`
    a. UPLOADING FILE
        1. hadoop fs –get training/NYSE.csv – from hadoop to terminal client
        2. hadoop fs –get training /eclipse.gz - same
        3. hadoop fs -put NYSE.csv training - putting file client to hadoop
        4. hadoop fs -put eclipse.gz training - same
        5. hadoop fsck /user/bigcdac432571/training/eclipse.gz –files –blocks -locations



```
[bigcdac43251@ip-10-1-1-204 ~]$ hadoop fs -put NYSE.csv training
[bigcdac43251@ip-10-1-1-204 ~]$ hadoop fs -put eclipse.gz training
[bigcdac43251@ip-10-1-1-204 ~]$ hadoop fsck /user/bigcdac43251/training/eclipse.gz -files -blocks -locations
WARNING: Use of this script to execute fsck is deprecated.
WARNING: Attempting to execute replacement "hdfs fsck" instead.

Connecting to namenode via http://ip-10-1-2-24.ap-south-1.compute.internal:9870/fsck?ugi=bigcdac43251&files=1&blocks=1&locations=1&path=%2Fuser%2Fbigcdac43251%2Ftrainin
g%2Feclipse.gz
FSCK started by bigcdac43251 (auth:SIMPLE) from /10.1.1.204 for path /user/bigcdac43251/training/eclipse.gz at Sat Nov 19 11:57:51 UTC 2022
/user/bigcdac43251/training/eclipse.gz 207106008 bytes, replicated: replication=3, 2 block(s):  OK
0. BP-1336065699-10.1.2.24-1610578884748:blk_1079987394_6259433 len=134217728 Live_repl=3  [DatanodeInfoWithStorage[10.1.2.209:9866,DS-e2077d08-3929-41ee-929a-90fca07ba
aaf,DISK], DatanodeInfoWithStorage[10.1.2.175:9866,DS-c3eacab0-05ea-46ed-a1c0-3e568bcbd8fc,DISK], DatanodeInfoWithStorage[10.1.2.40:9866,DS-f3a0f38d-0c38-4658-aa10-1bf4
57eb5e64,DISK]]
1. BP-1336065699-10.1.2.24-1610578884748:blk_1079987395_6259434 len=72888280 Live_repl=3  [DatanodeInfoWithStorage[10.1.2.40:9866,DS-f3a0f38d-0c38-4658-aa10-1bf457eb5e6
4,DISK], DatanodeInfoWithStorage[10.1.2.209:9866,DS-e2077d08-3929-41ee-929a-90fca07baaaf,DISK], DatanodeInfoWithStorage[10.1.2.194:9866,DS-37921f71-b7ec-4b37-a752-5f5ad
9075865,DISK]]


Status: HEALTHY
 Number of data-nodes:  5
 Number of racks:               1
 Total dirs:                    0
 Total symlinks:                0

Replicated Blocks:
 Total size:    207106008 B
 Total files:   1
 Total blocks (validated):      2 (avg. block size 103553004 B)
 Minimally replicated blocks:   2 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
```

6. How to change replica factor: `hadoop fs -setrep 4 training/a.iso`
7. **Configuration filed:**
   a. Core-default.xml
   b. Hdfs-default.xml
   c. Marpred-default.xml
   d. Yearn-deafult.xml
8. **Hdfs-default.xml:**
   a. This file consist of all the keywords and all the data of hdfs
   b. https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml
   c. Core-site.xml and hdfs-site.xml is used to changes the properties of cluster. (Properties like block size and replication factor etc.)

6. **YARN: YET ANOTHER RESOURCE NEGOTIATOR**
   a. YARN consist of master client services
   b. Master act as Resource Manager (one per cluster)
   c. Slave- node manager (one per data node)
   d. Name node and data node combines called as hdfs
   e. For mapper programmer resource is created in same node where data is present called as data locality. Mappers are run on same node where data is stored
   f. One application master handles one query or jobs



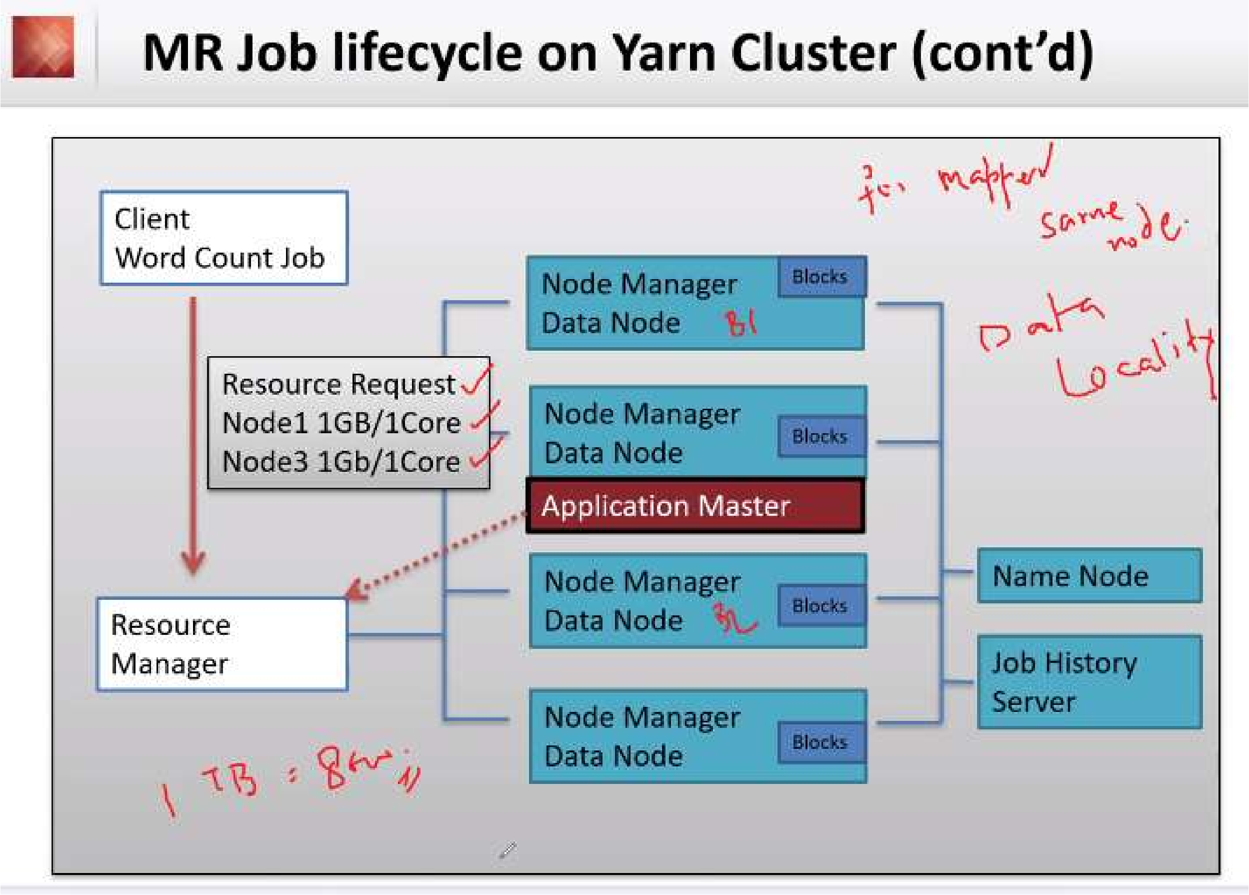
YARN High Level Architecture

Resource Manager: Runs on master node, Global resource scheduler, Independent system resource in application computing

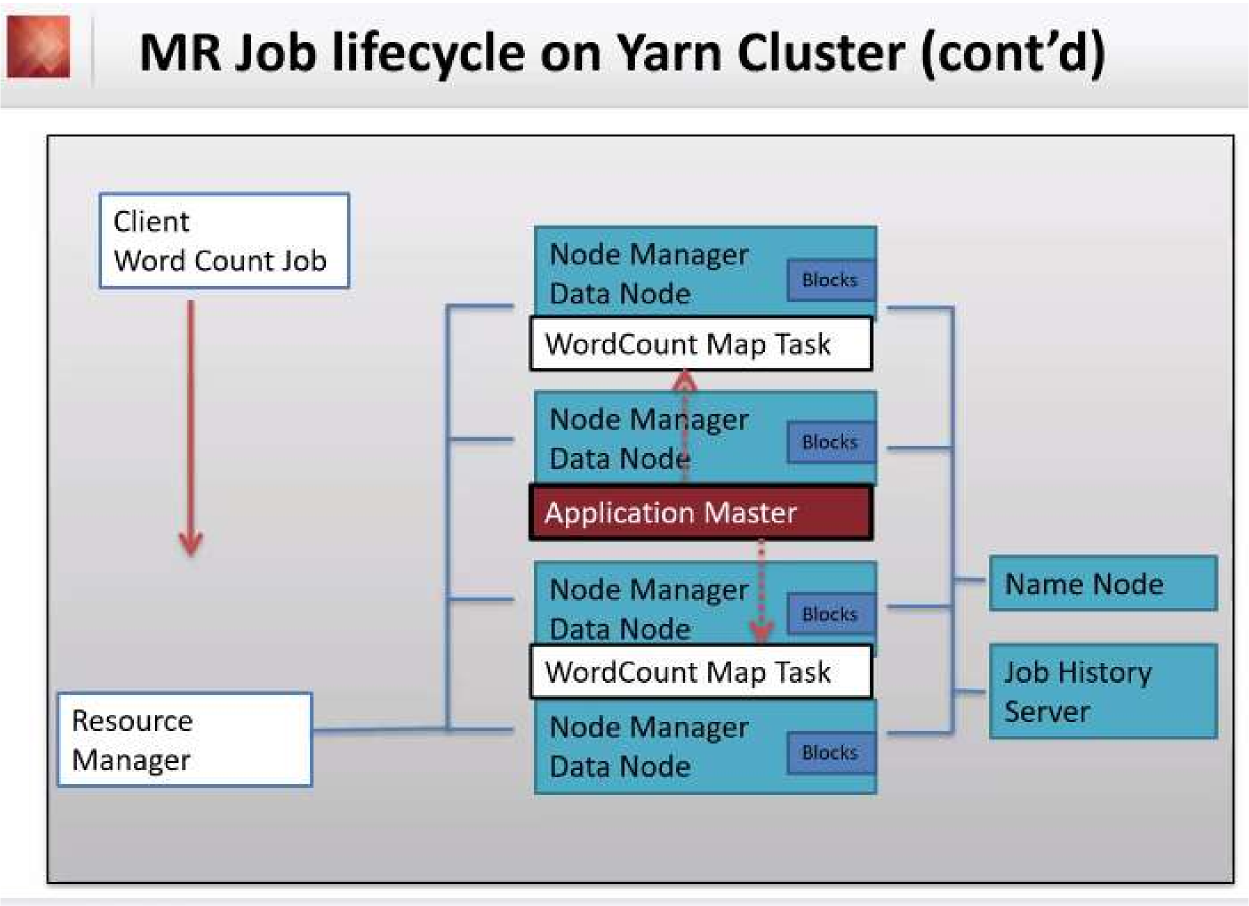
Node Manager: Runs on slave nodes and communicates with resource manager

Containers: Created by the resource manager upon request, allocate resources on slave node, application run one or more containers

Application Master: One per application, application specific, run in container, request more containers to run application tasks



MR Job lifecycle on Yarn Cluster (cont'd)



MR Job lifecycle on Yarn Cluster (cont'd)

Client
Word Count Job

Resource
Manager

Node Manager
Data Node
Blocks

WordCount Map Task

Node Manager
Data Node
Blocks

Application Master

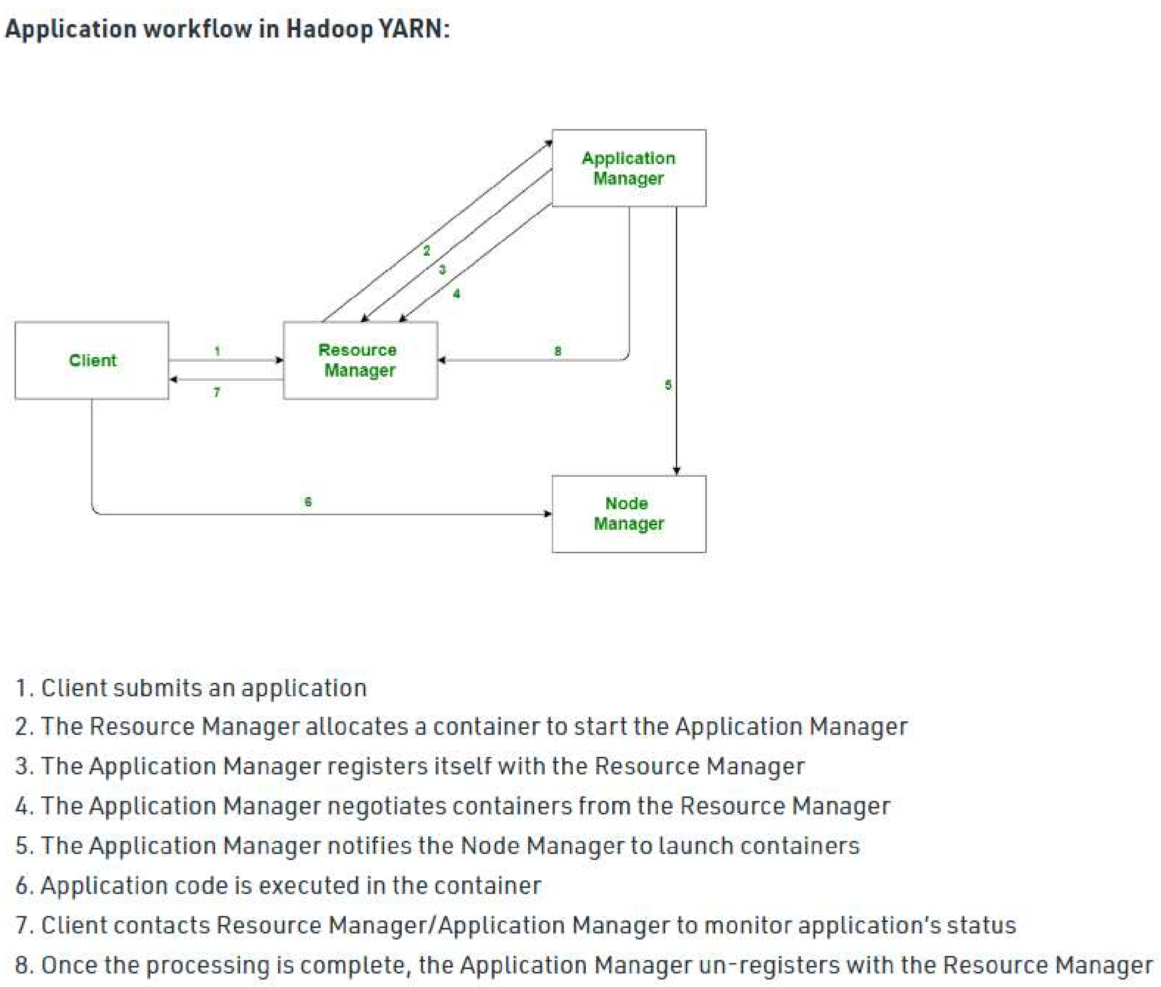
Node Manager
Data Node
Blocks

WordCount Map Task

Node Manager
Data Node
Blocks

Name Node

Job History
Server



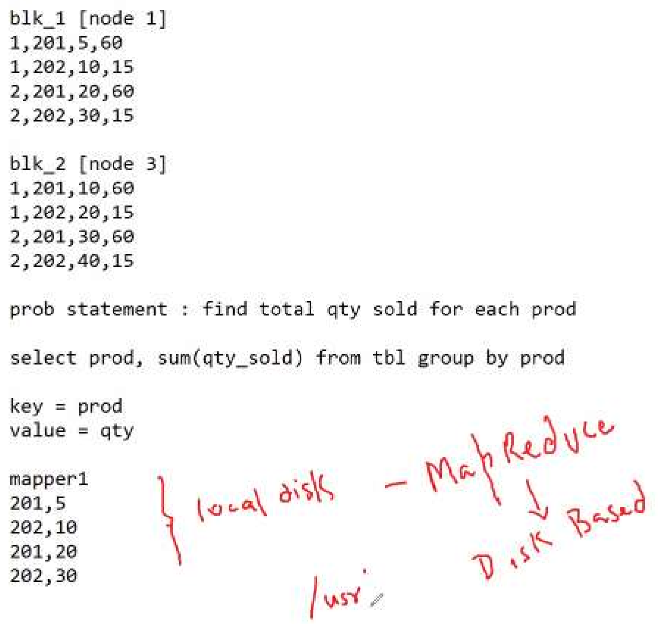
Application workflow in Hadoop YARN:

1. Client submits an application
2. The Resource Manager allocates a container to start the Application Manager
3. The Application Manager registers itself with the Resource Manager
4. The Application Manager negotiates containers from the Resource Manager
5. The Application Manager notifies the Node Manager to launch containers
6. Application code is executed in the container
7. Client contacts Resource Manager/Application Manager to monitor application's status
8. Once the processing is complete, the Application Manager un-registers with the Resource Manager

7. MAP REDUCE:

a. **Mapper:** Mapper is the programed which create the key value pair of data

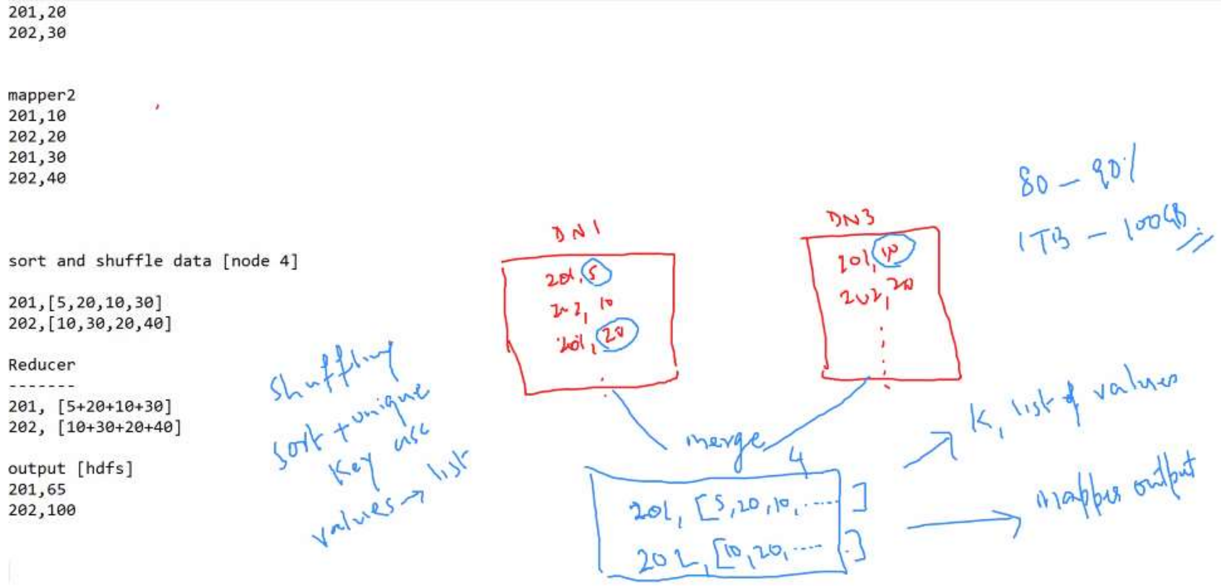
   1. In mapper the data is stored in disk memory so that it is called as disk based processing



```
blk_1 [node 1]
1,201,5,60
1,202,10,15
2,201,20,60
2,202,30,15

blk_2 [node 3]
1,201,10,60
1,202,20,15
2,201,30,60
2,202,40,15

prob statement : find total qty sold for each prod

select prod, sum(qty_sold) from tbl group by prod

key = prod
value = qty

mapper1
201,5
202,10
201,20
202,30
```

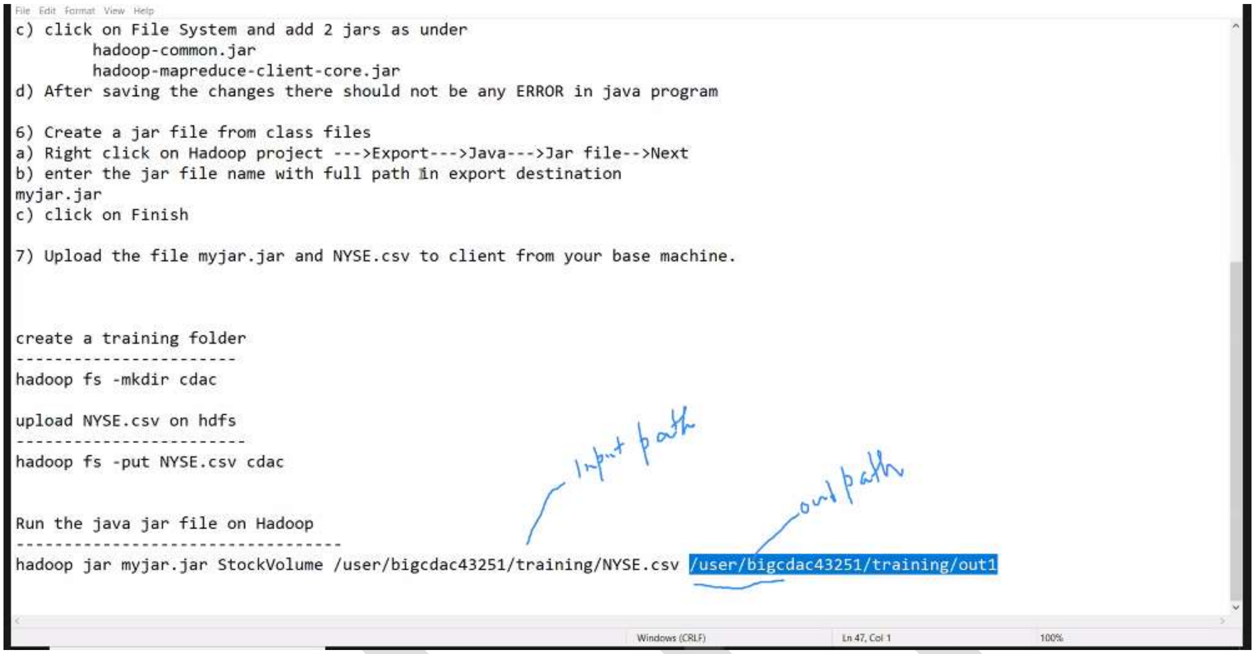
2. **Shuffling –** shuffling the data is called as moving data and merging the data on different location by.
3. **Sorting-** After shuffling the keys are sorted by acceding odder and each key the list of its value is made called as sorting.
4. **Output** of matter is always key value and which is filter data

b. Reducer: Reducer is programmed to process the data done by mapper



```
201,20
202,30


mapper2
201,10
202,20
201,30
202,40


sort and shuffle data [node 4]

201,[5,20,10,30]
202,[10,30,20,40]

Reducer
-------
201, [5+20+10+30]
202, [10+30+20+40]

output [hdfs]
201,65
202,100
```

8. **Jar tvf MyJar.jar -** will shows all class present in that jar comment



```
c) click on File System and add 2 jars as under
        hadoop-common.jar
        hadoop-mapreduce-client-core.jar
d) After saving the changes there should not be any ERROR in java program

6) Create a jar file from class files
a) Right click on Hadoop project --->Export--->Java--->Jar file-->Next
b) enter the jar file name with full path in export destination
myjar.jar
c) click on Finish

7) Upload the file myjar.jar and NYSE.csv to client from your base machine.



create a training folder
-----------------------
hadoop fs -mkdir cdac

upload NYSE.csv on hdfs
------------------------
hadoop fs -put NYSE.csv cdac


Run the java jar file on Hadoop
----------------------------------
hadoop jar myjar.jar StockVolume /user/bigcdac43251/training/NYSE.csv /user/bigcdac43251/training/out1
```

1. To run the jar program we have to used following command

```
 hadoop jar MyJar.jar StockVolume
/user/bigcdac432571/training/NYSE.csv
/user/bigcdac432571/out1
```

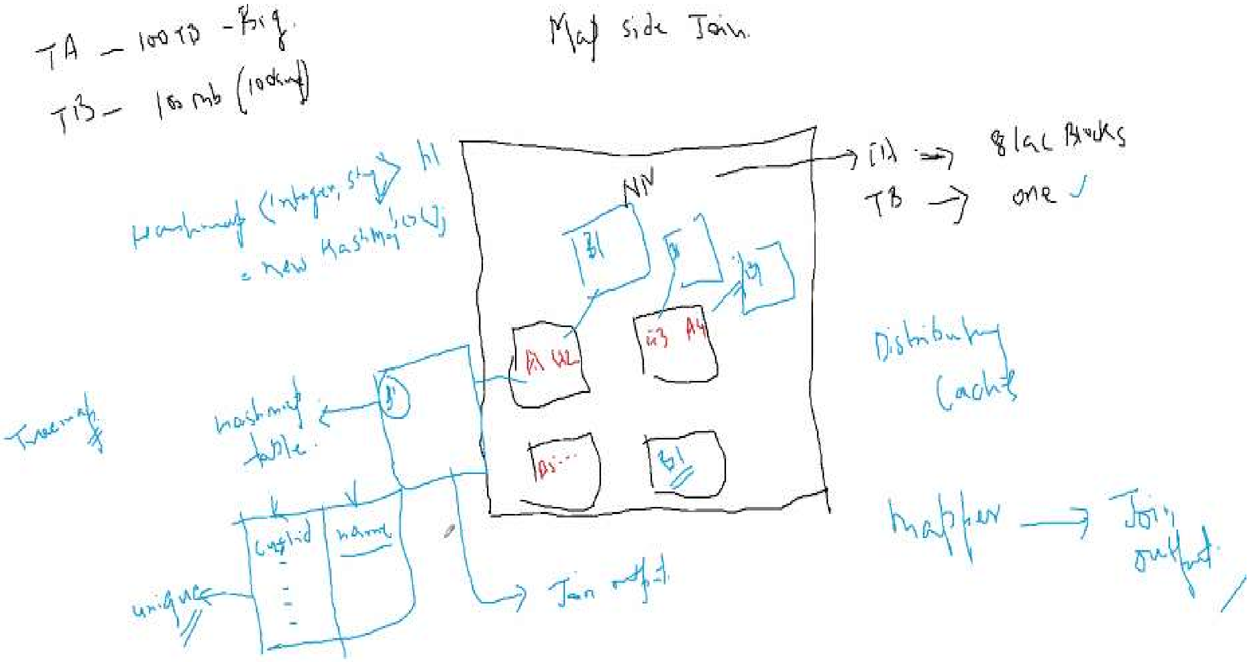
9. **COMBINER:**
   1. Combiner is mini reducer on min rescuer on mapper level.
   2. It reduces the work load of reducer.
   3. It called as optimization technique.
   4. Combiner which is used reducer logic called as default combiner

10. **PARTITIONER**
    a. **Default pardoner(Hash Portioning) (setNumReduceTasks(2))**
       i. Hash value % (modulo) number of portioning will decide portioning number
       ii. Automatically set portioning depends upon reduces task given

    b. **Customized portioning :**
       i. This is done by user by using portioning class
       ii. User need to take that number of portioning should be same as number of reduces task

11. **JOINTS IN HADOOP**
    a. We can join big data set with small data set 100tb data with 100mb (**MAP SIDE JOIN IN HADOOP**) E.g. joining customer table with sales data

1. We are distributing data to each block container called as distribution cashing
2. This is done in the mapping process
3. Cashing is done by using hash map level
4. By default it is left outer join
5. **Method in map class**
    1. **Setup method :** it will run only one it will not run loop
    2. **Map Method:** This method is to write mapper logic and it will run in loop
    3. **Clean Up method:** This method is use for clean up purpose.

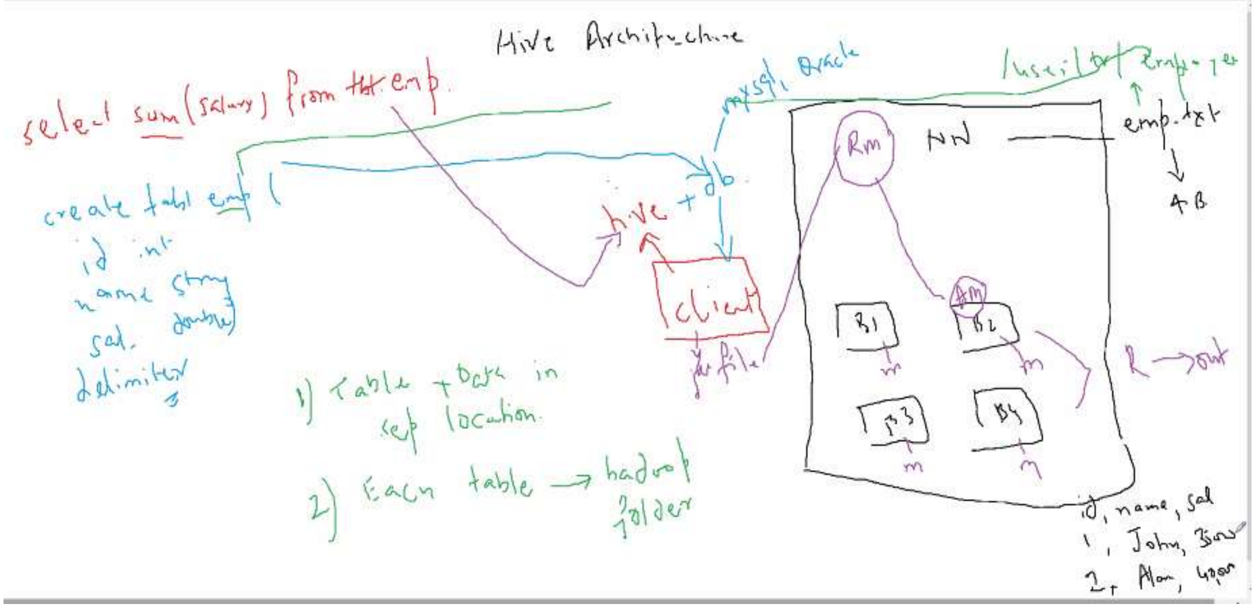


b. We can join big data with another big data 100tb to 100tb **(REDUCE SIDE JOIN)**

1. Here we are using multiple input class
2. Key of both class should be same is the condition
3. Here we have to add tag
4. By default it is full join by default
5. The data is gets sort depend upon original key, even the data is seems to be not sort sorted even though data is sorted manner

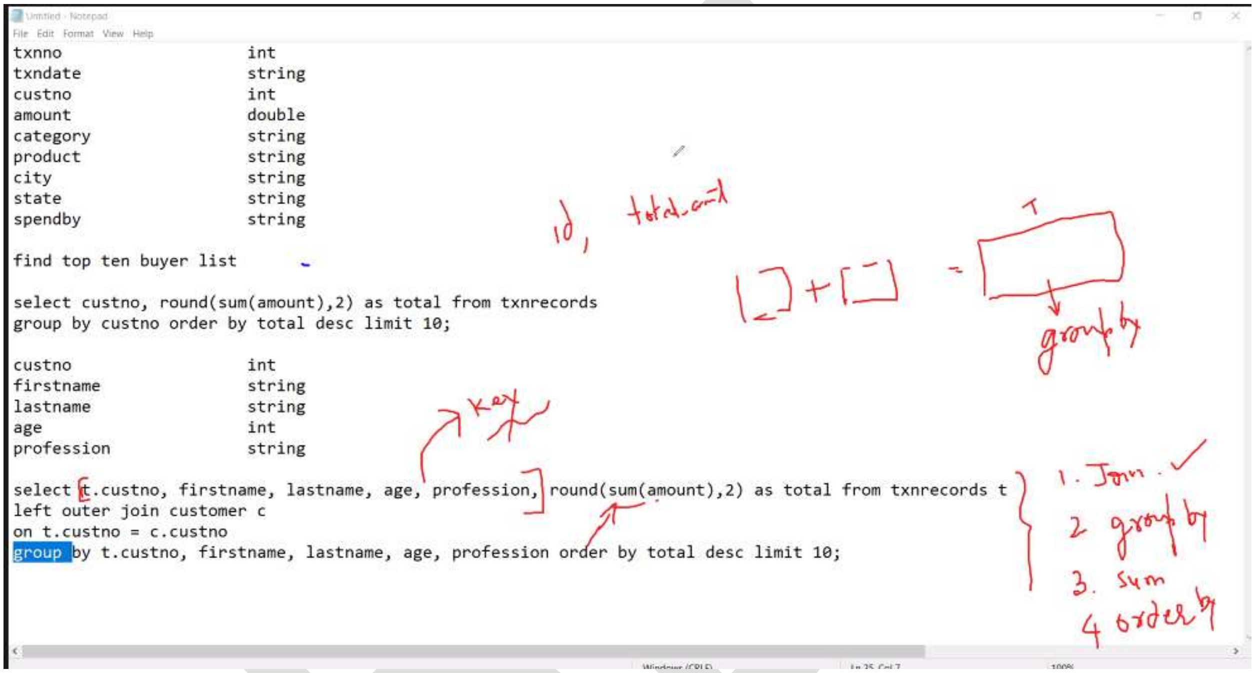
9) HIVE:

1. Hive is a tool which is used to analyses data
2. It uses sql query
3. It converted sql query into java
4. It is interface converting sql query into java code
5. It is used for only structure data
6. Hive is install on client side
7. In client site table is created in e store metadata of databases that is structure of table for support to hive
8. The table is created to local db and if supportive database is not here hive is not going to be launch this database is called as supportive database
9. Here data and table are not together they are in separate location (table is empty and data is store in hdfs)
10. When table is created each table is mapped to Hadoop folder it is automatically without mapping we cannot create table
11. **Orc and parquet is mostly used files system which uses compressed and binary format. orc has one advantage that we can write update and delete command on orc**



12. load data local inpath 'NYSE.csv' overwrite into table nyse;
13. row format delimited fields terminated by ',' this command will set the row delimiter by , by default it is save as tab serrated

14. When u want to add multiple in single table then all u need to add file to same folder where previous data is available.
15. This is done becoz we are reading data during runtime.
16. When we use location during creating mapping it is called as explicit mapping.
    `location '/user/bigcdac432571/sales';`
17. `User/hive/wearhouse is default path for implicit mapping.`



```
txnno                   int
txndate                 string
custno                  int
amount                  double
category                string
product                 string
city                    string
state                   string
spendby                 string

find top ten buyer list

select custno, round(sum(amount),2) as total from txnrecords
group by custno order by total desc limit 10;

custno                  int
firstname               string
lastname                string
age                     int
profession              string

select t.custno, firstname, lastname, age, profession, round(sum(amount),2) as total from txnrecords t
left outer join customer c
on t.custno = c.custno
group by t.custno, firstname, lastname, age, profession order by total desc limit 10;
```

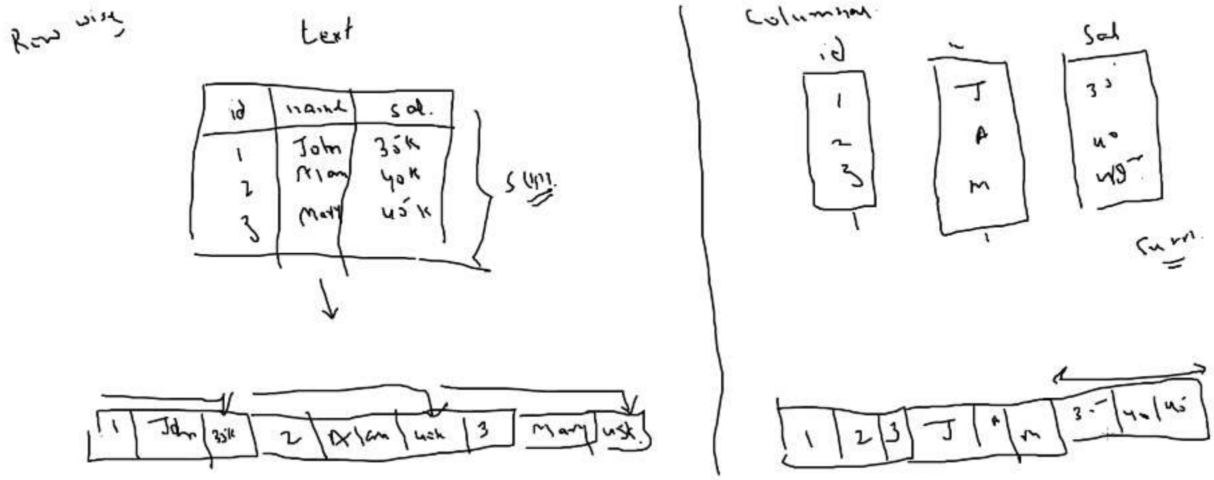
18. While doing string comparison in hive always use trim(upper()) function so that it will give right output.

1. `select * from airport where trim(upper(country))='INDIA' limit 10;`

19. HIVE DATA FORMAM

1.ORC FORMAT

1. ROW WISE DATA ANA COLUUM WISE DATA
2. Row wise data required more time to access and operation
3. Where Colum wise data was high speed for operation



4. orc and parquet data files are Coolum base files and they are secure and compress
5. Conversion and DE conversion of orc file done by serialization and deserialization libraries SerDe libraries'.
6. We can update and delete in orc format.
7. Data warehouse contains summaries data from which user can access data much faster than accessing row data.
8. Hive is also called as data warehouse in Hadoop.
9. Scd= slowly changing data
10. Etl- extract transformation tool like Hadoop.
11. We can use curd operation in hive table only on orc but only when transition true property is on.

2. **PARQUATE FORMAT**
   1. It less compress than ORC table

20. TYPES OF TABLE

1. **MANAGE TABLE:**
   a. When we drop table data are also get deleted called as manage table
   b. By default table get created as managed table.
2. **EXTERNAL ITABLE;**

a. It will not delete table it will just delete the structure.

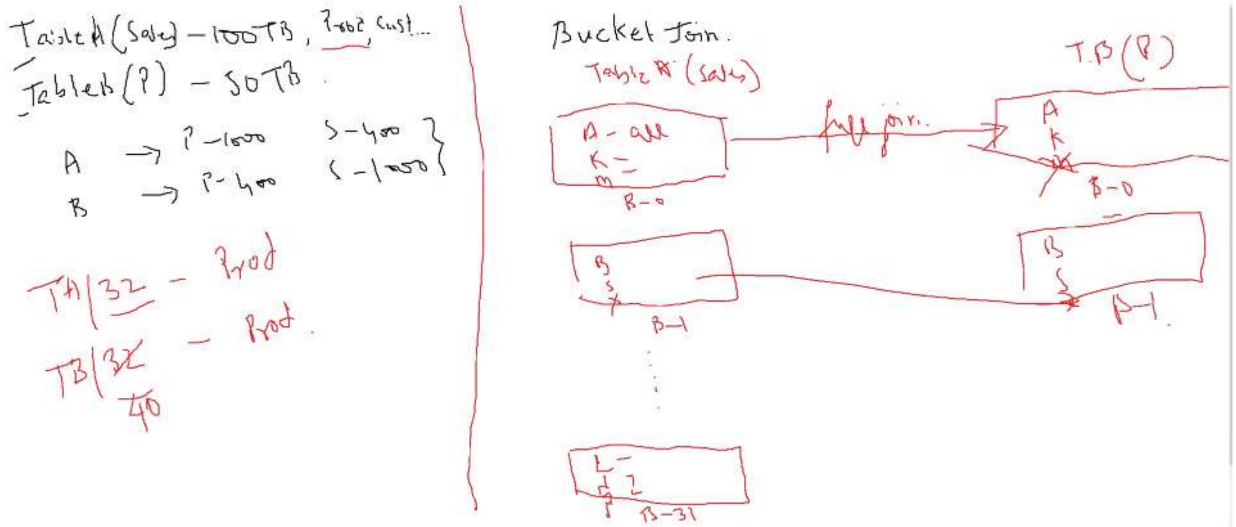
21. HIVE OPTIMISATION TECHNIQUE:

A. PARTITIONIG

1. Here we create separate table which consist of partition dependent files so that we can segregated table
2. Substring is used to get specific string from string e.g. substring("rohan",1,2) then its answer will be ro.

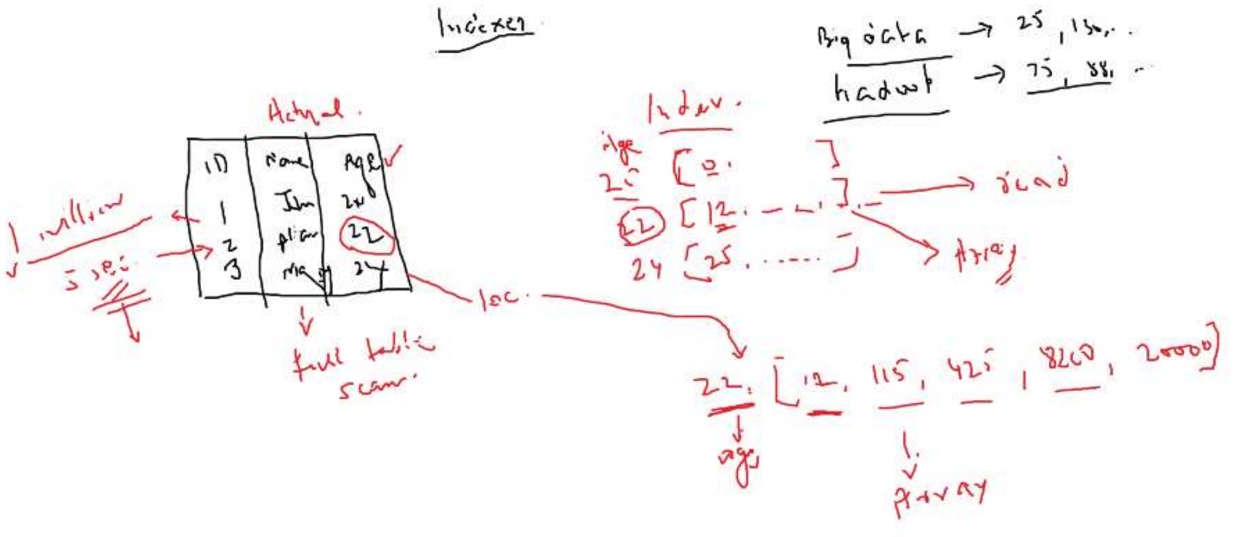
B. BUCKETTING

1. We define partition by our own called as bucketing
2. By default bucketing has dependent to hash value of key % no of bucket = bucket number
3. Bucketing can be done with two way with partitioning and alone
4. You can control partition by creating bucket
5. Bucketing join



C. INDEXES

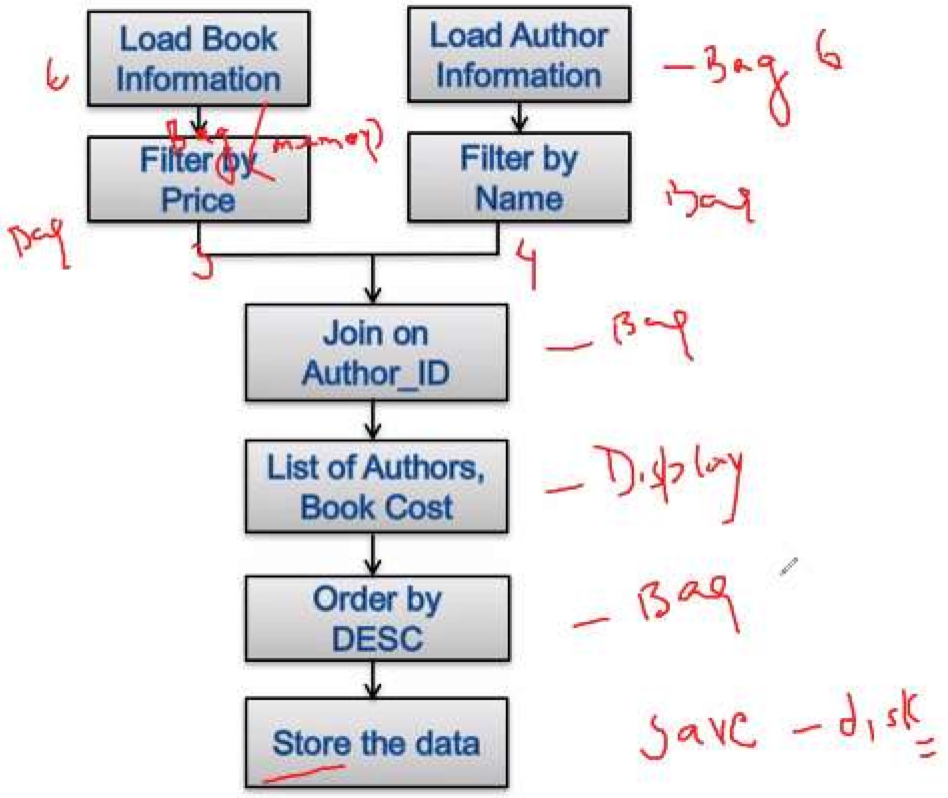
1. By use of indexes we access data much faster it like having index page of book cover.
2. Here we create index with Colum and stores the offset byte position of record as list to that key.



3. Index tables does not get automatic updated we have to run rebuild the index command.
4. alter index prof_index on customer rebuild; this command will update our index table .
5. orc does not need indexing in orc it gets automatically indexing.
6. If we drop table then indexes are also get deleted

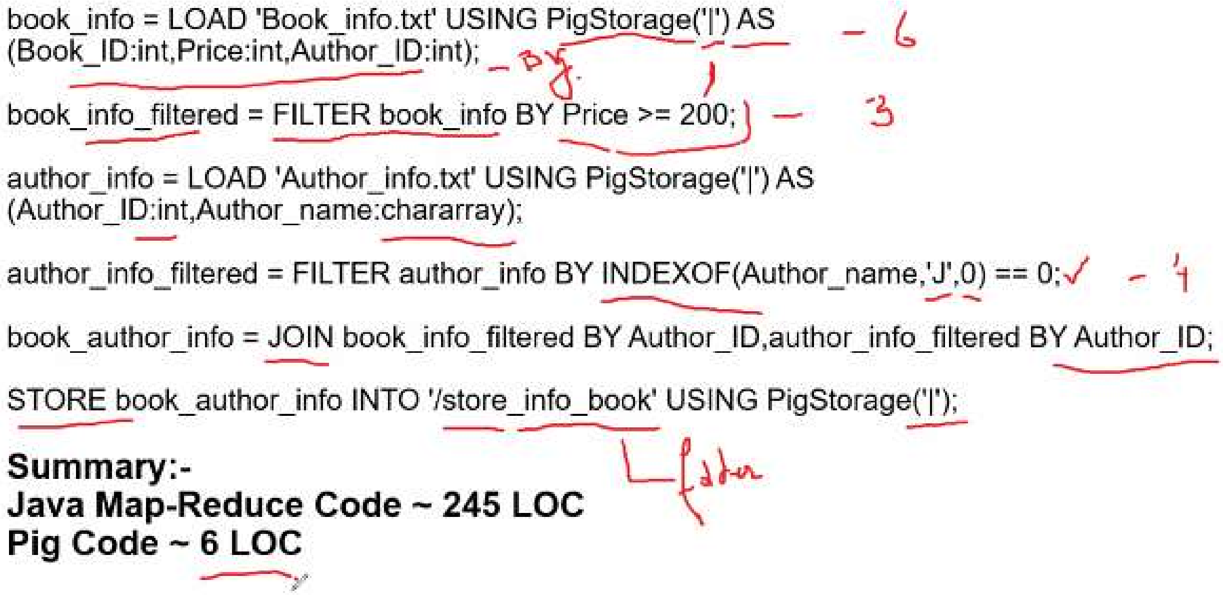
10) PIG:

1. Difference in hive and pig is, pig process all kind of data structure and unstructured.
2. It process data process by process i.e. step by step
3. We using pig Latin
4. It is very similar to hive
5. They used map reduce technology
6. Syntax is very easy to use.
7. We are loading data in memory it called bag;
8. Bag is like temporary table;



9. Logical and physical bag get load when and only when action command get execute. This concept is called as lazy evaluation.

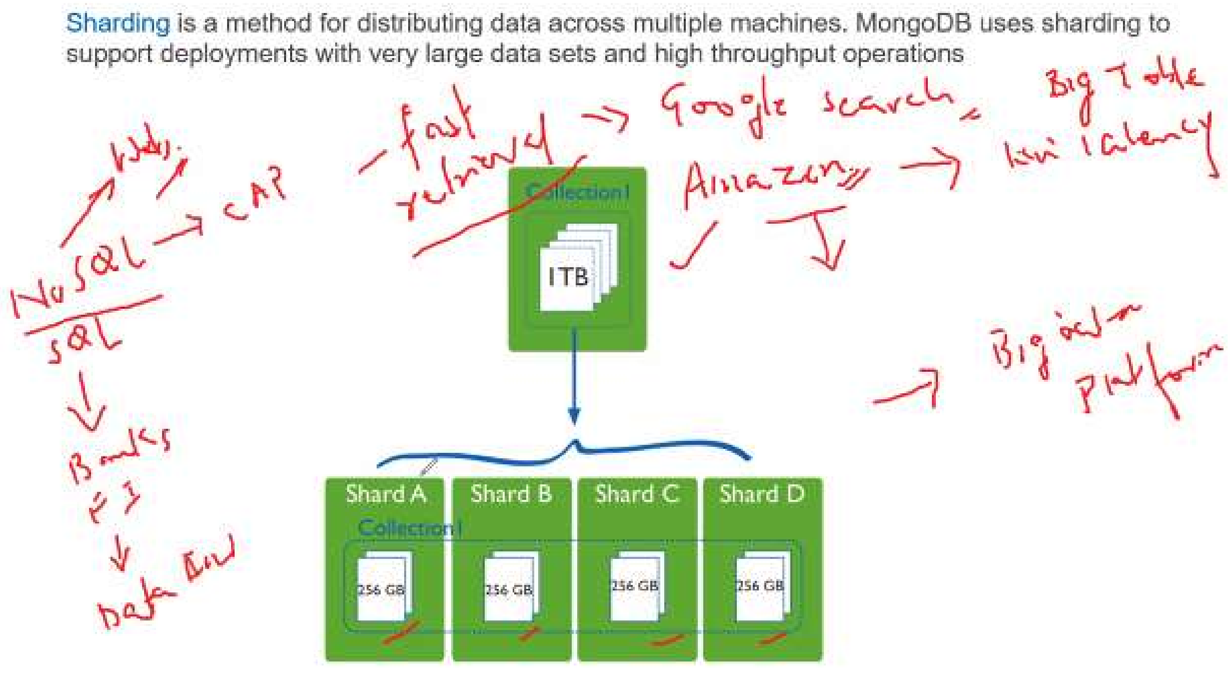
10. We occupy memory only on runtime and runtime get only when we run action command.



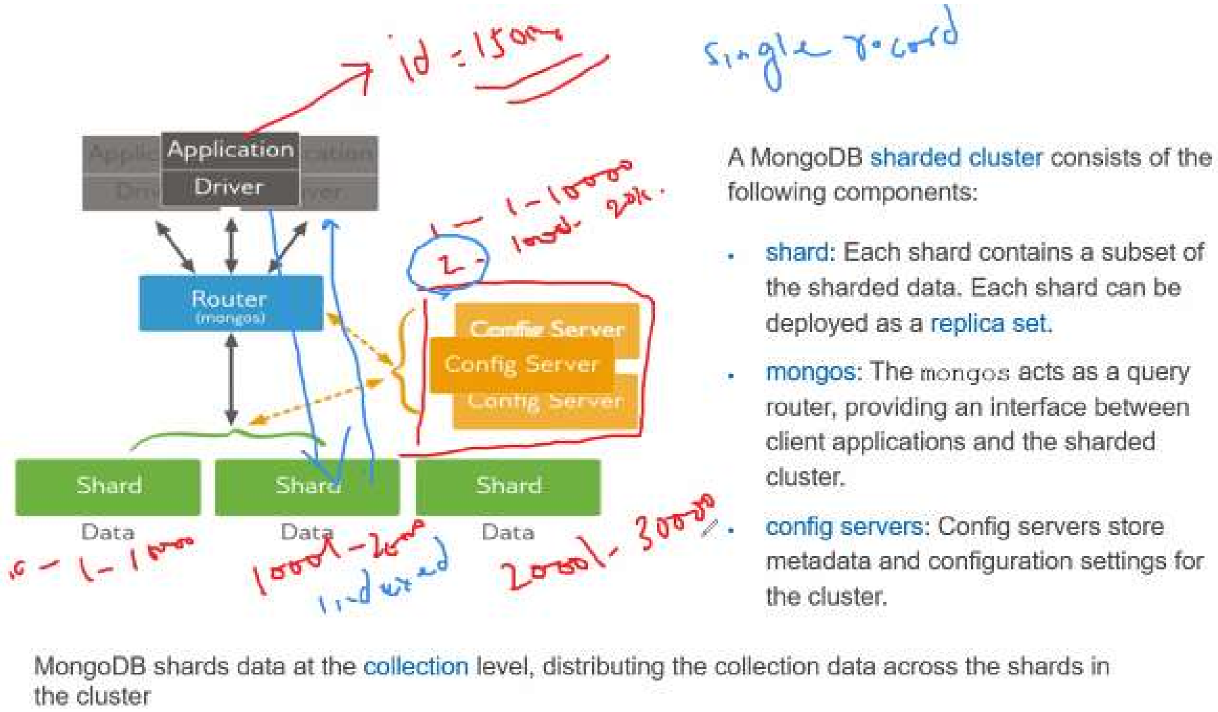
```
book_info = LOAD 'Book_info.txt' USING PigStorage('|') AS
(Book_ID:int,Price:int,Author_ID:int);

book_info_filtered = FILTER book_info BY Price >= 200;

author_info = LOAD 'Author_info.txt' USING PigStorage('|') AS
(Author_ID:int,Author_name:chararray);

author_info_filtered = FILTER author_info BY INDEXOF(Author_name,'J',0) == 0;

book_author_info = JOIN book_info_filtered BY Author_ID,author_info_filtered BY Author_ID;

STORE book_author_info INTO '/store_info_book' USING PigStorage('|');
```

Summary:-
Java Map-Reduce Code ~ 245 LOC
Pig Code ~ 6 LOC

11) HBASE:



Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations

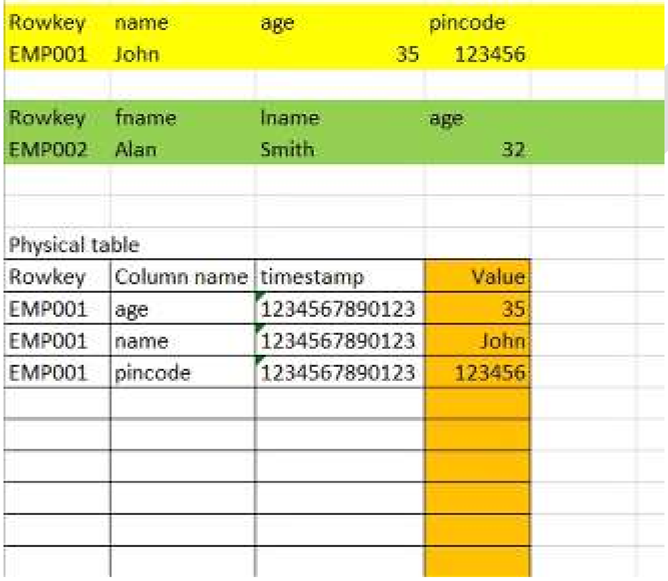


A MongoDB sharded cluster consists of the following components:

- shard: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.
- mongos: The mongos acts as a query router, providing an interface between client applications and the sharded cluster.
- config servers: Config servers store metadata and configuration settings for the cluster.

MongoDB shards data at the collection level, distributing the collection data across the shards in the cluster

1. Hbase is distributed Colum-oriented data store built on top hdfs
2. Hbase has its own primary key called as row key
3. We have four table which are fix in hbase which consist of rowkey , Colum name, timestamp ,value
4. What is Colum null:=



Rowkey	name	age	pincode
EMP001	John	35	123456

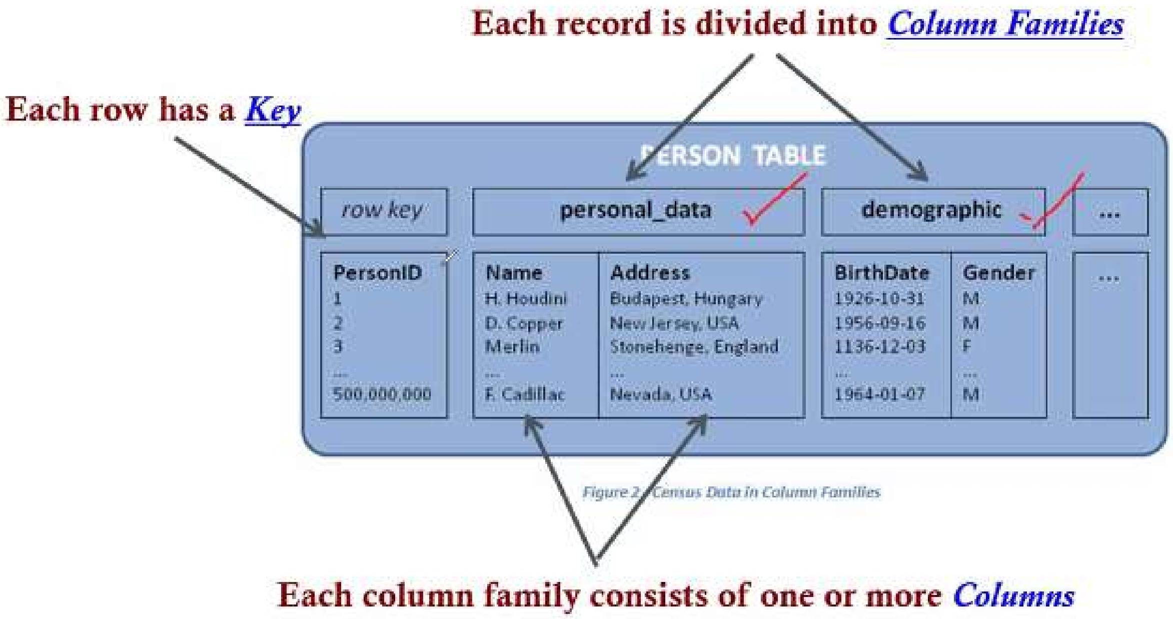
Rowkey	fname	lname	age
EMP002	Alan	Smith	32

Physical table

Rowkey	Column name	timestamp	Value
EMP001	age	1234567890123	35
EMP001	name	1234567890123	John
EMP001	pincode	1234567890123	123456

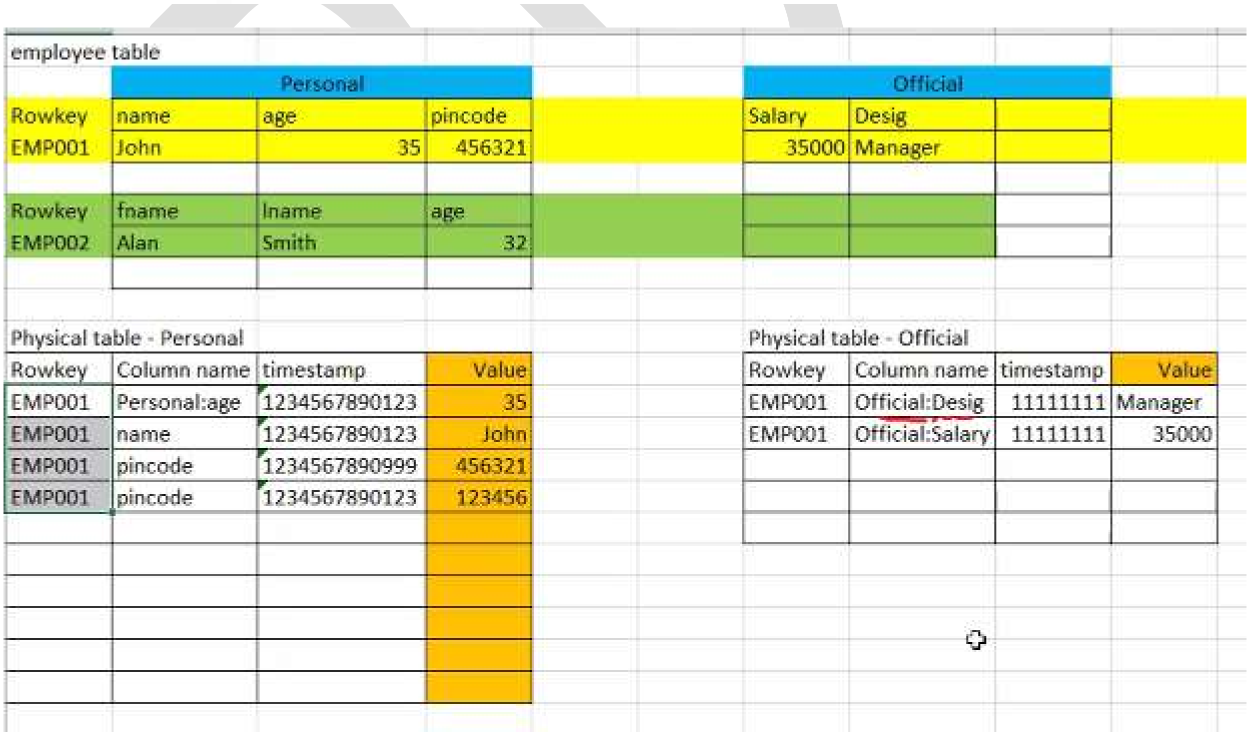
5. This table is called as h table and this table is get divided into region table and it has limit of 1gb and it get store in hdfs
6. All hbase data get store on hdfs as a blocks.
7. It has only store data as byte array it has only one data type as byte array

8. Hdfs used when we have bulk record storing where hbase is used in real time storing data
9. We does not used hbase in analysis it is used for inserting and retiring data. And hfs used for analysis
10. In hbase we can save old data as well as new data , i.e. it can store old data before updating and after updating and it is distinguish by timestamp


Each record is divided into Column Families
Each row has a Key
PERSON TABLE
row key
personal_data
demographic
...
Each column family consists of one or more Columns

PersonID	Name	Address	BirthDate	Gender	...
1	H. Houdini	Budapest, Hungary	1926-10-31	M	
2	D. Copper	New Jersey, USA	1956-09-16	M	
3	Merlin	Stonehenge, England	1136-12-03	F	
...	...	...	...	...	
500,000,000	F. Cadillac	Nevada, USA	1964-01-07	M	

Figure 2. Census Data in Column Families



employee table

	Personal			Official		
Rowkey	name	age	pincode	Salary	Desig	
EMP001	John	35	456321	35000	Manager	
Rowkey	fname	lname	age			
EMP002	Alan	Smith	32			

Physical table - Personal

Rowkey	Column name	timestamp	Value
EMP001	Personal:age	1234567890123	35
EMP001	name	1234567890123	John
EMP001	pincode	1234567890999	456321
EMP001	pincode	1234567890123	123456

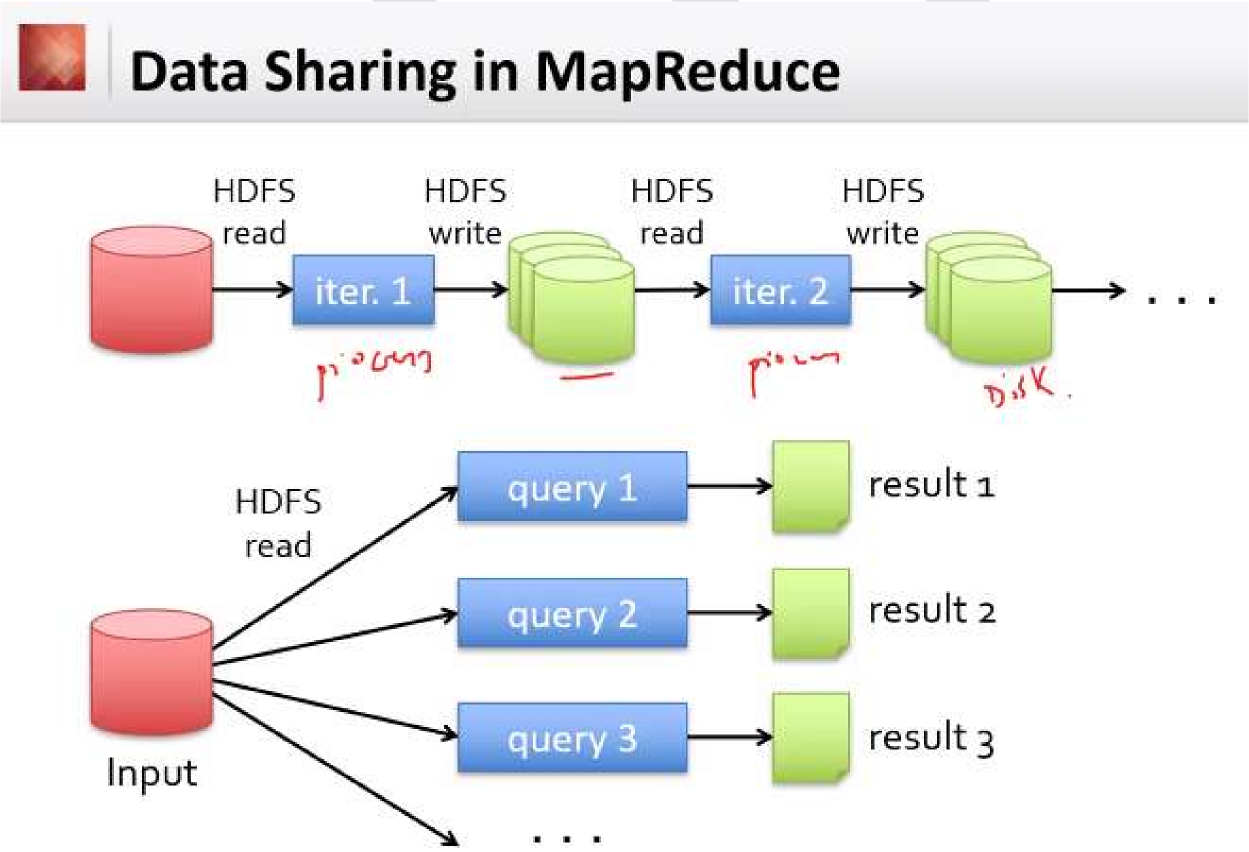
Physical table - Official

Rowkey	Column name	timestamp	Value
EMP001	Official:Desig	11111111	Manager
EMP001	Official:Salary	11111111	35000

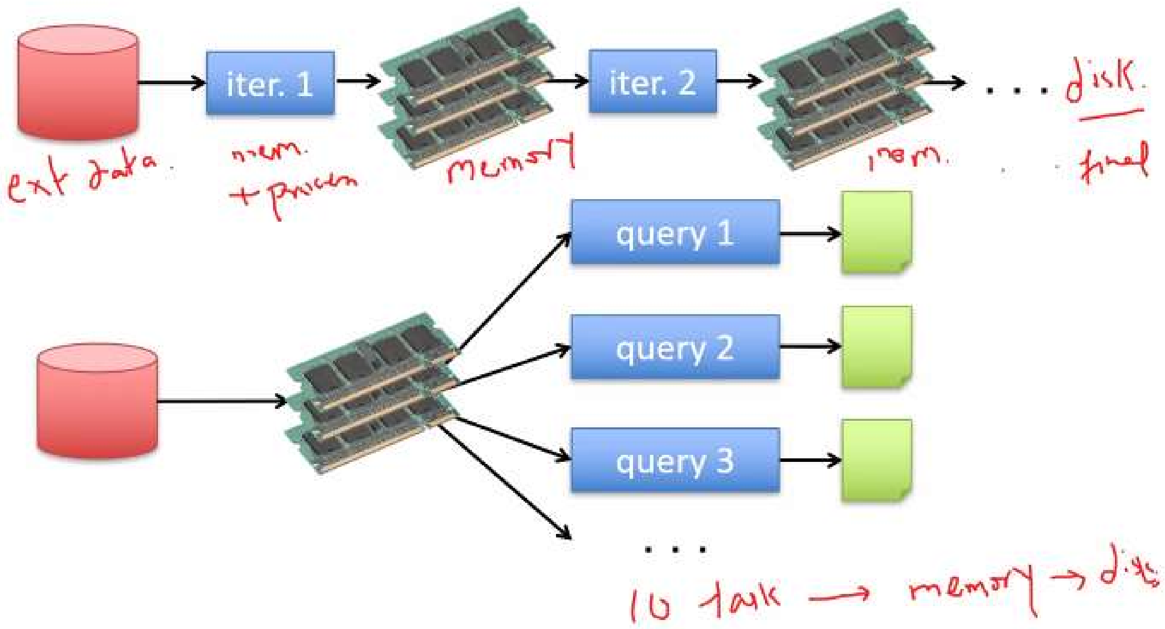
11. Creating table in **hbase create 'table2_432571', 'info','roles'** here table2_432571 is table name and info and roles are family name
12. To insert data **put 'table2_432571', 'cutting', 'info:height', '9 ft.'** here cutting is Colum name and info Is family name and height is Colum name and 9ft is value
13. To get or read data from table we have to give command **scan 'table2_432571'**
14. `get 'table2_432571','cutting'` it like where command in MySQL
15. if rowkey and Colum name is same then data will get sorted on descending order of timestamp
16. disable 'table2_432571'
    alter 'table2_432571' , {NAME=>'roles', VERSIONS=>3}
    enable 'table2_432571'
    this command will select version if version is 2 then we can save old data as well

12) APACHE SPARK:

1. Spark is also distributed computing framework
2. Independent from Hadoop but we can incorporate Hadoop and spark
3. It does not have file system data writing and reading is done through external file system e.g. rdbms , hdfs , local file, cloud data
4. A low latency cluster computing system


Data Sharing in MapReduce

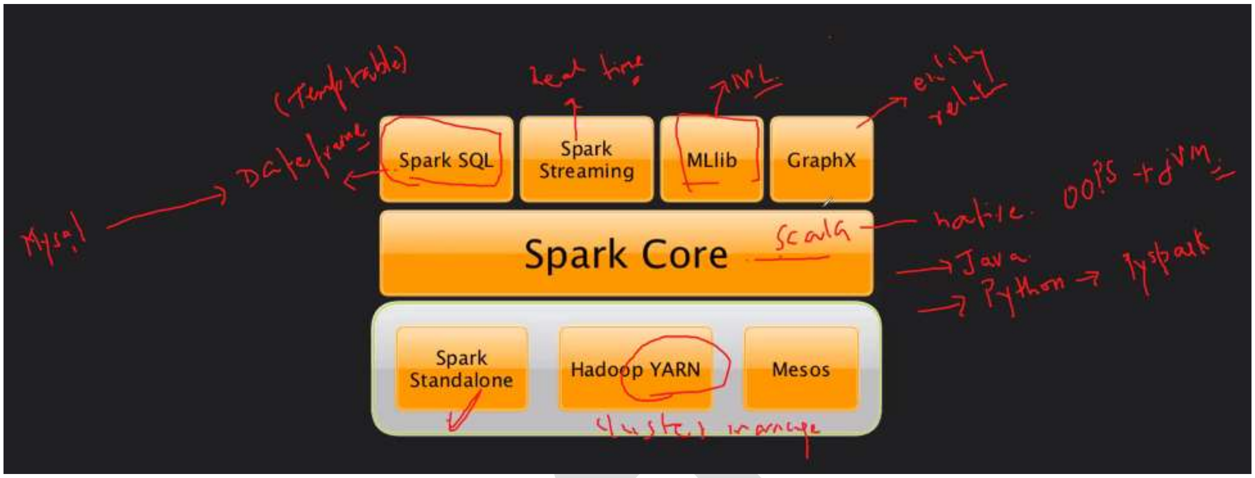
Data Sharing in Spark



Spark is …

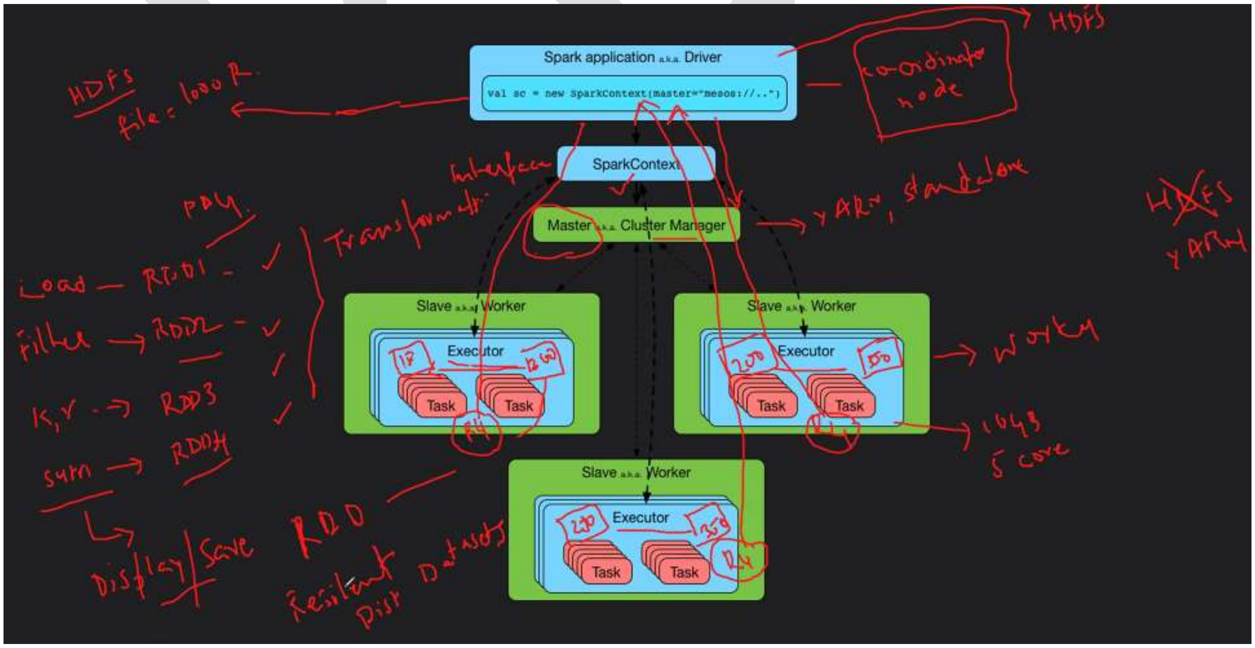
- Not a modified version of Hadoop.
- A low latency cluster computing system.
- Separate, fast, MapReduce-like engine
  - In-memory data storage for very fast iterative queries
  - General execution graphs and powerful optimizations
  - 40x to 100x faster than Hadoop*
  - 100 x faster than MapReduce for Iterative algorithms.
- Compatible with Hadoop's storage APIs
  - Can read/write to any Hadoop-supported system, including HDFS, HBase, SequenceFiles, etc

13) APACHE SPARK ECOSYSTEM:



A. PROCESSING OF DATA IN SPARK

1. **RDD:**
   - **a.** What are the data stored in memory stored in RDD (resilient distributed data) It is an object of data who consistent partition data
   - **b.** Once load command execute all RDD get erased
   - **c.** Loading of executor is done by cluster manager
   - **d.** Executor is basically a RAM
   - **e.** Until and unless the you run action command RRD will not get created
   - **f.** Creation RDD 1. RDD 2 called as Transform
   - **g.** Resilient means have capacity to bounce back



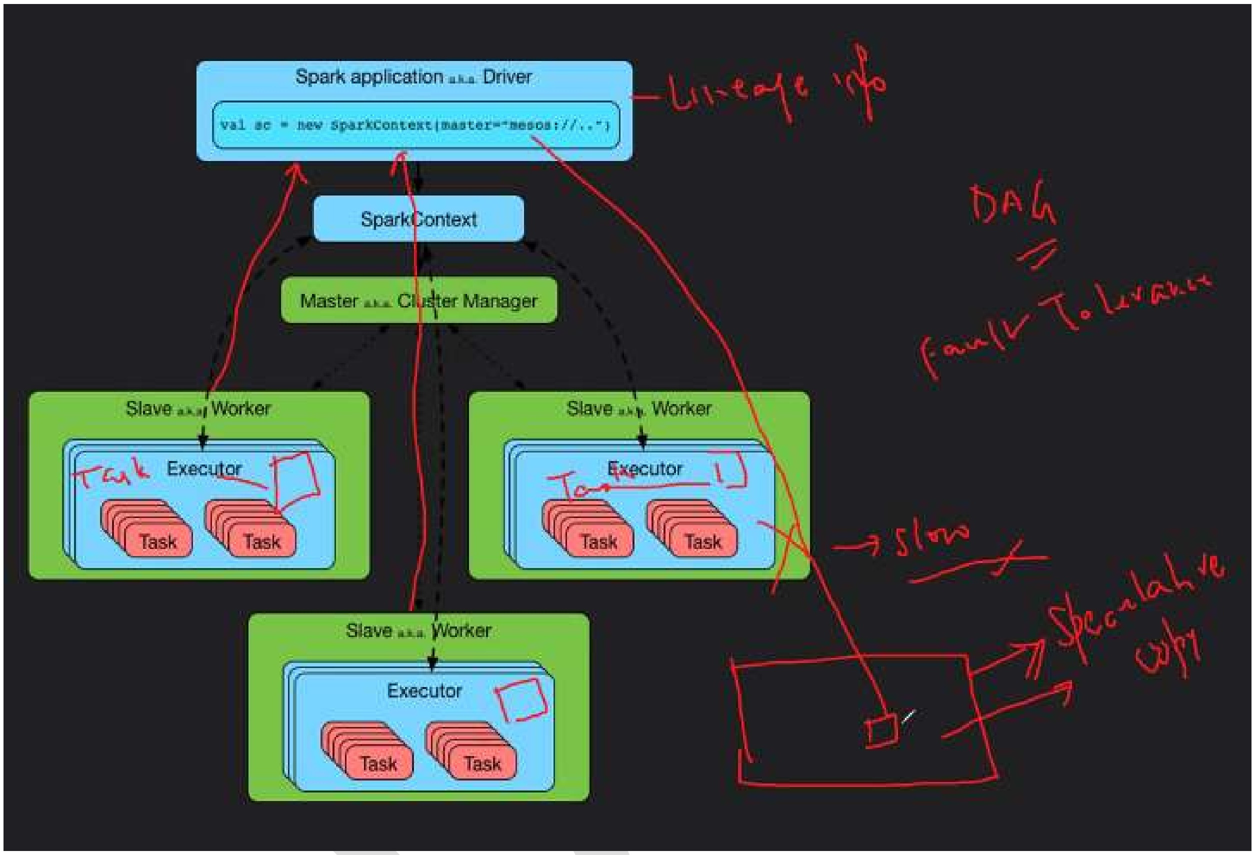
```
val sc = new SparkContext(master="mesos://..")
```

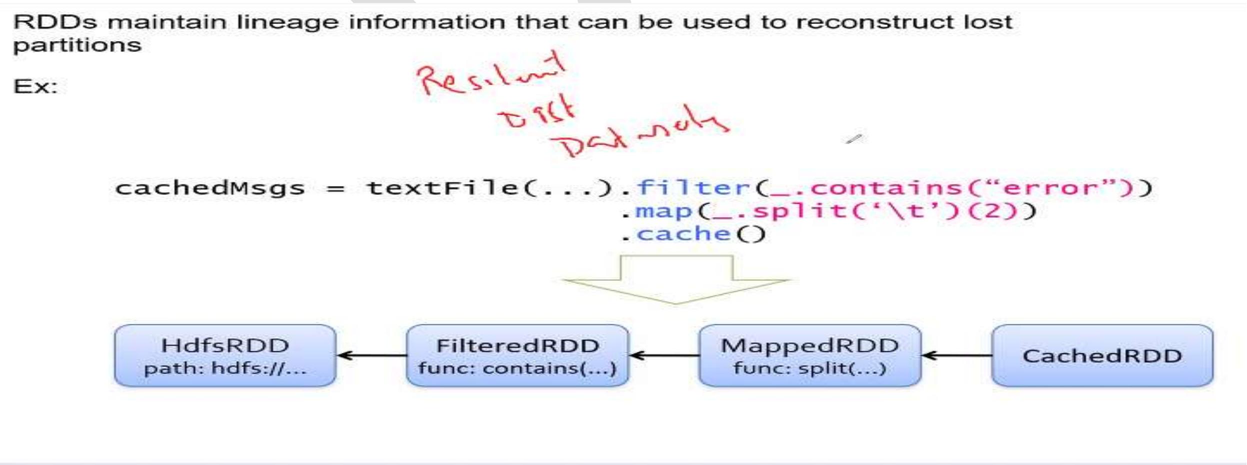
- **h.** **Transformation:** It is a function that produces new RDD from the existing RDDs.

i. **Action:** In Transformation, RDDs are created from each other. But when we want to work with the actual dataset, then, at that point we use Action.

B. HOW FAULT TULARANC HANDLE IN SPARK

1. We store data linear Information to driver
2. Once worker node get failed same information will create another worker node those this method is called as speculative copy
3. Lineage information is like DAG data plan



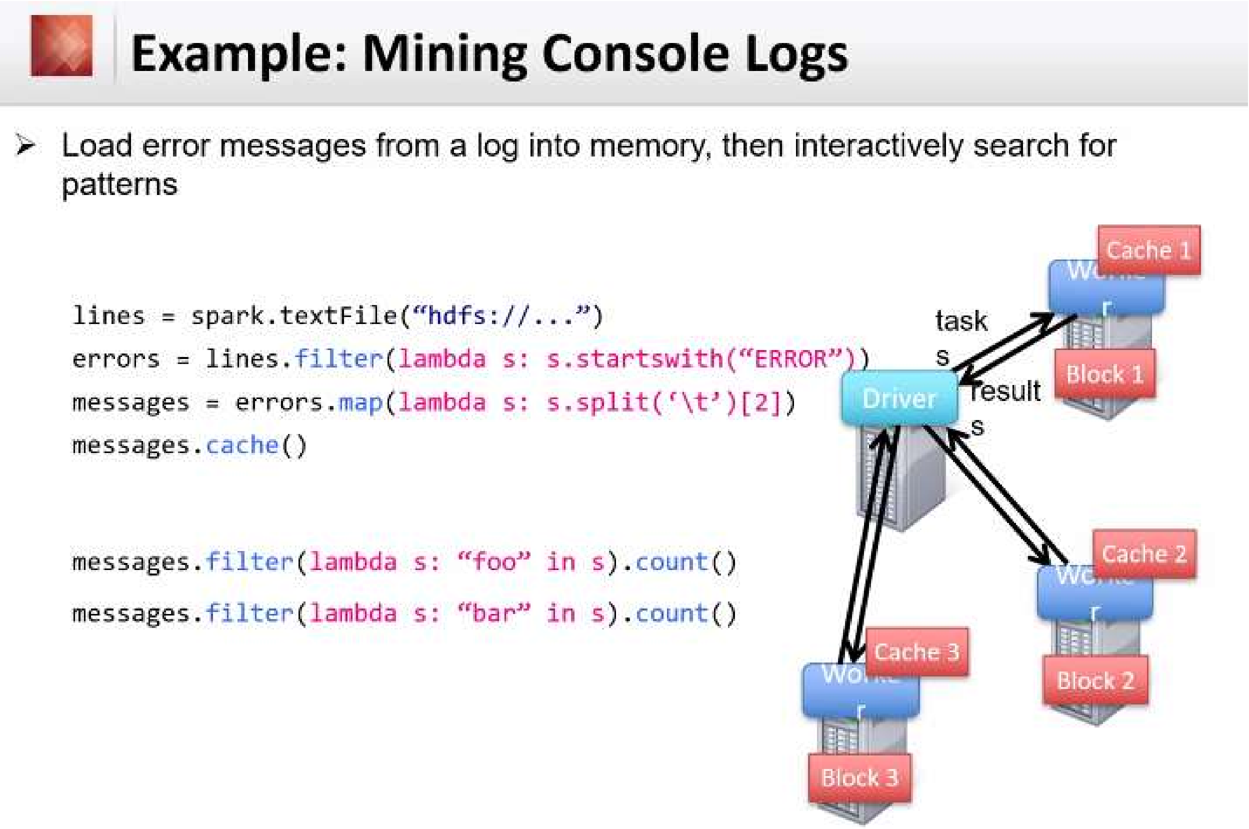


RDDs maintain lineage information that can be used to reconstruct lost partitions

Ex:

```
cachedMsgs = textFile(...).filter(_.contains("error"))
                          .map(_.split('\t')(2))
                          .cache()
```

C. EXAMPLE



Example: Mining Console Logs

- Load error messages from a log into memory, then interactively search for patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(‘\t’)[2])
messages.cache()


messages.filter(lambda s: “foo” in s).count()
messages.filter(lambda s: “bar” in s).count()
```

D. WORDCOUNT

```
dataRDD1 = sc.textFile("hdfs://nameservice1/user/bigcdac432571/training/sample.txt",1)
>>> dataRDD1.count() //it will give count of line

=================================================================
>>>dataRDD1.getNumPartitions()

mapRDD = dataRDD1.flatMap(lambda a : a.encode("ascii", "ignore").split(' '))
>>> for word in mapRDD.collect():
...     print(word)
...
=================================================================
mapRDD = dataRDD1.flatMap(lambda a : a.split(' '))



>>> keybyword2 = mapRDD.map(lambda word : (word,1))
>>> for line in keybyword2.collect():
...     print(line)

>>> counts = keybyword2.reduceByKey(lambda a,b : a+b).sortByKey() //it is a rdd
>>> for line in counts.collect():
...     print(line)

sortbyval = counts.sortBy(lambda a : -a[1]).collect()  //this is collected in driver node
sortbyval it is collecttion item it is not a rdd
>>> for line in sortbyval:
...     print(line)  //IT does not run on rdd and worker node

>>>
counts.saveAsTextFile("hdfs://nameservice1/user/bigcdac432571/training/pyspark1")
//this will only run with rdd
>>>
sc.parallelize(sortbyval,1).saveAsTextFile("hdfs://nameservice1/user/bigcdac432571/training/pyspark2")  //COLLECTION ITEM TO RDD WE USE PARALLELIZE
```

===

ACTION COMMANDS RETURNS DATA TO DRIVER NODE AND IT WORKS ON WORKER NODE
COLLECTION ITEM TO RDD WE USE PARALLELIZE

E. FINDING ERROR COUNT IN LOG FILE

```
lines = sc.textFile("hdfs://nameservice1/user/bigcdac432571/training/mapred-hduser-historyserver-ubuntu.log")

lines_lower = lines.map(lambda s : s.lower())

errors = lines_lower.filter(lambda s : "error" in s)

pattern = errors.filter(lambda s : "received signal 15" in s)

pattern.count()

for lines in pattern.collect():
```

```
...         print(lines)
...
```

F. TRASANCTION FIND TOP 10 SOLD ITEM

```
>>> txnRDD =
sc.textFile("hdfs://nameservice1/user/bigcdac43251/training/txns1.txt",1)
>>> txnKVRDD = txnRDD.map(lambda row : (row.split(',')[5], float(row.split(',')[3])))
>>> for line in txnKVRDD.take(5):  //it is action command and it shows only 5 element or
row
...     print(line)
...
('Cardio Machine Accessories', 40.33)
('Weightlifting Gloves', 198.44)
('Weightlifting Machine Accessories', 5.58)
('Gymnastics Rings', 198.19)
('Field Hockey', 98.81)
>>> spendbyProd = txnKVRDD.reduceByKey(lambda a,b : a+b)

>>> for line in spendbyProd.take(10):
...     print(line)
...
('Cycling', 42243.91000000003)
('Hunting', 39081.74)
('Foosball', 43055.95999999997)
('Boxing', 44516.869999999966)
('Springboards', 37890.76000000001)
('Scuba Diving & Snorkeling', 40239.799999999996)
('Dice Games', 40396.75)
('Boating', 43049.069999999934)
('Vaulting Horses', 41052.8)
('Portable Electronic Games', 41931.249999999985)
>>> spendbyProd.count()
125
>>> sortbyval = spendbyProd.sortBy(lambda a : -a[1])
>>> for line in sortbyval.take(10):
...     print(line)
...
('Yoga & Pilates', 47804.93999999993)
('Swing Sets', 47204.13999999999)
('Lawn Games', 46828.44)
('Golf', 46577.67999999999)
```

```
('Cardio Machine Accessories', 46485.540000000045)
('Exercise Balls', 45143.84)
('Weightlifting Belts', 45111.67999999996)
('Mahjong', 44995.19999999999)
('Basketball', 44954.68000000004)
('Beach Volleyball', 44890.67000000005)
```

G. UBER DATA PROCESSING

```
import datetime

dataset = sc.textFile("hdfs://nameservice1/user/bigcdac432571/training/uber_data")

header = dataset.first() //it is created at driver here fisrt is action command

print(header)

eliminate = dataset.filter(lambda line : line != header) //heare we are removing header

format_data = "%m/%d/%Y"  //here dromat data is variable here date is stored as mm-dd-yy

//eliminate2 = eliminate.map(lambda row : row.encode("ascii","ignore"))

split = eliminate.map(lambda a : (a.split(",")[0], datetime.datetime.strptime(a.split(",")[1], format_data).strftime("%A"), a.split(",")[3]) ) //this commands take data as string and convertes it into date

combine = split.map(lambda x : ( x[0] +" "+x[1], int(x[2]) )) // here our data is not int so we are converting it into int

arrange = combine.reduceByKey(lambda a,b : a+b)

sortbyval = arrange.sortBy(lambda a : -a[1])
```

```
===========================================================================
```

COMMAND EXPLAINATION

```
datetime.datetime.strptime(a.split(",")[1], format_data).strftime("%A"), a.split(",")[3])//== this commands take data as string and convertes it into date
```

```
>>> print(datetime.datetime.strptime("12/01/2002",format_data)
)

2002-12-01 00:00:00
```

14) RUNNING SQL QUERY IN SPARK (spark sql):

1. Instead of storing data in RDD we store data in data frame
2. Data frame is structure object created by spark
3. We can directly upload data from MySQL to data frame
4. Data frame is schema oriented data storing
5. Collect data from database, put it into spark, process it, and write data to again data bases.
6. By default the delimiter is ,

EXAMPLES:

A. NYSE DATA:

```
from pyspark.sql.types import StructType, StringType, IntegerType, DoubleType, LongType

schema = StructType().add("exchange_name",StringType(),True).add("stock_id",StringType(),True).add("stock_dt",StringType(),True).add("open",DoubleType(),True).add("high",DoubleType(),True).add("low",DoubleType(),True).add("close",DoubleType(),True).add("volume",LongType(),True).add("adj_close",DoubleType(),True)  // ON THIS COMMAND WE ARE DECLARING SCHEMA OF DATAFRAME

df_with_schema = spark.read.format("csv").option("header","False").schema(schema).load("hdfs://nameservice1/user/bigcdac432571/training/NYSE.csv") //HERE WE ARE READING DATA FROM ACTUAL FILE AND INSERTING DATA TO DATAFRAME HERE WE ARE USING SPARK.READ WHICH IS USED FOR READING SQL DATA

df_with_schema.printSchema() //THIS WILL PRINT THE SCHEMA IT IS LIKE DESCRIBE TABLE COMMAND IN SQL

df_with_schema.show()//IT WILL PRINT THE CONTENT OF THE TABLE IT IS LIKE SELECT * FROM TABLE, SHOW IS ACTION COMMAND, IT WILL SHOW BY DEFAULT ONLY 20 ROWS

df_with_schema.registerTempTable("nyse") // HERE WE ARE CREATING A TBALE WITH NAME NYSE

df_StockVol = spark.sql("select stock_id, sum(volume) from nyse group by stock_id")// WE ARE QUERING AND STORING VALUES TO RDD this is transforming command

df_StockVol.rdd.getNumPartitions()// by default 200 partition get created but we can change it it is a not true action command

df_new = df_StockVol.repartition(1)
```

```
df_new.write.csv("hdfs://nameservice1/user/bigcdac43251/training/spark1")
```

B. AIRLINES DATA

```
schema2 =
StructType().add("Year",StringType(),True).add("Quarter",StringType(),True).add("ARPS",DoubleType(),True).add("Booked_seats",IntegerType(),True)


df_with_schema2 = spark.read.format("csv").option("header",
"True").schema(schema2).load("hdfs://nameservice1/user/bigcdac432571/training/airlines.csv")

df_with_schema2.printSchema()

df_with_schema2.show()
df_with_schema2.count()
df_with_schema2.registerTempTable("airlines")

YrWiseRev=spark.sql("select year,round(sum(arps * booked_seats)/1000000,2) as
total_in_mil from airlines group by year order by total_in_mil desc")

YrWisePsx=spark.sql("select year,sum(booked_seats) as total_psx from airlines group by
year order by total_psx desc")
```

C. RETAIL PROJECT

```
from pyspark.sql.types import StructType, StringType, IntegerType,
DoubleType, LongType

schema10 =
StructType().add("txn_id",StringType(),True).add("cust_id",StringType(),Tr
ue).add("age",StringType(),True).add("zipcode",StringType(),True).add("cat
egory",StringType(),True).add("product",StringType(),True).add("qty",Integ
erType(),True).add("cost",IntegerType(),True).add("sale",IntegerType(),Tru
e)

retail
spark.read.format("csv").option("sep",";").option("header","False").schema
(schema10).load("hdfs://nameservice1/user/bigcdac432565/retail")
retail.registerTempTable("retail")
```

Count of unique customers and total sales for each age group and for a given month = Jan

```
>>> q1=spark.sql("select age,count(distinct(cust_id)) as Count,sum(sale)
Total from retail where month(txn_id)=01 group by age order by Total
desc")
```

Count of unique customers and total sales for one age group(A) for all products - [sort data on totalsales desc- find top 10]

```
>>> q2=spark.sql("select product,count(distinct(cust_id)) Count,sum(sale)
as total from retail where trim(age)='A' group by product order by total
desc limit 10")
```

3) Area wise sales

```
>>> q3=spark.sql("select zipcode Area,sum(sale) as total from retail group
by zipcode order by total desc")
```

4) find top 10 viable products (prod which give highest profit)

```
>>> q4=spark.sql("select product,sum(sale-cost) as Profit from retail
group by product order by Profit desc limit 10")
```

5) find all loss making products – Display all the loss making products from highest loss to the least

```
>>> q5=spark.sql("select product,sum(cost-sale) as Loss from retail group
by product order by Loss desc limit 10")
```

The repartition() can be used to increase or decrease the number of partitions, but it involves heavy data shuffling across the cluster. On the other hand, coalesce() can be used only to decrease the number of partitions. In most of the cases, coalesce() does not trigger a shuffle

15) CONNECTING SPARK WITH MYQL:

1. **mysql -u bigdatamind4385 -phadooplab234**
   (Do it in new shell this will open mysql)
2. **pyspark --jars mysql-connector-java-5.1.47-bin.jar**
   (Do it when loading pyspark from shell)
3. **studentDF = spark.read.format("jdbc").option("url","jdbc:mysql://ip-10-1-1-204.ap-south-1.compute.internal:3306/bigdatamind4385").option("driver","com.mysql.jdbc.Driver").option("dbtable","student_master").option("user","bigdatamind4385").option("password","hadooplab234").load()**
   (This will create data frame of table present in mysql)
4. **fyDF = spark.read.format("jdbc").option("url","jdbc:mysql://ip-10-1-1-204.ap-south-1.compute.internal:3306/bigdatamind4385").option("driver","com.mysql.jdbc.Driver").option("dbtable","fy").option("user","bigdatamind4385").option("password","hadooplab234").load()**
   (Loading another table to another data frame in spark from MySQL)
5. **studentDF.registerTempTable("student")**
   (Creating temp table with name student)

6. **fyDF.registerTempTable("fy")**
   (Creating temp table with name fy)
7. **df1 = spark.sql("select a.student_id, name, address, coalesce(result,0) from student a left outer join fy b on a.student_id=b.student_id order by student_id")**
   (This will join two tables)
8.
```
df1.write.format("jdbc").option("url","jdbc:mysql://ip-10-1-1-204.ap-south-1.compute.internal:3306/bigdatamind4385").option("driver","com.mysql.jdbc.Driver").option("dbtable","result").option("user","bigdatamind4385").option("password","hadooplab234").save()
```
   (To write the result to mysql)

16) DATA INJECTION GETTING DATA FROM RDBMS AND WRITING TO HIVE USING SPARK:

1. **HOW TO TRANSFER DATA FROM ORACLE OR ANY OTHER DB TO HIVE? (DATA INJECTION)**
   WHAT WE CAN DO WE HAVE TO GET DATA FROM DB TO SPARK AND HAVE TO WRITE ON HIVE THIS IS CALLED AS DATA INJECTION

```
pyspark --jars mysql-connector-java-5.1.47-bin.jar
```

```
studentDF = spark.read.format("jdbc").option("url","jdbc:mysql://ip-10-1-1-204.ap-south-1.compute.internal:3306/bigdatamind4385").option("driver","com.mysql.jdbc.Driver").option("dbtable","student_master").option("user","bigdatamind4385").option("password","hadooplab234").load()
```

```
studentDF.show()
```

```
+----------+-------+---------+

|student_id|   name|  address|

+----------+-------+---------+

|         1| Sanjay|Bangalore|

|         2|  Rajiv|    Delhi|

|         3|  Suraj|  Chennai|

|         4|Sandeep|    Delhi|

+----------+-------+---------+
```

-No need to register tempt able since we are not performing any query on data we are just reading/writing data

2. Writing data from spark--->hive (by default writes in parquet file format)

studentDF.write.mode ("overwrite").saveAsTable ("training_432571.newstudent") //It table is not there then it creates new table and if table is there then it overwrites the table

Open new shell 2: launch hive to verify

hive

use training_432571;

hive> select * from newstudent limit 10;

OK

When we write data to hive from spark the data get stored as parquet format

17) READING DATA FROM HIVE AND PROCESS IT USING SPARK AND WRITING BACK TO HIVE:

```
customerDF = spark.sql("select * from training_432571.customer")
//importing customer table from hive to spark and storing them to DF

txnDF = spark.sql("select * from training_432571.txnrecords"
//importing transaction record table table from hive to spark and storing
them to DF

customerDF.registerTempTable("customer")
//Registering data frame as temp table

txnDF.registerTempTable("txn")
//Registering data frame as temp table

toptenDF = spark.sql("select t.custno, firstname, lastname, age,
profession, round(sum(amount),2) as total from txn t left outer join
customer c on t.custno = c.custno group by t.custno, firstname, lastname,
age, profession order by total desc limit 10")
//Joining two table

toptenDF.write.mode("overwrite").saveAsTable("training_432571.topten")
//Storing data return to hive
```

write a program using shared variables like broadcast and accumulators to calculate total tax and total profit
using Retail data

```
retailRDD = sc.textFile("hdfs://nameservice1/user/bigcdac432571/retail")
retailRDD.count()
retailRDD.getNumPartitions()
gst = sc.broadcast(5.00)
totalTax = sc.accumulator(0.00)
totalProfit = sc.accumulator(0.00)
arrayRDD = retailRDD.map(lambda a : a.split(";"))
```

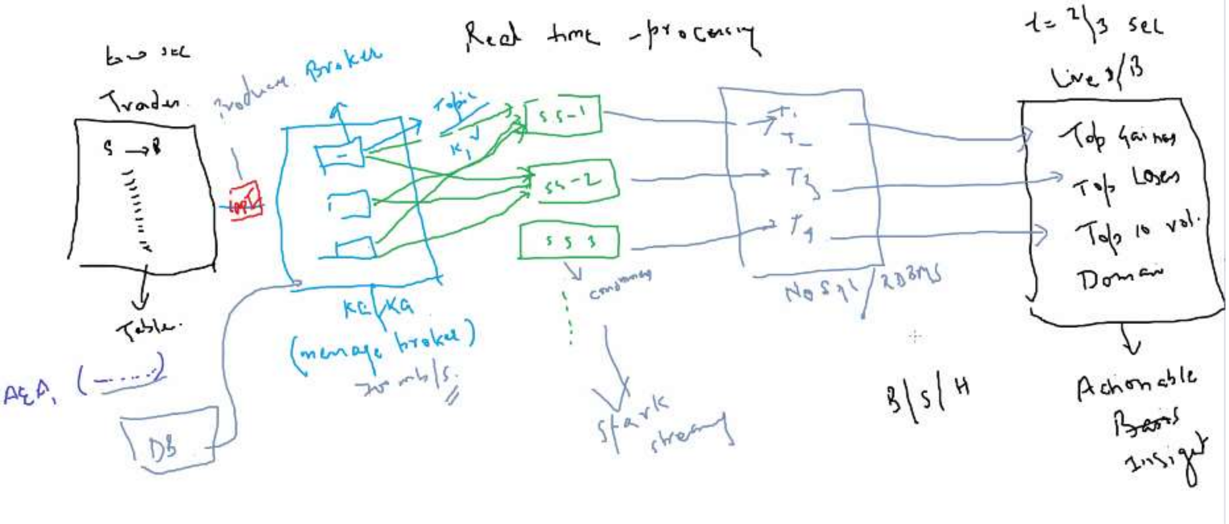
```
taxAndProfit = arrayRDD.map(lambda  a : ( float(a[8])*gst.value/100 ,
(float(a[8]) - float(a[7]))))
for line in taxAndProfit.collect():
...     totalTax += line[0]
print(totalTax)
for line in taxAndProfit.collect():
...     totalProfit += line[1]


print(totalProfit)
```

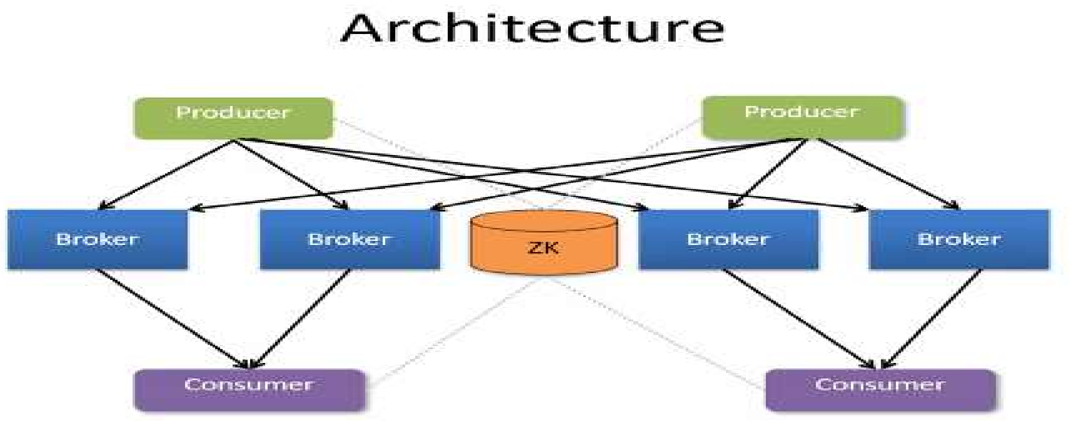
18) REAL TIME PROCESSING

Real time processing means processing data which continuously coming in real time

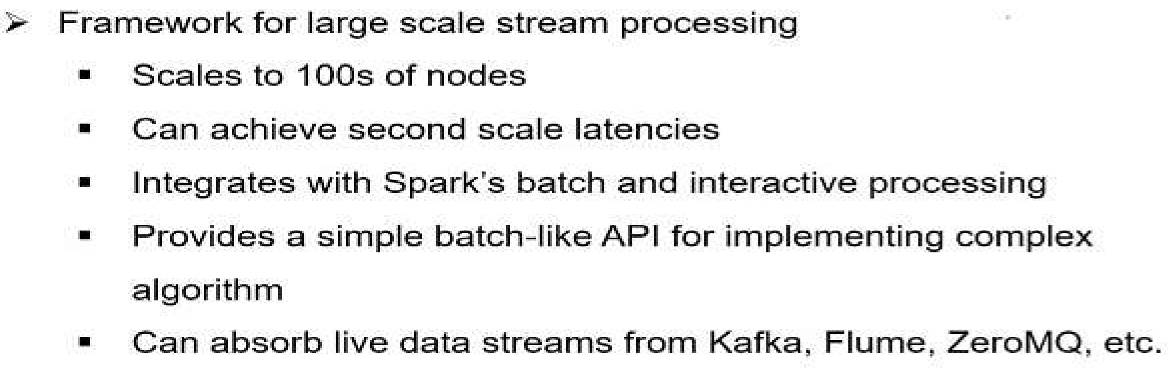
A. KAFKA

1. Kafka is cluster system and it is used for storing data
2. Partitioning done by using partitioning and it is called as broker
3. Data stored in Kafka as topics and it is key value paired
4. The data is process by spark and stored in rdbms as it has continuously updating system



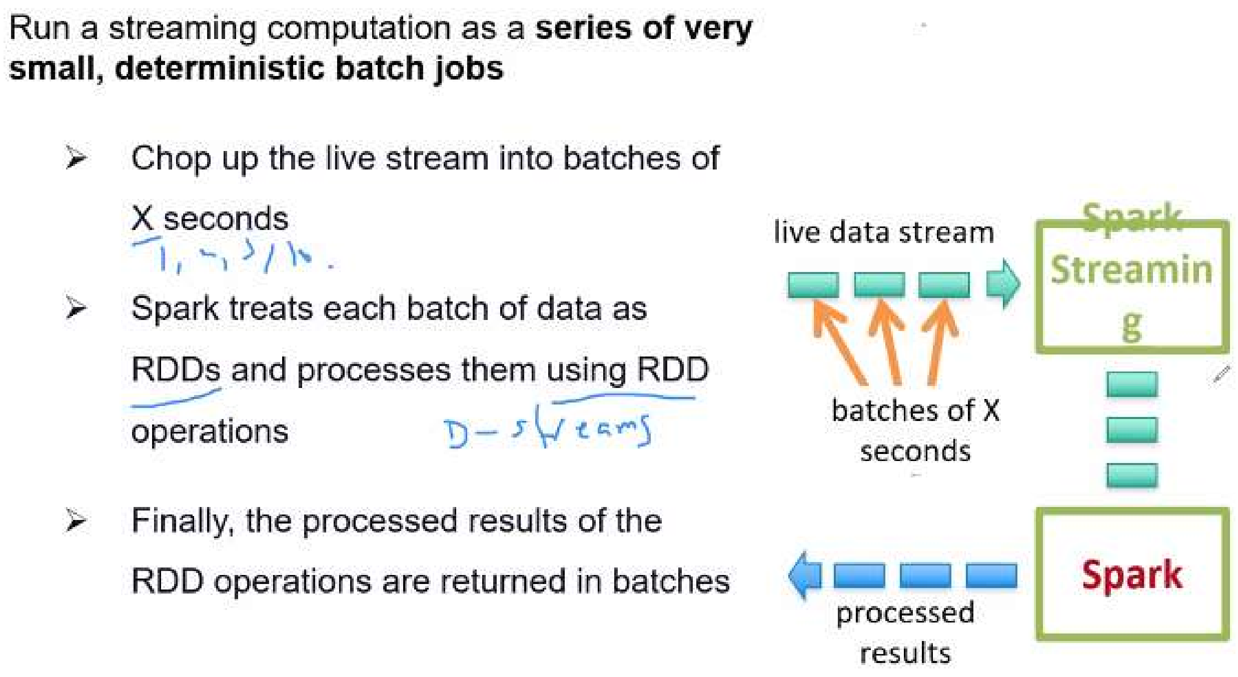


B. SPARK STREAMING



- Framework for large scale stream processing
  - Scales to 100s of nodes
  - Can achieve second scale latencies
  - Integrates with Spark's batch and interactive processing
  - Provides a simple batch-like API for implementing complex algorithm
  - Can absorb live data streams from Kafka, Flume, ZeroMQ, etc.

1. In spark streaming interval time is defined by which each interval time data is converted into batch and this batched are then processes.
2. There no data then also batch is created it might be empty set or have less data
3. This batches are called as d-streamed (RDD)
4. This RDD are called as d-streams
5. The most important factor in spark streaming is interval time
6. Interval time defines latency of spark



Run a streaming computation as a **series of very small, deterministic batch jobs**

- Chop up the live stream into batches of X seconds
  1, 2, 5/10.
- Spark treats each batch of data as RDDs and processes them using RDD operations D-streams
- Finally, the processed results of the RDD operations are returned in batches

WORDCOUNT EXAMPLE OF SPARK STREAMING

On a sep terminal

```
$ nc -lk 43257 (here we ae using our local terminal as real time data genrator)
```

On other shell

```
from pyspark import SparkContext

from pyspark.streaming import StreamingContext

sc.stop()

sc = SparkContext("local[2]", "NetworkWordCount") //here local [2] menas we are limiting thread two

ssc = StreamingContext(sc, 10)

lines = ssc.socketTextStream("localhost", 43257)

words = lines.flatMap(lambda line: line.split(" "))

pairs = words.map(lambda word: (word, 1))

wordCounts = pairs.reduceByKey(lambda x, y: x + y)

wordCounts.pprint()
```

```
ssc.start()             # Start the computation
ssc.awaitTermination()  # Wait for the computation to terminate
```

CONSUMER AND PRODUCER CREATION IN KAFKA

create a topic

```
/opt/cloudera/parcels/CDH/bin/kafka-topics --create --zookeeper 10.1.1.204:2181 --replication-factor 1 --partitions 1 --topic cdac_432571
```

list the topics

```
/opt/cloudera/parcels/CDH/bin/kafka-topics --list --zookeeper 10.1.1.204:2181
```

producer-api-console

```
/opt/cloudera/parcels/CDH/bin/kafka-console-producer --broker-list 10.1.1.204:9092 --topic cdac_432571
```

Multiple producer

```
/opt/cloudera/parcels/CDH/bin/kafka-console-producer --broker-list 10.1.1.204:9092 --topic cdac_432571
```

consumer-api-console

```
/opt/cloudera/parcels/CDH/bin/kafka-console-consumer --bootstrap-server 10.1.1.204:9092  --topic cdac_432571 --from-beginning
```

Multiple consumer

```
/opt/cloudera/parcels/CDH/bin/kafka-console-consumer --bootstrap-server 10.1.1.204:9092  --topic cdac_432571 --from-beginning
```

import from a text file into kafka

```
/opt/cloudera/parcels/CDH/bin/kafka-topics --create --zookeeper 10.1.1.204:2181 --replication-factor 1 --partitions 1 --topic cdac4
```

```
/opt/cloudera/parcels/CDH/bin/kafka-topics --list --zookeeper 10.1.1.204:2181
```

```
/opt/cloudera/parcels/CDH/bin/kafka-console-producer --broker-list 10.1.1.204:9092 --topic cdac4 < /home/bigcdac432571/custs.txt
```

```
/opt/cloudera/parcels/CDH/bin/kafka-console-consumer --bootstrap-server 10.1.1.204:9092  --topic cdac4 --from-beginning
```