BIG DATA

HADOOP

BY

Mr SREERAM

Data Scientist

Ventech IT Solutions

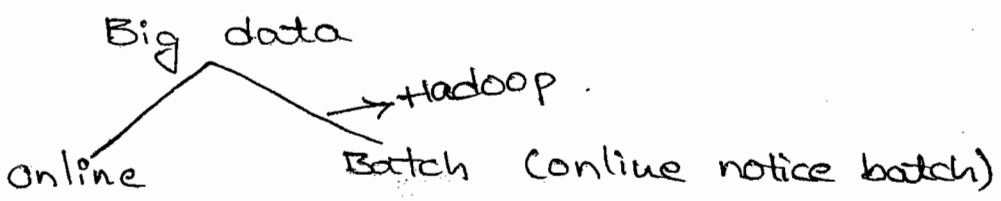
307/B Neelgiri block, Adithya enclave,
Ameerpet, Hyderabad-38

PH: 040-65356678, 9030056678

OM VINAYAKA

HADOOP

* To sort 1TB Tables
  - Oracle – 3½ days
  - Teradata – 4½ hrs
  - hadoop – 3-4 min
* Hadoop new version Apache announced it will take 1.2 mins.
* Oracle Hexadata :- its not successful
* Oracle BIGMISSIONS.
* HD msdos (microSoft) (higher definitions)
  1. Unlimited high speed
  2. very high speed
  3. All varities of data
     - * structured
     - * unstructure
     - * semistructured
* Big Data Dimensions (Big data frame work):-
  - * volume – unlimited ✓
  - * velocity – Speed is high ✓
  - * variety ✓
* Xadata / Hana :- partially Big data frame work.

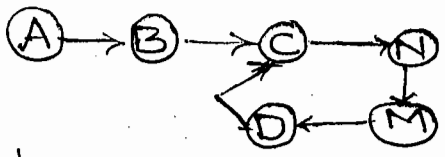


* User non-interactive Application is called
* Application take info to the user

* NO-SQL :- Data processing (not only SQL)
(mongo DB)

sign in → track → Risk (frequent updates)
(HBase)

* NEO4J :- To process Grapher data



* Map Reduce :- java, python, C++, Ruby these are developed or implemented.

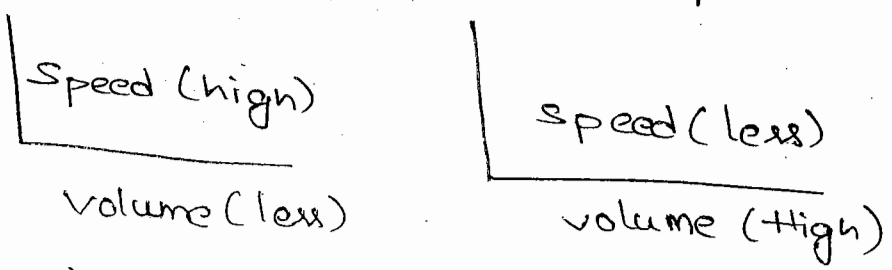
* GMR :- Google map Rating system
(implemented by python)

3/2/15:

Hadoop :- Hadoop is a bigdata frame work used for Bigdata & Analytics. performing analytic over Bigdata.

Bigdata :-

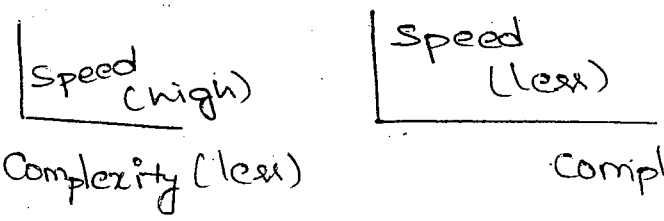
* Big is a Big volume + complex data.
* Complex data the large collection of huge volume of data sets with complex data structures.
* problem of RDBMS with Bigdata :-
* one has volume increase, speed is decreasing



* Queries :-
→ Select sum (amt) from Sales (100 rows)
→ Select sum (amt) from Sales 2)

In above queries first one is very fast this is called

* ② has complexity of Algorithm increase Speed decreases.



* Query:-

```
Select Sum (amt) from sales;
Select Std dev (amt) from sales;
```

* ③ has RDBMS can Better process structured data only, little features of Semi Structured (xml, JSON ---)

* Hadoop advantages:-
  * Unlimited data storage
  * very high Speed
  * All variety of data process
    * Structured
    * Un Structured
    * Semi Structured

* Big data Dimensions:-
* Volume (Huge volume data storage)
* Velocity (High Speed) mins
* Variety (All variety of data process)

→ Bigdata frame work means my technology Satisfy the bigdata dimension. Then we will called as "Big data".

→ Hadoop is Satisfying Bigdata dimensions thats why we are calling Hadoop is a "Big data"

* Scaling Models (Server):-

Two types of scaling models.

(i) vertical scaling model

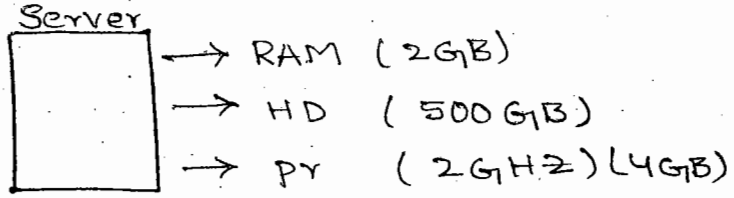
(ii) Horizontal scaling model

* Vertical Scaling Model :-

Server has

→ By Adding / Replacing components such as RAM/HD/Pr scaling op is done.

→ High level scaling is not possible because of its not Available it will act compactable

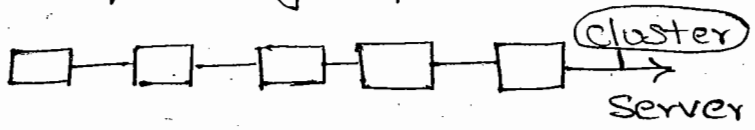


* Horizontal scaling model :-

It is a cluster Act as a server.

* Cluster:- A Group of Network (of cpu act as a server)

→ By adding Cpu's to cluster Scale up is done.

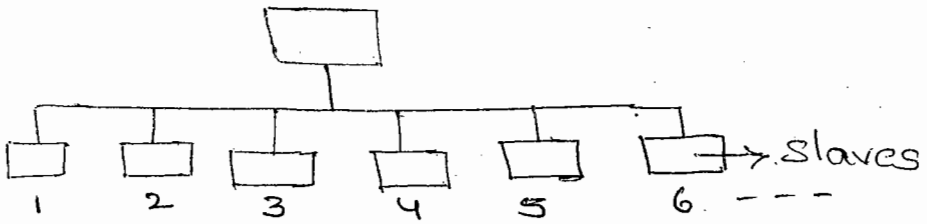


each cpu of cluster is called Node.

Cluster has two types of nodes.

1) master node

2) Slave node



* Master Node:- Its have lot of responsibilities Such as

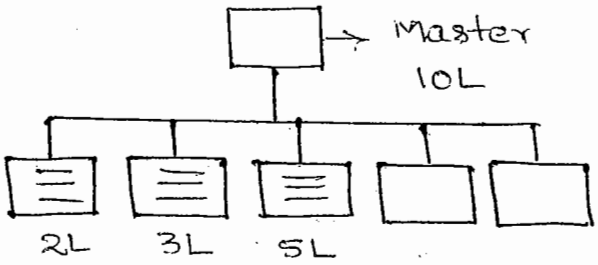
* work Assigning

* load balancing
* fault tolerance
* Health monitoring

Now a days each RDBMS/OS/App served is a cluster in its product place.

* Why unlimited storage not possible in RDBMS :-

Tera data (TD)



Sum (amt)

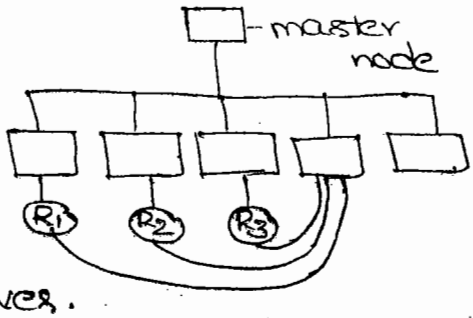
RDBMS	Max no. of slave
Oracle	256 → DWH
DB2	256
SQL Server	108 - 256
MySQL	54
MySQL Express	256 - OLTP
Teradata, Netezza, vertica	512

* Why hadoop unlimited data storage :-

In each RDBMS has it own limits for max no. of slaves thats why unlimited storage is not possible.

In hadoop the feature called Horizontally unlimited Scalability.

Horizontally adding slaves max no. of no limited for slaves.



Unlimited :- No limit for slaves unlimited storage table even though 1000 of slaves are there in hadoop, cluster still performance is not degraded, Because, independent another free slaves (not master).

Hadoop implemented normal capacity

=o=

6/7/15:

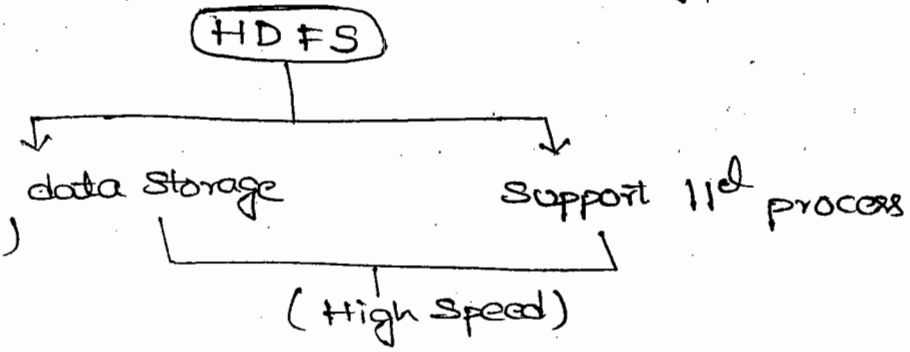
* Financial benefits :-

  * Open source
  * Hadoop → Storage
           → processing

SAN :- Storage Area Network (only store)

* Storage :- HDFS

Hadoop distributed file system



7/7/15:

* Why following RDBMS database don't support parallel process [Oracle / DB2 / SQL / Server / Sybays / my SQL / PG SQL] ?

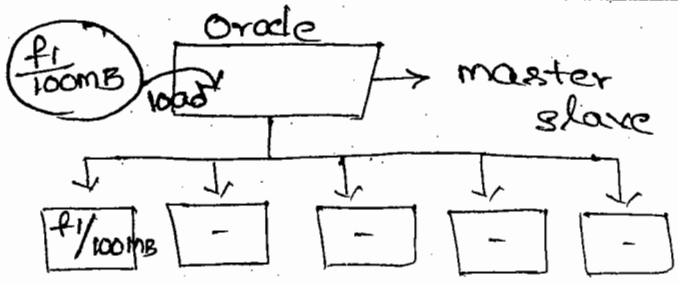
→ In RDBMS, Data is inserted in tables. But A Table is a Logical structure. Data is not saved in tables. Tables backend storage is "Table Space".

As per OS, each table space is a file, File type of above database is non-DFS. (Non-distributed file system).

* Behaviour of NON-DFS :-

A file data is completely kept in sigle slave.

oracle cluster have five slave notes. of each 1 TB storage.



when you created a table, the table placed at master. imeadiately one table space be assigned to the table. Table space is a file, the file to be stored in slaves, but this file to be dedicated for one sigle slave.

Ex:- Slave 1,

when you load 100mB data Sales table, The 100mB is stored in slave 1 immediately master will record tables data address.

EX:- Sales (table) → File 1 (table space) → Slave 1 (Slave)

when a job is (query) submitted, the request goes to master. master prepares a task for the query, assign to the file is available.

Ex:- Task is assigned to Slave 1 [which containes sales data]

Now the total process to be done by Slave 1. In this moment remaining slaves are free, still they can't share the load of first slave. because they don't have data of slaves table.

thats why IIel process is not possible in above data bases.

* Why Teradata, Netezza, Vertical, Storm, Spark, NO SQL, Hadoop support IInd process?

→ File Type = DFS.
Distributed file system

* Distributed file system behaviour :-

* File distributed into Block
* Blocks will be distributed across multiple slaves.

[The Block Size]

Tera data	32 KB
Oracle	4 KB
Netezza	64 KB
Hadoop	64 MB
(Advanced) (Hadoop)	128 MB

* In HDFS Block size = 64 MB

[from 2.6 (yarn) onwards Block size = 128 MB]

Ex:- OS file (local) = F1
Size = 100 MB

when this file moved into Hadoop.

As 64 MB block size, now the file requires 2 blocks.

F1 is divided as Block 1 and Block 2
(64 MB) (36 MB)

→ Block 1 is stored in slave 1.

→ Block 2 is stored in slave 2.

** when job is submitted, the job will be divided in-to tasks.

** Number of Task = No. of unique blocks.

** job is divided as Task1 and Task2

Task 1 $\xrightarrow{for}$ B_1 $\xrightarrow{in}$ S_1

Task 2 $\xrightarrow{for}$ B_2 $\xrightarrow{in}$ S_2

Now .

Task 1 is assigned to S_1

Task 2 is assigned to S_2

Both tasks will be parallely running by Slave1 and Slave2.

* Risk with parallel process :-

→ * During parallel process, if any active slave is down, All tasks of the job will be terminated. Otherwise, the results "will be in accurate".

* Solution expected :-

→ Even though some active slave is down the job execution should be continued with accurated results. This expectation is called "Fault tolerance (FT)".

* How Hadoop sopports Fault tolerance?

→ By Creating and distributed replicas into different slaves.

→ In HDFS, By default each Block is Replicated for 3 times.

→ [This can be configured]

decrease Increase

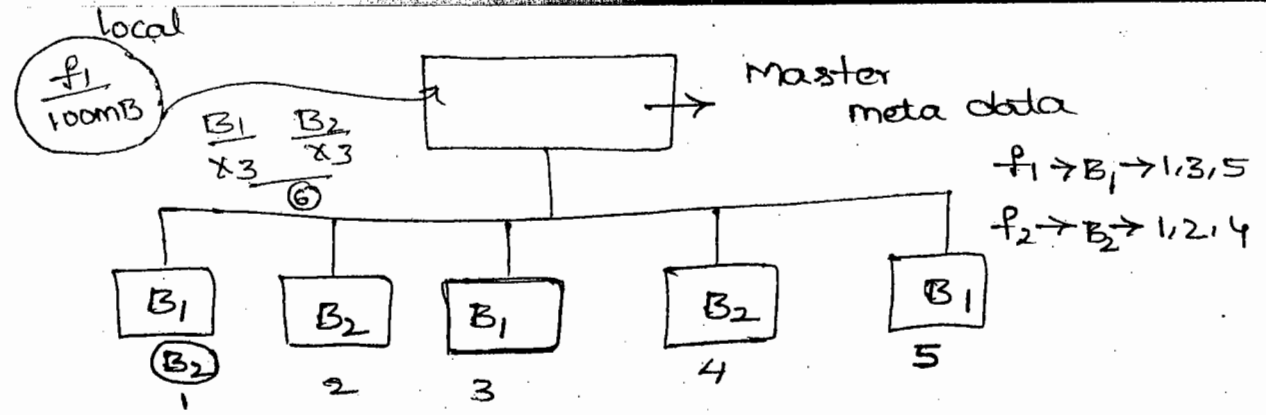
Ex:- F_1 → 100 MB (local) (means OS)

↳ Copied into HDP

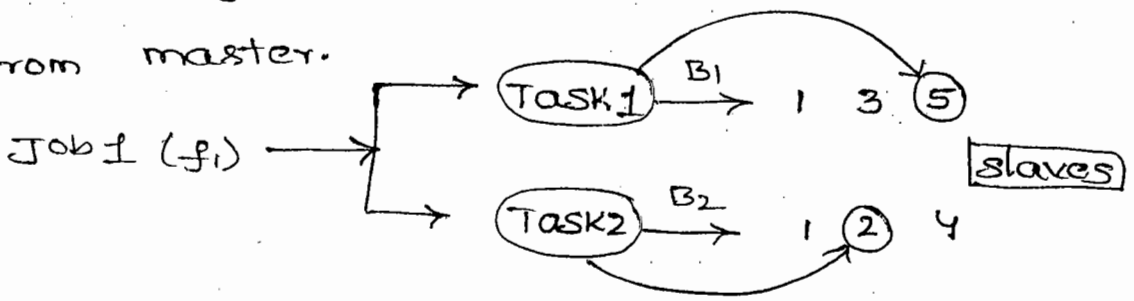
B_1 B_2

X_3 X_3

6 Blocks



when job is submitted job fetches metadata from master.



→ Task 1 to be assigned to any one of 1,3,5 slaves.

→ Task 2 to be assigned to any one of 1,2,4 slaves.

* Selection Cretaria:-

* Selection:-

priority Sequences:-

* High configuration (HW) (Hard ware)
* Nearest location
* Free Slave
* (All are not free) less busy
* (All are free) Health check
* All Health check busy Random selection.
* If all are high busy

(It tiggers non-localised process)

* Now Task1 by Slave 5

Task2 by slave 2

parallely processing, suddenly slave2 is down. Now the terminated task (Task2) will be reassigned to another slave, either 1 or 4, because still block2 available in these machine.

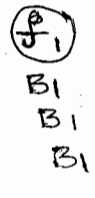
* Why Hadoop distributing file blocks?

→ To get parallel process.

* Why hadoop Maintains Replicas?

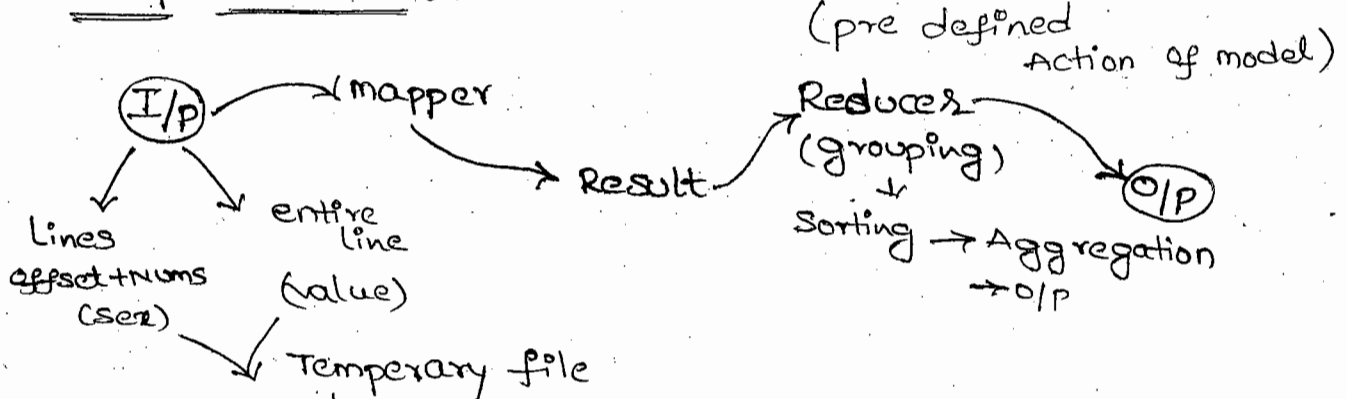
→ Fault tolerance

Hadoop → HDFS
→ MR (Map Reduce)



IMP:- The file Replicas can't be used as data backups. But can be used as process backups.

* Map Reduce:-



8/7/15:

* RDBMS Style of grouping Aggregations:-

Ex:- Select city, Sum (amt) from Sales group by city:-

Step ①:-

→ Sort on Raw Data by grouping column in ascending order. (painful)

*

Step ② :-

→ Identify No. of data groups.

Step ③ :-

→ Initiate Buffer

Step ④ :-

→ Buffer Accumulations

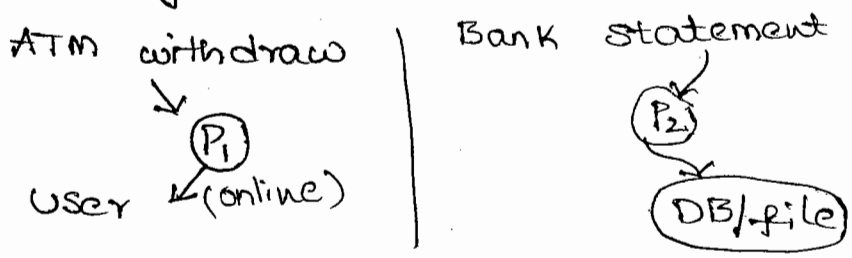
Step ⑤ :-

→ write Result

Map Reduce

* Map Reduce is a Batch job.
  → which is subdivided into two phases.
  * mapper task
  * Reducer task

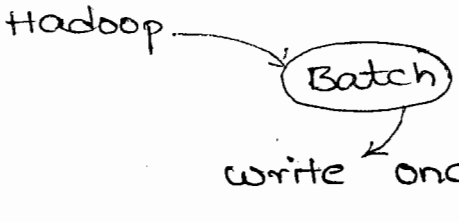
Ex:- → if you booked ticket and cancelled ticket, updation is batch.
→ Monthly bank statement



→ facebook

* Diff b/w ONLINE and BATCH?

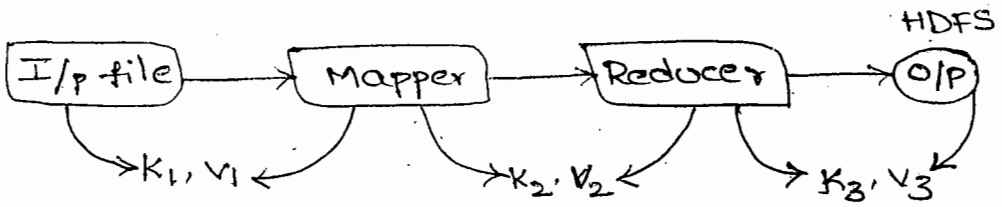
Online	Batch
→ User interactive applications are called online.	→ User non-interactive application are called Batch.
→ RUN time it takes input from user.	→ It takes i/p from external source such as files, DBS, etc.
Ex:- Adding a friend in facebook.	Ex:- *Credit card bill generation
→ ATM transaction	* preparation common friend list
→ Credit card transaction	* Recommendation list of users.
	* Bank statement.



write once Read many time

* Hadoop Ethic is write once Read many times, History should n't be updated and Deleted.

* Basic flow :- of Map Reduces



* Mapper :-

→ By taking input line, Mapper logic writes required o/p key and o/p value.

Ex:- city as key
amount as value

* Mapper logic to be written in "Map-function".

Map ()
is executed for n times, where n is no. of i/p rows.

→ The mapper o/p will be temporarily stored in local disk (OS).
Later this o/p will be sent into reducer.

* Reducer :-

Reducer is three functionalities.

* grooping
* Sort
* Aggregation (Result process)

* Reducer logic will be kept in "Reduce()".

Reducer() is executed for n-times where n is no·of input groups.

Ex:- group by sex

2 types (M/F)

NOTE:- → Reducer o/p stored HDFS.

→ ones Reducer phase completed mapper's o/p will be deleted.

Ex:- ① Structured emp

101, AA, 20000, M, 11
102, BB, 30000, F, 12
103, CC, 40000, M, 12
104, DD, 5000, F, 11
105, EE, 6000, M, 11
106, FF, 7000, F, 12

Query :- Select sex, sum (sal) from emp group by sex.

Map()
↓
write
sex as key
sal as value

Mapper o/p :- Temperaly stored in to local

M	20000
F	30000
M	40000
F	50000
M	60000
F	70000

∵ This is called Intermeadiate data

↓
Reducer → Grouping
→ Sort
→ aggregation

* Grouping :-

(K) Sex	(V) Sal list
M	<20000, 40000, 60000>
F	< 30000, 50000, 70000>

* Sort :-

F < 30000, 50000, 70000>

M < 20000, 40000, 60000>

* Aggregation :- Sum

Reduce ( )

O/P:- Find total of sal list write sum of Sal as key, total as value.

NOTE:- Among mapper and Reducer, Reduce phase is not always required. when ever data grouping is required then only we need to use Reducer.

If Reducer not required we need to suspend it other wise default reducer will be activated.

Reducer O/p

F	150000
M	120000

→ This is stored in HDFS.

[∵ mapper will be deleted]

Ex:

*

*

* M

Quer

Quer

8

Ex:- ② Select * Emp where sex = 'f'

* For this query Reducer not required because mapper output is final o/p.
* If Reducer is suspended mapper o/p permanently stored in 'HDFS'.
* Map logic :- to write sex as key and sal as value.

[101, A, 20000, M, 11]

map ()

String [] words = v. Split (',');

String sex = words [3]

int sal = Integer. parseInt (words[2]);

Context context. write (sex, sal);

Query :- Select dno, Sum (sal) from emp group by dno;

[101, AA, 20000, M, 11]

map ()

String [] words = v. split (',');

String dno = words [4]

String dno ;

int sal = Integer. parseInt (w[3]);

Context. write (dno, sal);

Query:- Select sal, count (*) from emp group by sal;

map ()

String [] words = v. Split (","); String sal = w[3];

Context. write (sal, 1);

10000	1
2000	1
1000	2

↗ Sum
10000 <1, 1>
2000 <1>

[: count = 1 has a valued]

* Reducer for Sum :-

K	V list
F	<10000, 2000>
M	<20000, 25000, 10000>

```
Reduce ()
    int tot = 0;
  for (int v : v list)
    tot += v;
    Con.write (K, tot);
```

O/p :- F : 30000

M : 55000

* Reducer for Avg :-

```
reduce ()
 int tot = 0;
 int cnt = 0;
for (int v : v list)
  {
   tot += v;
   cnt ++;
  }
int avg = tot/cnt;
 con.write (K, Avg);
```

* Reducer for max;

K	V list
F	<10000, 20000>
M	<20000, 25000, 10000>

```
Reduce ()
  int max = 0;
  int cnt = 0;
  for (int v = v list)
```

```
}
  cnt ++
  if (cnt==1) max=v
     max = math.max (max,v);
   }
Con.write (K, max)
```

O/p:-

F : 20000

M : 25000

* Reducer for min;

```
reduce ( )
int max =0; int min=0; int cnt=0;
   for (int v : vlist)
      {
   Cnt ++;
   if (Cnt==1) { max=V ; min=V; }
    min = math.min (min.v);
    max = math.max (max.v);
      }
int range =max- min;
   Con.write (K, range);
```

Query:- select dno, sex, sum(sal) from emp group by dno, sex;

[101, AA, 20000, M,1]

```
map ( )
String [ ] words = v.split (",");
  String dno = w[4];
  String sex = w[3];
  int sal =Integer.parseInt(w[2]);
  String mykey = dno + "\t" + sex;
```

11 F ?
12 M ?
11 M ?
12 F ?
| | | |

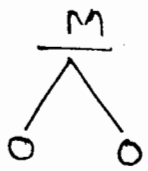
```
Con.write (myKey, Sal);
```

Query :- select dno, sum (sal), avg (sal), max (sal), min (sal), count (*) from emp group by dno;

[Sal, dno = seperated in mapper].

```
Reduce ()
 int tot=0 ; int max=0; int min=0; int cnt=0;
for (int v : v list)
      {
   cnt ++;
  if ( cnt == 1) { max = v ; min = v ; }
   min = math . min (min, v);
   max = math . max (max, v);
    tot + = v;
    }
int avg = tot / cnt ;
String res = tot + "\t" + avg + "\t" + max + "\t" + min + "\t" +
                cnt ;
      Con . write (K, res);
```

[This is multi aggregation purpose.]



How many map executed?

∴ no. of rows

R

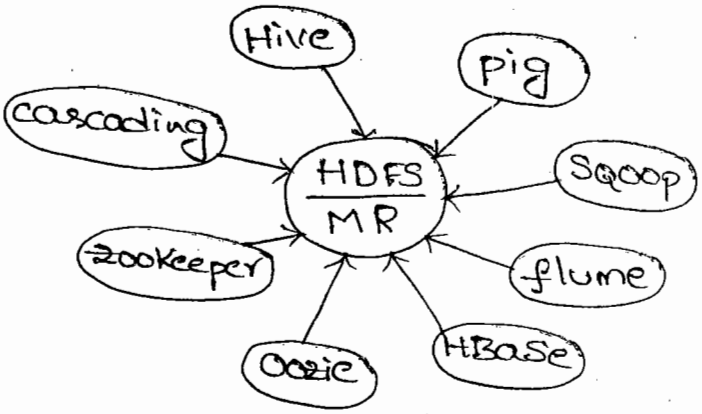
final o/p

no. of groups

9/7/15:

* Eco-Systems:-

* Is an independent software, which runs on top of HDFS and map Reduce.
* That means for the eco-system the backend storage is eco-system and execution model is "map Reduce."



* Hive:- * Hive is a data ware house enviroment in hadoop frame work. where you can store data in tables in "structured" formate. (raws/columns)
* "HQL" is used to manage and process.
  Hive query language.
  This is HQL is similar to SQL of RDMS.
* HQL can process:-
  * Structured data
  * XML
  * JSON
  * URLS → (web logs)

* pig:- * pig is data flow language (piping) in Hadoop.
  * Data flow is collection of pipes.
  * pipe is an operation.
  * The operation can be loading (loading data), transformation, grouping, aggregating (Avg, Sum) sorting, merging, --- etc.

* "pig latin" language is used to process data.

* Hive + pig 70%. data process

* Hive, pig custom functionalities can be done by UDFs. (User define functions)

* Hive UDFs can be developed by following languages.

  → Java
  → python
  → C++
  → Ruby
  → R

* pig UDFs can be developed by following languages.

  → Java
  → python
  → C++,
  → Ruby
  → java script
  → perl

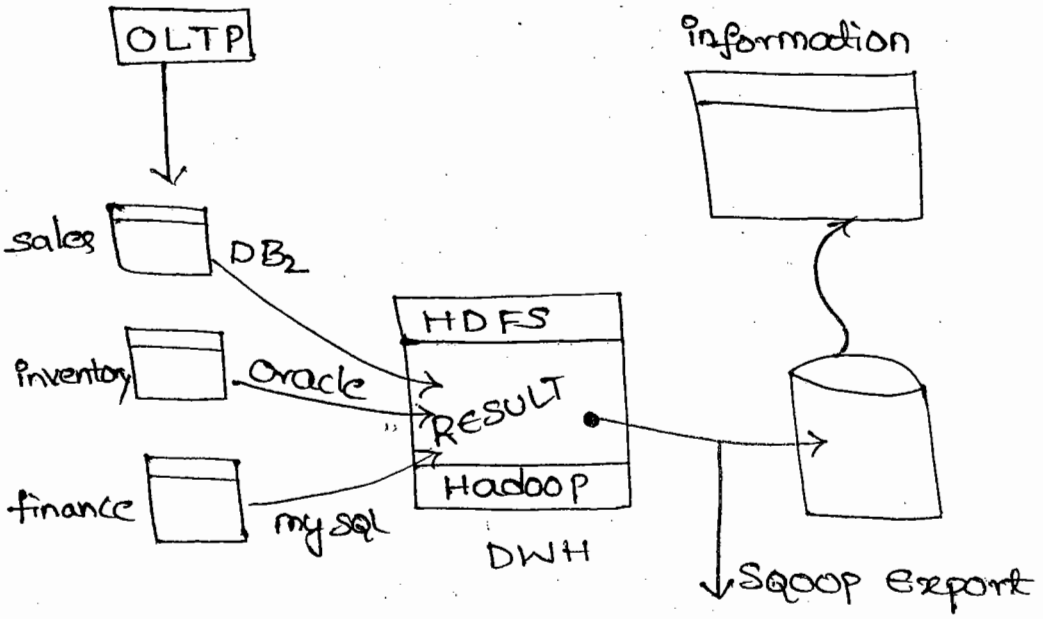
* Hive + Hive UDFs / pig + pig UDFs } 90%. data process

* [The major things in hadoop is Hive & pig.]

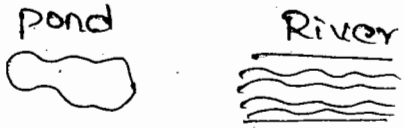
* Sqoop :- It has two tools

  * Sqoop import :- To import data from RDMS to Hadoop

  * Sqoop export :- To export data from Hadoop to RDBMS.



* Flume:— To import streaming data in Hadoop.



10/7/15:

Flume Drawbacks :—

To import streaming data into Hadoop. It event volume.

Disadvantages:—

* Interval based data is not possible.only event based Threshold / volume based Threshold possible.
* Flume can't do transformations while importing for live analytics it is not support. It will not under flume.
* Flume can't talk with db's (Reply back) bcoz of 2nd & 3rd draw backs flume can not be osed for live analytics, it simply supplies data to Hadoop.
* It keeps distrubuting streamming sources.
* NO Guarantee, that each data dumps hits the target.
* we are not recommend for

* Storm :-

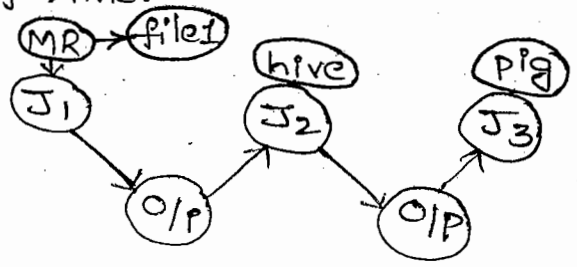
It's advance streaming data process system which can used

* oozie :-

  * to define work flows
  * to shedule work flows

→ Advantages of work flow to set dependency b/w to set different jobs.

→ The work flow is depended by interval base of point of time.



Ex :-

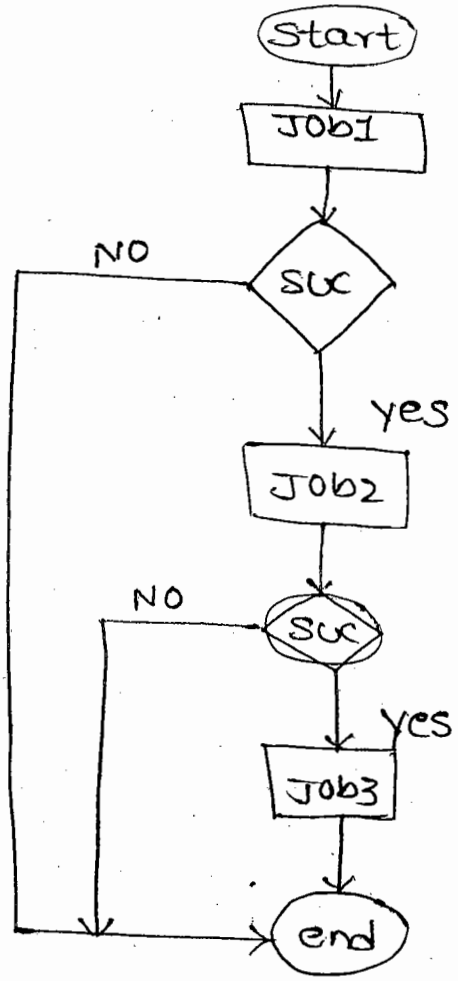
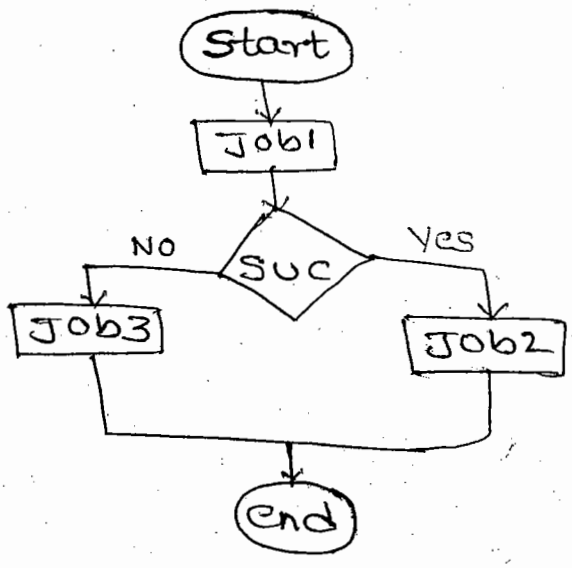


fig :- The work flow of a job

Ex:-



* Sheduling of work flow :-

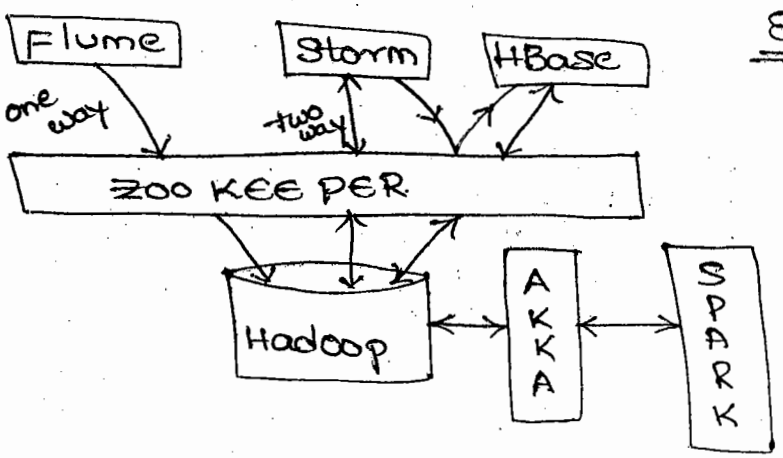
Interval based

POT
point of Time
To
Time

* Drawbacks Oozie :-

* Oozie can not follow custome calender.
* dynamic Scheduling is not possible.

* ZOOKEEPER :- To manage collections and locks between diff distributed Apps.



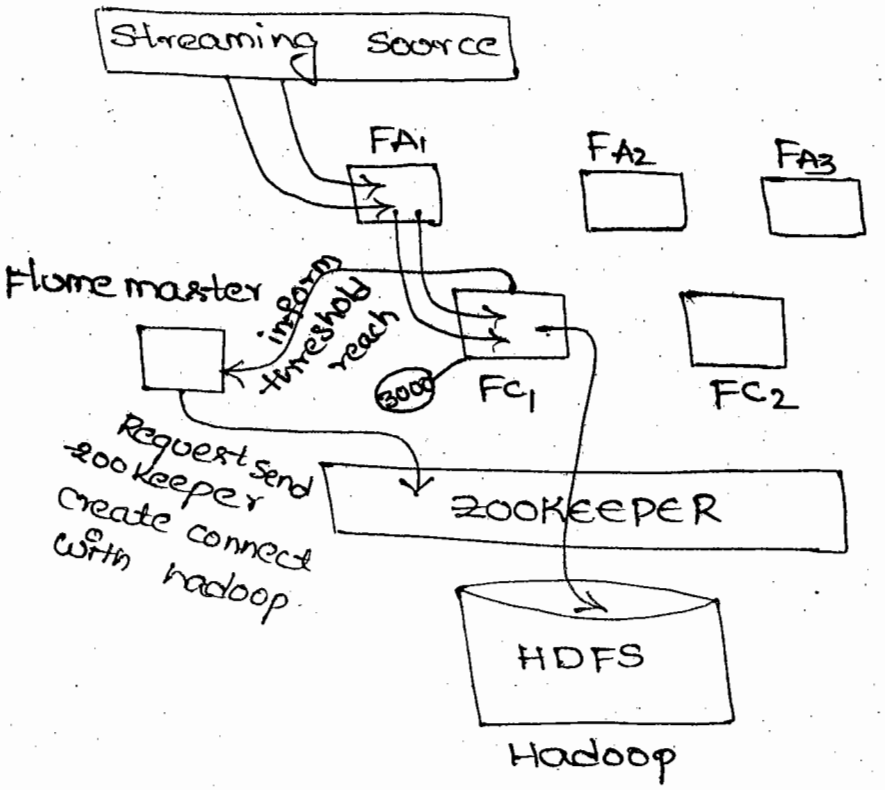
Ex:- Hadoop and Flume

Hadoop - HBase

Hadoop - Storm

Strom - HBase

* Flume Vs Hadoop :-



[∵ FC= Flume collector]

* Cascading :-

* It is a piping language similar to pig. (data flow)

* Cascading an API. Available for following language

→ Java
→ python
→ C++
→ Ruby
→ Scala.

most :-
Scala + cascading
= Scalding
→ to define complex flows.

* cascading API for Scala is called Scalding.

→ more complex operations made cascading

* NO SQL :- [NOT ONLY SQL] [ONLINE]

Basics of Adv of NOSQL :-

* High level Scalability (very large capacity)
* Schemaless, [Structure of table]
  ↳ Flexible schema

Ex:-

Tab

Name	Age	Sex
Ravi	25	M
Rani	24	F

* Faster Random CRUD operations.

[C R U D]

Create Read Update Delete

[* Table backend Table Space
Index backend Index space
Bufferpull :-

Facebook :- Random triler
Attempt → got error
(Read error) → again attempt.
It is viewing batch process]

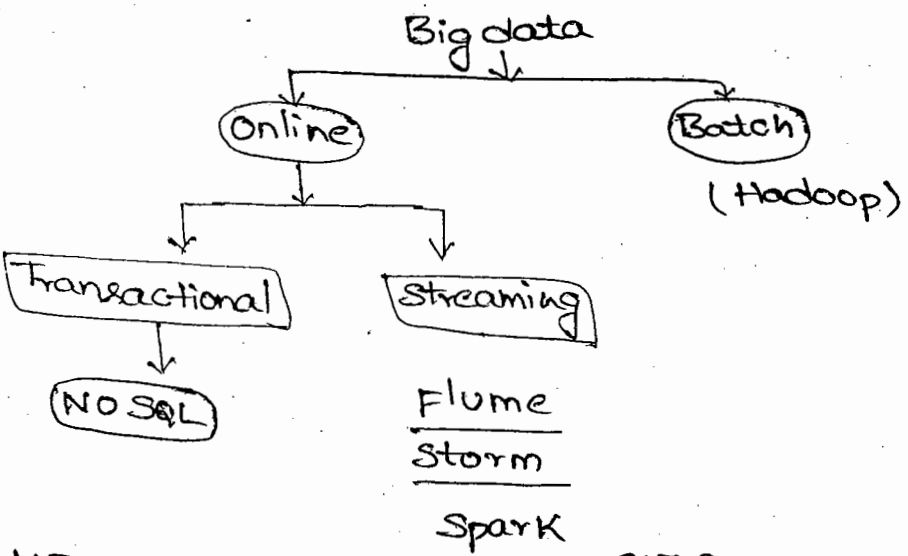
NO SQL data base havebeen classified in to four types :-

* Key value Store
  Ex:- * Riak (FB SignIn)
  * PNUT (exclusive product yahoo)
* columner Store
  * cassandra
  * HBase *** [Don't Neglect HBase - Apache Hadoop - Apache]

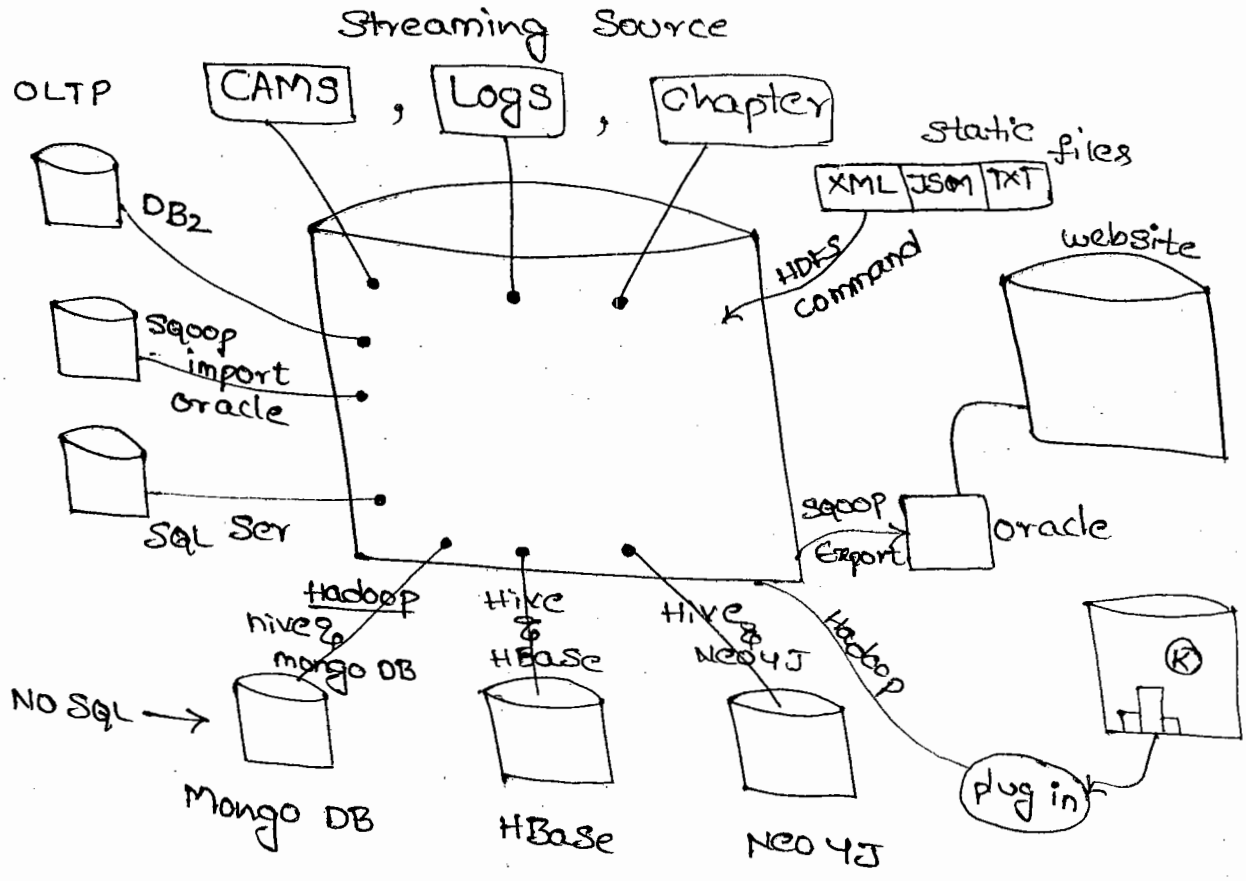
Ex:- Search / Frequent profile updations
Frequent profile crud.

* Document Store
  * Couch DB (Appache) Utterly plop
  * Mongo DB ***
* Graph Store
  * Neo4J

[10Gen.com → Mongo DB
Register Here --fecility]



* HBase :- is one a (NOSQL) OLTP data base, comes under columners Store category.
* KAFKA :- is a "messaging Service" in Hadoop.
  → simply message broker
  parallel to JSM
  → The integration hadoop and non hadoop



* Data cleaning
* Data validations
* Data preperations
* merging data sets
* making unstructure to structered.
* Analysis
* Result.

12/7/15:

* Linux COMMANDS:-

~]$ ls :- List of files and directories

* ls vinoth → Temp stats store

* hadoop fs -ls (checking hadoop files)

/ user / traing → user id

→ name space for HDFS

root directory

Ex:- / BMC / Devoo 1 → user id

root directory → name space for HDFS

In LINUX:-

* ~]$ mkdir local s1 (create a local directory)

~]$ ls local s1

In hadoop:-

* ~]$ hadoop fs -mkdir hadoopers 1.

Control +D → Save

$ hadoop fs -put

data - .csr

A branch of files copy.

[check out the file]:-

~]$ cat file1 (creating files)

[Add some content]:-

~]$ cat >> file1

[I want to copy in hadoop]

~]$ hadoop fs - copy*From*local file1 hadoopers1

~]$ hadoop fs - ls hadoopers1 [checking]

~]$ hadoop fs - cat hadoopers1 / file1

~]$ cat > file2 NOTE:- you can not directly create file in hdfs.

aaaaa

bbbbb

ccccc

ddddd

(move from local files)

~]$ hadoop fs - move↑from local file3 hadooper1

~]$ hadoop fs -ls hadoopers1 [Checking]

[I want Reverse it] [From hadoop to Local]

~]$ hadoop fs -copyToLocal hadoopers1/file1 locals1

[source (HDFS)] [Target (local)]

$ ls locals1

NOTE:- -moveToLocal is not at implemented.

[If u want delete a Hdfs file]

$ hadoop fs -rm hadoopers1/file2

[Check out either Keep in or not]

hadoop fs -ls hadooper1

[I want to Copy hadoop to hadoop] Copy paste

$ hadoop fs -cp hadooper1/file3 hadooper2 ↑directory

[I want to move] cut & paste

$ hadoop fs -mv hadooper1/file1 hadooper2

moved

[If u want delete directory]

~]$ hadoop fs -rmr hadoopers1

remove recursivity

[I want to rename] change the name

$ hadoop fs -mv hadoopers2/file3 hadoopers2/fz ← change name

~]$ hadoop fs -ls hadoopers2 [checking].

13/7/15:-

Map Reducer

* Map Reduce is a batch job.
  which is subdivided into two phases.
  * Mapper phase (Task)
  * Reducer phase (Task)
* For mapper and Reducer · Input and output should be key and value formate.
  → when line is submitted to mapper, lines offset number is key, entaire line is value.

* Mapper logic to be written in "map() function".
* Logic:- By taking input value, it seperates and writes required o/p key and o/p value.
* Map():- Excutes from n times where n is no.of input rows of input file.
* But in distributed systems (That means cluster) file data is not available in single machine.
* File is Divided into Blocks,
  Blocks will Replicated,
  These Replicas will be distributed into diff slaves.
* when job is submitted NO.of mapper task is initiated is = NO.of onique blocks.

Ex:- Local file = f_1

Size = 100 MB

when copied into hadoop, f_1 is divided into two Blocks each Block Replicated for 3 times.

So, f_1 is HDFS has 6 blocks.

Explanation:-

B_1	B_1	B_1	B_2	B_2	B_2

In this example job submitted on file1.job has to be divided into two tasks.

Initial tasks are mapper Tasks.

So First job is divided in to two mapper tasks, Because 'f_1' has two unique blocks. tasks, Each, mapper task, as map() function.

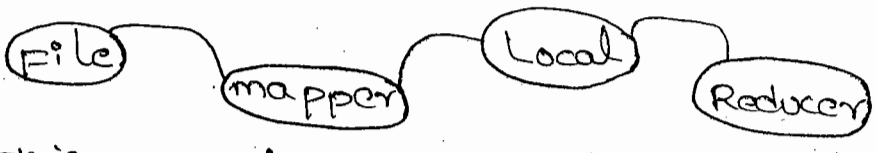
Each, map() function works with own block.

So map() function works with execute for n times where n is no.of input rows of the block.

* Map( ) :-
  * Mapper o/p is temperorily stored in local DISK (OS)
  * If Reducer is suspended, mapper's o/p is stored in HDFS perminantly.
  * map o/p should be key and value formate.
  * Org.apache.hadoop.mapreduce.mapper
    class package.
    This class is used to define custom mapper class.
  * when java class extended mapper class the java class will get mapper functionality.

Ex:- public class mymap extends mapper<--->
{
≡
}

* HDFS / Mapper Data types :-
  * HDFS uses special data types called "writables".
    In mapReduce process data stored in disk of slaves, this data to be transported accross the network.

Ex:- 

  * In this serialization and deserialization activity should be done.
  * mapReduce provides you, predefined Serialization and deserialization formates with HDFS, Network called "writables" and also called as "mapReduce data types."

* Map Reduce has writables for almost java primitive types.

Ex:-

Java	writable
Int	Int writable
long	long writable
boolean	boolean writable
null	null writable
String	Text
float	float writable
double	double writable.

* writable are available in following package

Org.apache.hadoop.io.*

* mapper class has 4 types of generic parameters. They are following below
  * Input Key data type
  * Input value data type
  * output Key data type
  * output value data type.
* These data types should be writables.

```
public class my map extends
        I/p K   I/p V   O/p K   O/p V
mapper <long writable, Text, Text, Int writable>
        offset  Line    word    1
        number                  value
```

NOTE:- Mapper O/p Key and value should be equalent to Reducer I/p Key and value.

* Mapper class following function :-
  * Set up ( ) :→ executed for one time before map ( )
  * map ( ) :→ no. of rows in Block
  * Clean up ( ) :→ one time of after completion of map ( )
* Map ( ) has 3 arguments :-
  * variable for i/p Key (offset no)
  * variable for i/p value (line)
  * Context variable to write results.

Mapper has an inner class context, context has write ( ) method, to write key and value to disk.

14171715 :-

Query :- Select sex, sum(sal) from emp group by sex;

```
public class sex sal map extends mapper
{                    < longwritable, Text, Text, Intwritable>
  public void map ((longwritable K, Text V, context
                                                  con ))
          throws IOException, Interrupted Exception
  {
    String line = v.to string ( );
    String [ ] w = line.split (",");
     String sex = w[3];
     int sal = Integer.parseInt (w[7]);
     Con. write ( new Text (sex), new Intwritable
                                          (sal));
        }
      }
```

* Mapper has a Inner class 'Context'

Org.apache.hadoop.mapReduce.mapper.Context
class one more class

Context has

* get Configuration ()
  → to get files HDFS Configuration.
* write ()
  → to write result into disk.

NOTE:- Result shape is K,V

Ex:-① Mapper for word Count
[Same like hello world pgrm in java]

Sample data:-

*/ I Love you
you love hadoop
*/

```
public class wordmap extends mapper<Long
        -writable, Text, Text, Intwritable>
{
  public void map(longwritable K, Text V,
                  context Con) throws
          Ioexception, Interrupted exceptions;
NOTE:-
[Converting Text to string]
  {
    String line = V.to string ();
    String tokenizer t = new String Tokenizer
                         (line);
    while (t.has more Tokens())
    {
      String word = t.next Token();
```

con.write (new Text(word), new IntWritable(1))

}
}
}

O/P :-

I	1
Love	1
you	1
you	1
love	1
hadoop	1

sent to Reducer

I <1>
love <1,1>
You <1,1>
hadoop <1>

Ex :- ②

*/ Sample data :-

trid	Prid	amt	qnt	location
tr_1	P_1	1000	2	hyd
tr_2	P_2	2000	5	del
tr_3	P_3	1200	1	del
tr_4	P_4	1100	2	hyd
tr_5	P_2	1200	3	del

Query :- select loc, sum(amt * qnt) from sales
group by loc;

*/ public class salesmap extends mapper
<longwritable, Text, Text, Intwritable>
{
public void map(longwritable k, Text v, Context con)
throws IO exception, Interrupted Exceptions
{
String line = v.toSplit();
String [] w = line.Split(",");
String loc = word[4];
int amt = Integer.parseInt(w[3])
int bil = qnt * amt;

```
con.write(new Text(loc), new IntWritable(bill))
    }
  }
```

O/P:-
```
Hyd   2000
del   10000
del   1200
Hyd   2200
del   3600
```

Ex:- ③ Sample data :-

```
ecode , name , sal, sex , dno
  101 ,  Ravi , 2000 , m , 11
```

Query:- Select dno, sex , sum (sal) from emp group by dno; sex;

```
*/
public class Empmap extends mapper
        <LongWritable, Text, Text, IntWritable>
{
public void map(LongWritable K, Text V, context con)
        throws IOException, InterruptedException
{ String line = V.toString();
  String [] w = line.split(",");
  String dno = word[4];
  String sex = word[3];
  int sal = Integer.parseInt(w[2]);
  String mykey = dno + "\t" + sex;
con.write(new Text(mykey), new IntWritable(sal));
} }
```

O/P:- 11 m 2000
mykey ↵ value ↵

Ex:- ① Sample data:-

ecode, name, sal, sex, dno

Query:- select * from emp where sex = 'F';

*/ public Emp map extends mapper< longwritable,
{ Text, Text, null writable>

public void map (longwritable K, Text V, context
con) throws Io Exception,
Interrupted exceptions

String line = V.to string ();
String sex = line.split (",") [3];
if (sex.matches ("[F, f]"))
con.write (v, null writable.get());
}
}

15/7/15

* Map Reduces practical sessions:-

*// word count programm :-

*// Wordmap.java * //*

package mr.demo
import java.io.Ioexception;
import org.apache.hadoop.io.Intwritable;
import org.apache.hadoop.io.longwritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapReduce.mapper;
import java.util.stringTokenizer;
public class wordmap extends mapper < longwritable,
Text, Text, Intwritable>

```
{
public void map (long writable K, Text, context con
        throws IOexception, Interrupted exception
 {
  String line = v.to string ();
  String Tokenizer t = new string Tokenizer (line);
  while (t.has more Tokens())
      {
   string word = t. next Token ();
   Con.write [new Text (word), new Int writable (1)];
      }
   }
}
```

// word Driver java //

```
package mr. Demo
import org. apache . hadoop . conf . configuration;
import org . apache . hadoop . fs . path;
import org . apache . hadoop . io . Intwritable;
import org . apache . hadoop . io . Text;
import org . apache . hadoop . mapReduce . Job; input.
                                    FileInput format;
import org . apache . hadoop . mapReduce . lib . output,
                                    file Dutput formate;
    publi  class wordDrives
          {
  public static void main ( string [] args)
                throws Exceptions
```

```
{
Configuration c = new Configuration ();
  Job j = new job (c, "my first job");
    j.setJarByClass (wordDriver, class);
    j.setMapperClass (word map.class);
//  j.SetNumReduceTasks (0);   * NOTE *
//  j.setoutputKeyclass (Text.class);
//  j.Setoutput value class (Intwritable.class);
    path in = new path (args[0]);
    path out = new path (args[1]);
fileInputFormat.addInputpath (j, in);
fileoutputFormat.setOutputpath (j.out);
  System.exit ().waitForCompletion (true)? 0:1);
      }
   }
```

In hadoop Execution :-

1) package name
2) driver name
3) directory name in file name
4) input file :-
5, 6) :- Directory name in o/p file :- result

```
~]$ ls comment
comment available.
~]$ hs cat comment
I Love India
I love hadoop
I love mother
~]$ hadoop fs -mkdir test (creating adirectory)
~]$ hadoop fs -copyfromlocal comment Text
~]$ hadoop fs -ls text (checking copy or not)
~]$ hadoop jar Destop/demo.jar mr.demo.word driver
    > text/comment test/result
                ①          ②
```

output file this execution is

map input record = 2

map output records = 9

output mapper is formed by

$ hadoop fs -ls test/result

-rw-r--r--1 training supergroup 59 2015-09-13 22:02/user/training test/result/part-m-00000

Checking for mapper;

~]$ hadoop fs -cat test/result1/part-m-00000

Output mapper:-

```
i |
love |
India |
i |
love |
hadoop |
i |
Love |
mother |
```

Reducer output checking;

~]$ hadoop fs -cat test/result3/part r-00000

Output Reducer:-

```
hadoop |
i |
i |
i |
India |
Love |
Love |
mother |
```

NOTE:- To Suspend Reducer we need to write this statement otherwise default reducer run if default running we need to provide (K,V)

16/7/15

*/ Column filter :-

```
{String line = v.toString();
String[] w = line.split(",");
String newline = w[1]+","+w[3]+","+w[4];
con.write(new Text(newline), NullWritable.get());
}
}
```

*/ News Filter :-

```
{
  String line = v.toString().toLowerCase();
  if (line.contains("dell"))
  con.write(v, NullWritable.get());
  }
```

17/7/15 — Map Reducer in Java PG Entering

* Open eclipse
* Create New project Name
* Java project [Give Name] [like column]
* Finish
  (Created)
* Right click
  → Click package
    (Creating)
  → Name [Give package Name] [heros]
  → Finish
* Right click
  → Create class [map class]
  → Click class
  → Name [ ]
  → Finish
* Create again another [Driver]
  → Click Class
  → Give any name [Row Driver]

Next Step :-

→ put on Crusher in project
→ Right click
→ Build path[→]
→ Configure build path

→ Add external jar

→ hadoop-0.20

→ hadoop-core.jar

→ Lib

→ commons-cli-1.2.jar

** → Column

→ Right click

→ Export

→ Give any "name"

18/7/15 → Finish

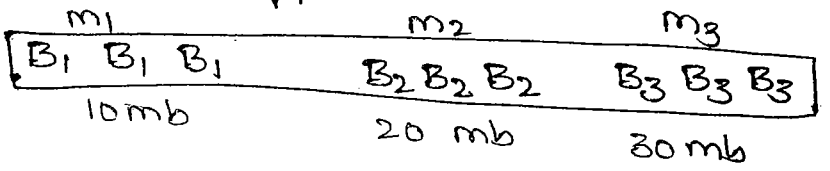
* Reducer Functionality :-

Reducer collects input from mapper outputs, between mapper outputs and Reducer inputs "partitioning" phase will happen.

* what is the partitioning phase?

A:- Directing proper keys to proper reducers.

* Number of Reducer Depends on total outcome of all mappers.



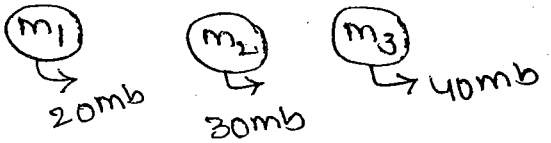
* Number of mappers = No. of unique blocks

total outcome of all mappers = 10+20+30

= 60mb < 64 mb

No. of Reducers = 1

*



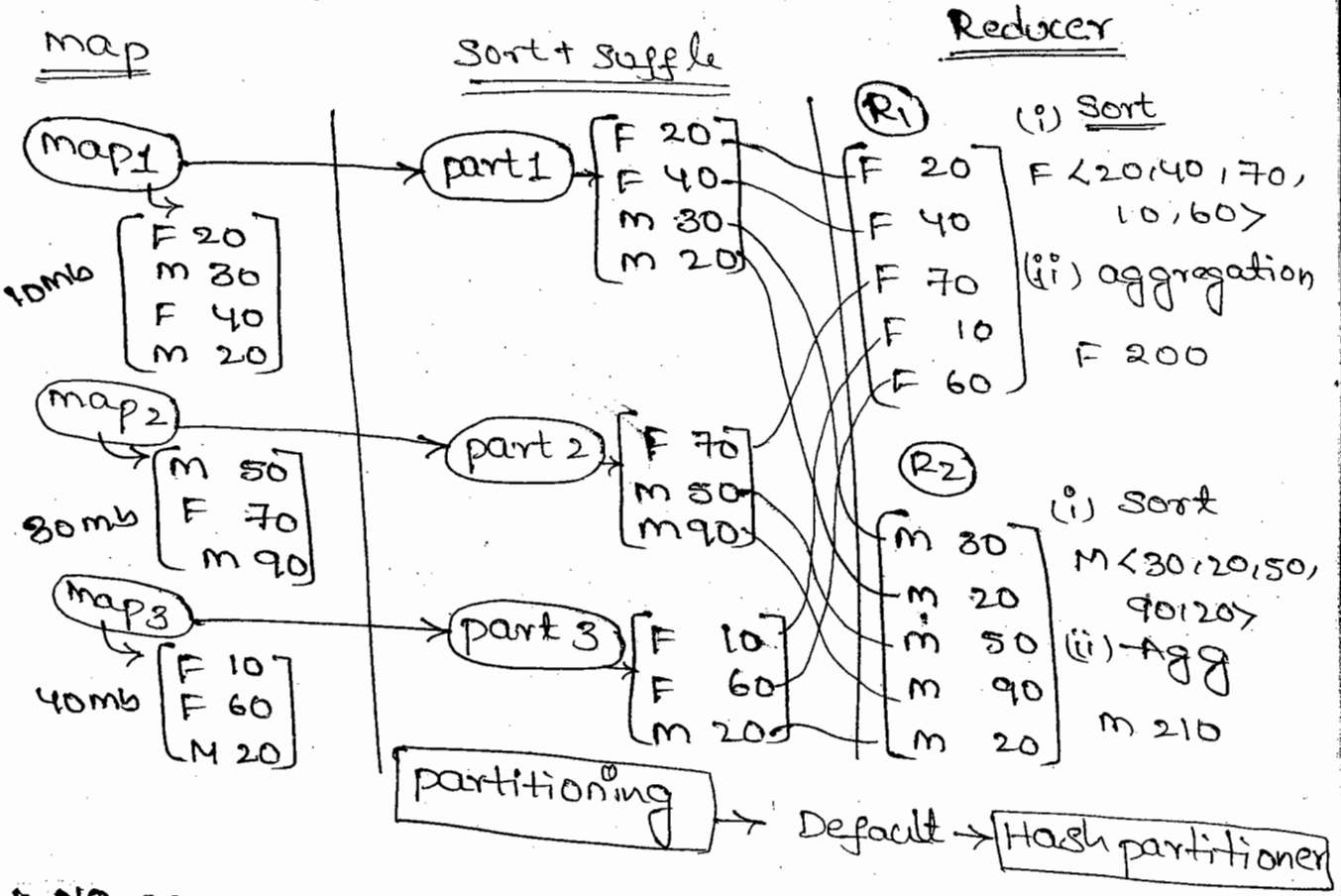
total outcome of all mapper = 20+30+40

= 90 mb > 64 mb

= <128 mb

NO. of Reducers = 2

* Sex (Key), Sal (val) :-

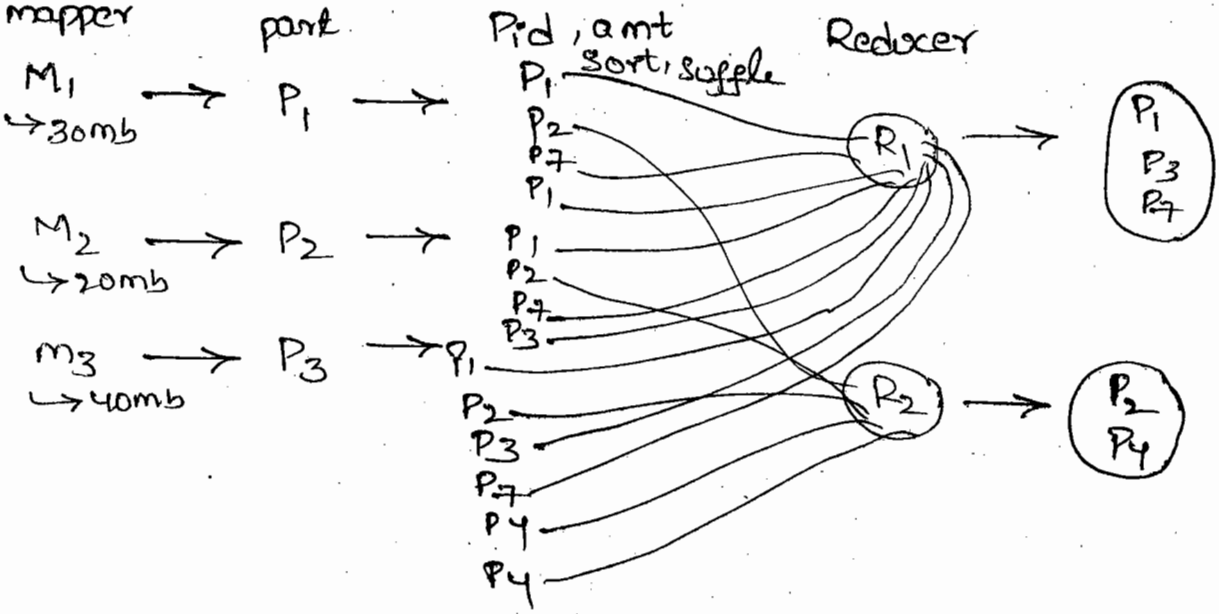


* NO. of partitions = NO. of mapper

* NO. of mapper = NO. of unique blocks

Ex:- If NO. of keys is greater than NO. of Reducers.

Pid (Key), amt (val)

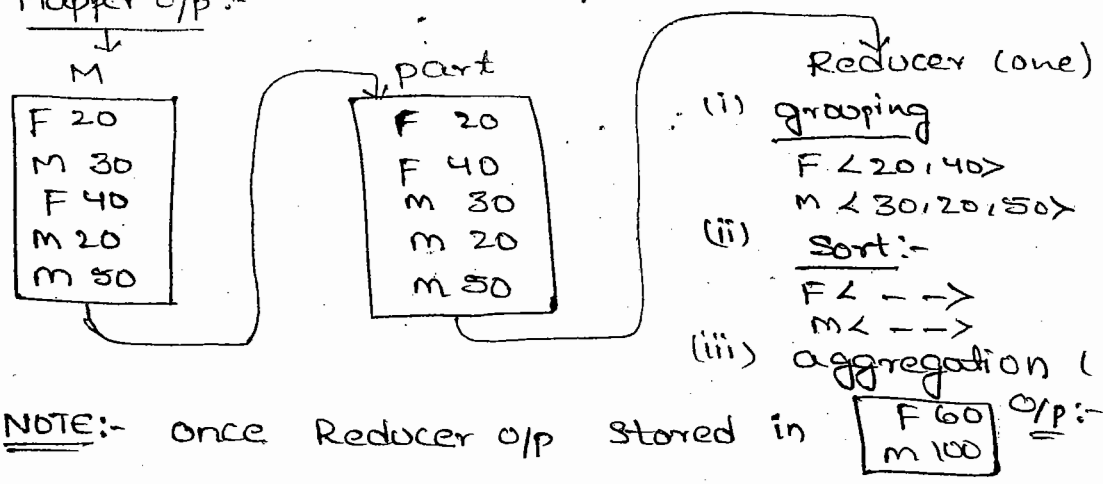


19/7/15:-

Reducer

* partitioner o/p is sent to Reducer
* The Reducer work
  - → Grouping
  - → Sorting :- *Sort is already done at partitioner level.
    * you can design group sort by writting your own algorithm.
  - → aggregating

Ex:- sum, Avg, max, min, count



NOTE:- once Reducer o/p stored in HDFS, mapper o/p will be deleted.

* Reducer Class:-

  org.apache.hadoop.mapreduce.Reducer.

  ↳ This is to define customs Reducer class.

* when java class extended Reducer the class will get Reducer functionality.

* Reducer class has 4 Generic parameters
  * input Key writable
  * input value writable
  * output Key writable
  * output value writable

NOTE:- Mapper output key, output value should be equal to input key, input value writable.

Ex:- public class mymap extends mapper <Longwritable, Text, Text, Intwritable>

```
{
==
}
```

public class myRed extends Reducer < Text, Intwritable, Text, Intwritable >

```
{
==
}
```

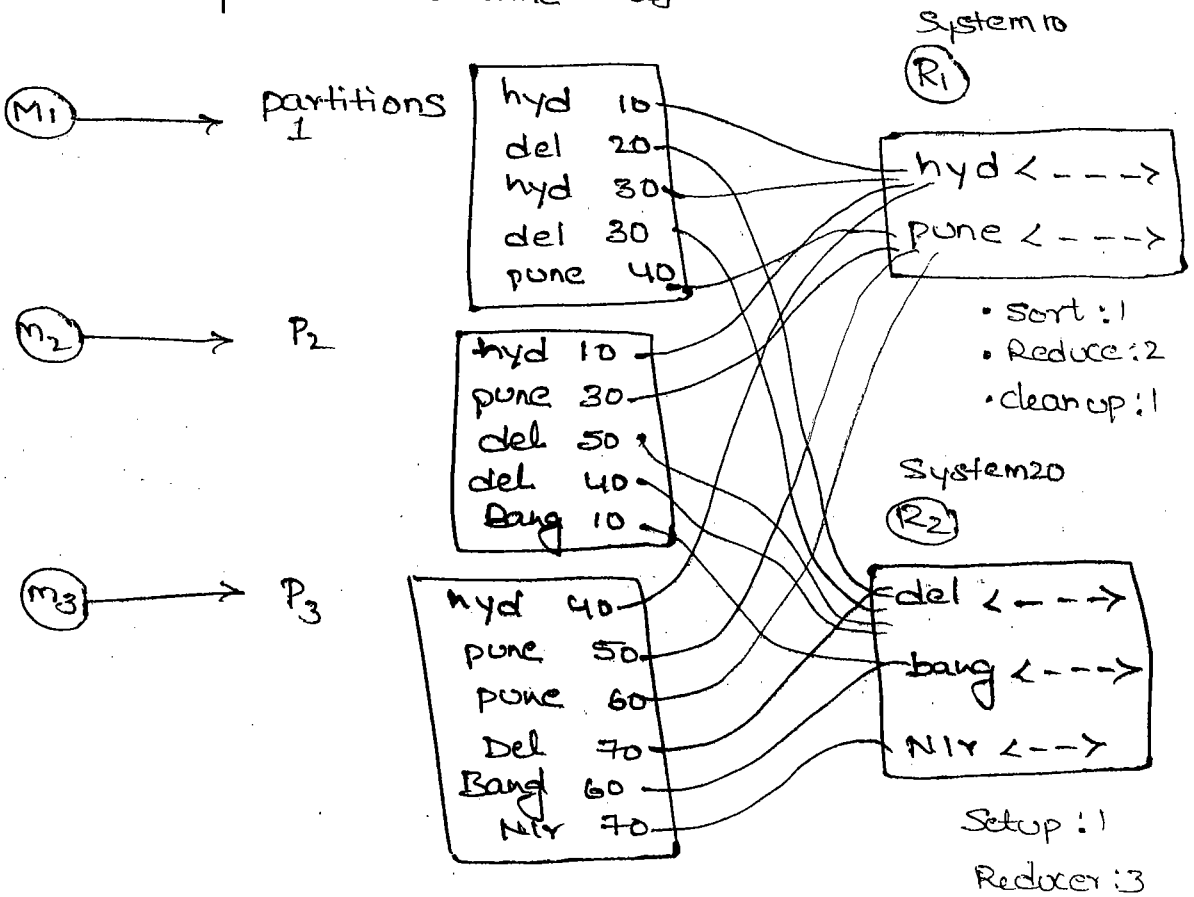
If not match IOException.

* Reducer class has 3 functions :-

→ setup() :- one time before Reduce

→ reduce() :- excuted for 'N' times where N is NO. of data groups.

→ Cleanup() :- One time after Reduce



* Aggregation logic to be written in Reduce function.

* Reducer functionality :-

→ Grouping :-

If key not existed a fresh row will be inserted if key already existed the value will be appended to (vlist).

Reduce function has 3 arguments;

* Input key variable
* Input value Iterator
* Context variable

Org.apache.hadoop.Mapreduce.Reducer.Context

* Context has write() method, to write result into disk

Ex:- Red for Sum

```
public class RedforSum extends Reducer <Text, Intwritable,
                                        Text, Intwritable>
{
public void reduce (Text K, Intevable <Intwritable> vlist,
                                        Context con) throws
                    IOException, Interrupted Exception
    {
    int tot = 0;
  for (Intwritable v: vlist)
  for (Intwritable v: vlist)*
      tot += v.get();
Con.write (K, new Intwritable (tot));
}}
```

NOTE:- get() is converting the Intwritable to Int.

21/9/15 : Map Reduce :-

* Map Reduce is a batch job. which is one of execution model in Hadoop frame work.

NOTE :-

* where MR bad, "DAG" will come
* where MR is bad for Iterative algorithms.

Ex :- For Iterative algorithms.

K- means
K - Nearest Nehibhour
Decision Tree
Random forest

* MR is also bad for "Adhoc" Querying Data.
  Asking again & again yesterday & today's sales data comparing.

* S :- "DAG"
* MR yearly Asking sales Report is Good [2011, 2012, ...]
* 0.14 hive, we can choose it, it will MR model or DAG.
* hive < MR / DAG → TEJ (very speed)
* pig < MR / DAG → Spark
* MR is batch job is batch job which is divided in two phases
  * Mapper phase (TASKS)
  * Reducer phase (TASKS)

  → Master → JT

* M/R :-
  * Takes I/P formate key & value and write I/P in key & value format

Ex :- Select name, age, sal, emp;

name, age, sal } my key = name + ',' + age + ',' + sal

* key - which is starting grouping.
* value - which is aggregation done.
* offset Num can not be dublicate, No Row will be missing.
* partitioning :-
  ↳ (sort & suffle) phase

→ Default partitioning is → Hash partitioner

This is partition phase, Direct proper key to specific Reducer.

Ex:- All females into Reducer 1
All Males into Reducer 2.

Ex:- All prid (P_1, P_3) into Reducer 1
All prid (P_2, P_4) into Reducer 2

→ In this moment
Reducer 1 aggregates P_1, P_3 groups & Reducer 2 aggregates P_2, P_4 groups.

→ If no.of grouping keys are 4, and if no.of Reducer initiated is 4, then each seperate key will be sent to a seperate Reducer.

All $P_1 \rightarrow R_1$
$P_2 \rightarrow R_2$
$P_3 \rightarrow R_3$
$P_4 \rightarrow R_4$

* If no.of keys are > no.of Reducer initiated

→ Hash code of Key / NO·of Reducer ,

if Remainder = 0, then the Key will be sent to → R_1
= 1, " " " " " → R_2
= n-1 " " " " " → R_n

hash (P_1) %.3 = 0 → R_1
hash (P_2) %.3 = 1 → R_2
hash (P_3) %.3 = 2 → R_3
hash (P_4) %.3 = 0 → R_1
hash (P_5) %.3 = 1 → R_2

once a Key is sent to a Reducer All similar Keys (same) will be sent to same Reducer.

* How many partitioners :-

NO·of partitions = NO·of mappers.

NOTE :-

* NO·of mappers / partitioner can not be manually controlled
* NO·of Reducer = int (total outcome of mappers / Block size) + 1

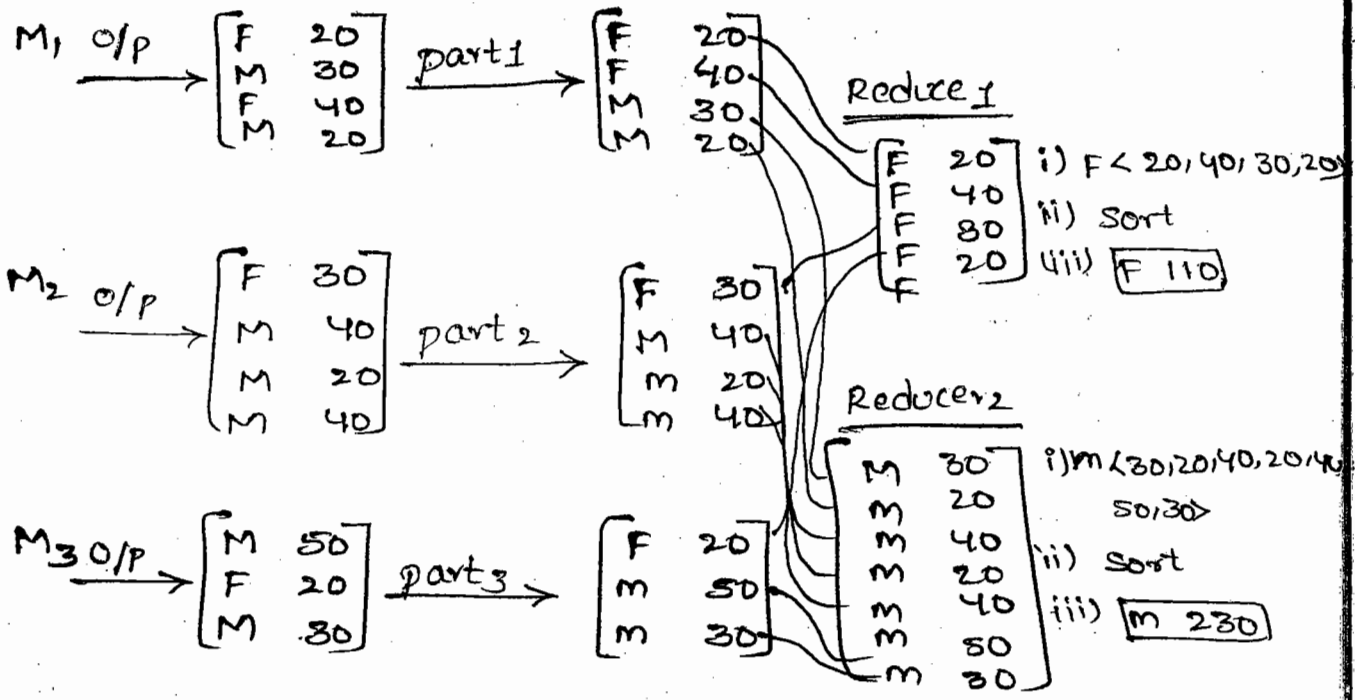
Ex:- $\left(\frac{90\,MB}{64\,MB}\right) + 1 = 2$

* NO·of Reducer can be controlled

Ex:- j.setNumReduceTasks(2)

* partitioners are executed at local → mappers
In which machine mapper executed
* with out mapper, HDFS API will be allowed to Run a partitioner will be Awoked by Reducer will there.

Ex:- ①



==o==

① Hadoop cluster (server) to another Hadoop duster :-

$ hadoop dfs -distcp hdfs://IBM.fin.com:3020/user/IBMS/data1
(host name, port num)

hdfs://IBM.Backup.com:9000/user/FBMS/backup1 ↵

NOTE:- distmv is not available.

Backup Statergies:-

* In Target Backup cluster, No. of Replicas configured as ①

Delete a file :-

$hadoop fs -rm /user/training/hd2/file1

→ Delete a directory

$ hadoop fs -rmr hd2
(remove recorssive)

→ How to increase / decrease no. of Replicas :-

$ hadoop fs -setrep hd2/filex -w 2

< 0.2	>= 0.2) versions
Fixed Block Size	variable block size
follow configuration file	file to file different configuration can be maintained.

Fixed No. of Replicas	variable No. of Replicas

* How to track HDFS file system using web interface:-

http://Local host : 50070 ↵

→ File system
→ Logs
→ Space availability
→ HDFS Health.

Live nodes dead nodes decommissioning nodes.

* How to monitor job from web consorn:-

http://Local host : 50030
→ jobtracker

→ It will give separate logs for each job
→ running job
→ mapper tasks < both running & finishing
→ reduce tasks < both

* Monitoring Task tracker :-

→ http://Local host : 50060
→ No port for data node

* From Hadoop to the ecosystem
→ real time monitoring it done
→ To manage the cluster & monitoring the cluster.

Ambari
↓
Admit tool

* Horton works / map R distributions use Ambari
* IBM Big Insights / cloudera distributions use Cloudera managers

Ambari
is a product up apache

Cloudera
* Automated
* GUI
* start stop
it is product of its self

* Combiner :-

Combiner is a mini Reducer.

It is executed at local to mapper.

In which mechine mapper executed.

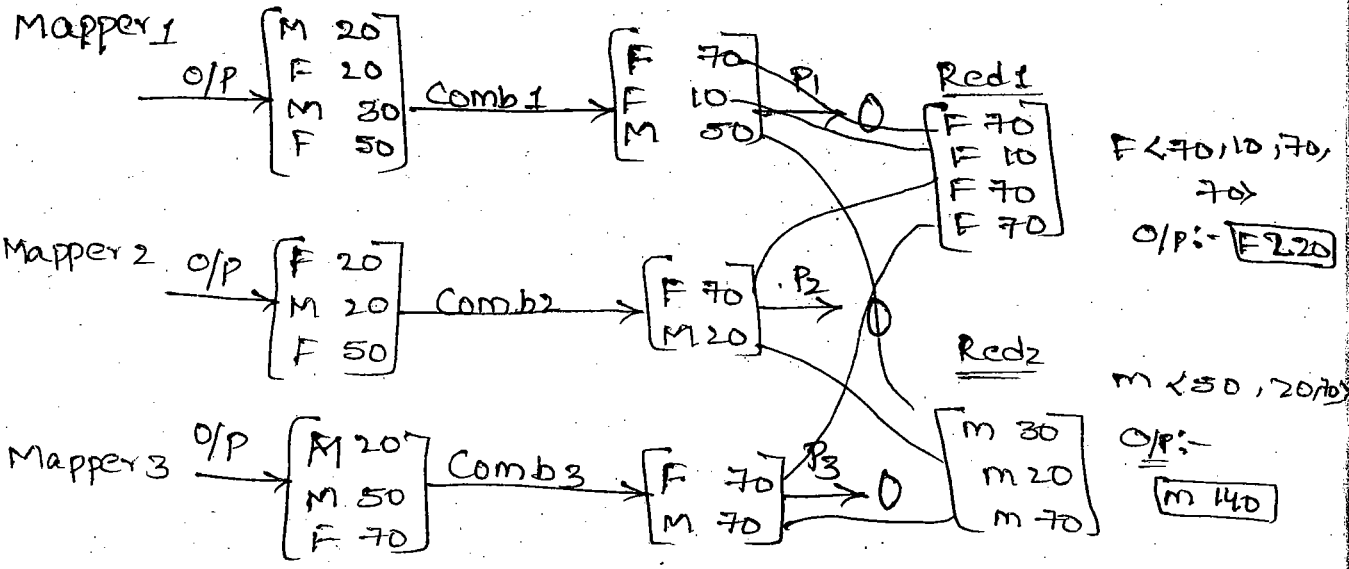
i.e No. of Combiners = No. of Mappers = No. of partitioning = No. of unique blocks.

* Advantages of Combiners :-

To decrease load on Reducers. we can choose how much percentage of load to be decreased.

Ex:- sex is key, sal is value from mapper o/p Instead of sending 1000 females to Reducer, send only 500 females to same Reducer by making sum of every two female salaries as one.

* Mapper - combiner - partitioner - Reducer :-



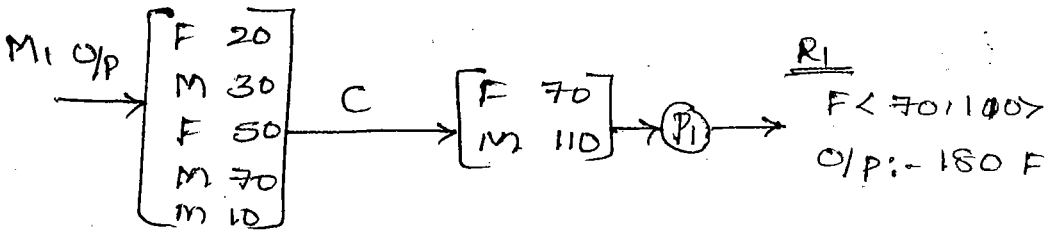

```
public void reduce (Text k, Iterable<IntWritable> vlist,
                    Context con) throws IOException,
                                        InterruptedException
{
  int tot=0
  for (IntWritable v:vlist)
      tot += v.get();
  con.write (k, newIntWritable (tot));
}
```

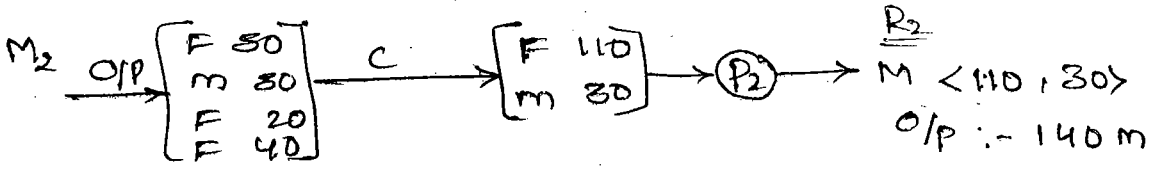
* Combiner for sum (make every two sum)

```
public void reduce (Text K, Iterable<IntWritable>
                    V list, Context con) throws IOE IE
{
  int tot=0; int cnt=0;
  for (IntWritable v: vlist)
    {
    cnt++;
  tot += v.get();
    if (cnt==2) {
  con.write (K, new IntWritable (tot))
    }
  }
if (cnt==1) Con.write (K, new IntWritable (tot));
  }
```

* Reducing 100% of each mapper:-

```
j.set Combiner class (R1.class)
j.set Reducer class (R1.class)
```





NOTE:- *Combiner's o/p is also stored in local.

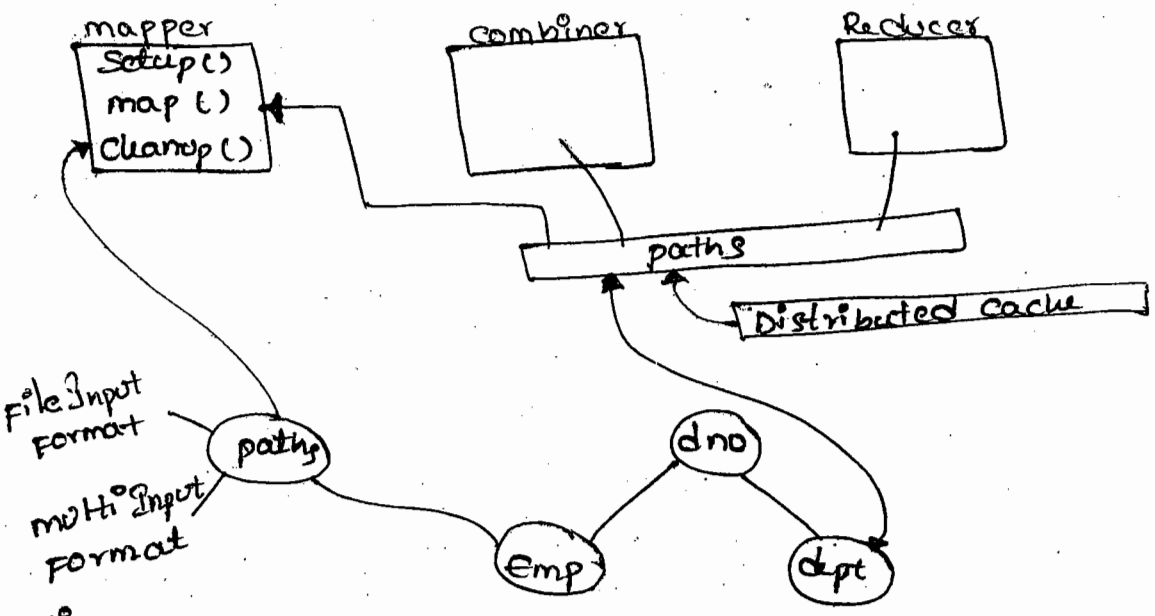
* Combiner should be used only for associative operations.

Ex:- A numeric point of view
a+b = b+a } associative
a*b = b*a
a-b ≠ b-a } dis associative
a/b ≠ b/a

Matrix point of view
A+B = B+A
A*B ≠ B*A (X)

8/10/15:

JOINS :-



Joins:-

Emp

Ecode	name	sal	sex	dno

dept

dno	dname	loc

PK — Unique

Emp → public void map(K, V, con) { = Record Reader }

dept → public void setup(con) { = } path[:] = Distribute cache get Local cache files (con.getconfiguration());

pass configuration information to context

Driver

Distributed cache.addCacheFile(Pnz to use().getconfiguration());

Combiners :-

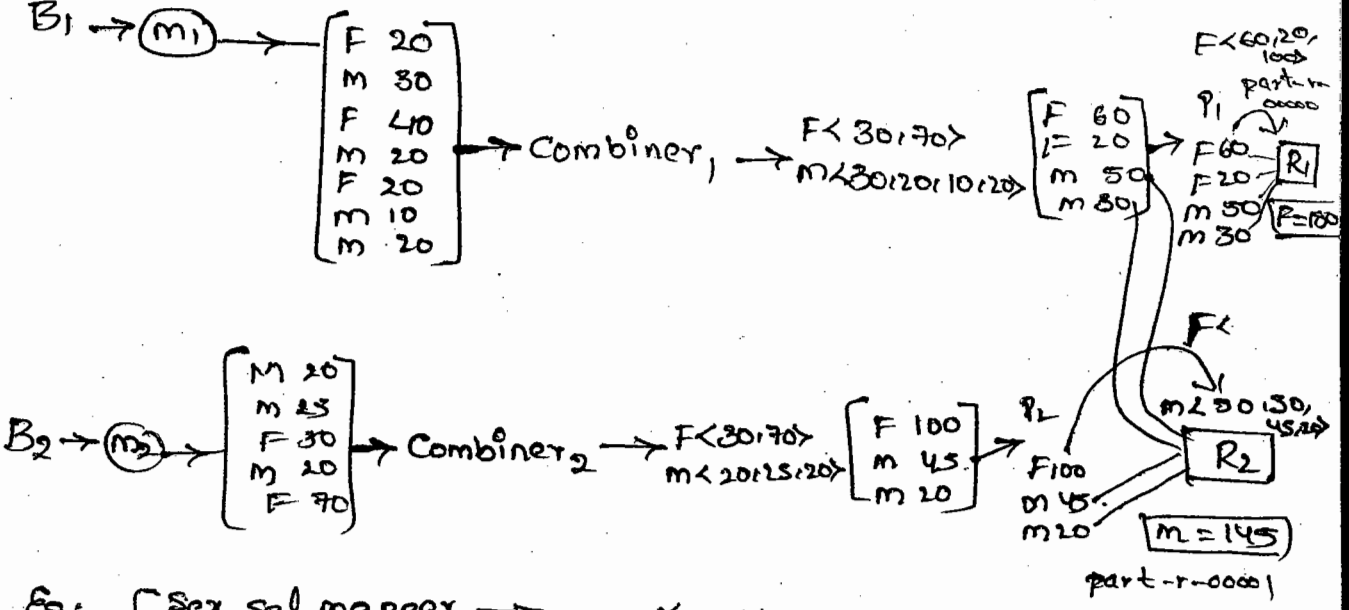
* is a mini Reducer to decrease load of reducers.

Combiners are executed at mapper level.

NO. of combiners = NO. of mapper =

mapper → Combiners → partitioner → Reducer

Same machine



Eg:-

- Sex sal mapper → K V
- Combiner for sum → (Sex, Sal)
- Red for sum → F <10,70,40,50,60>
- Driver Demo

M<

j.setcombiner class (Combiner for sum.class);

mapper 10 → combiner 6 → Reducer → 6

mapper O/P records | combiner O/P records | Reducer O/P records

NOTE:- Combiner should not be used in following situations

1) Small volumes of data
2) more keys but less frequency of each key

No Reducer
→ No combiner
→ No partitioner

Rule:- Combiners should be used for Associative operations

A+B = B+A

A*B = B*A

Combiner ForSum

public combiner forsum extends Reducer< IntWritable, IntWritable, . . . -->

```
{
public void reduce ( Text K, Iterable < IntWritable> vlist,
                                    Context con)
          throws IOException InterruptedException
{
  int total =0  coont = 0;
  For (IntWritable v: vlist)
       {
       coont ++;
       total + =v.get ();
       if (coont = =2)
    {
    con. write (K, new IntWritable (total);
            total = 0;
              coont =0;
            }
    if (coont = =1)
    {
      con.write (K, new IntWritable (total));
      }
   }
}
```

30/8/16

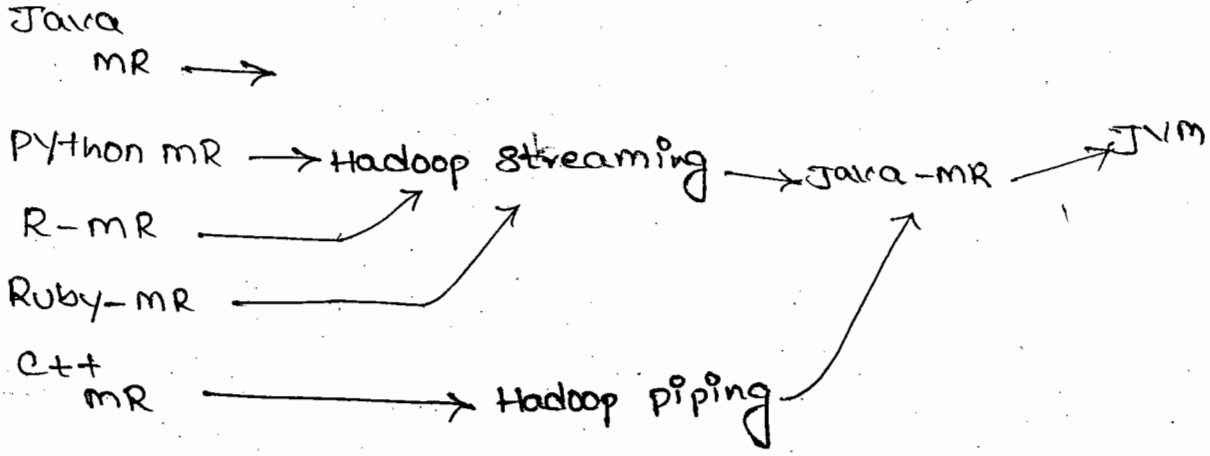
st, MapReduce API is Available for following Languages

- Python
- C++
- Ruby
- R-lang

But Hadoop runs on JVM. How to execute NON-Java MapReduce.

i) hadoop Streaming

ii) hadoop piping



* Java MR classes :-

- org.apache.hadoop.mapReduce.Mapper
- org.apache.hadoop.mapReduce.Reducer
- org.apache.hadoop.mapReduce.Job

MR data types also called "writables"

- org.apache.hadoop.io.IntWritable
- org.apache.hadoop.io.LongWritable
- org.apache.hadoop.io.Text
- org.apache.hadoop.io.BooleanWritable
- org.apache.hadoop.io.DoubleWritable

- org.apache.hadoop.mapReduce.lib.input.FileInputFormat;
- org.apache.hadoop.mapReduce.lib.output.FileOutputFormat;
- org.apache.hadoop.mapReduce.lib.input.MultipleInputs;

- org.apache.hadoop.util.GenericOptionsParser;
- fs path

- conf : configurations;
- file cache . Distributed cache;

org.apache.hadoop.mapReduce.lib.input.Text Input Format.
SequenceInput Format

mapReduce uses special datatypes called writables.
This writables are Serializable Format with hadoop distributed network.

Serializable (obj → byte) byte → Obj (DeSerializable)
mapper class inbuilt with Serializable Network Socket programming and multithreaded.

* Difference b/w Java Types & Writables :-

Java Types	Writables
* Not compatitable with HDFS & not seralizable	* Compatitable with HDFS
Int	Int Writable
long	Long writable
String	Text
boolean	Boolean writable

import org.apache.hadoop.mapreduce.mapper

3 functions

* Setup ( ) — 1 time before map ( )
* map ( ) — N No. of rows in file block
* cleanup ( ) — 1 time after map ( )

* Context is used to hold configuration into & to write results into Disk.

* Writable Serializable format with HDFS Network

In Hadoop Job process data is transporting from one slave to another slave. During this transportation data travels in "bytestream", when another task is going to take input of bytestream & captures into objects (map objects <K, VS) serialization to be done.

Same process while writting into disks (HDFS) object formats to be deserialized into "byte-streams". For all this Serialization & Deserialization developers need to code for them, which makes development process lengthy & complicated, so developers needs to concentrate only on "Business logics".

* IO Exception → Data types mismatches
* Interrupted Exception → when writting data to disk } Local (HDFS)

Job:

Job class will pass HDFS Input file configuration to "context" variable // Job class Driver //

Job j = new Job (new configuration ( ), "Batch Driver");

j.setMapper class (xxx);
j.setReducer class (xxx);
j.setcombiner class (xxx);

↓ context ref available
context is separately available for both mapper & reducer.

* Lifecycle of Mapper:

(1) Record Reader's initiation
↳ Converts file record into <K,V> format (deserialization)
offset no. — line
(2) setup( ) — initial process before map( )
(3) map( ) — Transformations & write <key, value>
(4) clean up( ) —
(5) Record writer initiations → converts <K,V> objects into byte stream (serialization)

Developers Life cycle:

* Input file —fetches data→ Mapper → holds values into writable
↓
Convert writables into Java types
↓
Apply Business logic for Required
↓
convert <K,V> results into writables (MR Types)
↓
Write Results into disk

But Reducer takes input from mapper outputs, after data is partitioned.

* Reducer Life cycle:

Mapper <K,V> O/P → Partitioned O/P <K,V>
↓
holds <K,V> in writables
↙ key should be single — should be Iterable ↘
↓
Convert only Value into Java Type
↓
Perform final Result (Aggregation)
↓
Convert Result variables into writables
↓
Write Results into disk

Assignment:-

Branch1, F, 20000
Branch1, M, 30000
Branch2, F, 30000
Branch2, M, 40000
⋮
n

} - multiple aggregation

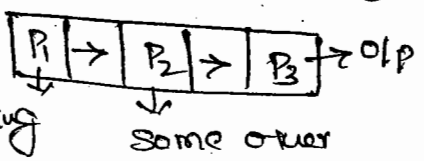
PIG

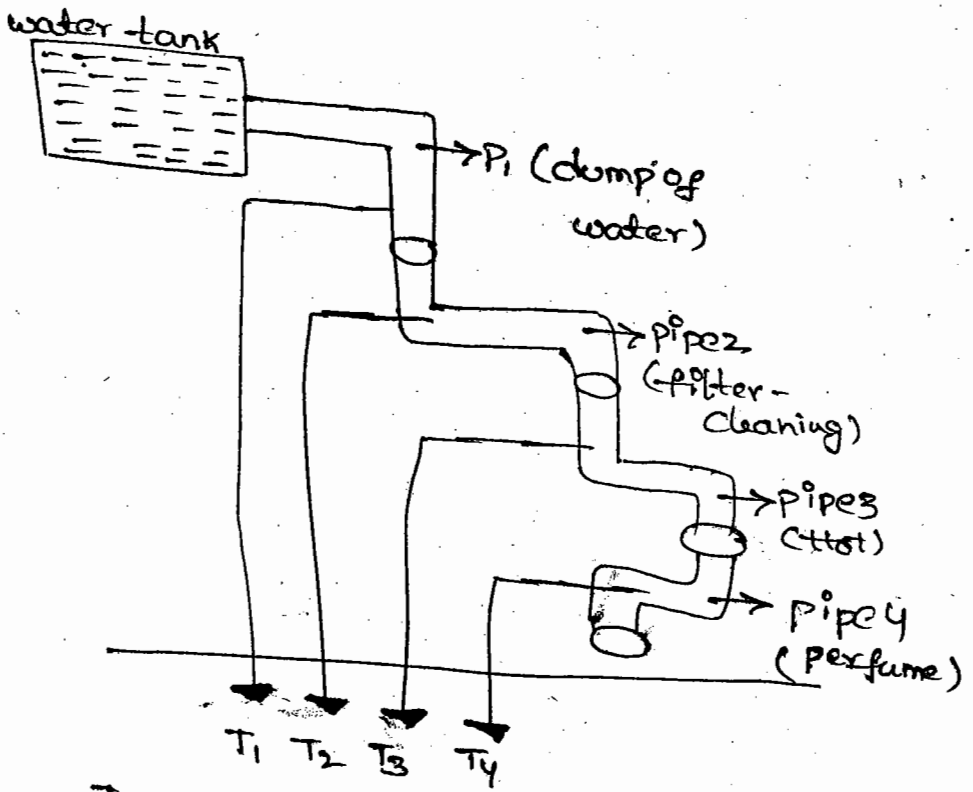
* **PIG** :- pig is a data flow language in hadoop framework.
* **Dataflow** :- collections of pipes.
  ↳ piping language
* **pipe** :- pipe is an operation (means data related operation).

→ This operation can be loading data, transformation, sorting, grouping, filtering, aggregating etc.

→ From one to another pipe data is transformed from one state to another state (transformation)

Ex :-





In above water flow once pipe4 is tigger by opening T4, the flow executes from P1 to P4. from one pipe to another pipe water is getting transformed.

Ex:- emp (ecode, name, sal, sex, dno)

101, Amar, 20000, M, 11
102, Amala, 80000, F, 12
103, Ankit, 40000, M, 12
104, Ankita, 50000, F, 11
105, Anuz, 60000, M, 12

(P1)

M, 20000
F, 30000
M, 40000
F, 50000
M, 60000

(F{(F, 30000), (F, 50000)})
(M{(M, 20000), (M, 40000)})

(P4) (sum)
(F, 80000)
(M, 120000)

(P5) (count)
(F, 2)
(M, 3)

(P6) (max)
(F, 50000)
(M, 60000)

Ex:- grunt > emp = load "emp.txt" using pig storage(",")
as (ecode:int, name: chararray) sal:int, sex: chararray
; dno:id);

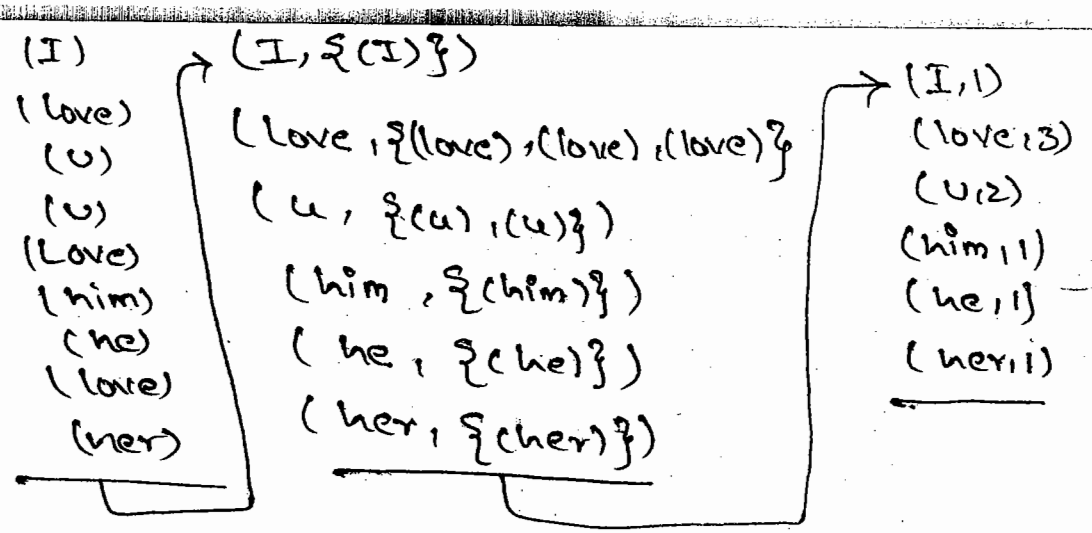
grunt > e = foreach emp generate sex, sal;

grunt > grp = group e by sex;

grunt > ressum = foreach grp generate group as
sex, sum(e.sal);

Ex:- Unstructured

I Love U
U Love him
he love her



Ex:- **WORD COUNT**

```
grunt > Lines = load 'comment' as (line : chararray);
      > words = foreach lines generate FLATTEN
                    (TOKENIZE (lines))
                    as words
      > grp = group words by word;
      wordcount res = Foreach grp generate group
                    as work count(words) as cnt;
      > Store wordcount res into " Test res1"
                                  ↳ HDFS directory
      > Line = " I Love U U love me"
```

Tokenize (line)

↳ {(I)(love)(U)(U)(love)(her)}

(",")

chararray

FLATEN (TOKENIZE (line))

↳ (I)
(love)
(U)
(U)
(Love)
(me)

as

(I,{(I)})
(love,{(Love)(love)}
(U,{(U),(U)})
(her,{(her)})

↳ (I,1)
(Love,2)
(U,2)
(her,1)

Ex:- emp.txt

(101, AA, 2000, m, 11)
(102, BB, 30000, F, 12)
(103, CCC, 40000, m, 12)
(104, DD, 50000, F, 11)
(105, EEEE, 60000, m, 12)

Query:- Select sex, sum(sal) from emp group by sex.

```
> emp = load 'emp.txt' using pig storage (',') as
        ecode :int, name :chararray, sal:in,
        sex :chararray, dno:int);

grp= group emp by sex;

f, {(102, BB, 3000, F, 12), (104, DD, 50000, F, 11)}
M, {(101, AA, 20000, M, 11), (103, CCC, 40000, m, 12),
    (105, EEEE, 60000, m, 12)}

Res = Foreach grp generate group as sex,
      sum (emp.sal) as tot;
```

O/p:-
(F, 80000)
(M, 120000)

* pig Terminology:-
  - → Relation
  - → Bag
  - → Tuple
  - → fields

* Field :-

→ Field is a data entity.

→ It is a place of data record.

* Sql Terminology = column

Ex:- ecode, name, sal, age, city

* Tuple:-

→ collection of field values formed as a tuple.

→ Tuple is represented by '()'

Ex:- (Ravi, 25, m)

(Rani, 30, F) → Tuple

field

* Bag:- Collection of tuples formed as a bag.

→ These are two types.

* Inner bag
* outer bag

outer bag:- Collection of all tuples of a data set is called outerbag.

→ outer bag is a referenced by a relation name.

Inner bag:-

A bag placed as a field, then the bag is called inner bag.

Ex:- (emp) ← Relation

Referenced by

(101, Amar, 20000, m)
(102 Amala, 30000, F)
(103, Ankit, 4000, m)
(104, Ankita, 50000, F)
(105, Aakash, 60000, m)

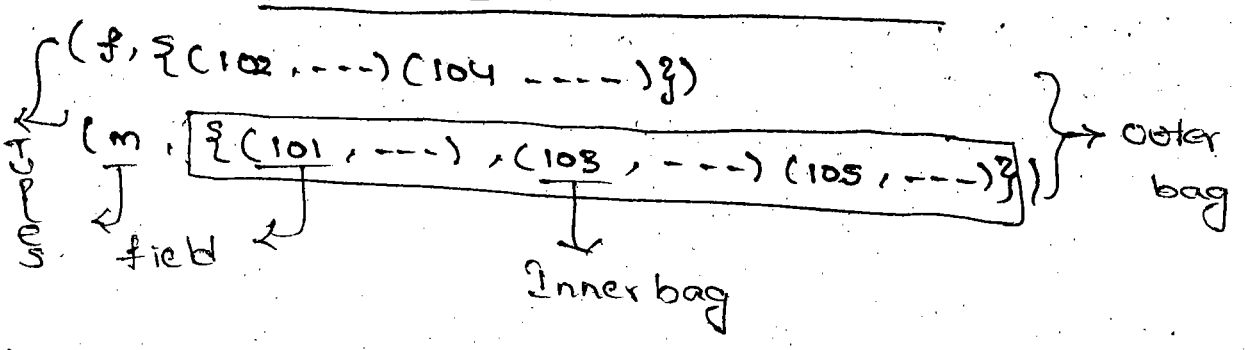
TUPLE

→ outer bag

Fields

Ex:- grp = group emp by sex;

grp → Relation



* Generally inner bags are produced when data is grouped.

Note:- pig aggregated functions should be applied on inner bags only.

Ex:- Sum / AVG / max / min / coont

Sum(sal) — invalid

Sum(e.sal) — valid

* pig Latin Statement :-

<Alias of Relation> = <operator along with Expressions;

The expressions are changing from operator to operator.

Ex:- A = Foreach B generate *;

alias of the relation — operator

* pig Startup modes (Environment) :-

* Local :- $ pig -x local ↵ (command)

pig takes i/p from local files. (operating system) and writes o/p into local.

↳ used for test Command

* HDFS:- It takes i/p from HDFS files & writes o/p in to HDFS.

→ used for production cluster., → which is default.

Command:- $.pig ↵

* pig data types:-

simple data types:
* Bytearray
* Chararray
* int
* long
* Double
* Boolean

complex data types:
* Tuple
* Bag

Ex:- (Ravi, 30, Hyd, (Rani, 26, SE) {(Sujan, 21, m)(Sagi, 4)}

Ravi, 30, Hyd → chararray, int, chararray
(Rani, 26, SE) → Tuple
{(Sujan, 21, m)(Sagi, 4)} → Bag

* pig operators:-

* Load * Store * Dump * Describe * illustrate
* Limit * Sample * filter * Union * foreach * Group
* Co Group * join * Left Order join * Right out join * Cross
* Fullouter join * pig * Register * Distinct * Exec
* Define * RUN * Order

NOTE:- Comparing hive/pig which is best?

→ Structured data can be handled by both hive and pig. But the data flow length is very high pig is best.

Bcoz entire data flow is created as single mapreduce job.

→ where in hive foreach select statement are map-Reduce job tigger.

Ex:- Create directory in pig

$ hadoop fs -mkdir pigLatin
$ cat emp

> emp = Load 'pigLatin/emp' as (ecode:int, name:chararray, sal:int, sex:chararray, dno:int);

O/P

Ex:- $ cat > samp

```
100  200  300
10   20   30
1    2    3
11   22   33
```

$ hadoop fs -copyFromLocal samp pigLatin

> dump > S = load 'pigLatin/samp' as (a:int, b:int, c:int);

> S = Load 'pigLatin/samp' using pigStorage ('\t') as (a:int, b:int, c:int);

> S = Load 'pigLatin/samp' using pigStorage() as (a:int, b:int, c:int)

> dump samp

NOTE:- ① pig storage :- pipe storage

bin storage :- compress storage

② Default storage in pig, pig storage

> delimitor ('\t')

↳ zero tabs the entairline treated as field.

Ex:- cat > emp

```
10000, 2000, 30000
1000, 300, 500
10, 39, 40
1, 3, 5
```

hadoop fs -copyFromLocal damp pigLatine

> d = Load 'pigLatin/damp' as (a:int, b:int, c:int)

> dump damp

* LOAD:- This is load data from file to pig relation

NOTE:- Load is logical

↳ at the time of excuting flow data is streamed from file.

* File :-

pig

Comm

Hdoo

$ pig

$ pig

Synta

No li

Ex:- $

$ ha

$ h

$ pi

> -- p

> --

> pig

> Bin

> Def

> --

> S1

> S2

* File :- Stored depends on Local / HDFS.

→ The File can be local or HDFS depend on pig start up mode.

COMMAND :- `$ pig -x Local` → Local

`$ pig ↵` → HDFS

How to Submit pig :-

`$ pig pScript1.pig ↵`

`$ pig pscipt1.pig -x Local`

Syntax :- <alias of Relation> = Load '<File path>' [using storage Method ( ) >] as (field1:datatype, file2 : datatype , --- fieldn : datatype);

No limit for max no. of fields.

Ex :- $ cat > Samp

100 200 300 (\t)

300 400 450

600 700 800

$ hadoop fs - mkdir pigLatin

$ hadoop fs -copyfrom Local Samp pigLatin

$ pig ↵

>-- pigLoad two Storage methods

>-- 1) pig Storage ( ) (2) Bin Storage ( )

> pig Storage → Text Input Format (Byte modes)

> Bin storage → Sequence Input Formode (Binary files)

> Default storage method is pig Storage ( )

>-- for the pig storage ( ) default delimitor is ('\t') (tabal space)

> s1 = Load 'pig Latin / samp' using pigstorage ('\t') as (a:int , b = int , c: int);

> s2 = Load ' pigLatin / samp' using pigstorage ( )

as (a:int, b:int, c:int);

S_3 = Load 'pigLatin/samp' as (a:int, b:int, c:int)

O/P:- of S_1, S_2, S_3 is same for pigStorage() of S_2, '\t' is applied as default.

For S_3 automatically pigStorage('\t') is applied as default storage method.

Ex:- Samp

100 200 300

300 400 450

> S_4 = Load 'pigLatine/samp' as (a:int, b:int);

> dump S_4;

(It takes first two fields of the file)

> S_5 = Load 'pigLatin/samp' as (a:int, b:int, c:int, d:int);

> dump S_5

O/P:- (100, 200, 300, null)
(a, b, c, d)

* Comma(,) is a delimitor :-

> emp = Load 'pigLatin/emp' using pigStorage (',') as (ecode:int, name:chararray, sal:int, sex=chararray, dno:int);

> dump emp

* dump :-

→ To execute pig's data flow. It executes, entire flow is (converted as a MR job) and writes o/p onto console.

O/P:-

> S_4

> l

> d

> cat

How t

> co

F
m

* I

> st

> ls

* Dif

dump

Storag

Syntax

Ex:

emp → e

> e = foreach emp generate sex, sal;

> grp = group e by sex;

> res = foreach grp generate group as sex,
Sum(e.sal) as tot
↳ bag applying

> dump res

O/P:- (f, 283000)
(M, 255000)

> store res into 'pigLatin/res1';
↳ O/P directory.

> ls pigLatin/res1
part-r-00000

> ls pigLatin/res2
part-m-00000

> cat pigLatin/res2/part-m-00000

How to check O/P :-

> cat pigLatin/res1/part-r-00000

F 283000
M 255000

* I want ',' delimitor;

> store res into 'pigLatin/r1' using PigStorage(',');

> ls pigLatin/r1

* Difference between dump & storage?

dump :- executes, O/p writes in console

storage :- executes, O/p writes in disk or Local.

Syntax :-

> dump <alias of Relation)

Ex :- > dump emp ↵

Ex:- > cat piglatin/emp

101, Amar, 2000, m, 11

⋮

emp = Load 'piglatin/emp' using pigStorage(',') as
(ecode:int, name:chararray, sal:int,
sex:chararray, dno:int);

e = foreach emp generate sex, sal;

grp = group by sex;

res = foreach grp generate group as
sex, Sum(e.sal) at tot;

dump res ↵

→ res - grp - e - emp,
the flow executes from emp to res, and writes o/p onto console.

* STORE:-

It executes flow, writes o/p into disk (files). The file can be local or HDFS. depends on startup mode.

Ex:- > store res into 'piglatin/res1';
new directory ↵

> ls piglatin/res1 ↵

> /user/training/piglatin/res1/part-r-00000

for store operator also by default pigstorage ('\t') is applied so tabspace is delimitor in o/p file.

> Store res into 'pigLatin/res2' using PigStorage (',');

* Describe:- To get schema (structure) of relation.

→ only that particular hirache structure).

> describe emp

Ex:- emp → e → ⊗ → ② → ②

emp :

grunt> describe e

* ILLUSTRATE:-

> illustrate result

→ It gives entire Hiraraohy of the flow

→ Schema of each flow, level.

→ Sample data at each flow used for debugging.
↳ not entaire data

Ex:- Illustrate pipe_25↵

NOTE:- Illustrate not triggered mapreduces.

* FOREach:-

* To copy tuples from one Relation (ds) to another Relation (dataset).

* To Filter fields (setting fields)

* to Generate new fields

* to transformations.

* to change datatypes

* to Rename fields

* to change schema order

Ex:- > emp = load 'pigLatin/emp' using pigStorage (',') as (ecode: int, name: chararry, sal: int, sex: chararry, dno: int)

* How to copy one relation to another relation :-

> e = foreach emp generate * ;

> -- selecting fields

> e_2 = foreach emp generate name, sal, dno;

> describe e_2

e_2: { name : chararray , sal : int , dno : int }

> e_3 = foreach e_3 generate *, sal + hra - tax as net;

NOTE:- New field alias cannot be reused same statement, following statement is invalid.

X = foreach emp generate name, sal, sal * 0.1 as tax, sal - tax as net;

above statement net is invalid expression.

* Transformations (condition transformations) :-

> cat pig Latin / xyz

100, 300
3000, 2000
123, 456
234, 120

> ds = Load 'pigLatin/xyz' using pigStorage (',') as (a:int, b:int);

> ds = foreach ds generate *, (a>b ? a:b) as big;

? → tenary operator conditional operator is used to perform conditional transformation.

* nested conditional operator :-

> -- nested conditional operator

$ cat >test

100,230,200

125,956,890

$ hadoop fs -copyFromLocal

> d = Load 'piglatin/test' using pigstorage (',') as (a:int, b:int, c:int);

> res = foreach d generate *;

a>b?(a>c? a:c) :(b>c ? b:c)) as biggest;

Ex:- e = foreach emp generate ecode, name, sal,

(sal >= 70000 ? 'A' : (sal >= 50000 ? 'B' :

(sal >= 30000 ? 'C' : 'D'))) as grade,

(sex == 'F' ? 'Female' : 'male') as sex,

(dno == 11 ? 'marketing' :

(dno == 12 ? 'Hr' :

(dno == 13 ? 'Finance' : 'others'))) as dname;

Checking output:-

> store e into 'piglatin/r10' using pigStorage(',')

> cat piglatin/r10/part-m-00000 ↵

* How to clean nulls:-

> -- working with nulls

$ cat >test2

10,40,50

100,800,

300,,500,

400,600

,400,600

100,,

,,400

100,500,600

→ * nulls can not be arthamatic operation.

→ * convert all nulls are zero.

$ hadoop fs -copyFromLocal testz pigLatin

grunt > cat pigLatin/testz

> ds = Load 'pigLatin/testz' using pigStorage

> ds2 = Foreach ds generate *, a+b+c as d;

> ds3 = Foreach ds generate

(a is null ? 0:a) as a,

(b is null ? 0:b) as b,

(c is null ? 0:c) as c;

> res = foreach ds generate *, a+b+c as d;

> dump result;

Ex:- getdit marksheet ↵

S1, 80,90,67,89,90,60,38,70,80,50

S2, 70,60,40,45,45,54,30,60,55,45,

S3, 45,56,49,90,40,56,80,90,45,10

S4, 55,30,56,49,90,38,67,45,49,54

Requirements :-

--- Sub1 --- Sub10

1-4, Subjects passmarks 50 --- minimum pass cnt:2

5,7, 40 -- at :2

8,10, 50 -- :1

aggregated pass count : 6

$ hadoop fs -copyFromlocal marksheet

> marks = Load 'pigLatin/marksheet, using pigStorage

(',') as (sid :chararry, m1:int, m2 :int, m3 :int,

m4:int, m5 :int, m6 :int, m7 :int, m8=int,

m9 :int, m10:int)

> ma

Check

>

>

> ma

> d

> res

> St

* Char

Ex:-

> marks2 = foreach marks generate
sid, (m1>=50 ? 1:0) as r1,
(m2>=50 ? 1:0) as r2,
(m3>=50 ? 1:0) as r3,
(m4>=50 ? 1:0) as r4;
(m5>=40 ? 1:0) as r5;
(m6>=40 ? 1:0) as r6;
(m7>=40 ? 1:0) as r7,
(m8>=50 ? (1:0) as r8,
(m9>=50 ? (:0) as r9,
(m10>=50 ? 1:0) as r10;

Checking the o/p:-

> dump marksheet

> marks3 = foreach marks2 generate sid,
r1+r2+r3+r4 as cnt1,
r5+r6+r7 as cnt2,
r8+r9+r10 as cntB;

> marks3 = foreach marks3 generate *, cnt1+cnt2+cnt3 as tot;

> dump marks3

> res = foreach marks3 generate sid, (cnt>=2 and cnt>=2 and cnt3>=1 and tot>=6 ? 'pass':'fail') as result;

> Store res into 'pigLatin/marks res, using pigStorage(',')

* Changing data types :-

emp → ecode (int) | name (chararray) | sal (long)

Ex:- emp = foreach emp generate (chararray) ecode, name, (int) sal;

> e3 = foreach e3 generate (chararry) ecode, (int) net;

> describe e3

e3: {ecode : chararry , net : int }

* Renaming :-

> x = foreach e3 generate ecode as id, net as netsal;

> describe x

x: { id: chararry , net sal : int }

* Changing the field order :- (merging to change to Statement)

> describe emp

emp : { ecode : int , name : chararry , sal: int , sex: chararry , dno : int }

> n = foreach emp generate ecode , name , sex , dno, sal;

> describe n;

n: {ecode : int, name : chararry , sex: chararry , dno: int , sal : int }

* Limit :-

To fetch top n - no. of tuples.

Ex:- Select * From Tab limit 10;
↳ limit

Ex:- e10 = Limit emp3;

by this ex first 3 raws will be get.

* I want 3 Last rows :-

1	6
2	5
3	4
4	3
5	2
6	1

* Limit fetch only top 3 rows only.

* Filter :- To seperate the rows the matched condition.

* Select * from x where x >= 200;

* x = filter emp by (sex == 'm');

If data is K-Sensitive

If I want F → 11
M → 12 & 13

Y = filter emp by ((sex == F and dno == 11) or (sex == 'M' and (dno == 12 or dno == 13)));

> emp = load 'piglatin/emp' using pigStorage (',') as (ecode : int, name : chararry, sal : int, sex : chararry, dno : int);

Accessing fields using index number;

> X = Foreach emp generate

Y = Foreach emp. generate $1 as name, $4 as deptnum;

Limit :-

> top3 = Limit emp 3;

> dump emp↵

* Sample :- To get random samples, this sampling is used for testing.

(Randomly selected by 50% of the data)

Ex :- > ss = sample emp 0.5;

> dump ss ↵

* Union :-

emp 2

101, AA, 2000, F, 11
102, BB, 30000, M, 12

emp3

301, ABA, 20000, F, 11
302, BCA, 30000, M, 12

> emp3 = load 'pigLatin/emp2' using pigStorage (',') as (ecode : int, name : chararry, sal : int, sex : chararry, dno : int);

> emp3 = Load 'PigLatine/emp3' using pigstorage (',') as (ecode : int, name : chararry, sal : int, sex : chararry, dno : int);

> emp3 = load 'piglatin/emp3' using pigstorage (',') as (ecode:int, name:chararry, sal:int, sex:chararry, dno:int);

> emp3 = foreach emp generate ecode, name, sal, sex, dno;

> e = union emp, emp2, emp3

> dump e

NOTE:- before merging schema should be same.

Ex①:- $ cat > file1

ravi, 30
rani, 40
mani, 40

$ cat > file2

siva, hyd
venu, del
giri, del

How manage missing fields in both files:-

$ hadoop fs -copyFromLocal

$ hadoop fs -copyFromLocal

> f1 = load 'piglatin/file1' using pigStorage (',') as (name:chararry, age:int);

> f2 = load 'pigLatin/file2' using PigStorage (',') as (name:chararry, city:chararry);

> fx = Foreach f1 generate name, age, 'null as city';

> fy = Foreach f2 generate name, 'null as age', city;

Now one can merge it;

> f = union fx, fy;

Check

> dump f;

O/P:- Siva, leyd
venu, del
giri, del

NOTE:- PigLatin's Union is equaliwd to union all of Sql.

Ex:- $ cat > file3

101, aa
102, bb
103, cc
104, dd
101, aa
102, bb
102, bbc

* I want eliminate duplicate rows;

$ hadoop fs- copyFromLocal files piglatine

> ds = load 'piglatin/file3' using PigStorage(',')
as (id :int, name : chararry);

> newds = distinct ds;

checking O/P;

> dump newds;

(101, aa)
(102, bb)
(102, bbc)
(103, cc)

* Distinct :- To eliminate duplicate rows (tuples)

→ If all tuples fields are matched then only tuple is treated as duplicate tuple.

→ If you want to eliminate duplicates based on key match, then take help of cdfs.

→ Performing aggregations.

* Pig aggregation functions :-

SUM, AVG, MAX, MIN, COUNT

NOTE:- pig aggregated function should be applied on inner bags.

→ So before applying aggregation data should be group. then only we get inner bags.

How to separate sex, sal :-

> e = foreach emp generate sex, sal;
> bysex = group e by sex;
> ressum = foreach bysex generate group as sex,
SUM(e.sal) as tot;
> dump ressum;

* Group :-

to group tuples into inner bag based on grouping field.

Ex:- single grouping multiple aggregation

> bysex = group e by sex;
> resall = foreach bysex generate group as sex, SUM(e.sal) as tot, AVG(e.sal) as avg, max(e.sal) as max, MIN(e.sal) as min, COUNT(e) as cnt;
> resall = foreach resall generate sex, (int)tot, (int)AVG, max, min, (int)cnt;
> dump resall;

Multi grouping :-

> e = foreach emp generate dno, sex, sal;

> by Dnosex = group e by dno, sex;

↳ (Invalid Statement)

NOTE:- pig does not allow you to group by multiple fields.

Sol:- Make multiple fields as a tuple. field group it by tuple field group it by tuple field.

> bydnosex = group e by (dno, sex);

(valid Statement)

* NO limit for no. of grouping fields

> describe bydnosex

> res = foreach bydnosex generate group.dno, group.sex, sum(e.sal) as tot;

> describe res

res: { dno: int, sex: chararry, tot: long }

> dump res

* multi grouping with multiple aggregation :-

> resall = foreach bydnosex generate group.dno, group.sex, sum(e.sal) as tot, count(e) as cnt;

* Entaire column aggregation :- (bag is required)

> e = foreach emp generate sal;

> grpall = group e all;

> describe grpall;

grpall : { group : chararry, e: { sal : int } }

> dump grpall

bt all sal formed as a one bag)

> res1 = foreach grpAll generate sum(e.sal) as tot;

> res2 = foreach grpAll generate sum (e.sal) as tot, AVG (e.sal) as avg, max (e.sal) as max, MIN (e.Sal) as min, count(e) as cnt;

> dump res2

Sum of multiple aggregations :-

> emp, emp2, emp3

> b1 = foreach emp generate 'branch 1' as branch, sal;

> b2 = foreach emp2 generate 'branch2' as branch, sal.

> b3 = foreach emp3 generate 'branch3' as branch, sal;

> b = onion b1, b2, b3

> describe b

> by Branch = group b by branch;

> res = foreach by Branch generate group as branch, sum (b.sal) as tot;

> dump res.

O/p:-

* Co-Group :- To get grouping seperate bags on each data set.

→ so that we can apply seperate aggregation on each datasets.

> cg = Cogroup emp by sex, emp2 by sex,

> res

s

>

O/P:-

Diff k

Ex:-

cg =

(A, {

(B,

(C, {

(D, {

R =

O/p:-

Ex:-②

S1 = f

S2 =

S3 =

S1

pid,

emp3 by sex ;

> res = foreach cg generate group as sex, sum (emp.sal) as tot1 , sum (emp2.sal) as tot2 , sum (emp3.sal) as tot3 ;

> dump res

O/P:-

Diff b/w Group / Co Group :-

Ex:- ds1 (K, A)

(A,10)
(B,20)
(A,25)
(B,80)
(C,70)
(A,40)

ds2 (K, V)

(A,15)
(B,25)
(A,40)
(D,70)
(D,90)

cg = cogroup ds1 by K , ds2 by K ;

(A, {(A,10)(A,25)(A,40)}, {(A,15)(A,40)}

(B, {(B,20)(B,20)}, {(B,25)})

(C, {(C,70)}, {})

(D, {}, {(D,70),(D,90)})

R = foreach cg generate group as K, COUNT (ds1) as ds1 , COUNT (ds2) as cnt2 ;

O/P:- (A,3,2)
(B,2,1)
(C,1,0)
(D,0,2)

Ex:-② Sals1 Sals2 Sals3

S1 = foreach sal1 generate pid , price * qnt as bill ;

S2 = foreach sal2 generate pid , price * qnt as bill.

S3 = foreach sales3 generate pid , price * qnt as bill ;

S1	S2	S3
pid, bill	—	—

```
Cg = Cogroup  s1 by pid;
              s2 by pid;
              s3 by pid;

R = foreach cg generate group as pid, sum (s1.bill) as totg
    Sum (s2.bill) as tot ; Sum (s3.bill) as tot;
```

* **Joins:-** joins are used collect inform action from two are more data sets.

* joins are basically two types
  - ** Inner joins
  - * Outer joins

  → As per Sql each join again two types

* **Equi-join :-**

In join condition " equals to" (=) is operator

Ex:- 11-11 , 12-12 , 18-18 --- etc

* **Non- equi-join :-**

In non- equi other than '=', >, <, >=, <=, != are operators in join condition.

11-13, 13-18, 18-12

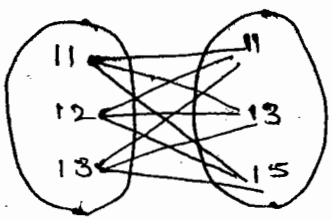
Ex:- For matrimony application males dataset to be compared with females dataset.

Ex:- male age to be compared with female age.

NOTE:- Both pig and hive ecosystems don't support not equi joins.

To get non equi functionality cross joins (Cross gives you cartesian product) used in pig.

Ex:-

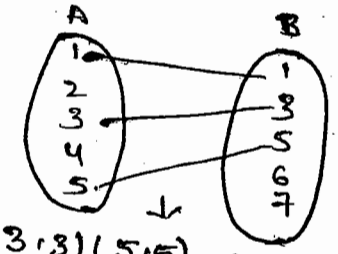


3 × 3 = 9

* **Inner Joins (inner & equi):-**

only matched tuples will be joined

Ex:-



(1,1) (3,3) (5,5) → only matching tuples.

* **Outer Joins:-**

will be matching & non-matching tuples.

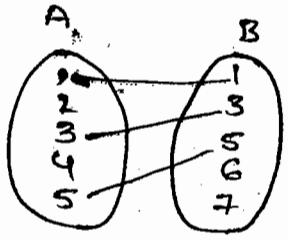
→ Outer joins are 3types:-

* Left outer join
* Right outer join
* Full outer join

* **LOJ:-** All matching + non matchings of left side

→ complete presence from left side dataset

Ex:-

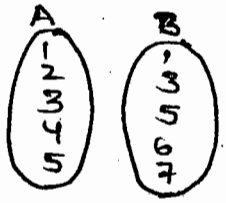


(1,1) (3,3) (5,5) → matching

(2,) (4,) → non matching from left

* **ROJ:-** All matching + non matchings from Right side

→ complete presence from Right side dataset.



(1,1) (3,3) (5,5) → matchings

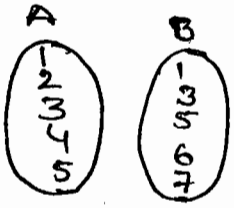
(,6)
(,7) } non-matchings from right.

* **Full outer joins:-**

→ matchings + non matchings of left & Right side

→ complete presence from both left & Right datasets.

Ex:-



(1,1) (3,3) (5,5) (2,) (4,) (,6) (,7)

Mathamatical rep :-

Inner join = $A \cap B$

Left outer join = $(A \cap B) + (A - B)$

Right outer join = $(A \cap B) + (B - A)$

FOJ = $(A \cap B) + (A - B) + (B - A)$

emp :-

(101, AA, 10000, 11)
(102, BB, 20000, 12)
(103, CC, 80000, 11)
(104, DD, 40000, 18)
(105, EE, 50000, 14)

dept :-

(11, marketing, Hyd)
(12, HR, Del)
(18, Finance, Hyd)
(17, accounts, Del)

J

(101, AA, 100000, 11, 11, marketing, HYD)
(102, BB, 200000, 12, 12, HR, Del)
(103, CC, 80000, 11, 11, marketing, HYD)
(104, DD, 40000, 18, 18, finance, HYD)
(105, EE, 50000, 50000, , ,)
(, , , , 17, Accounts, Del)

IJ = ①, LJ = ① + ②, RJ = ① + ③, FOJ = ① + ② + ③

NOTE :- To join two data sets at least one common meaning field should be existed.

dno	dept no
11	11
12	12

Ex :- IJ = join emp by dno, dept by dno;

LOJ = join emp by dno Left outer, dept by dno;

ROJ = join emp by dno Right outer, dept by dno;

FOJ = join emp by dno full outer join, dept by dno;

NOTE:- when join is applied

n (resultant ds) = δ(left ds) + n (right ds)

where

n = no. of fields

emp → 5 fields } 8 fields
dept → 3 fields

Schema:-

emp → ecode, name, sal, sex dno

dept → dno, dname, dloc

IJ = join emp by dno, dept by dno;

↓

Describe IJ

↳ {emp::ecode :int, emp::name :chararry,
emp::sal :int, dept::dno :int, dept::dname :
chararry,
dept::dloc :chararry }
↳ Ambigeror

NOTE:- Aggregated functions can not be applied on fields which are reference by other Relations.

Ex:- Sum (e.emp::sal) , thats why before grouping
invalid
eliminate reference operator.

Ex:- IJ = join emp by dno, dept by dno;

(I am going to removing reference)

nIJ = foreach IJ generate dept::dloc as city, emp::sal as sl;

Grouping applying:-

bycity = group nij by city;

Res = foreach by city generate group as city,

SUM (nij.sal) as tot;
(valid)

Eg:- > dept = load 'piglatin / dept " using pigstorage (',') as (dno: int, dname: chararry, dloc: chararry);

Grouping applying joins:-

edij = join emp by dno, dept by dno;

edlj = join emp by dno leftouter, dept by dno;

edfj = join emp by dno full outer, dept by dno;

testing purpose

> dump edfj

ed2 = foreach ed generate ::dloc as city,
emp::sal as sal;

applying grouping:-

grp = grp ed2 by city;

res = foreach grp generate group as city,
Sum (ed2.sal) as tot;

> store res into 'piglatin/x1';

Check the result

> cat piglatin/x1/part-r-00000

O/p:- del -118000
hyd -20000

> getid dept
11, mrkt, hyd, m1
12, fin, del, m2
13, ftr, hyd, m1,
21, acc, del, m2,
22, admin, hyd, m1

> dept = load 'pig Latin / depart' using pigstorage (',')
as (dno: int, dename: chararry, dloc: chararry,
mid: chararry);

storage

loc:

mry');

dno;

0;

xity,

e(',')

```
> mngrs = load '
> ed = join emp by dno, dept by dno;
> ed2 = foreach ed generate dept::mid as mid,
        emp::sal as sal;
> edm = join ed2 by mid, mngrs by mid;
> einfo = foreach edm generate mngrs::mname as
        mname, ed2::sal as sal;
> grp = group einfo by mname;
> res = foreach grp generate group as mname,
    SUM(einfo.sal) as tot, COUNT(einfo) as cnt;
> Store res into 'pigLatin/x2';
> res = foreach grp generate group as mname,
    SUM(einfo.sal) as tot, COUNT(einfo) as cnt;
```

Check o/p:- > cat pigLatin/x2/part-r-00000

```
eds = foreach edfull generate (emp::dno as dno1,
        dept::dno as dno2 emp::sal as sal;
> einfo = foreach eds generate
        (dno1 is not null and dno2 is not null?
        'working': (dno2 is null ? 'bench Team':
                'Bench project')) as stat, sal;
> dump einfo
```

> grp = group einfo by stat;

> res = foreach grp generate group as stat, sum(einfo.sal) as tot, COUNT(einfo) as cnt;

> res = foreach res generate stat, (tot is null? 0:tot) as tot, cnt;

* **Cross:-** Cross is used to get cartesian product.

Ex:- ds1 (1,a) (2,b) (3,c) ds2 (1,x) (2,y)

cr = cross ds1, ds2;

(1,a,1,x) (1,a,2,y) (2,b,1,x) (2,b,2,y) (3,c,1,x) (3,c,2,y)

Use case(1):- To make availe of aggregated results to each row

e = foreach emp generate sal;

grp = group e sal;

avgds = foreach avgds generate (int) avg;

cr = cross e, avgds;

> cr = foreach cr generate e::sal as sal, avgds :: avg as

> cr2 = foreach cr generate (sal >= avg ? 'above' : 'below') as stat, sal;

> grp = group cr2 by stat;

> res = foreach grp generate group as stat, COUNT(cr2) as cnt;

Assgr

Ex:-

> Sc

> S

> S

> gr

> re

> re

> c

> c

> cr

> re

> c

Assignment:- I want total occupancy

10000 5%
90000 45%
100000 100% } $\left(\frac{Sal}{tot} \times 100\right)$

Ex:- Sales

01/01/2011, 5000
02/04/2011, 90000
05/06/2011, 60000
06/01/2011, 80000
07/06/2011, 40000

> Sales = load 'piglatin/sales' using pigstorage (',') as (dt: chararry > amt: int);

> S = foreach sales generate SUBSTRING(dt, 0, 2) as mon, amt

> S = foreach S generate (int) mon, amt;

> grp = group S by mon;

> report = foreach grp generate group as mon, Sum(a.amt) as tot;

> report2 = foreach report generate *;

> Cr = Cross report, report2

> cr = foreach cr generate report::mon as m1, report2::mon as m2, report::tot as t1, report2::tot as t2;

> cr2 = filter cr by (m1-m2==1);

> res = foreach cr2 generate *, ((t1-t2)*100)/t2 as change;

> dump res

Use Case ②:- with the reference to above example diff rows after same data can be compared.

* Order :- used to sort data in ascending or descending order.

alias = order < relation > by < field1, field2, ---fieldN > [sort mode];

Here

Sort mode: can be ascending or descending default Sort mode is ascending.

Ex:- grunt > e1 = order emp by name;

Ex:- > e2 = order emp by sal desc;

Ex:- > e3 = order emp by sal desc, sex, dno desc; (multi Sort)

NOTE:- No limit of max no.of Sort fields.

Ex:- sales (schema) --> trid, prid, amt, qnt, city

sales = load 'pigLatin / sales' using pigStorage(',') as (trid : chararry, prid : chararry, amt : int, qnt : int, city : chararry);

bycity = group sales by city;

SalesRep = foreach bycity generate group as city, Sum (sales.amt * sales.qnt) as tot;

OrderRep = order SalesRep by tot desc;

top3 sales = limit OrderRep 3;

→ in above example if two cities have same sales

volume you will miss third (top) rank city.

Sol:①:- develop udf which generates, mathRank (equal score should get equal Rank)

Sol②:- take help of distinct and join (Bad way).

Ex:-
```
h  89000  1
d  89000  1
m  89000  1
c  30000  2
m  30000  2
b  20000  3
mn 10000  4
```

Ex:- $ cat > temp
```
Amar   90000
Amala  70000
Kiran  90000
Kiran  7000
```
hadoop fs -copyFromLocal

```
> info = load 'piglatin/temp' pigStorage(',') as
        (name:chararry, sal:int)
> info2 = order info by sal desc;
> top3 = limit info2 3;
> --- select * from info order by sal desc limit 3;
> dump top3;
```
(∵ this is correct for each one getting equal sal)

==o==

```
> sals = foreach info generate sal;
> sals = distinct sals;
> sals2 = order sals by sal desc;
> top3sals = limit sals2 3;
> rep = join info by sal, top3sals by sal;
> report = foreach rep generate
```
> dump report

I want eliminate duplicate records:-

```
101 , 10000
102 , 30000
101 , 10000
102 , 30000
101 , 10000
102 , 230000
```

x = distinct info ;

```
101 , 10000
102 , 30000
102 , 23000
```

=o=

Based on particular key match

first sort data;

```
101 , 10000   1
101 , 10000   2
101 , 10000   3
102 , 300000  1
102 , 30000   2
102 , 230000  1
```

} group count (udfs used)

can't apply filter.

Distinct:- to eliminate duplicate tuples based entire row match.

→ to eliminate duplicate rows based on some field.

take udf help which generate "group count"

→ pig udfs can be written in java, python, C++, ruby, javaScript, perl.

Script executions:-

we have 3 operators;

---→ pig, exec, run

* pig:- To submit script from command prompt.

'$' pig script

→ relation alias are not available grunt shell.

→ $ pig script1.pig

* exec:- To submit script from grunt shell.

→ relation alias are not available in grunt shell. so that you can not reuse.

grunt > exec script1.pig

* **RUN**:- to submit Script from grunt shell relation alias are available in grunt.

So that you can reuse.

NOTE:-

* To call pig Scripts in linux based shell script pig operator is used.
* To executes scripts from grunt and not to over the existed relation. exec is based.
* To reuse repeatedly used relations run operator is used.

(:: These alias are Independent)

exec / run ? exec / pig → to execute alias not available, every command is available in

↓ ↓

I want reuse If u submit shell script

Ex:- emp = load '' piglatin/emp' using pigStorage (',') as (ecode:int, name:chararry, sal:int, sex:chararray, dno:int);

esex = foreach emp generate sex, sal;

edno = foreach emp generate dno, sal;

ednosex = foreach emp generate dno, sex, sal;

bysex = group esex by sex;

bydno = group edno by dno;

byDNOsex = group edosex by (DNO, sex);

ressex = foreach bysex generate group as Sex,

Sum (esex.sal) as tot;

resDNO = foreach byDNO generate group as dNo, sum(edno.sal) as tot;

resDNOsex = foreach byDNO generate group as dno, sum(edno.sal) as tot;

resDNOsex = foreach byDNOsex generate group.dno, group.sex, sum(eDNOsex.sal) as tot;

* Pig UDFS :-

→ pig Custom functionalities are developed by UDFS.

→ pig UDFs Can be written in

* Java
* C++
* python
* Ruby
* javaScript
* perl

* Java based UDFS :-

Step①:- Develop UDFs class:-

```
package pig.UDFS;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import java.io.IOException;

public class concate extends EvalFunc<String>
{
public String exec(Tuple v) throws IOException
 {
String fname = (String) v.get(0);
String lname = (String) v.get(1);
```

Step②

Step③

Step④

Step⑤

* How-

NOTE:-

Ex:-

* UDF

S1 →

S2 →

S3 →

S4 →

S5 →

```
        String name = fname + "-" + lname;
            return name;
        }
    }
```

Step②:- Export into jar file.

ex:- /home/training/Desktop/myudfs.jar

Step③:- Register external jar file into pig

> REGISTER Desktop/myudfs.jar;

Step④:- Create a temporary function for udfs class.

grunt > Define Conpigudfs.concat();
(function name ↙) (↘ udf class)

Step⑤:- Calling function

Once a function is defind through out the session any no. of time the function can be called.

* How to call?

NOTE:- functions can only be called from FOREACH operator.

Ex:- profiles

fname	lname
(Ravi,	Kumar)
(Swetha,	Rani)
!	!

N = foreach profiles generate can (fname, lname) as name;

* UDFs life cycle:-

S_1 → Develop udf class → java class eclipse

S_2 → Export into jar file → eclipse

S_3 → Register jar file in pig → REGISTER at grunt

S_4 → Create or Define tempory function → Define at grunt

S_5 → calling the function → foreach at grunt.

user/lib/pig/pig - core.jar.

Ex:- register Desktop / pig udfs.jar;

define rowbig pig.analytics.Rowmax();

res = foreach S generate *, rowbig (*) as max;

Ex:- import org.apache.pig.Evalfunc;

import org.apache.pig.data.tuple;

import java.util.list;

public class Rmax extends Evalfunc <Integer>

{

public Integer exec (Tuple v) throws IOException

{

list <object> lobs = v.get All ();

int max = 0;

for (object o: lobs) * v.get

{

int val = (Integer) o;

max = math.max

}

return

}

}

① register Desktop / pigudfs.jar;

② define rmax pig.analytics.Rmax();

③ r = foreach f generate *, rmax (*) as max;

④ dump r

Unstructured data :-

```
$ cat > news
hadoop use cases are more
```

datawarehouse house is to store historical data in warehouse hadoop plays major role.

Teradata and netezza are compitators to hadoop

```
$ hadoop fs -copyFromLocal news piglatin
> news = load 'piglatin/news' as (line : chararray);
> news = foreach news generate *, INDEXOF(line,'hadoop')
                                    as idx;
> ds=filter news2 by (idx>=0);
```

Lformate:-

Line
ABC XYZ ABC

```
L= foreach Line generate*,
   INDEXOF (line, 3, ABC)
   as Ln;
```

INDEXOF(Line, 'ABC')
↳0 ↳③

INDEXOF (line, 3, 'AB')
↳ String by

> INDEXOF (String, Start Index, 'search String')
↓
default is 0

Ex:- Sentimental keywords using multi INDEX :-

```
$ cat > comments
we love hadoop
we hate coding
we don't work
I hate to love hadoop
```

```
$ hadoop fs - copyFromLocal Comments piglatin
> lines = load 'piglatin / comments' as line : chararry);
> lines2 = load 'piglatin/ Comments' o foreach lines generate
        *, INDEXOF (line, 'love') as idx1;
           INDEXOF( line, 'hate') as idx2;
```

> f = filter lines2 by (idx1 >=0 or idx2 >=0);

Ex:- cat >> comments

hadoop is cooco
hadoop is wonder
still i hate coding
java is ugly.

prepare sentiment dictionary:-

$ cat > sentiments

Love, 10	handsome, 12
hate, -10	ugly, -10
wooo, 20	
wonder, 15	
beautiful, 10	

$ hadoop . fs -copyFromlocal

$ hadoop fs -copyFromlocal

grunt> lines = load 'piglatin/comms' as(line:chararry)

> words = foreach lines generate FLATTEN (TOKENIZE(line)
as word;

> Sents = load 'piglatin/sentiments' using pigStorage (',') as
(word:chararry, score:int);

> Swords = join words by words by word, Sents by word;

> describe Swords;

> Swords = foreach Swords generate words::word as w1,
Sents::word as w2, Sents::score as score;

> Scores = foreach Swords generate score;

> grp = group Scores all;

> dump grp

> Sentiment = foreach grp generate
Sum (Scores.score) as Sent;

> dump Sentiment

Assignments:- ① calculate Sentiment foreach comment
② calculate the sentiments by considering negations.

Ex:- cat > dictionary
pyar, love
ishq, love
kadal, love
mohabbat, love
nafrath, hate
koobsurath, beautiful

]$ cat > comments
i pyar you
you ishq her
she kadal him
we nafrath love
she is very koobsurath

$ hadoop fs - copyFromlocal comments piglatin/ncomment
$ hadoop fs - copyFromlocal piglatin
> comm = load 'piglatin/ncomment' as (line:chararry);
> words = foreach comm generate FLATTEN(TOKENIZE(line)) as word;
> dict = load 'piglatin/dictionary' using pigstorage(',') as (hindi:chararry, eng:chararry);
> dwords = join words by word left outer, dict by hindi
> describe dwords
> dwords = foreach dwords generate words::word as word, dict::hindi as hindi, dict::eng as eng;
> dwords =

> trans = foreach dwords generate (eng is not null ? eng:word) as word;
> dump trans

Hive

Hive is a data warehouse environment in hadoop framework..

→ where you can Storage & management data in structured formate.

→ Hive is a ecosystem which runs of top of HDFS and mapreduce.

→ For Hive tables, Backend Storage in HDFS, and execution model is mapReduce.

→ 'hql' = hive query language is used manage and process the data.

Similar to sql to "RDBMS".

to develop custom functionalities Hive as UDFS.

* Hive UDFS can be written in

Java, C++, Python, R, Ruby

→ Hql can process

→ Structed data
→ XML
→ JSON
→ urls (weblogs)

→ Hive is weak is unstructred Test
pig.

* Hql tables are two types
↳ are two types
* INNER tables → Default
* External tables

→ when a table is Create in HDFS, in following location one directory will Created:

/user/hive/warehouse

* $ hive ↵

hive>

, when file is loaded into table, the file will be copied into it's backend directory that mean for hive tables

backend storage in HDFS.

hive > load data local inpath 'file1' into table mytab;
> load data local inpath 'file2' in table mytab;

In HDFS, → /user/hive/warehouse/mytab/file1/file2

* hive> select * from mytab;

→ when select statement is applied on table hive will read data from all files of Table's backend directory.

→ Hive have intelligence when to use mapper only & mapper and Reduce.

→ from hive 0.94 version Onwards Hive can use both MR & TEJ models to execute query.

→ This common behaviour for both Inner & External Tables.

* Inner Table :-

when table is droped both metadata and data will be deleted. that means from hive table will be deleted.

hive > drop table mytab;

from HDFS,
/user/hive/warehouse/(mytab) → will be deleted

* External table :-

→ If external table is droped only metadata will be deleted.

that means,
from Hive table will be deleted,
In HDFS, still table's backend directory is available with its data files.
So that, you can Reuse the data.

Ex:- hive> Creat external table urtab (line String);
In HDFS,
/user/hive/warehouse/urtab → (Dir)

hive, load data local inpath 'filex' into table urtab;
In HDFS, user/hive/warehouse/urtab/filex → data file

hive > drop table ortab;

hive > Select * from ortab;

$ hadoop fs - ls /user/hive/warehouse/ortab ↵

available with all its data files file x

[creation wise, load wise hive will same]

NOTE:- Generally intermeadiate data tempory data is kept in inner tables.

→ Regularly used data is kept in external data.

How to Reuse drop data (table):-

hive > Create table ortab(line String)

> Select * from ortab;

→ In HDFS, if directory existed, Hive will used it, if not existed Hive will create it.

→ for both inner and external table we can use custom location.

NOTE:- Inner table also called as manage table. To get complete structure.

hive > describe extended Otab;

Ex:- Create table mmm (line String) local '/user/mydir';

→ multiple tables can use same location.

* Loading data into tables:-

* Loading from local files:-

hive > load data local inpath '<file path>'

[overwrite] into table <table name>;

Ex:- Create table empl (ecode int, name String,

sal int, sex String, dno int)

row formate delimited fields terminated by ',';

> load data local inpath 'emp' into table emp;

* load to data hdfs file to table :-

hive> load data inpath '/test/emp2' in table empl;

NOTE:- when a file is loaded into table from HDFS location,

— the file will be moved to table backend directory.

A file is from local the file will be copied into table directory that means still available in local.

Data types :-

Simple data types	Complex data types
String	Array → collections.
int	Struct → collections.
long	map
float	
double	
boolean	
Small int	

* Array :- Collection of homogeneous items.

Ex:- [100, 250, 300]
product prices
→ Array<Int>

* Struct :- Collection of heterogeneous items.
→ each element purpose is different.

Ex:- < P1, 700, 5>
pr id ↓ price ↓ quantity.

* map :- also called as "associated Array".
Collection of key & value pairs.

Ex:-
{ P1 : 250 , P2 : 470 , P3 : 500 }
map K V

* Working with the collections (complex datatypes):-

Ex:- Cat > prof1

Ravi, 28, Btech # MTech # phd

Rani, 24, BSc # MSc # MTech

> Create table prof1 (name String, age int, qual array <String>

row format delimited fields terminated by ','

collection items terminated by '#';

> load data local inpath 'prof1' into table prof1;

> select * from prof1;

O/P:-

Ex:- cat > prof2

Ravi, 26, Rani # 24 # BTech, MTech

Giri, 86, Vani # 29 # MSc, BTech

>Create table prof2 (name String, age int,

wife struct < name : String, age : int, qual : String >,

qual String,

row formate delimited fields terminated by ','

collection items terminated by '#';

> load data local inpath 'prof2' into table prof2;

> select * from prof2

O/P:-

Ex:- $ cat > prof3

mohan, 84, btech $ 2001 # MTech $ 2003 # 2008

mohana, 83, bSc $ 2000 # MSc $ 2002

> Create table prof3 (name String, age int, qual map

<String, int>) row

formate delimited fields terminated by ','

collection items terminated by '#' map

keys terminated by '$';

> load data local inpath 'prof3' into table prof3;

> Select * from prof3;

O/P :-

By using index number accessing the array.

hive> select name, equal [0] from prof1

=o=

> Select name, size (qual) from prof1;

O/P:- Ravi 3
Rani 3

> Select * from prof1 where array_contains (qual, 'phd');

* Accessing elements from Structs:-

Select * from prof2;

hive> select name, wife.name, age, wife.age from prof2;

* Accessing elements of map :-

Select * from prof3;

> Select name, map_keys (qual), map_values (qual) from prof3;

Size apply:-

Select name, size (qual) from prof3;

Accessing particular element :-

(Key passed) :-

Select name, qual ['mtech'] from prof3;

O/P:- Mohan [Btech, MTech, phd] [2001, 2003, 2008]

> Select name, size(qual) from prof3;

O/P:- mohan 3
mohana 2

> Select name, qual ['mtech'] from prof3;

O/P:- Mohan 2003
mohana null

> show ~~tables~~ databases

> Create database mydbs

> use mydbs → database is selected

> Create table tab1 (line string);

-Check in backend

- $ hadoop fs - ls /user/hive/warehouse/mydbs.db

Load data from table to table :-

> Create table emp2 like emp;

> insert overwrite table emp2

> select * from emp where sex = 'm';

> whenever table to table, data is loaded, the hive generated file name as 000000 -0

Table to Table Copy techniques :-

> Create table emp2 like emp;

(i) Copy all data from one table to another :-

$ hadoop fs - cp /user/hive/warehouse/mydbs.db/emp/emp
/ " / " / " / " /empy

→ If Copy is done for all data, then copy the files in the backend.

(ii) Appending data of one table to another table :-

hive > Create table dummy like emp;

> insert overwrite table dummy

> Select * from emp where sex 'f';

[training @ local host ~]$ hadoop fs -cp /user/hive/warehouse/
mydbs-db/dummy/000000 -0 /user/hive/warehouse
/mydbs-db/emp2/new

NOTE:- In hive 0.14 version, "insert into" statement is available.

(iii) Table to local directory :-

> insert overwrite local directory 'mynew'

> Select * from emp where dno in (11,12);

(iv) Table to HDFS (directory) :-

> insert overwrite directory 'ornew'
> select * from emp where dno in (11,12);

(v) Table to directory (local/HDFS) with ',' delimiter :-

> insert overwrite table dummy
> select * from emp where dno in (11,12);
$ hadoop fs -copyToLocal /user/hive/warehouse/mydbs.db/dummy/000000 -o /home/training/desktop

(vi) Merging tables of different schema :-

Schema :- file 1 - ecode, name, sal, sex, dno
file 2 - " } delimiter
file 3 - ecode, name, dno, sex, sal
file 4 - same as file ① & ② — delimiter = tabSpace

hive > Create table t1 (ecode int, name string, sal int, sex string, dno int)
> row format delimited fields terminated by ',';

hive > Create table t2 (ecode int, name string, sal int, sex string, dno int)
> row format delimited fields terminated by ',';

hive > Create table t3 (ecode int, name string, sal int, sex string, dno int)
> row format delimited fields terminated by '\t';

hive > load data local inpath 'emp1' into table t1;
> " " " " 'emp2' " " t1;
> " " " " 'emp3' " " t2;
> " " " " 'emp4' " " t3;

NOTE :- ** Hive doesnot have union only union all which allows duplicate rows.
** Unions (Hive) should be as a sub-query
** Sub-query should have an allias

> Create table mytab like t1;
> Insert overwrite table mytab
> Select * from (
> Select ecode, name, sal, sex, dno from t1
> Union all
> Select ecode, name, sal, sex, dno from t2
> Union all
> Select ecode, name, sal, sex, dno from t3) t;
alias

* XML data processing using hive:-
* working with xml file:-
extensible marker language
(or)
data point of view:-
* Advantages:-
→ flexible schema (not find fixed schema.
↳ In RDBMS, fixed schema (structure)
record to record different attributes can be maintained.
→ Compatable formats with all databases
→ Compatable with all languages.
(C, C++, java, Python)
→ Can do journy in any network protocols.

(flexible schema) → schema less

XML1:-
]$ gedit xml Script1 ↵
$ cat > xml1
<rec><name> Rani</name> <age>25</age><sex>m</sex></rec>
node begin | node text | node end
<rec><name> Rani</name> <sex>f</sex><city>Hyd</city></rec>
> Create database xmls;
> use xmls;
> Create table raw (line String);

> load data local inpath 'xml1' into table raw;

> Create table info (name String, age int, sex String, city String);

> row format delimited fields terminated by ',';

> insert overwrite table info

> select xpath - String (line, 'rec/name'),
xpath - int (line, 'rec/age'),
xpath - String (line, 'rec/sex'),
xpath - String (line, 'rec/city'), from raw;

> select * from info;

> $hive -f (filename) is used to execute script from command line.

XML2:- cat > xml2 (with nested tags)

<rec><name><fname>Ravi</fname><lname>kumar</lname></name>
<age>25</age><contact><email><personal>
ravi@gmail.com</personal><official>
ravi@ibm.com</official></email><phone><mobile>1234</mobile><office>1235</office><residence>1236</residence></phone></contact><city><hometown>hyd</hometown>
<worktown>del</worktown></city></rec>

> Create table xinfo (fname String, lname String, age int, personal-email string, official-email string, mobile String, office string, residence String, hometown string, worktown String);

row format delimited fields terminated by ',';

> insert overwrite table xinfo
select xpath - String (line, 'rec/name/fname')
xpath - string (line, 'rec/name/lname')
xpath - string (line, 'rec/age')

xpath - string (line, 'rec | contact | email | personal').
xpath - string (line, 'rec | contact | email | official')
xpath - string (line, 'rec | contact | phone | mobile')
xpath - string (line, 'rec | contact | phone | office')
" " (" " " /residence)
" " (" , "rec/city/hometown')
" " (" " " (worktown') from raw;

> select * from kinfo;

xml3:-

<rec><name> Ravi </name><qual> Btech </qual><qual> Mtech </qual></rec>

<rec><name> Radha </name><qual> Btech </qual><qual> mba </qual></rec>

<rec><name> Rani </name><qual> BSc " " msc </qual><qual> mba </qual></rec>

[xpath - datatype → single value
xpath → array
xpath always load into string]

> Create table xml raw (line String);
> load data local inpath 'xml3' into table xml raw;
> Create table raw2 (name string, qual array<String>);
> insert overwrite table raw2
> select xpath - String (line, 'rec/name')
xpath (line, 'rec/qual/text()') from xml raw;
> Create table xml info (name string, qual string);
> insert overwrite table xml info
> select name, myq from raw2
lateral view explode (qual) q as myq;
> select * from xml info;

XML4 :-

<rec> <cid> 101 </cid> <price> 1000 </price> <price> 2000 </price> </rec>
" " > 102 < " > 2000 < " 500 " 7000 </price> </rec>

<rec> <cid> 101 </cid> <price> 3000 </price> </rec>

> Create database sales db;
> use sales db;
Create table raw (line String);
load data local inpath 'xml4' into table raw;
Create table raw2 (cid String, pr array < String >);
> insert overwrite table raw2
> select xpath - String (line, 'rec/cid/name')
xpath (line, 'rec/pr/qnt/text()') from xmlraw;
> Create table salesinfo (cid name String, pr qnt int String);
> insert overwrite table salesinfo
> select cid name, mypr from raw2
lateral view explode (pr) as mypr;
> select * from salesinfo;
Create table report (cid String, totbill Int);
insert overwrite table report
Select cid, sum(pr) from sales group by cid;
select * from raw;
select * from raw2;
select * from sales;
sclect * from report;

XML5:- (xml with multiple collections):-

<rec> <cid> 101 </cid> <pr> 1000 </pr> <qnt> 5 </qnt> <pr> 2000 </pr>
<qnt> 10 </qnt>
<rec> <cid> 102 </cid> <pr> 2000 </pr> <qnt> 10 </qnt> <pr> 500 </pr>
<pr> <qnt> 10 </qnt> <pr> 7000 </pr> <qnt> 5 </qnt>
</rec>

<rec> <cid> 101 </cid> <pr> 3000 </pr> <qnt> 10 </qnt> </rec>

Ex:-]$ getit News;

```
Create database xd;
use xd;
Create table raw (line String);
load data local inpath 'xml5' into table raw;
Create table raw2 (cid string, pr array <String>,
                   qnt array <String>);
insert overwrite table raw2
    select xpath_string (line, 'rec/cid'),
           xpath (line, 'rec/pr/text()'),
           xpath (line 'rec/qnt/text()') from raw;
Create table cidpr (cid String, pr int);
insert overwrite table cidpr select cid, mypr from
    raw2
    lateral view explode (pr) p as mypr;
Create table cidqnt (cid String, qnt int);
insert overwrite table cidqnt select cid, myqnt from
    raw2
    lateral view explode (qnt) q as myqnt;
add jar Desktop/hudfs.jar;
Create temporarily function sno as 'hive.analytics.Sequence';
Alter table cidpr add column (n int);
  "     " cidqnt  "   "   "   ;
insert overwrite table cidpr select cid, pr, sno()
                    from cidpr;
insert overwrite table cidqnt select cid, qnt, sno() from
                    cidqnt;
Create table sales (cid string, print, qnt int, bill int);
insert overwrite table sales select l.cid, pr.qnt, pr*qnt
    from cidpr l join cidqnt r
        on (l.n = r.n);
```

Create table report (cid String, totbill int);

insert overwrite table sales report select cid, Sum(bill)
from sales group by cid;

Select * from sales;

Select * from report;

* hive udfs for the program :-

```
package hive.analytics;
import java.io.Ioexception;
import org.apache.hadoop.hive.ql.exec.UDF;
insert org.apache.hadoop.io.Intwritable;
// udf to generate onique sequence number.
    public class Seqnom extends UDF
        {
        int cnt=0;
    public Intwritable evalute() throws IOexception
        {
        C++;
    return new Intwritable(cnt);
            }
        }
```

Explode() → one of UDTF

Ex:- SAMP

Name String	qual array<String>
Amar	["BTech","mTech"]
Amala	["BSc","msc"]

* Types of HIve function :-
(i) UDF
(ii) UDAF
(iii) UDTF

* UDF :- is one Scalar functions foreach row one value will be returned.

Ex:-

Tab x	Sqrt(x)
100	10
1000	100
25	5

Other ex
Size()
length()
Substr()
:
:

UDAF:- User defined Aggregated functions.

Ex:- Sum, AVG, MAX, MIN, COUNT

for entire column one value will be returned.

x
10
20
30

Sum(x) → (60) → Singly

UDTF:- user defined table generated functions.

→ for each row a table will be returned.

→ table can have multiple rows or columns.

Ex:- Arr → ["10", "20", "30"]

explode(Arr) → O/p

10
20
30 } multiple rows → table

Other UDTFs:-

explode()
json_tuple()
parse_url_tuple()
} which returns multiple columns or multiple rows

explode():- input argument is Array

↳ It splits array element, into rows each element into a seperate row.

Note:- Explode() should have an alias

Tab

x → Array
[10, 20]
[10, 30, 40]

Select explode(x) as y from Tab;

explode([10,20]) → 10, 20
explode([10,30,40]) → 10, 30, 40
union → merging

y
10
20
30
40
10
30
40

Tab

name(String)	goal(array)
Amar	["BTech", "MTech"]
Amala	["BSc", "MSc"]

select name, explode (qual) as q from Tab;

↳ is invalid statement

because → name is single value

explode ()'s result is a table

along with UDTF, column expressions should not given.

But each row of explode() can be joined with a column using "Lateral view"

↳ Lateral view a temporary table construct.

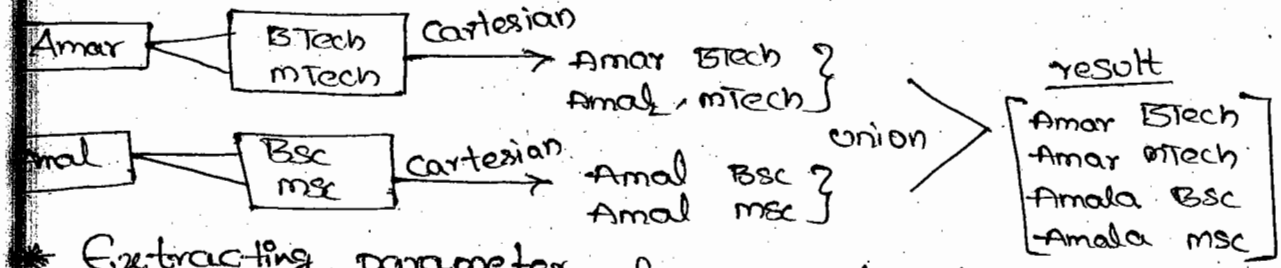
using lateral view, independent result of each row's explode, will be cartesian with given colum expressions.

Lateral each row result will be merged.

```
select name, myq from Tab
    lateral view explode (qual) q as myq;
```

(myq: Column alias for explode; q: alias of lateral view.)

Cartesian :- each row will be joined with



* Extracting parameter from xml Nodes :-

Ex:- <rec><cid> 101 </cid> pr id = "p1" >1000 </pr><pr id = "p4">
2000 </pr></rec>
<rec><cid > 102 </cid ><pr id = "p3" >2000< /pr><pr id =
"p2" 500 </pr><pr id = "p4">7000 </pr></rec>
<rec><cid> 101 </cid><pr id = "p4" > 3000 </pr></rec>

]$ gedit news;

```
Create database ndb;
    use ndb;
Create table raw (line string);
load data local inpath 'xmL6' into table raw;
Create table info (cid string, prid array <String>,
                   price array <String>);
```

insert overwrite table info select xpath_string
(line, 'rec/cid'),
xpath (line, 'rec/pr/@id'),
xpath (line, 'rec/pr/text()') from raw;

Select * from info;

XML

Now adays JSON will taken by the data.

* JSON data processing :-

(Java Script object Notation)

$ gedit JSON1

{"name":"aaaa", "age":25, "city":"del"}

{"name":"bbbb", "sex":"f", "city":"hyd"}

hive> creat database jsdb;

use jsdb;

Create table raw (line String);

load data local inpath 'JSON1' into table raw;

Create table info (name String, age int, sex String,
city String);

insert over_write table info select get_json_object
(line, '$.name'),

get json_object (line, '$.age'),

get_json_object (line, '$.city') from raw;

Select * from info;

hive> select X.* from raw
lateral view json_tuple (line, 'name', 'age', 'sex', 'city')
X as n, a, s, c

* Nested JSON records :-

{"name":"ravi", "age":26, "wife":{"name":"Gita", "age":24,
"city":"Hyd"}, "city":"Del"}

{"name":"mohan", "age":29, "wife":{"name":"mohana", "age":
28, "city":"Hyd"}, "city":"Hyd"}

```
J$ gedit jSON2 ↵
  Create table jraw ( line String );
  load data local inpath 'jSON2' into table jraw;
  Create table raw2 ( name String , age int , wife String,
                      city String );
  Select X.* from jraw
    lateral view json_tuple ( line, 'name', 'age', 'wife', 'city')
                  X as n,a,w,c;
Create table info ( hname String , Wname String , hage int;
                    wage int, hcity String , wcity String);
insert overwrite table info
  Select name, get_json_object (wife , '$.name'),
               age, get_json_object ( wife , '$.age'),
               city, get_json_object (wife , '$.city')
                from raw2 ;
  Select * j info;
-- when you want to collect series of json fields better
     use json_tuple ();
--- when you want a single field , use get_json_object()
```

* JSON with Arrays :-

```
J$ gedit jSON3;
{ "name": "Ravi", "qual": ["BTech","MTech"]}
{"name": "Rani", "qual" : ["BSc", "Msc","MTech"]}
Create table jsraw ( line String);
    load data local inpath 'jSON3' into table jsraw;
Create table jsraw2 ( name String , qual String );
insert overwrite table jsraw2
    Select X.* from jsraw
    lateral view json_tuple (line,'name', 'qual')
                  X as n,q;
```

```
Create table raws (name String, qual array<String>);
insert overwrite table raws
Select name, Split (qual, ',') from jsraw2;
Create table raw4 (name String, qual String);
insert overwrite table raw4
    Select name, myq from raws
Lateral view explode (qual) q as myq;
Create table raws like raw4;
Insert overwrite table raws
    Select name, Split (qual, ',')[i] from raw4;
Select * from raws;
```

Assignment :-

```
{ "name": "ravi",
  "age": 26,
  "qual": ["BTech", "MTech"],
  "wife": {
    "name": "Githa",
    "age": 24, "children": [
                {
                "name": "veni",
                "sex": "f",
                "age": 4
                },
              { "name": "venu",
                "sex": "m",
                "age": "1"
                }
              ]
            },
  "city": "hyd"
```

* URL data processing :-

http : // traing.com / bigdata / hadoop ? name = ravi ? age = 26 ? city = hyd ? Company = ibm

(host) (path) query information.

http://traing.com/bigdata/mongodB?name=aaa&age=28
&city=del

http://traing.com/bigdata/oops/java?name=aaa&age=29

* Url has 3 basic information.
  ** Host :- traing.com
  ** path :- bigdata/hadoop
  ** query string :- ?name=ravi&age=26&city=hyd&company=ibm

] $ gedit urls

```
Create database urldb;
        use urldb;
Create table raw (line String);
load data local inpath 'urls' into table raw;
   Select parse_url(line,'HOST'),
          parse_url(line,'PATH'),
          parse_url(line,'QUERY') FROM raw;
hive> insert create table raw2 (host String, path String,
                                query string);
hive> insert overwrite table raw2
    > select x.* from raw
    > lateral view parse_url_tuple
          (line,'HOST','PATH','QUERY') x as
          h,p,q;
Create table raw3 (host String, path array<String>
                   query map<String,String>);
insert overwrite table raw3
    select host, split(path,'/'),
[how to show] str_to_map(query,'&','=') from raw2;
                  show function
   Create table info (host String, Category String,
          Coarse String, City String, Sex String);
insert overwrite table info
 select host, path[1], path[2],
  query['name'], query['age'], query['company']
```

query ['city'], query ['sex'] from raws;

Select from info;

15/11/2015

How to Select database :-

hive > Used mydb;

hive> Create table samp (line String);

/user/hive/warehouse/mydb.db/samp

-- we can change location for table.

hive> Create table mytab (line String)
location '/user/mydir'

Ex:- Select * from etab;

Select name, sal from etab;

Select * from etab where sex = 'm';

-- where dno = 11 or dno = 13 or dno = 14;

-- where dno in (11, 13, 14)

-- where dno != 11 and dno != 13

where dno not in (11, 13) --- invalid

-- Sales -- trid, pid, dot, price, qnt
2001 to 2015

-- Select * from Sales
where (year (dot) = 2005 and month (dot) >= 6)
or
--(year (dot) >= 2006 and year (dot) < 2014)
or
(year (dot) = 2010 and month (dot) <= 7)

Select * from etab
where sal >= 40000 and sal <= 80000;

Select * from etab where sal between
4000 and 8000;

-- between clause not available

Select * from etab
where sal < 4000 or sal > 8000;

erforming Conditional transformations:-

]$ hive

ive> Create database practice

> use practice

ive> Create table emp2 (ecode int, name String, sal int,
grade String, sex String, dname String);

ive> Insert overwrite table emp2

> Select ecode, name, sal,

> if (sal >=7000, ('A'),

> if (sal > =3000, ('C', 'D'))),

> if (sex ='f', 'Female', 'male'),

> if (dno =11, 'mrkt',

> if (dno =12, 'Hr')

> if (dno =13, 'Fin', 'others'))) from emp;

Check it o/p:-

hive > Select * from emp2;

Ex:-]$ cat > Samp

100, 200,
, 8000, 7000
, , 100
10, , 50
, , 35
1, 2, 3
4, 5, 6

hive> Create table tab (a int, b int, c int)

> row format delimited fields terminated by ',';

Check it :-

hive> Select * from tab;

]$ hadoop fs -ls /user/ hive/ ware house/ practice.db/
tab;

Conditional transformation :-

```
hive> insert overcomite table tab
    >select if (a is null , 0 , a),
  >if ( b is null , 0, b),
    > if ( c is null , 0, c) from tab;
```

[cleaning the nulls]

Check this tab :-

```
hive > select * from tab;
```

(1/0 Gives infinity) [infinity is one of the predefined function]
(0/0 Gives NAN)

Aggregations in hive :-

```
select sum(sal) from emp;
Select Sum(sal), avg(sal), max(sal), min(sal),
        Count(*) from emp;
```

```
select count(*) from tab; -- counts all rows
select count(sal) from tab; -- does not count nulls.
Select avg(sal) from tab; -- does not count nulls.
select avg(sal) from tab; -- it does not considers nulls
                              tot/(not nulls count)
```

Grouping Aggregation :-

Group by "clause"
↳ is part of select Statement

-- to perform aggregations seperately on each data group.

Ex:- Seperate total for females and Seperate total for males.

```
hive> select sex , Sum(sal) from emp group by sex;
```

Multigrouping :-

```
hive> Select dno, sex, Sum(sal) from emp group by
        dno, sex;
```

NOTE:- No limit for max no of Grouping columns.
in Rooms - 16

```
hive> Select dno, Sum(sal) from emp
        where dno in (11,18,20)
          group by dno;
```

where --- filter happens at mapper level *

```
hive > select dno, Sum(sal) from emp having dno in
                                    (11,18,20);
```

having --- filter happing happens reducer level.

NOTE:- " having" should be applied with "group by" clause only.

```
Select dno, Sum(sal) from emp where
        sex = 'f'
        group by dno
          having dno in (11,18);
```

NOTE:- In having clause,
you can use
* grouping column
* aggregated functions

Ex:-
```
Select acno, count(*) from atm
  group by acno
    having count(*) > 2;
```

Distinct :-

- to eliminate duplicate values of a column

```
> Select distinct(dno) from emp;
> Select count (distinct (dno)) from emp;
> Select count (*) - count ( distinct (dno)) from emp;
```

> Select count(*) > c

How to get duplicate count :-

hive> Select dno, count(*) from emp

> group by dno having count(*) > 1;

Ex:- I$ cat > info

101, aa
102, bb
103, cc
103, cc
103, cc
102, bb
101, dd

(entire row match pwant eliminate)

hive> Create table samp (id int, name String)

> row format delimited fields terminated by ',';

hive> Select * from samp;

hive> Select distinct (id), name from samp;

hive> select id, name from samp group by id, name;

(it is best in MR point of view) — best model in hive

The backend process of eliminating dupes using groupby:-

(Internal process of MR)

(In data point of duplication)

mapper key = offset (0), val = 101, a

map()

con.write (val, *)

101, a*
102, b*
103, c*
102, b*
103, c*
103, c*

Reducer

(i) grouping

K	vlist
101, a	<*>
102, b	<**>
103, c	<***>

(ii) Sort

K	vlist
101, a	<*>
102, b	<*, *>
103, c	<*, *, *>

(iii) Reduce()

con.write (K, NullWritable.get())

o/p:- 101, a
102, b
103, c

Eliminating duplicates based some column match :-

```
]$ cat > info2
101 , 10000        103, 8000
102 , 30000        101, 7000
101 , 50000        103, 9000
101 , 6000         102, 1000
102 , 5000         106, 9000
```

```
hive> Create table damp (acno int, amt int)
    > row formate delimited fields terminated by ',';
hive > insert Overwrite table damp
    > Select * from damp Order by acc;
```

GroupCount UDF :-

```
package java.io.IOException;
public class GroupCount extends UDF
    {
     int cnt = 0 ;
     int prev = 0;
   public IntWritable evalute (IntWritable v)
        throws IOexception
         {
       int cv = v.get ();
        if (cv != prev) cnt = 0;
          cnt ++;
          prev = cv;
    return New IntWritable (cnt);
          }
    }
```

To call this of first of all data will be sorted.

```
hive> add jar Desktop / hives.jar;
hive> Create temporary function grpcnt as 'hive.you.GroupCount';
hive> Select * from damp;
hive > alter table damp and columns ( n int);
hive > insert Overwrite table damp
    > Select acno, amt, grpcnt (acno) from damp;
```

hive> select * from damp;

101	1	103	1
101	2	103	2
101	3	106	1
101	4		
102	1		
102	2		
102	3		

hive > insert overwrite table nodupes
> select acno, amt from damp
> where n=1;
hive> select * from nodupes.

How to get last 3 duplicates:-

hive > select l.acno, amt from
> damp l join (select acno, max(n) as m
> from damp group by acno) r
> on (l.n = r.m and l.acno = r.acno);

Sorting :-

"Order by" clause is used for sorting:-

Ex:- hive> select * from emp order by name;
hive> select * from emp order by sal desc;

-- multi Sort (secondary Sort)

hive> Select * from emp Order by Sal desc, dno, sex desc;

"Order by" and "Group by" combination:-

Ex:- Select city, Sum(amt) as tot from sales
group by city
order by to desc;

limit :-

to feach top n no. of row.

Select * from emp limit 3;

Ex:- add jar hives.jar;

```
create temporary function Sno as 'hive.you.SeqNum';
Alter table emp add columns (n int);
Select * from emp order by Sno() desc limit 3;
insert overWrite table emp
        Select ecode, name, sal, sex, dno, Sno() from emp;
```

to featch top3 Salaried employers.

```
Select * from emp Order by sal desc limit 3;
```

↓

above query is true, If all salaries are unique.

Ex:- cat > info3

```
101, aaaa, 10000
102, bbb, 9000
103, ccc, 90000
105, ddd, 89000
106, dv, 89000
107, KKK, 90000
```

```
hive> Create table etab ( ecode int, name String, sal int)
    > row format delimited fields terminated by ',';
hive> select l.* from
    > etab l join ( Select distinct (sal) as s from etab
    > order by s desc limit 3)
    > r on (l.sal = r.s);
```

↳ above approach is bad in performance because Hive is triggering multiple jobs.

* for Distinct
* for Order by
* for joins

desc;

Sd is UDF by generating math rank

```
public IntWritable evalute (IntWritable v) throws IOException
  {
  // equal score should get equal rank.
      int (cv != prev) cnt ++;
          prev = cv;
        return newIntWritable (cnt);
          }
```

hive > add jar Desktop / hive52.jar

> Create temporary function rank as 'hive.you.maturank';

hive> insert overwrite table etab

> select * from etab order by sal desc;

hive> alter table etab add columns (r int);

hive> insert overwrite table etab

>select ecode, name, sal, rank (sal) from etab;

hive > Advantages of UDF :-

* Custom to define functionalities
* to imporving performance
* reusability.

* Life cycle UDF :-

* Develop UDF class
* Export into jar
* Register jar hive
* Create temporary function for UDF class
* Calling function.

* Develop UDF class:-

hive > -- when java class extends UDF, the class will get hive udf functionality

> -- udf logic to be overriden in evalute() function

> -- evalute() executed for n-times

>-- where n is number of rows feached by select statement.

> -- add jar < jar file path >

hive > -- create temparary function < fun name >

> -- as 'udf class'

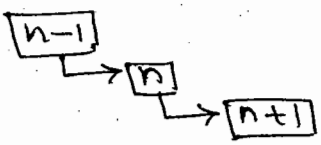
Hive partitioned tables:-

Using this

* Hive Creates Seperate partition for each data group
* partition is a Subdirectory of table directory.
* When you request a Seperate data group, Hive directly Contacts particular partition and reads only that partition data instead of reading partition.
  ↓
  total table data.

Advantages of partitions:-

* Time Reading Saved.
  No Need to Scan entire table data.
* you can Creates partitions using Single column and also by multiple columns
* Sub partition is Subdirectory of prior partition.
* Sub partition



* you can load data into partitions statically & dynamically

Ex:- 100 product transaction in Sales table.
regular products are P_1 and P_9
remaining are rare products.

→ load all P_1's into P_1 partition
→ load all P_9's into P_9 partition
→ load all remaings all into P_x partitions.

* needs Seperate partitions for each product in single load.
  Dynamic load
  By default dynamic load in feature is disabled.
* partition table can be inner or external
  → Inner and NON partition
  → Inner and partition
  → External & partitioned
  → External & non partitioned

Ex:- hive > Create table emp (ecode int, name String,
> Sal int, sex String, dno int)
> row format delimited fields terminated by ',';
hive> load data local inpath 'emp' into table emp;
hive> Select * emp;

--- Creating partitioned tables:-

hive > Create table epart (ecode int, name String,
Sal int, sex String, dno int)
partitioned by (s String);

-- loading data into partitions:-

hive > insert overwrite table epart partition (s='f')
Select * from emp where sex = 'f';

hive> insert overwrite table epart
partition (s='m')
Select * from emp where sex = 'm';

hive> Create table mpart (ecode int, name String, Sal int)
partitioned by (dno int, sex String);

Enabling dynamic load feature:-

hive > Set hive.exec.dynamic.partition =true;

This option enables dynamism for all partition columns except primary partitioned column.

-- to primary also dynamic, use following option.

hive> Set hive.exec.dynamic.partition.mode = nonstrict;

-- load dynamically

hive > insert overwrite table mpart partition (dno, sex)
Select ecode, name, Sal, dno, sex from emp;

Ex:- $ Cat > Sales
01/01/2001, 3000000
01/02/2001, 6000000
:
11/22/2016, 8000000

hive> Create table rawsales(dt string, amt int) raw format
delimited fields terminated by ',';
hive> load data local inpath 'Sales' into table rawsales;
-- data transformation :-
hive> Create table rawsales2 (dt array<string> amt int);
hive> insert overwrite table rawsales2
select split(dt, '/'), amt from rawsales;
-- cleaning data :-
> Create table sales like rawsales;
> insert overwrite table sales select concat [dt[2], '-', dt[0], '-',
dt[1]), amt from rawsales2;
> Create table Salespart (dt string, amt int)
partitioned by (y int, m int, d int);
> Set hive.exec.dynamic.partition = true;
> set hive.exec.dynamic.partition.mode = non strict;
> insert overwrite table salespart
partition (y,m,d) select dt, amt, year(dt), month(dt),
day(dt) from sales;
> select * from sales part where y=2014;
> select * from sales part where y=2014 and m=1 and d=25;
> set hive.exec.max.dynamic.partitions.pernode = 10000;
> Set hive.exec.max.dynamic.partitions = 100000;
---> cluster level limitation.

* Bucketing :-
Used for multiple sampling techniques.
* each bucket is a file in table directory.

hive> Create table bucks (ecode int, name string,
> sal int, sex string, dno int)
> clustered by (dno)
> into 2 buckets;

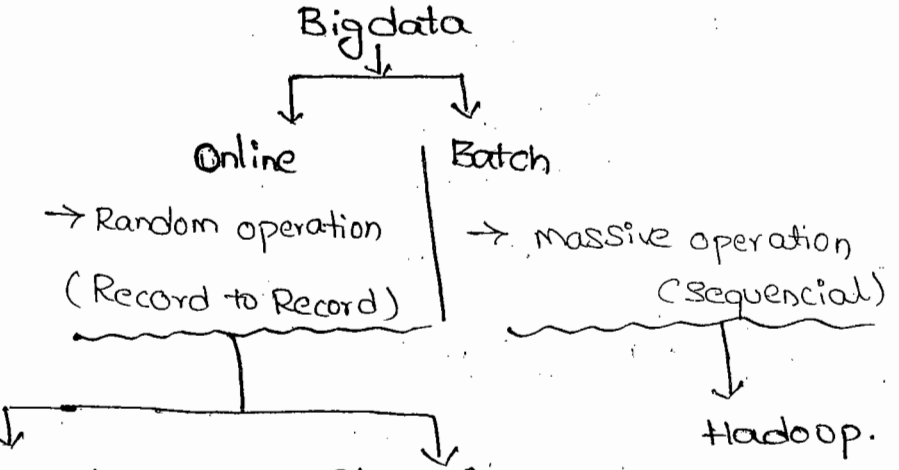
How to enabled buckets :-
-- by default bucketing feature is disabled.

To enable:-

```
hive > set hive.enforce.bucketing = true;
hive > insert overwrite table bucks
     > select * from emp;
```

NO-SQL

4/11/15



* NO-SQL Advantages:-

→ This is very highly scalable (Thousand of Nodes you can keep in to claster).

→ High capacity to the servers.

→ Schema less database

Structerd (flexible schema)

→ Read Record to Record different Structures maintained.

→ The full fledge Schema Available in MongoDB.

→ Faster Random CRUD operation.

Create / Read / Update / Delete.

* These NO SQL databases classified into four types.

* Key and value Store

Riak [FB/LN]

PNUT - [NOT open Source
yahoo product]

* Document Store (couch, MongoDB) ↓TOP
* Graph Store (NEO 4J)
* Columner Store (Cassandra, HBase)

NOTE:- By integrated all these types.

* Key and value Store:-

→ The purpose of key and value Store is faster Random Retrival.

Ex:- Sign In ← User ID (key)
pwd (value)

→ Storage object = MAP

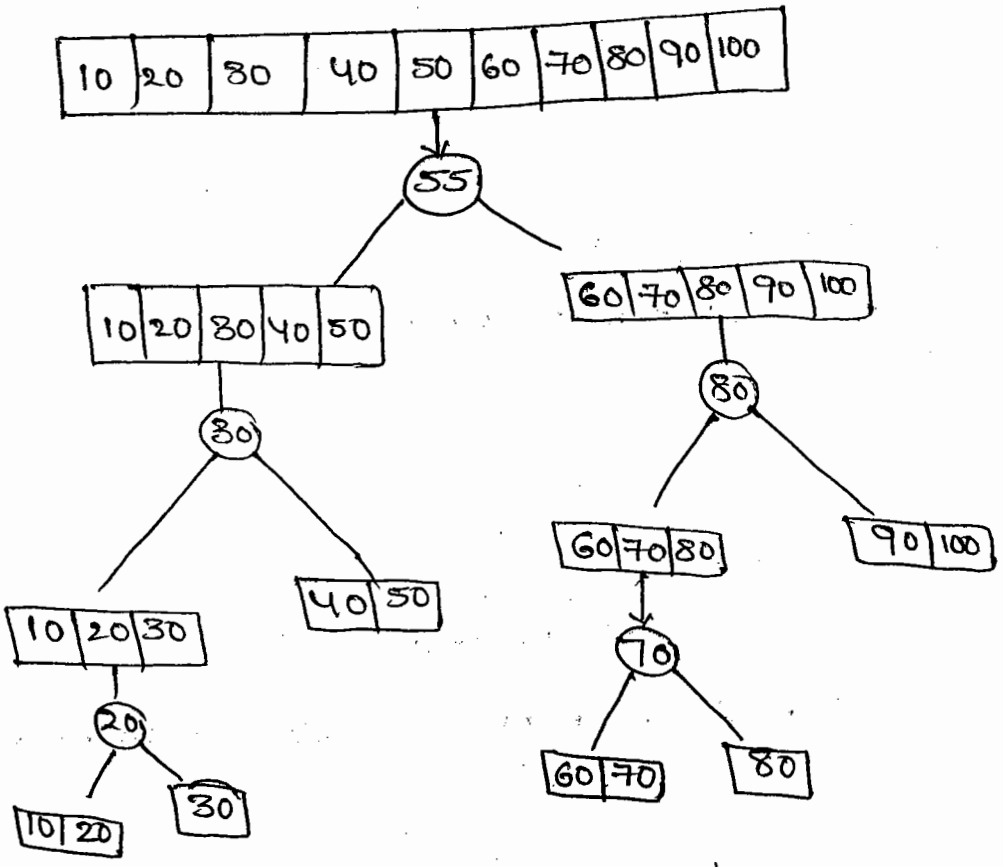
→ Algorithm it applys → Random Trailer

In RDBMS,

from Random Retrivals BTREE ↓ Binary algorithm is used (applied).

Ex:- B-TREE

5/11/15



The BTree algorithm gives latency for hive volumes of index.

Queries:- * select * from x where city = 'hyd';

* select * from x where ACCNO = 2004;

1 - 1000 - P_1 - HYD

1001 - 2000 → P_2 → D

2001 - 3000 → P_3 → D

3000 - - → P_4 → B

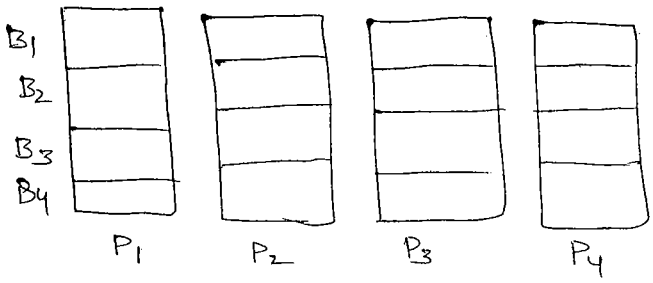
ACCS table → table space

ASP Table (Space)

AINDX

AINDXSP

Key	Address
107	ASp + P_1 + B_1 + 0



* How to Random trials works?

Based on INDEX Size

→ estimate and attemp to read if error occure, learn from error and Re attempt.

→ In just few trails a Record will can be read randomly identify.

5/11/15

* Columner Store :-

→ It is faster Random

→ CRUD over Specific columns of a Big Tables.

Bigtable → A table which has Converts lacks of tables.

→ google invented

↳ 4 millian columns in a single table.

* Cassandra (Old one)

↳ By over come improved

* ↓

HBase (New one)

↳ Recently introduced another one Accumulo

Ex:- Queries:- select * Table;

select C120, C978 from Tab;

In above two queries requesting all column of a table is faster rather requesting specific column of a table. Because identification of a columns starting possition and ending possition and filtering them is painfull process for RDBMS.

* Columner databases such as HBase and Cassandra are good speed at in CRUD operation over specific columns.

* In Columner Store

CRUD is applied on columns not on rows.

→ In these database Columns Stored as Rows.

So that RUN time any new column can be added to a particular Record.

→ In cassandra the Storage object is → MAP

Cassendra algorithm is → map of map

(In java that means nested map)

* In Cassandra Vs RDBMS (Data modeling):-

In RDBMS:

emp

ecode	name	Sal
101	A	1000
102	B	2000
103	C	3000

In cassandra

↓

logical model

Key	value
101	Name = AA
101	Sal = 20000
102	Name = BB
102	Sal = 30000
102	City = HYd
103	Name = CC
103	Sal = 40000
103	Age = 26

* Physical model :-

① Map of map model :-

Query :- select city from CTab when ecode = 102);

<101, << name, AA >, < Sal, 20000 >>>

<102, << name, BB >, < Sal, 30000 > < Sex, M >>>

< 103, << name, CC >, < Sal, 40000 > < age, 26 >>>

② Tabular formate of map of map :-

CTAB

ecode (Key)	m2	
	col name	col value
101	name	AA
	Sal	20000
102	name	BB
	Sal	30000
	Sex	M
103	name	CC
	Sal	40000
	Age	26

Query :- select sex from CTAB when ecode = 102

Explanation :-

When a column is requested of a particular row, two random reads will happend.

→ In first random read the row key is identified the returned value will be second level map.

→ On second map column name is key column value is value.

→ column name passed as key so that column value will be returned.

Eventhough Crores of rows and lacks of columns for each row in just two random reads. The required

Column is identified.

6/11/15

Cassandra:-

In cassandra the query language is "Cql" similar to "Sql."

→ Cassandra is very good faster if table is single. While collecting data from multiple tables. (emp)

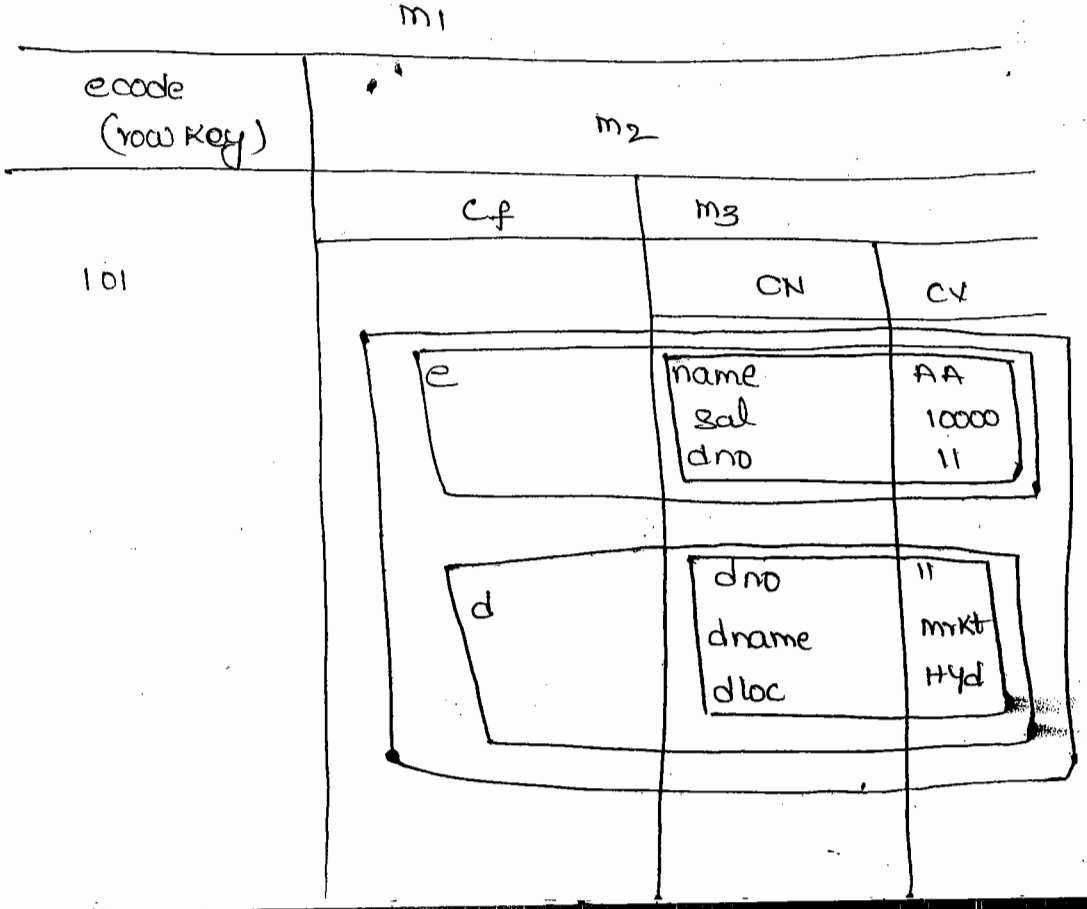
→ Joins to be applied.

Joins are very good when compared to subqueries. Joins are very bad for Online Bigdata.

→ (To eliminate) To over come this draw back joins next product introduced called HBase. with model "map of map of map"

→ Bcoz of 3 level maps joins can be eliminated.

* Map of map of map:-



6/11/15:-

Columner database :-

↳ Stored documents

↳ columns are store in row

* HBase :-

In HBase similar to Cassandra columns store as rows.

→ Each column should be associated with a column family.

Column family is a collection of column used for categorised (logical grouping) of columns.

Ex:- All personal details of a profile as "column family "p"

⇒ P.name
P.city
P.age ⇒ and similarly education summary as column family "e"

e.first qual
e.2nd qual

→ for a HBase table any no.of column families can be created one column family can have any no.of columns.

100cf × 100c

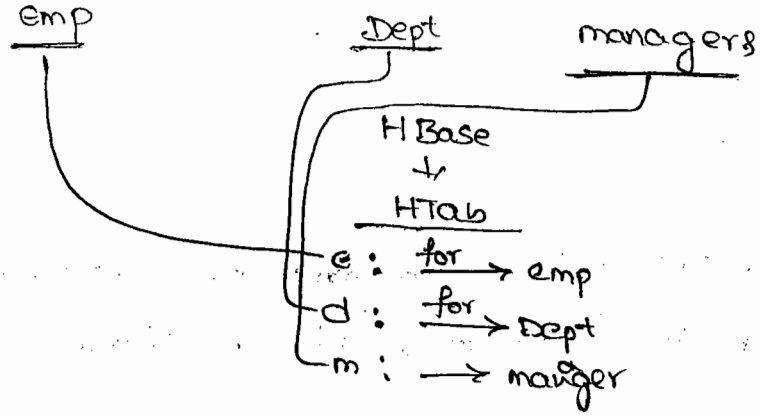
→ each cf should be associated with a row key.(access key)

→ Storage of object is = map

Algorithm is "map of map of map".

→ while migrating RDBMS schema into HBase table we keep RDBMS table as column family of HBase Table.

Ex:- RDBMS in normalised form



RDBMS vs HBase (data modeling) :-

emp

ecode	name	sal	DNO (FK)
101	A	10000	11
102	B	20000	12
103	C	30000	13
104	D	40000	14

Dept (PK)

DNO	dname	dloc
11	mrkt	Hyd
12	HR	Del

HTAB

(logical mode)

(Key)	call
101	e: name = A
101	e: sal = 10000
101	e: DNO = 11
101	d : DNO = 11
101	d : DNO = mrkt
101	d : loc = Hyd
102	11
103	11

physical model :- map of map of map :-

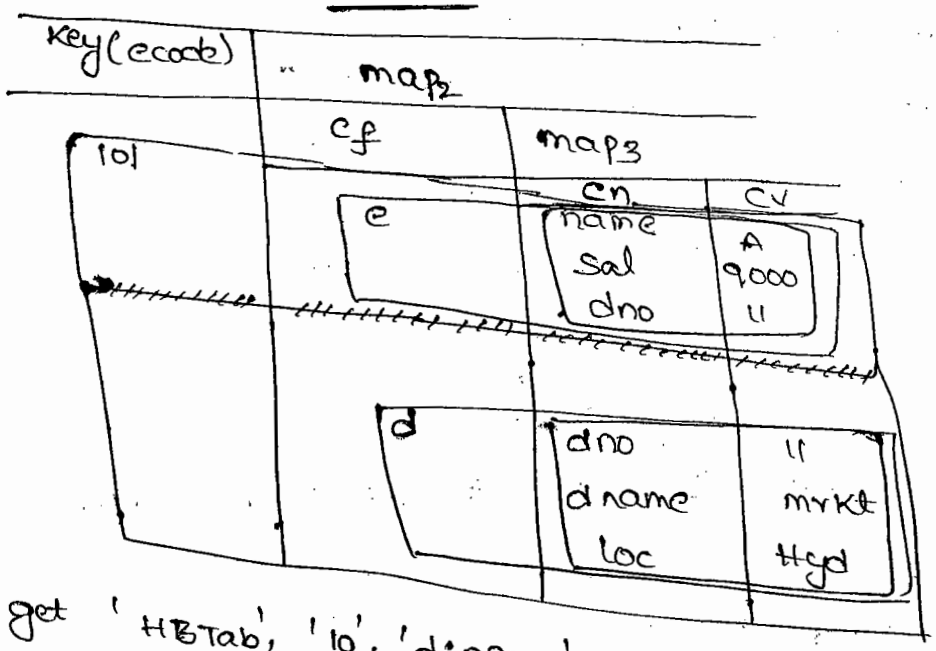
<101, <<e, <dno, 11>, <name, A>, <sal, 10000>>>,
<d, <<dloc, Hyd>, <dname, mark>, <dno, 11>>>>

<102, <<<e, <dno, 12>, <name, B>, <sal, 20000>>>,
<d, <<dloc, D>, <dname, Hr>, <dno, 12>>>>

<103,

Table formate :- (m of m of m)

Map1



get 'HBTab', '10', 'd:name'

get 'HBTab', '101', 'd'

get 'HBTab', '101'

get 'HBTab', '101', [e:sal ; d : name]

In Sql query:-

select e.sal, d.name from emp e join dpt d when e.dno = d.dno and ecode = 101;

9/11/15 :-

→ Columner is very faster

↳ faster CRUD over specific column of a Bigtable.

↳ it is a single table cassandra is good to eliminate join in online bigdata HBase

↳ 3 random records.

* Document Store :-

→ MongoDB (Mongo.org)

↳ Support providers

↓

10gen.com.

→ CouchDB (couchBase) → Apache.

* Advantages :-

→ It is pure Schemaless database

→ Runtime, user/developer is able to add

new, columns to table.

→ So that, Record to Record different Structures (Schemas) can be maintain.

Ex:- products

	Name	Brand	Size	price	model
①	TV	LG	21inches	80000	LX1
②	pendrive	LG	10GB (volume)	200	—

product to product different attributes maintained.

Ex:- profiles

①	Ravi	27	Hyd	single
	(name)	(age)	(city)	(mstate)
②	Giri	Del	married	2	Se
	(name)	(city)	(mstate)	Children	Design

Based on one attribute, other attribute is required

Ex:- mstat & children.

Document :- JSON Record → is called one document.

In mongoDB data stored in collection. (Tables) in RDBMS.

In collections data stored in BSON formate.

BSON → Binary notation of JSON record is called BSON.

Ex:- ObjectID

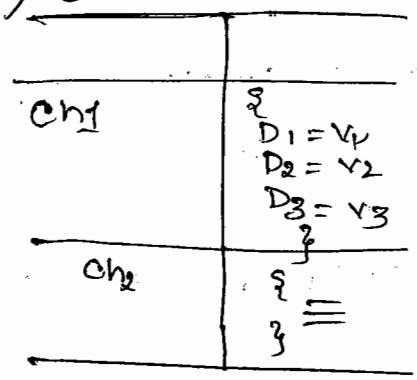
Ravi@gmail.com

```
{ "name" : "Ravi"
  "age" : "26"
  "sex" : "m"
  "pwd" : "x123"
}
```

Rani@gmail.com

```
{ "name" : "Rani", "Design" : "se"
  "age" : 26 "pwd" : "AxxB"
  "city" : "Del" }
```

Ex:- Youtube:-



Bigdata → {V1, V3, V9, V20, V30}

* Graph Store (DB):-

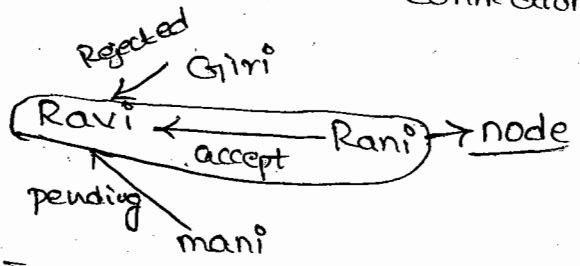
* NEO4J
  ↳ to store and process graph data.
  ↳ Linked records of table. (Record to Record)

Ex:- FB
Linked in

→ In NEO4J nodes are stores.
↳ one connection
↳ connection created between 2 called node.

Ex:-

U	F
A	C
C	B
B	A



Ex:- NEO 4J

Nodes	
U	F
Ravi	Rani
Rani	Giri
Giri	mani
mani	venu

N.get (Ravi)
↳ Rani

N.get / N.get / N.get / N.get (RAVI)))
↳ Rani
Giri
venu
mani

FRIENDS
my concern is mutual friends
U know the people.

MongoDB does not support Sqoop tool
↳ Hive will be integrated with all types of NoSQL database.

Hive & HBase } integration will accept

10/11/15

HBase

→ Is one of NoSql database comes under catagory "Column Store".

→ In HBase, columns Stored as Rows.

→ Each column should be associated with a column family.

→ HBase table should have atleast one column family.

→ No limit for max no. of column families

→ for one column family any no. of column can be given.

→ means that for a HBase table unlimited column can be given.

→ HBase has API for java python, even available /.net (c#) / Ruby / javascript / C++, nodejs / Angularjs.

→ This HBase is front end application (online)

↳ HBase is Bad for batch process.

How to Import Data from HBase to Hadoop :-

Those types are

* Using HBase API
* Hive & HBase integration
* Impala & HBase integration
* Pig has additional jar, by embedding jars into pig, pig also can work with HBase table data.

* HBase Interactive Environment :-

→ HBase shell

→ HBase shell command

How enter in HBase in Command Prompt :-

$ hbase shell ↵

hbase(main)>> // shell commands

In Sql
ddl
operations
- → Creating table (Creat)
- → Alter table (Alter)
- → drop table (drop)
- → Enabling and disabling tables.
  - Enable & disable
- → To manuplate
  - CRUD
    - → Scan
    - → GET
    - → PUT
    - → Delete

* (we are work with columns)

* performing CRUD operation HBase will work.

* Scan :- Reads all rows

* GET :- Random reading of a single row

→ to Record a record

* put :- * to insert a value to a column
* to update a column value.

* Delete :- → to delete column

→ to delete a column family all columns of the CF should be deleted.

similarly if delete all CF's of row, then row will be deleted.

→ put & delete : no aggregation function in available in HBase.

→ NO Row-filters.

Creation of table :-

hbase> Create 'Tab1' ↵

(Invalid)

↳ Because table should have at least one column family.

Ex:- hbase> Create 'Tab1', 'cf1' ↵

Ex:- hbase> Create 'Tab2', 'cf1', 'cf2' ↵ [single codes only]

Tab1 → cf1

('insert a data command)

Ex:- hbase > put 'Tab1', 'K1', 'cf1:a', 100

> put 'Tab1', 'K1', 'cf1:b', 200

> put 'Tab1', 'K1', 'cf1:c', 800

> put 'Tab1', 'K2', 'cf1:a', 400

> put 'Tab1', 'K2', 'cf1:b', 500

Ex:- Tab2 → cf1, cf2

emp

ec	n	s	dn
101	A	20	11

dept

dno	dname	dloc
11	mrkt	Hyd

hb> Create 'emp', 'e', 'd'

> put 'emp', 101, 'e:name', 'Amar'

> put 'emp', 101, 'e:sal', 2000

> put 'emp', 101, 'd:dname', 'mrkt'

> put 'emp', 101, 'd:design', 'Se'

18/11/15

How to

I$ hb

hbase

* if yo

> Crea

> Crea

> Cr

Check

> list

> Des

I want

> put '

> put

hbase

> Sca

> put

NOTE:-

Ex:- ①

> Sc

② if

> Sca

③ part

> Sca

> get

> get

> get

i want

> get

* Seque

* part

update

> put

13/11/15 :-

How to open HBase :-

∃ $ hbase shell ↵

hbase > list ↵ (All existed tables will show)

* if you want check particular table :-

> Create 'hbtable1', 'cf1' ↵

> Create 'hbtable1', 'cf1', 'cf2' ↵

> Creat 'hbemp', 'e', 'd' ↵

Check

> list ↵

> Describe 'hbempl' ↵

I want insert column record

> put 'hbempl', 101, 'e:name', 'Ravi' ↵

> put 'hbempl', 101, 'e:sal', 1000 ↵

hbase > put 'hbempl', 101, 'e:dno', 11

> Scan 'hbempl' ↵

> put 'hbempl', 101, 'd:dname', 'mrkt'

NOTE:- RUN time we can add any record to column.

Ex:- ① How to track specific columns :-

> scan 'hbempl', {COLUMNS ⇒ 'e:name'}

(upper case only)

② if you want multicolumn :-

> scan 'hbempl', {COLUMNS ⇒ ['e:name', d=dloc']}

③ particular complete cf want :-

> Scan 'hbempl', {COLUMNS ⇒ 'd:'}

> get 'hbempl', 102

> get 'hbempl', 101, 'd:dloc'

> get 'hbempl', 101, ['e:name', 'd:dloc']

i want complete employ details :-

> get 'hbemp', 102, 'e:'

* Sequencial All Record to Record read :- SCAN

* particular Record Read randomly we used :- get

update the values (like Sal) :-

> put 'hbempl', 102, 'e:sal', 40000

Check it :-

> get 'hbempl', 102

I want delete a column :-

> delete 'hbempl', 102, 'd:dloc'.

Check it

> get 'hbempl', 102

Data migrated in RDBMS to NOSQL,

How to bring data RDBMS to HBase

How to enter into MySql :-

J$ mysql ↵

$ mysql -u root ↵

Show database

mysql > Create database hbdemo;

> use hbdemo;

Creating one sample table :- (This is joined table)

> Create table samp (ecode int primary key, name char(10), sal int, dname char(10), dloc char(10));

> insert into samp values (101, 'aaa', 2000, 'mrkt', 'hyd');

> insert into samp values (102, 'baaa', 30000, 'mrkt', 'hyd');

mysql > select * from samp;

ecode	ename	esal	ednam	e location
101	aaa	20000	mrkt	hyd
102	baaa	30000	mrkt	hyd

= o = o =

Open 3 training hosts.

(1) HBase
(2) mySql
(3) Sqoop

data import from RDBMS to Hbase by using Sqoop import :-

feaching the data :-

J$ sqoop import \

> -- connect jdbc : mysql :// local host / hbase \
(jdbc: Uri; local host: hostname; hbase: database)

> -- username root \

> -- table samp \

> -- hbase - table hbnew \ → Create a table at runtime

> -- column - family e \

> -- hbase - create - table ↵

* Uri :- uniform resourcing identifier.

O/P :-

To imp

J$ s

* Hive

1) J$ my

mysql >

->

mysql >

J$ s

--

--

--

--

--

--

--

--

Ex :-

2) mys

mysql

mysql

mysql

mysql

mysql

mysql

mysql

mysql

o/p :- to check in hbase tab ↵

> list ↵

> scan

To import into a specific column-family :-

]$ sqoop import \

-- connect jdbc:mysql://localhost/hbdemo \

-- username root --table samp \

-- columns ecode, dname, dloc \

-- hbase-table hbnew3 \

-- column-family d- hbase- create -table ↵

* Hive and HBase integration :-

1) 14/11/15

]$ mysql -u root pdmdk

↳ if you have give any password

mysql > Create table samp2 (ecode int, name char(10), sal int,

-> dname char(10), dloc char(10));

mysql > insert into samp2 select * from samp;

] $ sqoop import \

-- connect jdbc:mysql://localhost/hbdemo \

-- username root \

-- table samp2 -m 1 \

-- hbase -table hbtest1 \

-- hbase -row-key ecode

-- column -family e \

-- hbase -create -table

Ex:-

2) mysql > Create table emp (ecode int primary key,

-> name char(10), sal int, dno int);

mysql > insert into emp values (101, 'aaa', 1000, 11);

mysql > insert into emp values (102, 'baaa', 2000, 12);

mysql > insert into emp values (103, 'caaa', 3000, 13);

mysql > insert into emp values (104, 'daaa', 4000, 14);

mysql > Create table dept (dno int, dname char(10),

-> dloc char(10));

mysql > insert into dept values (11, 'mrkt', 'hyd');

mysql > insert into dept values (12, 'hr', 'del');

mysql > insert into dept values (13, 'finance', 'hyd');

mysql > Create table denorm (ecode int, name char(10),
→ sal int, dname char(10), dloc char(10));

mysql > insert into denorm
→ select ecode, name, sal, dname, dloc
→ from emp e join dept d
→ where (e.dno = d.dno);

Check it:-

> select * from denorm;

hive > Create table hvimage1 (ecode int, name String,
salary, dloc String, dname String)

hive > Stored by 'Org.apache.hadoop.hive.hbase.
HBaseStorageHandler'

hive > with serdeproperties ("hbase.columns.mapping" =
":key, e:name, e:sal, e:dname, e:dloc")

> tblproperties ("hbase.table.name" = "hbase-tab1");

J$ sqoop import \
-- connect jdbc:mysql://localhost/hbdemo \
-- username root \
-- table denorm -m 1 \
-- hbase-tbase-tab1 hbtest1 \
-- hbase-row-key ecode \
-- column-family e \
-- hbase-Creat-table

Check data in hive;

Check data in hbase;

Ex ③ :- hive > Create table hvimage2 (ecode int, name String,
> sal int, dname String, city String)
> Stored by 'Org.apache.hadoop.hive.hbase.
HBaseStorageHandler'
> with serdeproperties ("hbase.columns.mapping"=
":key, e:name, e:sal, d:dname, d:dloc")

Check

J$ s

Check
hiv
HBase
olddc

RDB

NOTE:-
Hive
* how
hive >
$Sqoop

① Cre
② d
③ by
Q:- Im
Step①

> tblproperties ("hbase.table.name" = "hbase_tab2");

Check in hbase :-

list hbase_test2

]$ sqoop import --connect jdbc:mysql://localhost/hbdemo --username root --table denorm -m 1 --columns name, sal, ecode --hbase-table hbase_tab2 --hbase-row-key ecode --column-family e - hbase-create-table

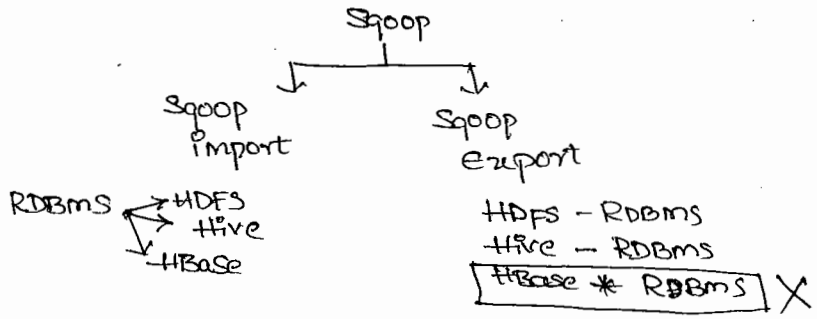
Check in hive :-

hive > select * from hvimage2

15/11/2015

HBase Syntaxis :-

olddata in RDBMS —sqoop import→ HBase



NOTE :- To export data from HBase to RDBMS. Hive & HBase integration is used.

* how to write data Hive to HBase :- file (101◊Ravi◊1000◊HYD◊-\001 00000-0

hive > Insert overwrite directory 'myhdfs' select * from hvtab1.

$Sqoop Export --connect jdbc:mysql://localhost/hbdemo
--username root --table etab1
--export-dir myhdfs/00000-0
--input-fields-terminated-by '\001' ↵

① Create Hive table

② dump data in hdfs

③ by using sqoop export data

Q:- Importing data from HBase to Hive table?

Step① :- Create integrated table (hvtab1)

→ Htab1 — HBase

Step②:- Creat hive table

Hive> Create table purehive (ecode int, name String, sal int, city String);

Step③:- Loading data

hive> Insert overwrite table purehive select * from hvtab;

Step④:- importing data from HBase to local directory

hive > insert overwrite local directory 'mylocal' select * from hvtab1;

Reverse Process:-

Loading data from (localfile /to HBase table)

Step①:- > Create integrated table (hvtab1) ↑table

> load data local inpath 'mylocal / emp.txt'

into table hvtab1;

↳ In this load, the load statement is expecting ↕ delimiter.

If file has (,) delimiter. --

Step①:- hv-tab1

Step②:- hive > Create table temp (ecode int, name string, sal int, City String)

row formate delimited fields terminated by ',';

Step③:- hive > load data local inpath 'mylac /emp.txt' into table temp;

Step④:- hive > insert overwrite table hvtab1 select * from temp;

(massive file we can load this type)

Copy data from one Hbase table to another HBase table :-

Step①:- Create hive integrated tables for both source and Target hbase tables.

table	table backend
h1 →	hb1
h2 →	hb2

Hive HBase

step② :- hive > Insert overwrite table h_2 select * from h_1;

Using HBase Shell commands ↑ (CRUD) we can delete a specific column fly. to delete rows massively.

step① :- mass deletion

> hive -> Integrated table (hvtab1)
↳ htab1

step② :- hive > Insert overwrite table hvtab1
Select * from hvtab1
when city != 'hyd';

* performing mass updation :-

put - shell command can only update a single column of a row randomly but not massively.

step① :- hive > Integrated table hvtab1 - htab1
↳ ecode, name, sal, city

step② :- hive > Insert Overwrite table hvtab1 select ecode, name, sal + (sal * 0.1), city from hvtab1;

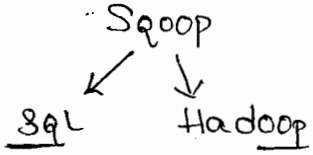
Aggregation over HBase table :-

hive > select Sum(sal) from hvtab1;

hive > select city, Sum(sal) from hvtab1 grap by city;

Sqoop

* Sqoop :-



Sqoop has two tools.

* Sqoop import
* Sqoop export

Sqoop import :-

→ To import data from RDBMS sources to hadoop.

Sqoop import can do:-

* RDBMS → HDFS
* RDBMS → Hive table
* RDBMS → HBase

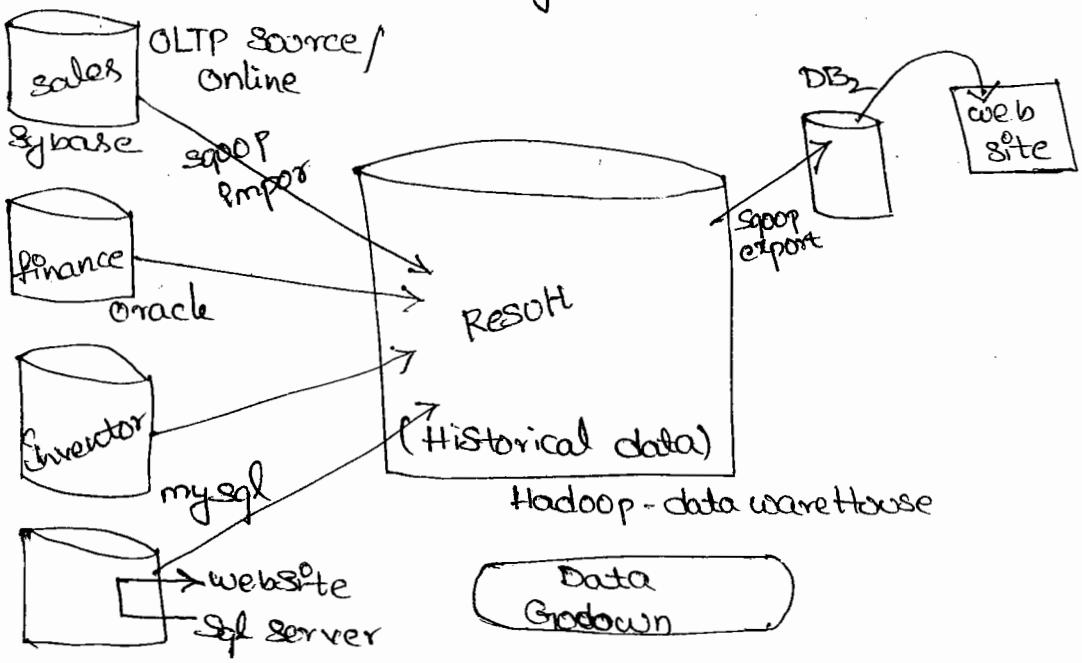
Sqoop export :-

→ To export data from Hadoop to RDBMS.

→ It can do

* HDFS to RDBMS table
* Hive table to RDBMS

→ To export data from HBase to RDBMS, we use Hive and HBase integration.



* Sqoop importing options:-

* **--connect** :-

→ to specify connect URI
(service point of view URL)

Ex:- ① jdbc : oracle : //db.yahoo.com / user db
↓ ↓ ↓ ↓
database drive — RDBMS — hostname — db name

② --connect jdbc : mysql : //192.300.400 / my db.
↓ ↓ ↓ ↓
DBdrive — RDBMS — HostIP — DBMS

* **--username** :-

→ To specify DB username.

Ex:- --username Dev001

* **--password** :-

→ To specify password text

Ex:- --password A123

* **--table** :-

→ To specify source table to RDBMS

Ex:- --table sales.

* **-m** (Sub option):-

→ To control no.of mappers by default sqoop imports initiate four mappers. Should be done if importing data.

Soboption
-m<n>
-m 1

* Why no.of mapper should be 1 if table doesnot have a primary key.

A:- By Default Sqoop initiate 4 mappers.

when multiple mappers are initiates DBC RDBMS) after Authentication & previllige is done, it Splits the data into small chucks.

→ No·of Splits = No·of mappers

→ 4 Splits = 4 mappers.

→ If table has 1000 Rows,

1-250 → Splits 1 $\xrightarrow{for}$ map1

250-500 → Splits 2 $\xrightarrow{for}$ map2

500-750 → split3 $\xrightarrow{for}$ map3

750-1000 → Split4 $\xrightarrow{for}$ map4

* map1 → has to start Reading begining of first Split.

map1 directly pointed to split1 begining because it's first Record.

* map2 → has to start Reading from 251th Record which is begining of Split2.

* Table has no primary key so map2 can not be printed to 251st Record. So map2 will be failed.

* Ilely map3, map4 will also be failed.

* Now only sequential Reading is allow.

To do so, no·of mappers should be 1.

* `---target-dir`

To specify HDFS O/p directory,

Ex:- -- table (sales)

-- target-dir [my dir]

O/p:- part-m-000000

* `--where` :-

Ex:- * -- where 'esal, 5000'

* -- where ser = '"F" and sal >= 7000'

* `--columns` :-

To import selective columns.

Ex:- * -- columns name, sal, dno

* -- columns name, upper(name)

* -- columns names, sal, sal * 0.1

The above expressions are invalid. In columns expression automatic and function expression not allow. Sqoop Reads tuan column name.

* `-- query` :-

It executes given select statement and imports data into target.

Ex:- for query option " $ condition" is mandatory with where clause.

```
-- query ' select * from emp
            where sal > 5000
            and $ condition in above
```

Statement there are two expression.

* emp → valid
  sal >= 5000 → valid
  } $ condition = true.

→ the default $ condition is false, when ever all given expressions valid, then these boolean variable turns to true. then equality will be started.

* if query start with (" ")

-- query " select - - -
- - - - -
from Tab
where $ conditions and city = "Hyd" "

* --hive - import :-

To specify destination has hive

* --hive - table my tab
↓
default

Ex:- ① -- hive- table urdb . my tab

* -- hive - create - table :-

To specify, if a given target table not exist create the table.

* --hbase - -table :-

to write into Hbase

Ex:- * -- hbase -table Hemp
* -- column - family e
* --hbase - row - key
↳ three are depend in Hbase.

* --bindir :- Locy
↓
To write object files (-class) into local directory.

* -- code loc 2 :-

↳ to writes source code into local directory.

* -- append :-

to write in existed directory.

* -- hbase - create - table

Sqoop Exports:-

* -- connect
* -- username
* -- password
* -- table
* -- bindir
* -- code
* -- export-dir
  to specify impat HDFS path.

* Sqoop exports, delimitor, which is that follow

`-- import -fields - terminated - by '\t'`

hive → ordb → mytab → mytab → ✧

my Sql

Tab* / abc (a,b,c)

-- target tab x

-- export -dir | user | hive | wh | ordb.db. mytab

-- input -- field - terminated - by "\001"

Ex:-

]$ sqoop import

-- connect mysql://localhost/sqdb

-- username jdbc:root

-- table sample

-- target-dir import31.

mysql > Create table damp (ecode int, name Char(10), Sal int, Sex char(1), dno int)

Ex:- ~]$ sqoop import

-- connect jdbc.y sql://local host/sqdb

-- username root - table damp -m1 -- hive -import -- hive-table dumps --append

NOTE:- while sqoop writting in hdfs ',' will delimitor, by writting in hive table "/001 (☆)" will the delimitor.

* If table has ',' delimitor then use this Statement:-

~]$ sqoop import -- connect jdbc:mysql://local host/sqdb - user name root --table damp - m1 -- target - dir / user / hive / warehouse / bamps --append

* If you chang the delimitor:-

-- target - dir import -- fields - terminated by 'it'

* NOTE:-

* both query and table are mutually execlusive.

* "- m1" is Soloption of table and query.

Ex:- ~]$ sqoop import -- connect jdbc : mysql :// local host / sqdb -- user name root -- query ' select name, sal, sal *01, sal * 0.2, sal + (sal * 0.2) - (sal * 0.1) from damp where $ conditions ' - m1 - -target - dir - import 13

Ex:- ~]$ Sqoop export -- connect jdbc : mysql :// local host / sqdb - - username root - - table sommary -- export -dir / user / hive / warehouse / result 1 / 000000 - 0

db.

name
into int]

C

=o= o=

* Hadoop Hardware configuration:-

Hardware Components	Meadium	High End
CPU	8 physical cores	12 physical cores
Memory	16 GB	48 GB
DISK	4 disks X 1TB = 4TB	12 disks X 3TB = 36TB
Network	1 Gb Ethernet	10GB Ethernet

Oozie

* Oozie is an ecosystem to define work flows and Scheduling work flows.

→ work flows :- Sequence Collection of steps.

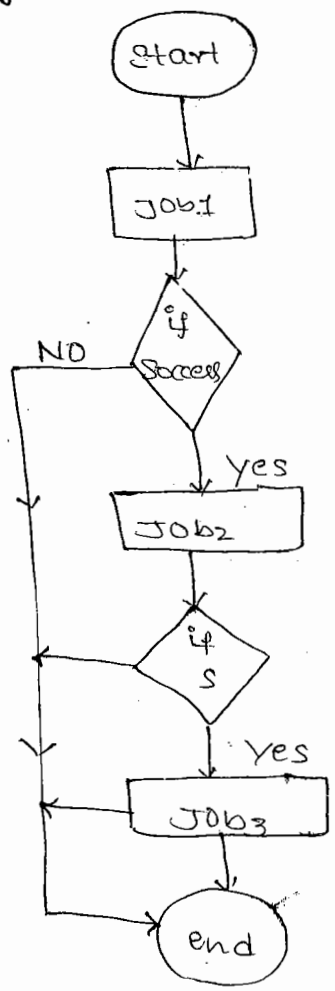
* A step is a Batch job
* The job can be hive / MR / pig / cascading / crunch etc.

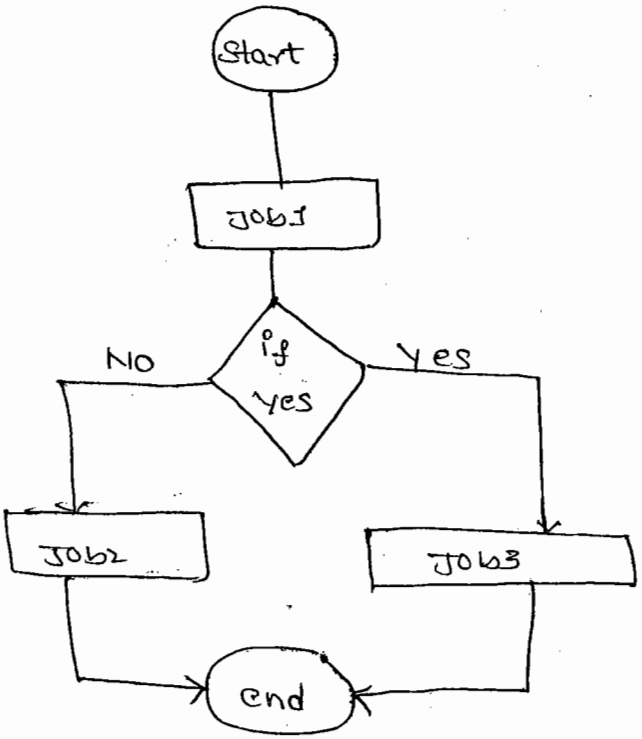
→ Oozie currently not supporting SPARK jobs.

* using work flows,

we can set dependency b/w jobs. But there is no direct connectivity between jobs. inform of input files and output file the dependency is set.

* visi Diagram of a work flow:-





* Scheduling of work-flows:-

NOTE:- Directly jobs can not scheduled

* Scheduling can be interval based or point of Time based.
  - interval → Hourly jobs
  - point of Time → So and so date & Time

* Oozie limitations:-

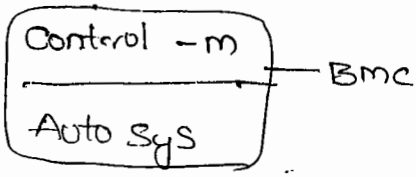
* Dynamic Scheduling is not possible.

Ex:- A flow is scheduled on every month first by 9.00 AM. There are some flows which should not be executed on holydays.

If next month first is holydaly (Sunday) The Scheduled should be prephoned or postphoned.

→ custome calender can not be defined

[follows standard calender]



→ Oozie can work with only hadoop jobs.

* Oozie -y → GUI

↓ Oozie wfs and Schedules can be coded in "XML"

*// To define workflow

<oozie - workflow --->

=} → actions

</oozie - work_flow>

*// To define Schedule

< oozie - coordinator --->

=

</oozie - coordinator>

* In workflow Tags used to define Jobs.

① Start → To start used to define execution of a flow from specific action (Job) onwards.

② End → to terminate the flow after final action.

③ If any action failed, it shows error message on console and moves flow execution control to end point.

④ Action → To declare a job as an action

<action> has two SubStages.

(i) <OK ---> → Success

(ii) <error ---> failure

If job success where to move flow control.

If job fails where to move flow control.

```
<action name = "act1">
<ok to = "act2"/>
<error to = "kill1">
</action>
```

* Style of killing :-

```
<kill name = "kill1">
<message> write program by having mind you
</message>
</kill>
```

Define a last job:-

```
<action name = "act10">
<ok to = "finish">
<error to = "kill10">
</action>
<end name = "finish"/>
```

⑤ FLOW → To call a workflow from another workflow. to define nested flow.

Ex:-
```
<action name = "act1">
<flow name = "workflow1.'xml'
<ok to = ?/>
<error to = ?/>
<action name = "act2">
<---->
<---->
```

wf1 → a1 - a100

wf2 → a1 - a100 | J101 | J102

ontrol.

trol.

nd you

ner

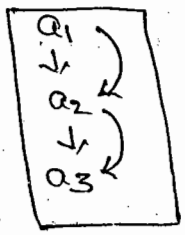
⑥ path → To define parallel actions.

< path actions = " act1 , act2 , act3" >

< action name = " act1" >

< act = act2 >

< act = " act3" >



* Deffning a flow :—

< oozie - workflow - - - - > parameters

< start action = " act1" /> [NOT mondatory]

① < action name = " act1" >

< ok to = " act2" />

< error to = " kill1" />

< /action >

② < action name = " act2" >

< ok to = " act3" />

< error to = " kill2" />

< /action >

③ < action name = " act3" >

< ok to = " finish" />

< error to = " kill3" />

< /action >

< kill name = " kill1" >

< message > First job failed < / message >

< kill name = " kill2" >

< message > 2nd job failed < / message >

< / kill >

< / kill name = " kill3" >

<message> 3rd Job failed </message>

</kill>

<end name = "finish" />

</oozie -workflow>

* Flow Scheduling :-

→ Is called coordinator Jobs.

<oozie - coordinator - ---->
≡ ↳ Scheduling parameters

</oozie - coordinator>
name = "cord1"

Ex:- < oozie -coordinator name = "cord" Frequence = 60 >
No. of Minutes

< workflow > flow1.xml </ coork flow>
↳ a1 a2 a3

</oozie - coordinator>

* Scheduling parameters :-

① Frequence :- To schedule on interval based.

② Mode :- daily (defaut) [It don't Say option]
= monthly
= yearly
= weekly

ⓐ Mode = " daily", Frequency = " 60"

ⓑ Mode = " daily", Time = " 6:00 Am"

ⓒ mode = " daily", Time = " 6:00 Am",
" Holidays =" wednesday"

(d) mode = "daily" Holidays = "Saturday"

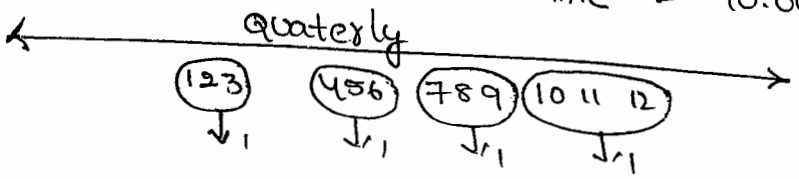
(e) mode = "weekly" (SUNDAY)

(f) mode = "weekly" day = "wednesday"

(g) mode = "monthly", day = "1", Time = "9:00 pm"

(i) mode = "yearly"

(j) mode = "yearly", month = "3", day = "15",
time = "10:00 pm"



No. of Minutes
ence = 60>

< oozie - coordinator name = "Q1"
mode = "yearly", month = "1", day = "1" >
< workflow name = "flow 10.xml" />
< /oozie - coordinator >
< ——— name = "Q2".
mode = "yearly", month = "4", day = "1" >
< workflow name = "flow 10.xml" />
< /oozie - coordinator >

FLUME

* Streaming data & Analysis :-

* what is Streaming data?

A) Data which keeps generated by other, Sources is called "Streaming data"

Ex:- logs, -- App log, DB log, os logs, web log---
live cams, chapter, files.

* Streaming data Collection tools :-

Flume, Kafka

* Flume can collect Streams and can write into Hadoop Can/ JMS (Java messaging Service)

* Kafka Can collect streamed events, Can write in hadoop / non-hadoop systems.

In both cases,

Source systems are called producers Target systems are called Consumers.

Target systems are called consumers.

Flume/ Kafka are called ~~consumers~~. broker Systems. are also called "Buffered Systems".

* why production applications can not write event into hadoop?

---(i) problem is with block size of hadoop 64mb(old) / 128mb (new)

--- (ii) hdfs is not accepting records.
it accepts only files.

--- (iii) if production application writes 1000 events. into hdfs by using hdfs API.
So 1000 files will be created.

As per hdfs behaviour, each file to be blocked as 64mb and each block to be replicated for 3 times.

So 3000 blocks will be generated.
3000 meta records to be maintained by Namenode Server.

Sol:- single event should not be as bunch of events should be written to hdfs.
that means, events to hdfs be buffered.
What happens if events buffering is done at production Servers?

A:- High stress on production server
-- production application performance will be down.

What happens if events buffering happend at hadoop, once buffered threshold reached, if data written into hdfs?

Because Hadoop has high capacity, that too H can buffer events in distributed passion.

while application write into hdp buffers if network connection failed with app and hdp.
all buffered data will be lost.

→ Thats why buffering of events should not be done at hadoop level.

---→ If hadoop is buffering events from 100s for 1000s of production apps (Server)

very high load will be hadoop · which can dominate hadoop capacity.

Thats why we need a seperate broker system which has very large capacity stream and buffer.

once
into
* Big
*→
Online
(a)
(b)
Flume
Stori
Triden
* The
event
into
Storm
* Anal
Re
m
k
* Rec
Beoz
later
Ex:-
base
Ex:-
into
* Mic
Ex:-

once buffer threshold reach, these brokers can write into target systems.

* Bigdata application can be classified into two
  *→ online *→ batch (Hadoop)

Online:-

(a) transaction -- nosql

(b) Streaming -- flume, kafka, Storm / Trident / Spark - Streaming.

Flume / kafka -- category 1

Storm --- category 2

Trident / Spark - Streaming - category 3

* the flume and kafka can only Stream filtering events and little transformation and write into hadoop. bcoz longterm they cannot hold data.

Storm --→ live Analytics
  └──→ Real time Streaming Analytics

* Analytics process types:-

  Real times - Storm

  micro batch process -- Spark Streaming

  batch -- Hadoop

* Real time Analytics (near to real-time)

  Bcoz transaction or event first hits appServer, later Storm can collect.

  Ex:- Smart gps systems, which can guide you root based on current traffic.

  Ex:- Maintaing attendance, when ever user entered into office.

* Micro batch System in real time:-

  Ex:- Atm user should not do more than 5 transaction

and we should overcross limit of 50000.

Micro batch System in Batchs:-

From each branch of the bank, in last one hour, how much deposites collected.

Batch System:-

- Recommender Engivers.
- common products of two pairs of users.
- monthly bank statements.

to perform micro batchs in real time, you should run "Spark Streaming" (processor) on "Storm" clustor.

To perform micro batch in Batch..
you should run "Spark Streaming" on "hadoop"
Need of Storm and flume or Storm and kafka to integrate Storm with hadoop and also non hadoop.

Spark

Spark is execution frame work, which is replace for mapReduce.

→ Spark execution model is DAG
Direct Acyclic Graph.

→ Spark needs a cluster, cluster can be Aws, Hadoop, mesos, Storm --- etc.

* Spark Componerts:-

* Spark Core:-

on top of "Spark Core" diff Components.

(1) Spark Sql:-

--- to access data of Hive and Impala.

by running Sql statements, you can process in batch style.

* Spark ML Lib:-

Is a machine learning library. all predefined algorithms for machine learning Available.

Ex:- Credit card fraud Detection.

* Spark Graphx:-

Is a library to process graph data. All "Graph data algorithms" available.

* Spark Streaming:-

Can collect data from online streams (Storm). and from batch systems (Hadoop) and perform "micro Batch".

Ex:- detecting "Fraud Transaction" in live., is responsibity of "Storm", in recent hour, how many "Fraud Transactions" happend :- is responsibility of Spark Streaming.

Spark != Spark-Streaming.

* Flume and Kafka ---->
(old) (new)+(advanced)
(purpose is same)

* Flume:-

Flume initially developed by "Cloudera", later it hand overd Flume to Apache.

Apache developed "Flume-NG" (next generation) by which you can build mountains of data.

In Flume-NG -->

no limit for nodes, similar to Hadoop.

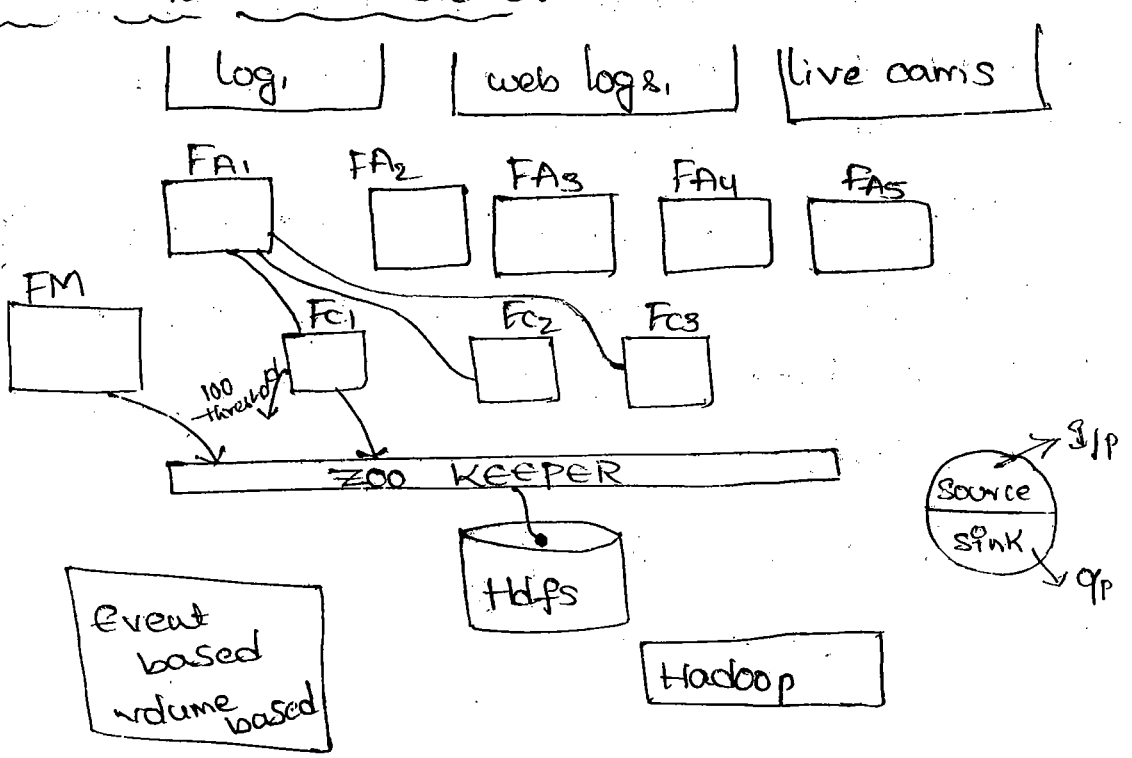
upto flume version 0.9 is just a flume, after flume 0.9, it is flume-NG.

Flume drawbacks:-

* can only import Streams.
* can not perform transformations.
* can not reply back to databases. (via JMS)
* No guarenty that each dump hits the target.
* data dump is based on, only when threshold is reached, not as interval based.

Threshold can be, event based or volume based.

Flume Old Architecture:-



* Flume has two types of nodes.
  (1) physical node
  (2) logical node

* physical node:-

Each jvm machine is called physical node. physical nodes are two types.

* manager node (master)
* worker node

* manager node responsibility:-

* Task assignment
* load balancing
* allocating resources.
* fault tolerances
* health checkup of workers.

* Workers:-

worker nodes are two types.

* flume agent
* flume collector

→ flume agent:-

It collects events from streaming source and writes into flume collector.

→ flume collector:-

It collects events from flume agent, and buffer it till threshold reach it and writes into object once threshold reached. as on single file. Threshold can be event based or volume based.

Ex:- 5000 events

2GB volume

* Threshold is user choice can be configured at flume configuration file.

Once data is written into hdfs, the buffer data at flume collector will be deleted.

So that it can collect new events.

Before data dump if coll clashes the buffer data will miss. thats why no guaranty all events will reach the target.

* Logical nodes:-

In the each worker node two types of logical nodes are running. There are

* Source
* Sink

Ex:- <ln1>

<source> Twitter </source>

<sink>

hdfs:// Local host : 8020 /user/mydata

</sink>

</ln>

* Flume - NG :-

In flume -ng only two types of nodes.

* manager node.
* worker node.

** In flume -ng, Data flows with be executed.

A data flow is called an "agent".

Agent has 3 components.

* Source
* Channel
* Sink.

Source :- Is a flume System Service which is responsible for recieving events from 'Streaming Source" (ex : logs).

and writes to channel.

On each attempt, Source collects one event, when that event is written to channel, source gets an ack (acknowledge) next, and starts attempting to recieve next event.

* Channel:-

Channels are "buffered Areas" For events based on capacity configured for channel.

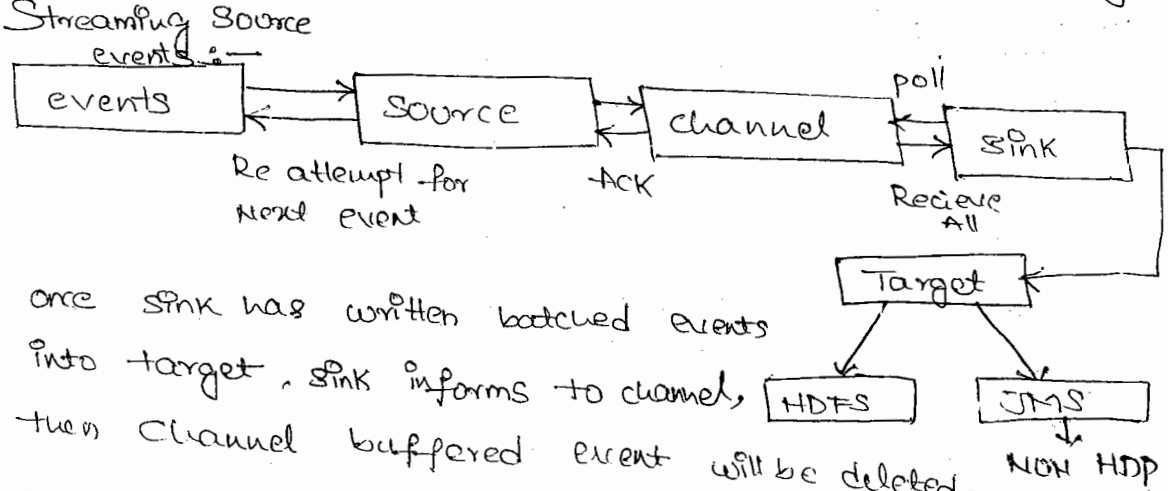
channels buffered for channels buffers all events. once capacity threshold reached, informs to Sink.

* Sink:-

Collect all events of channel as mini batch. and write to Target.

* Streaming Sources:-

Source ⟶ channels ⟷ sink ⟶ Target.

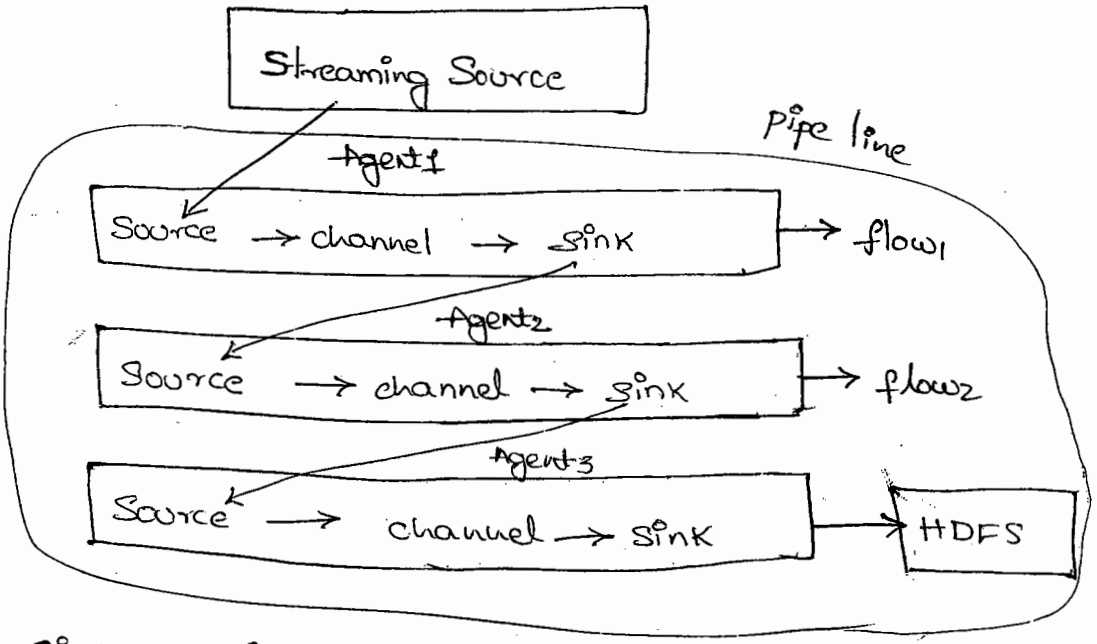


once sink has written batched events into target, sink informs to channel, then channel buffered event will be deleted.

* Flume pipe line:-

Agent:- An agent can write a target or write to another agent the data journey b/w agent to agent is called pipeline.

unlimited no of agents you can keep in a pipeline.



Sink of first agent writes to Source of agent2
Sink of agent2 writes to Source of agent3.

once each sink has written to its next hop, channel data will be deleted.

to perform Complicated filters or transformation these data pipelines are used.

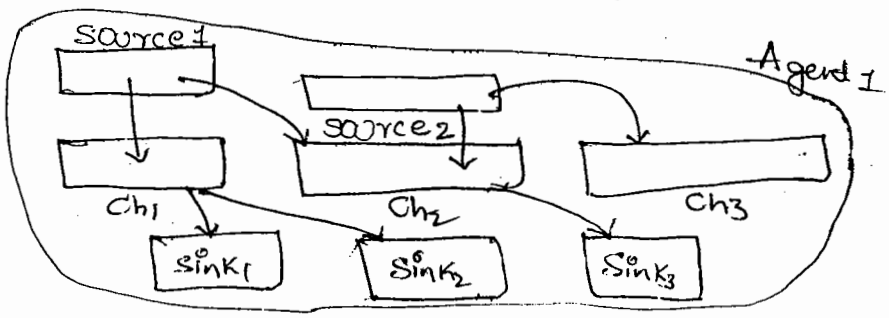
Advantages pipelines:-

* Complex trans-formations.
* reusability

A single agent can have multiple sources. Multiple channels and multiple sinks.

That means

Single source can write to multiple channels
multiple sources can write to single channel
multiple sinks can collects events from single channel.
one sink can collect one particular channel.



* Event :-

Event has two info:-

headers, data

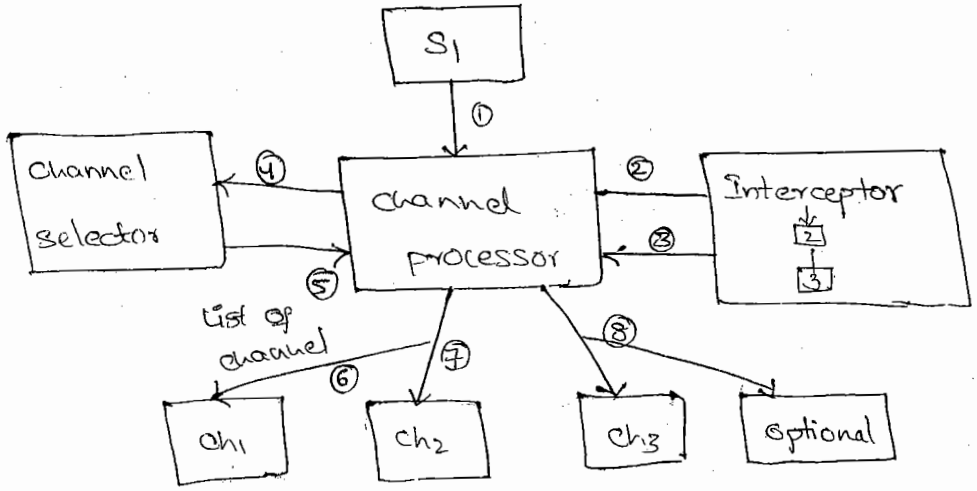
Headers ---> time stamp, Source

* before source writing into channel following process will happen :-

→ Channel processor,

→ interceptor,

→ Chamel Selector.



* process b/w Source and Channel :-

* Channel process
* Interceptors
* Channel selectors

Channel processor :-

Each Source as one Channel processor. when ever Source collect event it passes to channel processor.

if no transformations / filter are required. channel processor pass the event to channel Selector,

it transformations are required it passes to interceptors.

A Source can have multiple interceptors. but these multi interceptors executed one by one.

once final interceptor completed its execution, it passes to channel processor.

Channel process will be the modified event to Channel Selector.

Based on headers of event, channel Selector will select channels for the event.

Ex:- ch1, ch3, ch7

These channels for can be regular or optional (dependson Configuration)

The channel processor will wright event to given channels.

during this write, if write process failed, channel processor will throw channel Exception to Source

The Source will recollect the event from Streaming Source.

If this failure happend for optional channel simply Channel processor will ignore it.

* Interceptors:-

-- it does filter,
-- modify data or headers
-- adding new info to data or hearders.

Ex:- Filtering based on regex (regular expression)

we have predefined interceptors, at same time you write custom interceptors.

based on Order given in configuration file for a particular Source, these interceptors will be executed on after one.

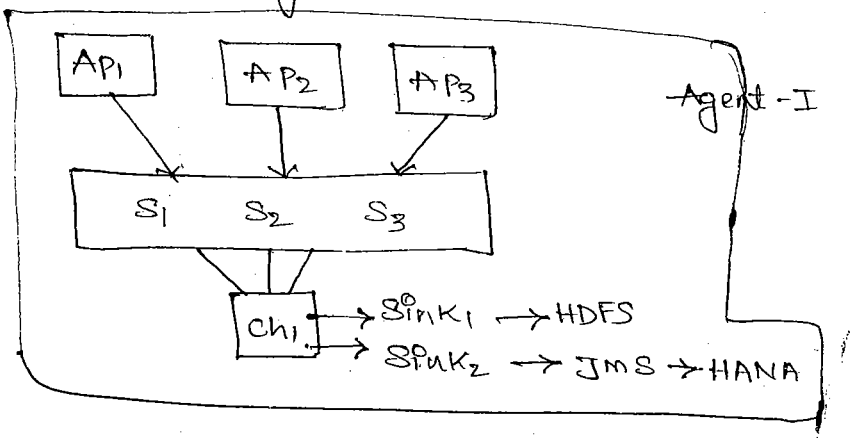
during this interceptor flow if any interceptor failed, interceptor flow to be re executed from begining.

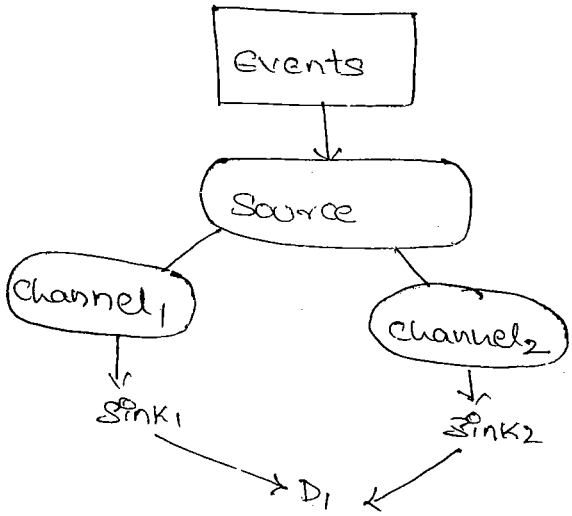
the last interceptor will write event to channel process.

* Streaming Source → Source → Channel processor → Interceptor flow → channel processor → channel selector → channel processor → channels.

All components of an agent, can be run on a a single worker, or can be run on different work-ers, Flume manager keep tracking of all these processer. Fault tolerance implemented by manager.

* Each source has only one channel processor. One agent has only one selector.





duplicate channel for high availability

Process between Channel & Sink:-

* Sink runner
* Sink group
* Sink processor (Sink).

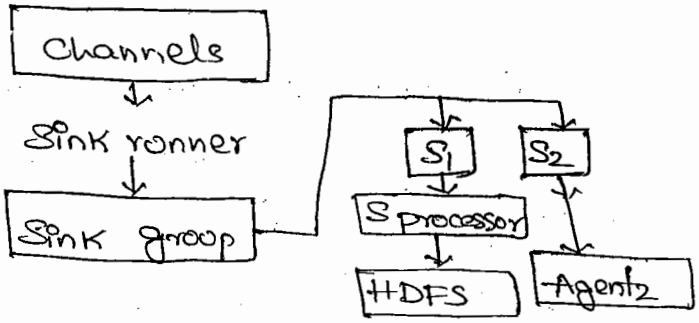
* Sink runner:-

When channel reached threshold (limit) channel informs to Sink runner.

Then Sink runner initiates Sink group.

Sink group can have multiple Sinks.

all Sinks will write data into next hop or target.



Events → Source → Chaprocess → interceptor → ch-pr → ch-selector → ch-pr → channels → Sink runners → Sink groups → Sinks → Sink process → Target / next-hop

* How to Submit flume -ng Jobs:-

$ Flume -ng help

$ flume -ng version

$ flume -ng agent a1 -conf -c
< configuration file path>
--- other parameters.

* Sample configuration file:-

Save this file writes extension .conf

agent.sources =s

agent.channels = c1

agent.sinks = k1

agent.sources.s1.type = netcat

agent . sources . S_1 . channels = C_1

agent . source . S_1 . bind = 0.0.0.0

agent . source . S_1 . port = 12345

agent . channels . C_1 . type = memory

agent . sinks . K_1 . type = logger

agent . sinks . K_1 . channels = C_1

SPARK

↓

Execution Framework

Cloud-Application
Data Retails

Day to Day

Eg:- HipKart
BigBasket

Hadoop petabytes

SAN → Bigdata → Storage / processing

HDFS

process

unlimited

Mesos → Limitations "4000 slave" nodes.

Server → clusters

* Why Spark + Hadoop required :-

Best solution why Spark + Hadoop ?

Hadoop suits & fits best?

Distributing process System

Spark + Hadoop + mongo DB + mesos + AWS cloud

parallel process

* can be used with any Spark + TD + Netezza + Hadoop + Storm + Vertica

* Load balancing

* Health monitoring

* Fault Tolerance

Shared Nothing arch → DFS

Distributed file System

master

slaves

cluster

File → Blocks

divided into Block Size

B1 B2 B3

- 64 MB
- 128 MB (latest v.2.0)

replica (x3) default)

Name Node | Job Tracker | SNN

DN1 TT1 | DN2 TT2 | DN3 TT3 ...

* Blocks → replicas — can not used as Backup / Storage

↓

Localized processing

↓

Shared NOTHING

Diff B/W

Simultaneously	parallel process
5sec | 5sec | 5sec	XXX
executing	XXX
	5sec

Record distributed

Define Streaming process

Eg:- Live Analytics

Storm TB (32 KB)

Block distributed

Type I

Type II

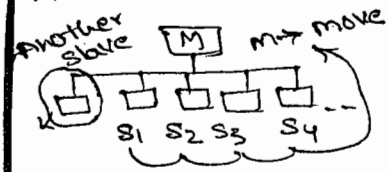
Hadoop ✓

Spark ✓

Spark Hadoop

Netezza (64KB)

Netezza cluster → Type I not suitable



M → move load → to avoid it

512 nodes limitation

Type II — SPARK + Hadoop → integrate together → Big data storage with great speed.

Direct Acyclic Graph → DAG → super speed calculations operations → But NO storage

unlimited storage

processing ← MR / DAG (V2.0) ← maths * Abacus

Spark → DAG → 1TB (Sort)

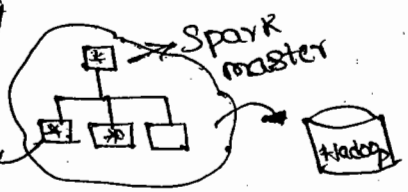
* Very fast Execution (than MR)
* NO storage
* Can run on → AWS (Amazon web service) → cloud system
  → Mesos
  → Hadoop
  → mongoDB

Hadoop → MR → Transformations cannot be re used
*NOT good for Iterative Algorithms

Hadoop → DAG → TEJ / IMPALA / SPARK → pig / Hive (honeybee) → Disk Based
SPARK → you can use in → Disk Based
SPARK → In memory computing Based ***

Faster → Eg:- brain in memory computing

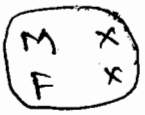
* vast / more transformations complicate → slaves / workers
* algorithms



DAG model

Eg:- Sorting

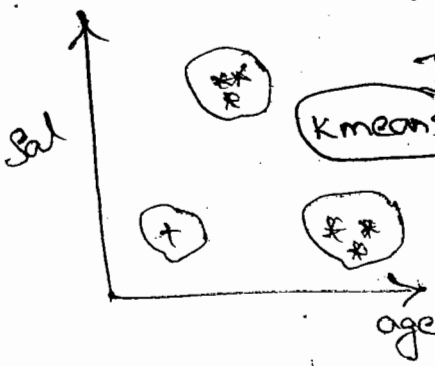
Sal



* select sal, max(sal) from emp group by sex;

Ex:- JOBS → get / will n't get

Bank loans → will pay / will n't pay (defaulter)

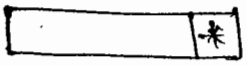


Kmeans — max

KNN | DT/RF → are iterative algorithms highly

Eg:- flipkart, youtube → Recommender

Search Engine about product's based on History.



iteractive algorithms →
1) Unique products
2) product count
3) frequency — Ranking
4) Count with threshold

C1	P1 P2 P7
C2	
C3	P2 P3 P5 P1
C4	P1 P2
C5	P3 P5
C6	P3 P2 P5
C7	P1 P3 P4
C8	P1 P2 P5 P8

products

P1 P2 P3 P4 P5 P7 P8
(7) count

timer

P1	5
P2	5
P3	4
P4	1
P5	5

* Based on priority eliminate the rare item used

Set priority

P1 - 6 - 1
P2 - 5 - 2
P5 - 5 - 3
P3 - 4 - 4
P4 - 1 - 5

P1 P2
P1 P2 P5
P1 P2
P1 P5
- - - -
- - - -

after Elimination

Again add

Same

Again add

P1 - 4
P2 - 3
P5 - 1

* Frequent pattern Algorithm
* Advance Analytics
* Recommender search Engine
* predictions

* Spark :- is Effective execution framework for large clusters such as [mesos / AWS / Hadoop] Spark has an execution model called

DAG [→ Direct Acyclic Graph] which maintains graph (Links) of RDD's (Resilient distributed Datascience). Similar to HDFS blocks.

RDDS :-

→ will be distributed across different worker Nodes
→ Can be partitioned as Sub pieces of data. These partitions will be distributed.
Default no. of partitions ('1')
→ are distributed into RAM ; of worker. If RAM Space is not available then RDD's Stored in Disk.

DAG engine maintains graph of RDD's

Eg:- which RDD (name of RDD)
How many partitions
Address of partitions
location of partions (RAM/HD)
what is Input RDD

Similar to meta data of HDFS file

RDD's are of two types:-

* Transformations — Sort, map, flatmap, filter, groupBy --- expressions
* Actions — count | saveAsTextFile | collect | save (aggregations ---)

*RDD's are LAZY it means RDD's will be computed & distributed when ever actions are performed on RDD's

Eg:- FIP → file1

(RDD1) ← (RDD2) ← (RDD3) ← (RDD4)

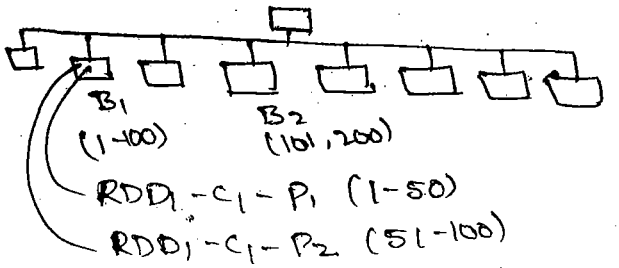
Transformation

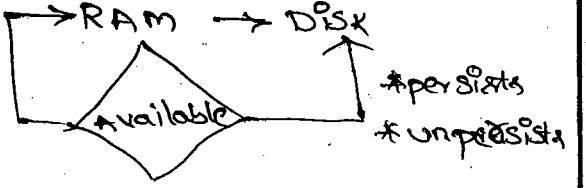
* RDD1 count()
  action *
* pig Relation ~ dump Store

RDD4 Save As Text file(',') → Action

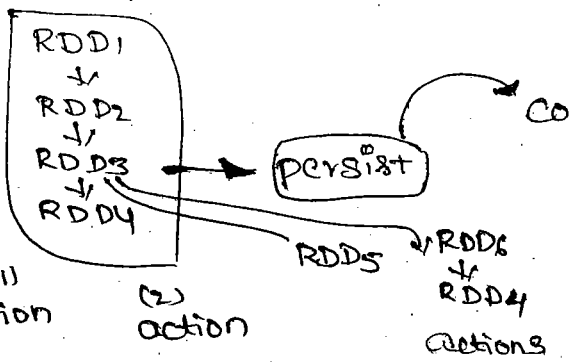
Eg:-
```
val RDD = sc.textFile("hdfs://-/user/f1",2);
RDD1.count()
```
declaration
action

P1 (10, -150)
P2 (15, -200)





Eg:-



common | Regularly using make it persist

* Once RDD's stored in RAM it will be available there till another RDD comes (dependent | independ). Later prior RDD will be moved to DISK. When RDD is in RAM if any action ↓ performed on the RDD, no need to transformation

Recompute.

Once RDD is available in DISK then Recomputation is required.

Eg:- RDD1 → RDD2 → RDD3

RDD1 → available in System → S1 (RAM)

RDD2 → " → S2 (RAM)

Now performing action of RDD2 Save

In this example RDD2 o/p is available in

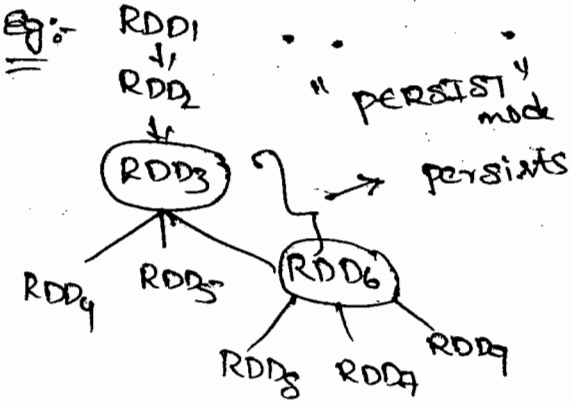
RAM, so no need of Re-computing.

* When RDD's will be terminated from RAM?

1) when next RDDs comes
2) when flow execution is completed

* To keep regularly used RDD into RAM permanently keep the RDD in persist mode.

Eg:-



From above eg:

action is performed on (RDD4)

RDD4 →
1) RDD1
2) RDD2
3) RDD3
4) RDD4

When action is performed on RDD4 i.e RDD1 - RDD4 will be completed, if not persist all remove from RAM but RDD3 is persist so RDD3 stay there.

when action is performed on RDD8 which inputs from RDD6 input from RDD3. Now following flow will be executed

RDD3 (RAM)	persists (stay there in RAM)
RDD6 (RAM)	
RDD8 (RAM)	

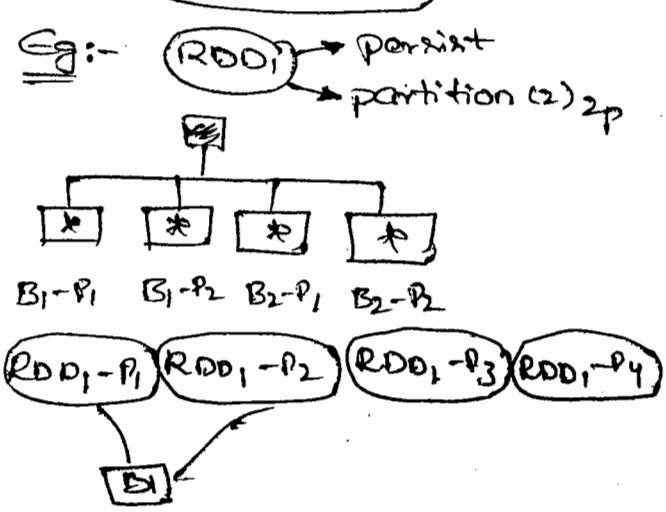
* without maintaining Replicas (x3) - Spark capable of fault-tolerance maintainence

Mode
* persist
* UNPERSIST

(RDD3) - PERSIST mode
1) Forcebly unpersist
2) Shut-down server
3) System Crashed

These ways we can stop

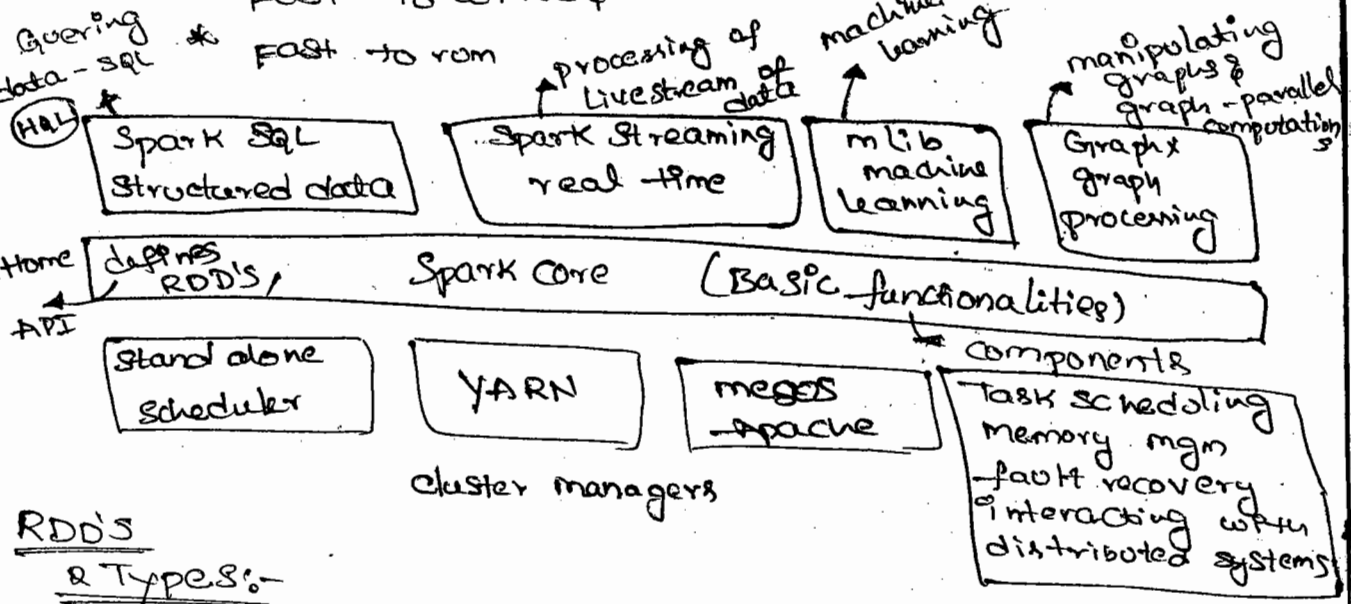
Eg:-



tackle Bigdata sets - API's in Python Java & Scala

Apache Spark – open source cluster computing system that make data Analytics.

* FAST to write &
* FAST to run



RDD's & Types:-

1) Transformations:-

- map()
- filter()
- flatmap()
- Sample()
- GroupByKey()
- reduceByKey()
- Union()
- Cogroup()
- Crossproduct()
- mapvalues()
- Sort()
- partitionBy()

2) Actions:-

- Count()
- Collect()
- reduce()
- lookup()
- Save()

* Spark :- RDD1 RDD2 X RDD3
System down recomputing

* fault tolerance - No Block replica's needed RDD's
→ means RAM

PORSIST :-

- MEMORY
- MEMORY-SER
- MEMORY-AND-DISK
- MEMORY-AND-SER

MEMORY → RDD stored in RAM (all partitions) in unserializable formate more memory consuming

MEMORY SER → (Deserialization) RDD stored in RAM in serialized form (less space (so less) all partitions)

→ In both cases if RAM space not available RDD stored in DISK.

Eg:- RDD1 [P1 P2 P3 P4]

* Note:- when RDD contains multiple partitions. If partition is not available able to load in RAM. then all partitions of RDD will be stored into DISK.

* If memory space is concern then prefer memory-SER Recovery is concern then use MEMORY.

* Generally serialized data will occupy 2 to 5 times less space than deserialized data, so less occupancy in RAM.

MEMORY-AND-DISK → If RAM space is available then partitions will be loaded into RAM.
For any partition if RAM not available then only that partition will be loaded into DISK.

RDD

MEMORY_DISK_SER ~ same as MEMORY_DISK

Storage format → Serialized

→ Storage format → unserializable

Latest versions Spark 2.0 onwards . MEMORY – DISK – SER2

SER3 → Replicas for partitions

* How to Create RDDs :-

* when a file is loaded into object then RDD will be created.
* when you parallelize an object the RDD will be created in LAZY mode.

Eg:- val rdd1 = sc.textFile ("hdfs://myhost/object's user/mydata");

val a=10;

1RDD

Eg:- val =10;

val n = Array (10,20,30,40,50,60)

valy = sc.parallelize (x,3) no. of partitions

40 objects ; 2 RDD's

y.count(); → action performed

Eg:- val v1 = sc.textfile ("samp.text"); → 101 AA 12000 / 102 BB 13000

val v2 = v1.map (_.split (","));

val a = Array (10,20,30,40,50);

val b = a.filter (_<30);

2 RDD's

* Converting one scalar to RDD (sc.parallelize (...))

Sc.parallelize (a,4) ← partitions

(Sc.textfile(...))

RDD.collect() → Action
RDD.saveAsTextFile("_") → Action

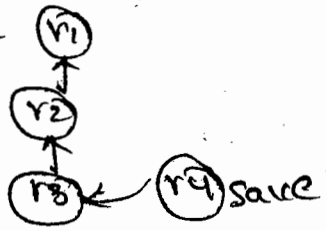
Books (download)
- Learn Spark
- " Scala
- Spark cookbook
- Scala cookbook

Eg:- Comment.txt

Spark is great
Spark good with hadoop

```
val r1 = sc.textFile("../comments.txt")
val r2 = r1.flatMap(_.split(" "))
val r3 = r3.map(x=>(x,1))
```
Tuple → collection of fields

RDD




```
val r4 = r3.reduceByKey(_+_)
r4.saveAsTextFile()
```
pig pt of view → sum aggregation
↳ action

In Scala every thing is an object

* Func()
* objects()
* r4.count()

Launch cloudera manager

```
Scala > sc
Scala > val a = 10; → immutable
        var a = 10; → mutable
```
val : is immutable i.e cannot be reassigned / changed
var : is mutable

```
Scala > val b = 3000
      > val b = 4000  (Killing & Reassigning)
      > val a = Array(10,20,30,40,50,60)
```
Scala object

To make it RDD :-

```
> val R1 = sc.parallelize(a); → 1 partitions
  val R2 = sc.parallelize(a,2); → 2 partitions
> R1
> R1.collect(); → action
> a
> R1.saveAsTextFile("Local Directory");
Scala> ls localdirectory  [part-00000] , partition
     > cat localdirectory/part-00000
     > R2.saveAsTextFile("Local dir2")
```

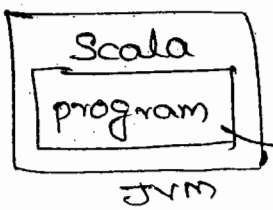
part - 00000
part - 00001 → 2 partition.

Scala > cat comment
-Spark is great, spark is good with hadoop

Scala > val read1 = sc.textfile ("comment")
Scala > read1 ↘ lazy mode
Sala > read1.collect()
" > val read2 = read1.flatmap (_.split (" ")) → RDD
> read2.collect();
> val r3 = read2.map(x ⇒ (x,1))
word,1
Compile shape
> val r4 = r3.reduceByKey (_+_)
> r4.saveAsTextfile ("wcresult1");
> ls wcresult1
> cat wcresult1/part-00000 → Output

(Spark 2)
(is 2)
(great 1)
(good 1)
(with 1)
(hadoop 1)

functional programming + oops features

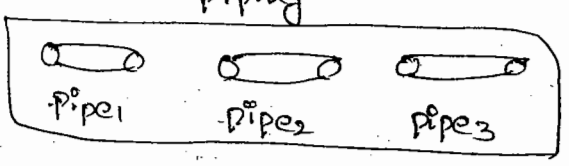


* Simple form
Single Statement :-

> val read1 = sc.textfile ("comment")
> val res = read1.flatmap(_.split(" ")).map(x⇒(x,1)).
reduceByKey (_+_);
> res.saveAsTextfile ("hdfs://quickstart.cloudera/user/cloudera/wcresult2)
↳ saves in HDFS
local host

> hadoop fs -cat wcresult2/part-00000

Sala > val r1 = sc.textfile ("hdfs://quickstart.cloudera/user/cloudera/linkage);
> r1.count();
> :quit



> cat
ravi
Rani
Raju
Lath

> va
> val
> va
> tes
> tes

Eg2:-

Scal

simple form

Eg3:-

>

> cat emp

ravi	M	50000
Rani	F	100000
Raju	M	20000
Latha	F	10000

Select sex, sum(sal) from emp group by sex;

Cloudera@quickstart > spark-shell ↵

Scala > * spark-shell --master ↵

* spark-shell --master yarn-client → Hadoop 2.0

> val arr = Array(10,20,30,40,---100);

> val rtest = sc.parallelize(arr, 5); ↳ 5 partitions - [part-00000, part-00001, ≡]

> val testres = rtest.collect(); Now I want only 1 output data

> testres // one scala object

> testres // make scala object as RDD as one object

Eg2:- > val e = sc.textFile("emp");

> val e2 = e.map(_.split(",")) // e2 is RDD

> val test = e2.collect();

> test

4 Rows → as single Array

Array 0,1,2,3 4 elements

Array(ravi M 50000)

M

> test(0) - 1st element

> test(0)(1) - In 1st element 1st element

(∵ test.map // local scala object)

Scala > val e3 = e2.map(x => (x(1)), x(2).toInt))

> val test3 = e3.collect();

> val res = e3.reduceByKey(_+_)

> val testres = res.collect()

> testres

(M 50000) (F, 10000) (M, 20000) (F, 10000)

simple form > val result = e.map(_.split(",")).map(x=>(x(1),x(2).toInt)).reduceByKey(_+_)

Eg3:- * max, * min In java [math.max(,) / math.min(,)] available

> val resultmax = e.map(_.split(",",-1).map(x=>(x(1),x(2).toInt)).reduceByKey(math.max);

(math.min)

Scala > val arr = Array(1,2,3)

> for (i ← arr)
  | println(i) (<-) ~similar to System.out.println()

o/p 1 2 3 sum the array element.

Array → arr

> var total = 0
> for (i ← arr)
  total = total + i; | total += i

> total
⑥ o/p

simpler one
> arr.foreach (x ⇒ println(x))
> arr.foreach (println)

Eg:- > for each element *10;
in array - - [1 2 3 - -]

> For (i ← arr)
  i * 10 (or) for (i ← arr)
  yield i * 10 (or) arr.map (x ⇒ x*10)
  (Or) arr.map(_*10)

Eg:- > arr.flatmap (_*10)
Error

* Difference b/w flatmap & map()

> def operation (x:int) = [...]
> def operation2 (x:int) = list (x-1, x, x+1)
> operation2(1)
[0,1,2]

> val arrl = list (2,4,5)
arrl.map(operation2)
[(1,2,3) (3,4,5) (4,5,6)]

> arrl.flatmap (operation2) → give flat collection of elements
[1,2,3,4,5,6]

Eg:- > filter
filter Not

> val arr2 = Array (1,2,3,4,5,6,7,8)
> arr2.filter (x ⇒ x<5)
> arr2.filter (_<5) (Or)
> arr2.filterNot (x⇒x<5)
arr2.filterNot(_<5)

Logical AND | OR | && | ||
we can use

```
Scala > val lines = sc.textFile ("comment")
      > val test = lines.collect();
      > test
```

Spark is great with hadoop
Spark is great
Hadoop is a Batch job

```
Scala > def isItHadoop(x: String):
                                Boolean = {
          x.contains ("hadoop")
        }
```

```
Scala > isItHadoop ("Spark is great") → false
        isItHadoop ("hadoop is Batch job) → True
      > test.filter (isItHadoop); → display the only hadoop lines
      > test.filterNot (isItHadoop);
```

```
Scala > val a=10
      > a.getClass
      > a.getClass.getName
```

"Name" "age" "sex" "sal" → first
ravi, 23, m, 10000
rani, 34, f, 20000
vani, 30, f, 50000
mani, 25, m, 40000

→ Test1

```
Scala > val lines = sc.textFile ("test1");
      > lines.first
      > val heads = lines.take (3)
      > heads
      > def isItHeader (x: String): Boolean = {
                    x.contains ("name")
                }
      > val noheaders = lines.filter (isItHeader); (NOT)
      > val noheaders = lines.filter (x => !isItHeader(x))
      > val x = Test.map (_.split (","))
      > val y = x.map (_(3)).toInt)
      > y.sum → Entire array → column Aggregations
```

* Working with Strings :-

```
Scala > val str = "Spark", "Hadoop"
      > str.length
      > for(c <- str)
          println(c)s
```

Scala treats string as a sequence of (characters Array)

```
> str.foreach (println);
> str.toupper case | st.No wer case
                   st.to
> "h".to upper case → for string
> 'n'.to upper → for char
> val s1 = 'Hadoop'
> val s2 = "hadoop'
> val s4 = null
> val s4 : String = null
    s1 = s4 → false
```

$s_1 == s_2$ → True

NOTE:- In scala 'null' values can be compared when you call a method on null values, Scala throws NULL pointer inception

Eg:- s4.length

Eg:-

Eg2:- Scala > val name = "ravi"
> val age = 25
> val res = name + "is" + age + "years old"

* Embed a variable into String :-

```
> val s = " name is age years old"
```

name is 25yrs olds

```
> val s = "s $name" is $ age years old"
```

Ravi is 25 years old

NOTE:- Start the string with s for float f

```
> val age = 25000
> val s = s"$name is $ age years old"
> val s = s"$name's next yearage & ${age+1}
                                & ${age+1}
```

* Multiple String if multiple lines

```
> val x = """ xxx xxx xxx
            yyy yyy yyy
            zzz zzz zzz """ (3 double quotes)
```

```
scala> val s = " sparks scala"
 > s.filter (_ != "s")
 > s.capitalize
 > s.drop (2)
 > s.take (5)
 > s.drop (1).take (4).capitalize
```

```
> s.split (" ")
> s.split (" ").foreach(println)
> s.split (" \\s+")
```
(`\\s+`: removes spaces using regex expression.)

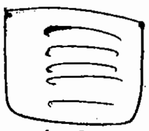
Eg:-
```
> val lines = """ line
                  Spark
                  |scala
                  |hadoop """.stripMargin
> val lines = """ line
                  Spark
                  #hadoop """.stripMargin('#');
> lines.replaceAll ("\n", " ")
```

Eg:-
```
> val s = " one, two, three , a,b, c,d ,e,f,g,f,g,h")
> s.split (",")
> s.split (",").map (x => x.trim)
                .map (_.trim)   (or)
```
(space → Ltrim | Rtrim)

| Strings & regex patterns regex expressions | → practice well. ** → unstructured data |

(REPL) → console
* Read Evalute Print Loop


which we work
pykon | R-lang | Scala

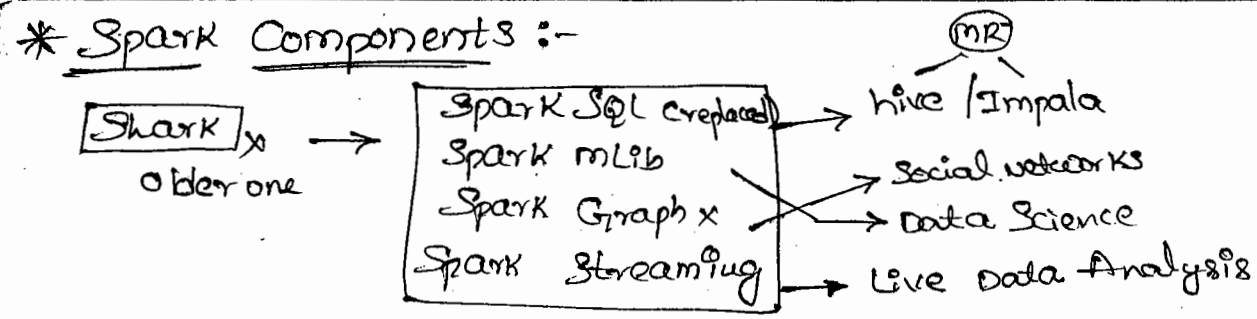

file.name.scala
filename.java

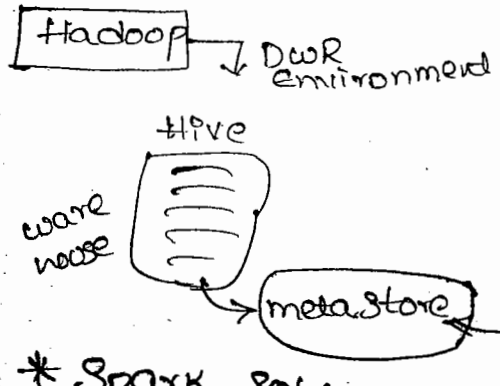
Compile
scalac>
javac> >

.class

run
scala>
java>

* Spark Components :-





* without Hadoop we can also run Spark in AWS/ mesos also ---.

* Spark SQL :-

By using Spark SQL we can connect with Hive metastore. So that we can perform (I/O) operations with hive-tables Data Storage into Hive tables.

* Reading from hive tables.
* processing by spark.
* Rewrite in Hive tables.

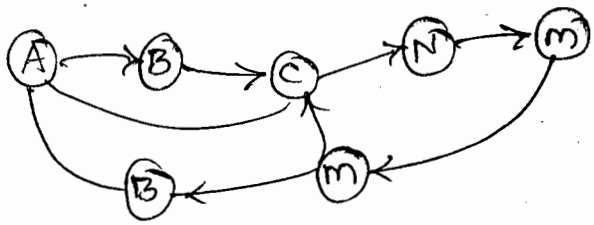
So Spark SQL we can also work with Impala (Cloudera products) ↳ faster

Impala & Hive will use same metastore so that we can access tables.

Spark GraphX :-

to work / process the graphdata. GraphX is library with all graph algorithm (Graph means → links of Record

Eg:- facebook users connections



Common friends

* 1st level friends
* 2nd level
* 3rd level

Graph Data processing is vastly used in Topic Indexing / Social meadia / online Retails / Gen / Logistics / page ranking

So Spark GraphX has predefined library for Graph Data process.

Spark MLib :- M - machine learning

Eg : Gmail → Gmail Email Engine (library) → promotions, Spams, Social, Inbox

Is library for machine learning

In MLib all Statistical & predective algorithms

Data patterns → machine

clustering algorithms.

Neural algorithms, Recommder algorithms ... are available. (Engine Builder)

MLib → used for data science.

Spark Streaming :-

can Stream data from flume / Storm & other Streaming Systems.

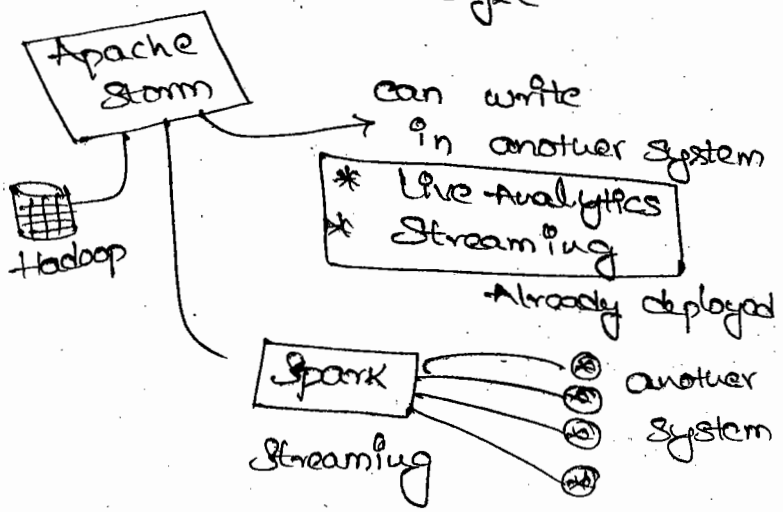
Apache Flume / Storm → Live Analytics

more powerful (X) (better) ✓

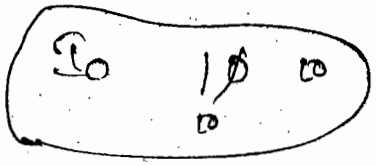
flume not suitable / recommend

Eg :- GPS predictive algorithms

* Can only Stream data from live-Analytics
* Cannot analyze



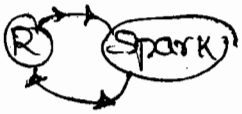
* Adhoc Live's

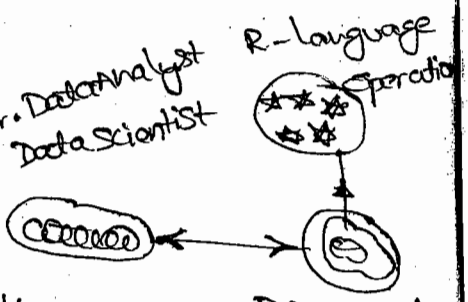


* machine understands different

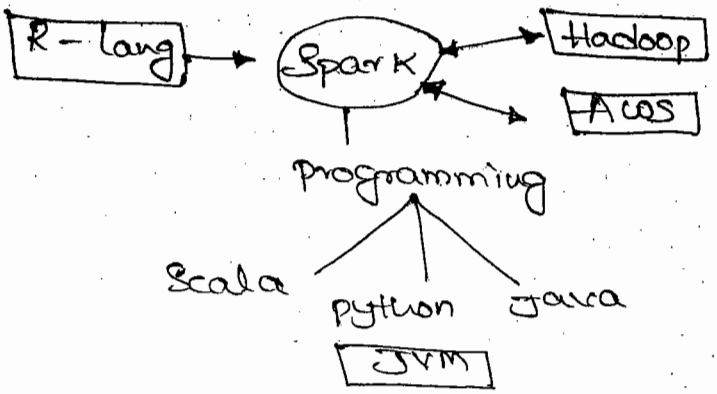
R-Language :-

Spark → 3rd party tool

Converting 



Using SparkR R can be integrated with Spark.
R processer will be converted as DAG - RDD models



Eg:- http://bit.ly/1Aoywaq d oration.zip

Data cleansing → deep duplication eliminating.

Eg:-

name	phone	city	
name			
Ravi	21	hyd	same
Rari	21	hyd	

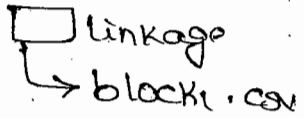
Scala> ls linkage

> gedit linkage/block1.csv

> val lines = sc.textFile("hdfs://quickstart.cloudera/user/cloudera/linkage");

> lines → is a RDD

> lines.count()



Scala > lines.first

> val f = lines.first

> f.length

> f.split(",").length

> val head = lines.take(11)

> for(x ← head)
println(x)

> def isHeader(x: String): Boolean = {
x.contains("id_1");
}

> head
> head.filter(x => !isIt Header(x));

Simpler form
> head.filter(!isIt Header)
> val noheads = head.filter(!isIt Header(-));
> noheads.foreach(println)
> val noheads = noheads.map(-.split(","))
> val f = noheads.map(x => x(0))
To convert it into integer
> val f = noheads.map(x => x(0).toInt)
> val f = noheads.map(x => x(p1).toBoolean) to Boolean)
> val f = noheads.map(x => x.slice(2,11))) foreach(println);
x => x.slice(2,11).first);
> f.map(x => x.toDouble) x
> "1.34".toDouble
> "?" toDouble (x : String) : Double = {
if ("?".equals(x)) Double.NON;
else x.toDouble
}
> Convert Double ("?")
> convert Double ("10.45");
> f.map(convert Double)
> val scores = noheads.map(-.split(",")).slice(2,11);
> scores(0)
val test = scores(0);
> test.map(-.slice(2,11));
> scores(0)
val test = scores(0);
> test.map(-.slice(2,11));
Scala> scores.map(-.slice(2,11)).map(convert Double);
> s(0).map(convert Double)

name sale grade xxxxx
37291, 47641, ?, , , , , true
47691, 46724
xx false
x false

Eg:- > def parse (x : String) = {

val words = x.split (",")

val id1 = words(0).toInt

val id2 = words(1).toInt

val scores = words.slice (2,11).map (wx => convertDouble (wx))

// Pending val matched = words(11).toBoolean

(id1, id2, scores, matched)

}

Scala > parse (head(1)) → tuple int, array, boolean

> val mydata = head.filter (!isItHeader(-))

.map (parse(-))

*** Eliminates the false value items & displays me only TRUE value records.

Scala> case class matchedData(id1: Int,

id2 : Int,

scores : Array [Double]

matched : Boolean)

Scala > def parse x (x : String) = {

val w = x.split (",")

val id1 = w(0).toInt

val scores = w.slice (2,11).map(convertDouble(-))

val matched = w(11).toBoolean

matchedData (id1, id2, scores, matched)

}

> val mydata = head.filter (!isItHeader(-))

.map (parse x(-))

↓

Each element

defining through case class

& a = new A()

> mydata(0)

> mydata(1) → results

last

> mydata(0).matched

> mydata.map (x => x.matched)

> mydata.map (x => x.scores)

*** Littles bit of cleaning is done & now I want to perform aggregations.

> val r = head.parallelize (head) need to convert it into RDD

> val res r .filter (!is It Header(-1)).map (-.Split(",",")")
.map (r => (r(11) .toInt))
.reduce By Key (-+-)

> val test result = res. collect ()

> test result

Eg 3:- Scala > lines

Scala > val my raws = lines .collect ()

ravi, m, 10000
Rani, F, 2000
Govi, m, 30000

> my rows .foreach (println)

> val mydata = my raws.map (-. Split (",",""))

> mydata .SortBy (-.-1) .foreach (println)

> mydata . Sort By (-.-2). reverse .foreach (println)

2nd field

Sorting by the field

> mydata. map (r => r. Score 3()). Stats()

* raw blocks :-
raw data —
- coont()
- collect()
- save AS Text file

> val head = lines. collect (500);

> head .length

- filter
- filter Not
- foreach (println)
- first
- length

> def convert Double (r.String)
: Double = {

> def convert Double (r: String)
: Double = {
if (" ?". equals (r))
Double. NAN
else
r. to Double
}

r => (r)
Space should be there.

> val rs = arr.slice (2,11) .map (convert Double);

> val r = (1, 2, 3, 4)

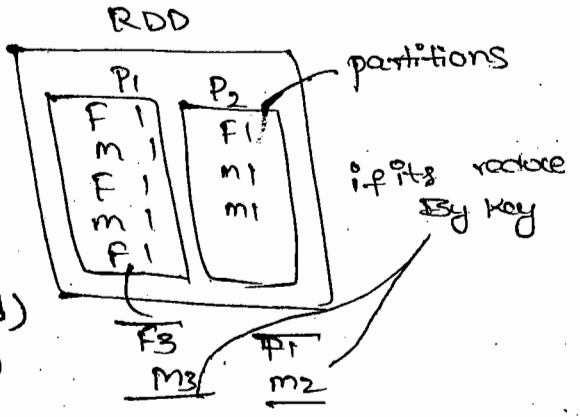
> r. product Arity

> r . product Element (2)

* parsed
cache () → rdd. persist (≡) storage level . memory ***

Eg :- Aggregations

group By → map (K,V)



Scala>

>val grouped = mds. group By (md => md. matched)

>grouped. map values (x => x. size) foreach (println)

x.sum
x.mean
x.avg
x.min
x.max

part-00000 part-00001

Independent aggregation

* (Select sex, max(sal), sum(sal), min(sal), count(*) from table, group by sex)

Select dno, sex, max(sal) from emp group by dno, sex;

multiple aggregations :-

> val parsed = sc. parallelize (mds)

> val matched counts = parsed. map (md => md. matched). countByValue()

↳ only on RDDS

> val match counts seq = matched counts. to seq

> match counts. seq. sort By (-._1) foreach (println);

" " . Sort By (-._2). reverse . foreach (println);

* Sorting By ↳ field positions

* Select * from emp order by sal, dno, sex, city

couple collaborative filtering

ALS → Algoritum

Alternative least square

Eg :- Recommender Engine

phase 1 — phase 2 — phase 3 — phase 4 — phase 5 → phase 7

ALS Algoritum

you-tube / flipkart portal

available in mlib