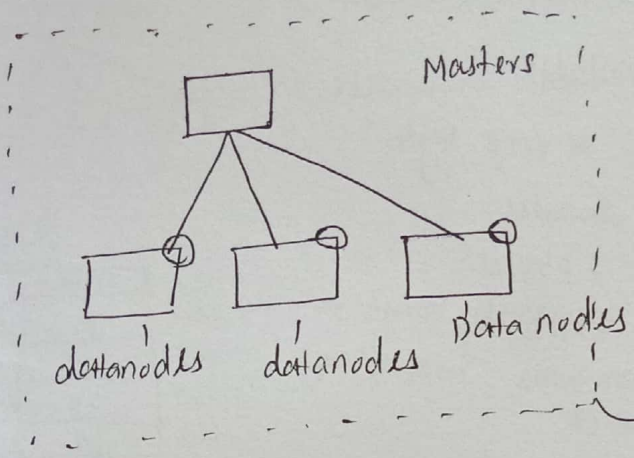
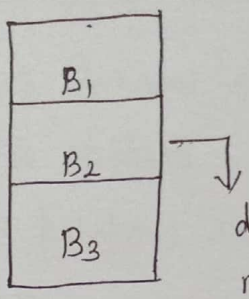
HDFS

HADOOP ARCHITECTURES

05.12.20
wednesday

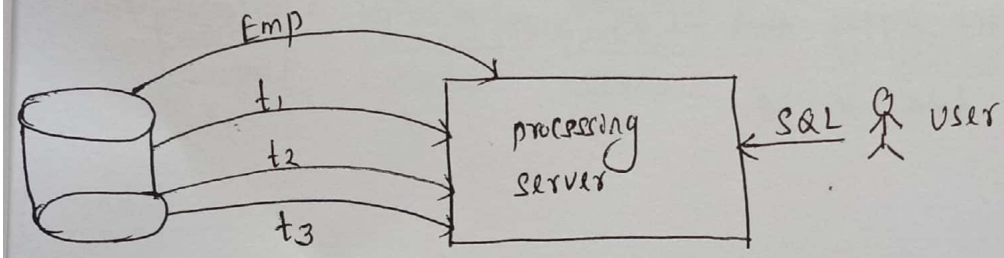




distributed multiple slaves

→ Clusters → set of machines / computers

* Masters will have the metadata for all the nodes. Metadata will contains filename, size, no of blocks, block ids, block locations, etc.



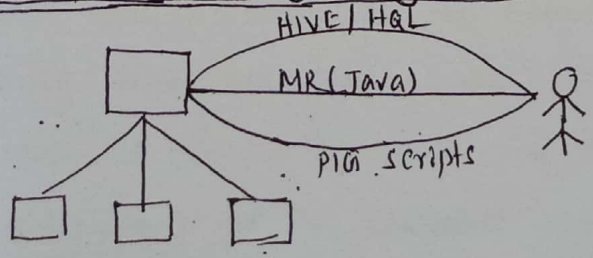
→ Huge content of data is travelling over the network.
→ Leads to network traffic / congestion.
→ poor query performance

* So to solve this issues data locality comes into pictures.

Datasize	MB, GB, TB
Logic size	B, KB

⟹ It will do the logic travel, not the data travel.
⟹ Logic will be passed each and every slave and processing happens locally at each slave.
⇒ Data + Logic happening in one server is called data locality.

PPM (parallel programming Model)



Cheap and commodity H/W DNs => slaves

↳ easily available

Advantage – Low cost & easily available

Drawback – chances of failure is very high

Space contributed by all the DNs | 2 nodes
7 physical
7 × 1021 TB

Heart beat = 3 sec interval of the CF.

* A TCP-IP protocol that each and every DNs are connected.

DN failure

DN sends to NN saying that we are alive & can do a piece of t

To overcome this data loss RF.

Replication factor

Advantage – Data high availability

Disadvantage – occupies more spaces.

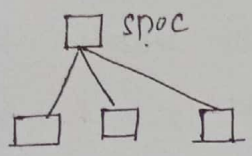
Backup cluster to take the data (DR), disaster recovery

wait time :- it will wait for a while & take the Replica.

hadoop fs -setrep 2 file

↳ file level Replication.

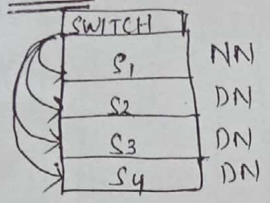
06.12.2018
Thursday



→ All the information is coming to the Name node first.

→ Replication factor = 3 → copy of same data in 3 nodes within the clusters.

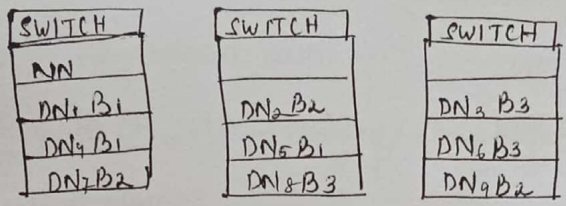
RACK

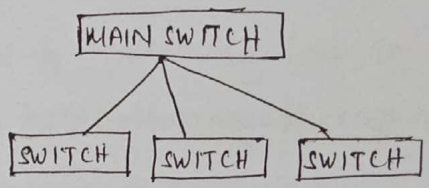


→ Actual placement for the servers.

→ physical equipment to hold the servers.

9 DNS, 1 NN, 3 Racks, 3 RF, 150MB, BS = 64MB





Conditions

→ No 2 copies of the same block will be stored in same data Node. Always a Replica should be stored in a different datanode.

→ No 3 copies of the same block will be stored same rack. At least 1 copy of the block should be stored in remote (different) rack.

Identification Unique

Blocks – Block ids

Nodes – IP address or host names

Rack – Rack ids

Datacentre – Datacenter

→ Block Algorithm for placement

→ Rack Awareness

→ Hadoop will follow the Awareness by following the block placement Algorithm.

→ Name Node will have all the metadata.

→ config files will bear all the information for the master and slaves.

→ Block size should be configured at the starting. It can't be changed later, same for the replication factor.

→ Space wastage from 0 to 63.999... MB

B_1 - R_1/DN_1 (X), R_1/DN_4 (X), R_2/DN_5

B_2 - R_2/DN_2, R_1/DN_7, R_3/DN_9

B_3 - R_3/DN_3, R_3/DN_6, R_2/DN_8

- If one racks failure B_1, th will be different location for as well to recover.

NN
- fail/ Down - Temporarily unavailable
  - → 07.12.201
  - → friday
  - → wait time re
  - → May need Re
- crash → permanently unavailable
  - → Hardware required
  - → Dataloss will be there
    - ↳ (Metadata)

→ In first generation of hadoop 1 NN.

→ In 2nd generation of hadoop 2 NN.

1 NN Crash → Metadata loss

→ So new cluster will be configured.

2NN → It is not hot backup of Name node

→ It takes metadata backup at some definite intervals of t (check pointing time = 1 hr by default)

→ Metadata stored in SNN can be used for rebuilding a new node in case of NN crash.

→ It will take only the delta information.

NN
- More RAM
  - Being a poc more RAM is preferred
  - Holds all metadata in RAM for faster access
- Less HD

DN
- Less RAM
- More HD

SNN
- Less RAM
- Less HD

NN configuration

→ 16 core in one machine - Hex core system

→ 1X2 TB HDD, 16X16 GB RAM

→ Total capacity: 2TB HDD, 256 GB RAM

Data node

1X2 TB, 4X16 G

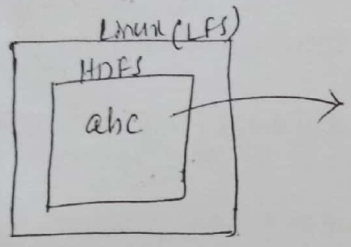
= 2 TB HDD, 64 GB

Datanodes - 32TB - 30x4
= 120TB

① $ start-all.sh

② $ jps — Java related processing

3004 NameNode



→ creating metadata.

LFS
(or)
HDFS

① Hadoop
- ls
- cat
- rm
- copy from local
- move from local — cut+paste
- put
- get
- cp] HDFS → HDFS
- mv

Hadoop fs -mkdir abc

$ hadoop fsck /user/training/abc/emp.csv -files -blocks -locations -racks

Localhost:50070

- Browse the file system

HDFS - commands

$ jps
→ JPS
→ Datanode
→ Jobtracker
→ Secondary Namenode
→ Tasktracker
→ Namenode

① $ hadoop fsck → check the status of the file like size, dirs, files, blocks etc

$ hadoop dfs -cat /user/training/abc/emp.csv

→ To read the content of the file.

$ hadoop fs -ls /user/training/abc/emp.csv

→ To see the content for the specified path, like how many files are found, access level

$ hadoop fs -cp /user/training/abc/emp.csv /user/training/abc/emp1.cs

↳ Source Destination ↲

The content from emp.csv will be directly transferred to empl.c

→ $ hadoop fs -cat /user/training/abc/empl.csv.

→ $ hadoop fs -rm /user/training/abc/empl.csv /user/training/abc/empl

→ Removing the files from the specified location.

→ $ hadoop fs -cat /user/training/abc/emp.csv

→ The file will not be present more in the specified location.

→ $ hadoop fs -put /user/Hive/warehouse/abc.db/empl/emp.csv /user /ab

→ $ hadoop fs -cp /user/hive/warehouse/abc.db .....

* HDFS to HDFS will be done by the -cp command, -put will not work for HDFS to HDFS.

→ $ hadoop dfs -put /home/training/f1.txt /user/training/abc

$ hadoop dfs -cat /home/training/emp.csv

→ $ hadoop dfs -get /user/training/abc/emp.csv /home/training fo

→ file already exist

→ $ hadoop dfs - copyFromLocal /home/training/f1.txt /user/training/ab

→ $ hadoop dfs -moveFromLocal /home/training/f2.txt /user/training/abc H

→ $ hadoop dfs -ls /user/training/abc

* file will be present in this location.

→ $ hadoop fs -mv /user/Hive/warehouse/abc.db/emp/emp.csv /user/trai

→ $ hadoop fs -ls /user/training/abc empl

----- emp.csv
----- empl.csv
----- f1.txt
----- f2.txt

*

*

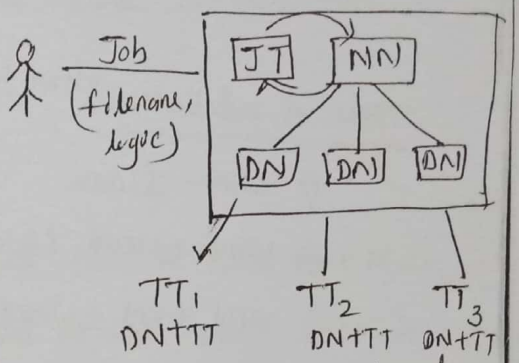
10.12.2018
Monday

① Bigdata storage (HDFS) — master : NN, slave : DN ↑ Heart beat

② Bigdata processing (MR) — Master : Job tracker, slave : Task tracker ↑ HB

NN will do 2 tasks → check file is there / not → file already exist
→ check storage capability - then RF factor ....

storage jobs → hadoop fs -
processing jobs → hadoop jar -



→ Job Tracker will connect with Task Tracker

Job = ~~TT1~~ + TT_2 + TT_3

TT_2 + TT_3 + TT_4

If the DN failed 1/3 of data not completed, so it will find for the next node for the same task.

JT URL → username - Ramesh
→ (web user interface)

Job_2018 1210_0815_001
Jobtracker id

Hadoop config files

1) HDFS-site.xml (BS, HB, RF ....)
2) core-site.xml
   101 (NN)
3) Masters
   103 (SNN)
4) mapped-site xml
   102 (JT)
5) Slaves
   104 (DN+TT)
   105 "
   106 "

6 servers

192.168.133.101 (NN)
.... 102 (JT)
.... 103 (SNN)
.... 104 (DN1)
.... 105 (DN2)
.... 106 (DN3)

↓

Java + Hadoop both needed

* Hadoop config folders will be there across the machines all the machines will have.
* No need to start the services at each node. On one machine you can do as all the nodes have the config files.

It will run 5 services → DAEMONS (A back ground job)
services
processesss

(SPOC) NN – Metadata & tracks all DNs
DN → stores all datablocks
SNN → stores all metabackup at some definit interval of time.
JT → Accepts all the jobs & tracks to completion.
TT → accept task from JT and executes in same DN.
and support NN with HB

Types of installation

1) ~~Wint~~ Linux
2) WIN + Linux (DUal Boot)
3) WIN + VM + Linux

Centos – Community enterprise operating system

$ ifconfig → provide ip address.

→ $ stop-all.sh → as JT is not working

localhost:50030 → job details → GUI for Map reduce

→ $ exit; → then logout from vmware

Next

Client node / Dummy node

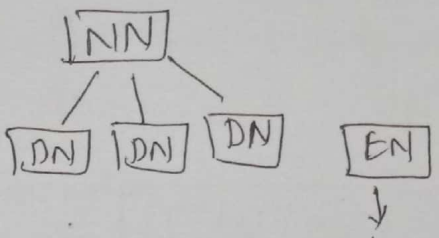
HDFS writes
HDFS reads
HDFS Deletes
Scaling < VS / HS
Load balancing

Namenode access

→ Namenode is busy for other task
→ Namenode having the sensitive data
→ So no access given to developers

11.12.2018
Thursday

→ Edge node
client node
Dummy node
Gateway node
connecting node
Application node

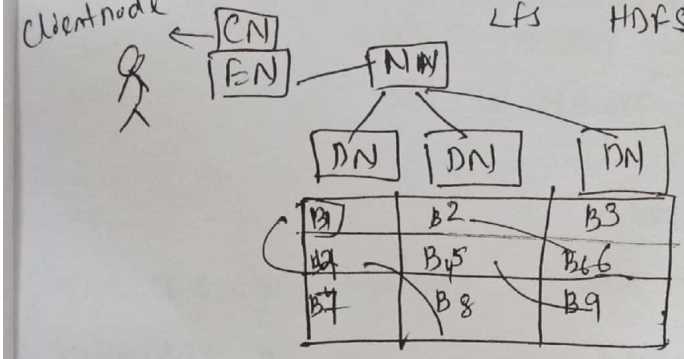


↓
not used for storing data

* all the nodes & EN are interconnected
* can be accessed with ssh command

HDFS Writes

hadoop fs -copyFromLocal file.txt abc → to client node
(LFS) (HDFS)



* EN will split a files, store in a location and as the other block to keep the copy of the file with them.
* After data replication all the information/ metadata will be stored in NN.
* CN will ask for available resources from the main node.

B_1 - R_1/DN_1, R_1/DN_4, R_2/DN_8

B_2 -

B_3 -

DN_8 acknowledge to DN_4 then
DN_4 report to DN_1 then
DN_1 report to client node the blocks are replicated.

* Block placement algorithm
* NN will provide excluding dead and full datanodes, to client node
* CN have the config files.
* the DN confirmation to CN then CN report to NN to update the metadata.

HDFS Reads

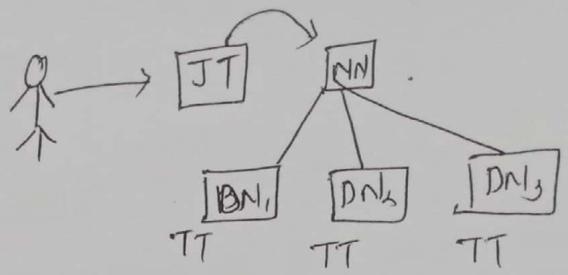
→ hadoop fs -cat abc/file.txt
* it will give the nearest node

HDFS deletes

→ hadoop fs -rm abc/file.txt * All the blocks will be removed.

processing

PIG (script)
MR (Java)
Hive (sql)



job = $TT_1 + TT_2 + TT_3$

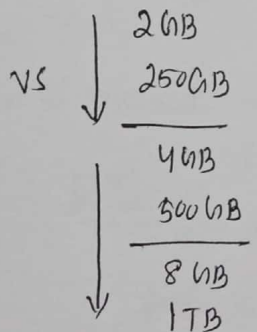
1) soft delete ⇒ hadoop fs -rm abc/f1.txt — (soft delete)
2) Hard delete ⇒ hadoop fs -skipTrash abc/f1.txt — (Hard delete)
⇒ hadoop fs -cp Trash/f1.txt abc
(HDFS) (HDFS)
⇒ hadoop fs -mv Trash/f1.txt abc

* Blocks will be generated in ascending order.

§ 12.12.2018
* Wednesday

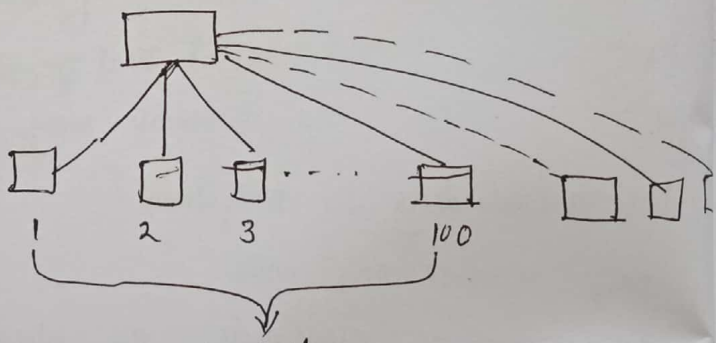
Scaling

→ Increasing / decreasing the capacity



vs

(Vertical scaling)



Nodes full

(Horizontal scaling)

Load imbalancement

Cluster resources under utilization.

Safemode (Read-only mode)

→ hadoop dfsadmin -safemode enter (ON)

If not safemode then data will come to the newly added node

→ hadoop dfsadmin -safemode leave (OFF)

→ hadoop dfsadmin -safmode get (status)

balance.sh

→ some of the blocks from the present nodes moved to newly nodes.

→ How many are newly added

→ how much block should be copied to new nodes.

stop-all.sh

→ some CLI tools will be used to change the node configurations and clustering will be happened

→ NN will have new data.

→ In this way Load balancing will be done.

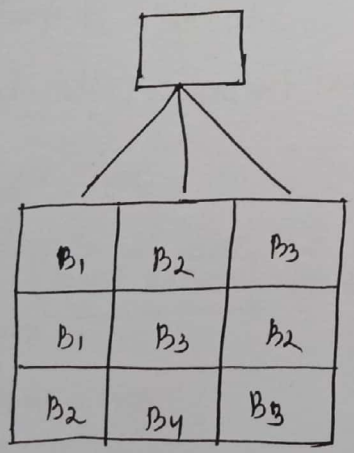
Slaves
101
:
200
201
202
203
204
205

During cluster start-up BLR → Block Level Report

every 10th heartbeat as BLR

How many data blocks particular node is holding. $3 \times 10 = 30$ sec

→ when the cluster starts it will be in safe/Readonly mode, till one heartbeat it should get.



$3 \times B_1 - 1 = 2$

$3 \times B_2 = 3$

$3 \times B_3 = 3$

RF < 3

→ All the blocks lost on their particular node called "under replication blocks".

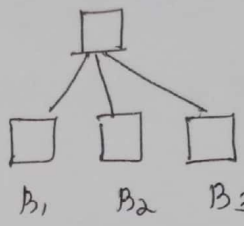
→ when the node come to normal condition then

$3 \times B_1 + 1 = 4$ nodes

→ All the blocks present in the reconnected node are called as "OR blocks"

over replication blocks

RF. 73



RF = 1 ✓
RF = 2 ✓
RF = 3 ✓
RF = 4 ✗
RF = 5 ✗

RF=1 B_1 B_2 B_3

RF=2 B_1B_3 B_2B_1 B_2B_3

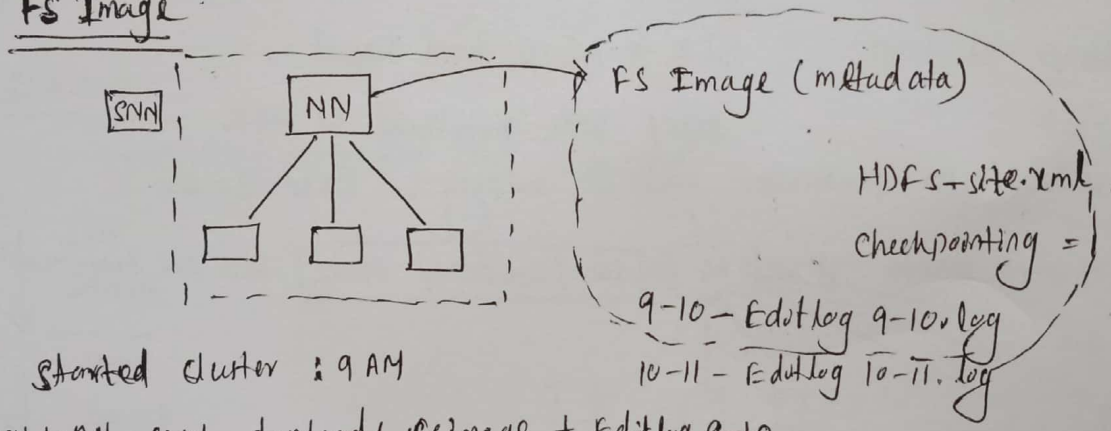
RF=3 $B_1B_2B_3$ $B_1B_2B_3$ $B_1B_2B_3$

RF=4 $B_1B_2B_3$ $B_1B_2B_3$ $B_1B_2B_3$ [000] → missing Replocation = 1

RF=5 $B_1B_2B_3$ $B_1B_2B_2$ $B_1B_2B_3$ [000] [000] → missing replocation = 2

→ Rf for a cluster is always less than node DNS or atmost it can be equal no of DNS, else it will be lead to "MR".

FS Image



→ at 10 AM SNN downloads fsimage + Edit log 9-10
= new FSImage
→ Uploads then a copy of new fs image to NN.
→ at 11 AM SNN downloads Edit log 10-11.log
- merges with new FSImage & creates one more [new fs image]

→ This is the recurring process of fsImage.

Edit log (transaction log)

→ Add a file
→ delete a file
→ move a file
→ change in metadata

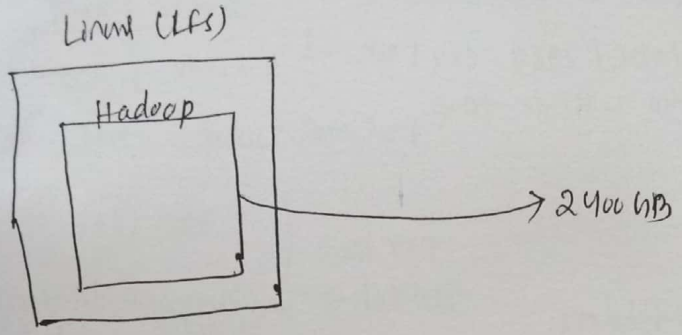
Transaction in NN.

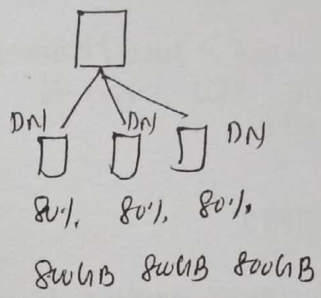
Checkpointing

→ process of merging edit logs to the fs Image & creating a new fsImage.
The node which is doing this activity is called as "check pointing Node".

namenode.data.dir
sec namenode.data.dir

13.12.2018
Thursday

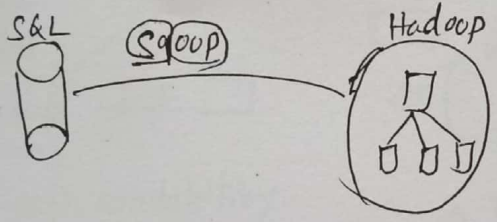




file will be broken and distributed across the datanodes.

Single node cluster (Pseudo distributed mode) ← training practice

fully distributed mode (multinode cluster) ← dev, prod, POC, Test



$ hadoop fs -rmr abc

deleted hdfs://localhost:54310/user/training/abc

$ mudar abc

$ hadoop fs -ls abc

1024B = 1KB

1024 KB = 1MB

1024 MB = 1GB

1024 GB = 1TB

1024 TB = 1PB

1024 PB = 1 EB (Exa)

1024 EB = 1YB (Yotta)

1024 YB = 1 ZB (Zetta)

1024 ZB = 1BB (Brontо)

dvs] $ hadoop fs -copyToLocal abc/emp1.csv .

↳ home directory dvs

$ cat emp.csv → file will displayed.

$ hadoop fs -cat /user/training/abc/1987.csv | head -10

| tail -10 → bottom 10 lines

$ wc -l 1987.csv

1048576 1987.csv

$ hadoop fs -cat /user/training/abc/1987.csv | wc -l

→ 1048576

| wc -c

→ 102664080

$ Recursive RF will be applied to subfolders. setrep → To change replication

-R - Recursively apply the same RF, sensitive data and subfolder.

-w → waittime where we increate the R.F.

· before copying the files

hadoop fs -D dfs.replication=5 -copyfromLocal emp.csv abc

Storage (HDFS) — Master - NN, Slaves - DN — HB

Processing (MR) — Master - JT, Slave - TT — HB

14.12.2018 Friday

→ All storage jobs directly land in NN.

→ All the processing jobs directly land in JT & indirectly lands in NN.

Drawback of 1.x Hadoop

→ NN is SPOF → JT is SPOF

→ No horizontal scaling of NN. - No load sharing of metadata in dist NNs.

→ JT is SPOF, in case of failure all jobs are restarted again.

→ It is suitable for Batch processing of huge amount of data.

→ It supports 4000 nodes for the cluster.

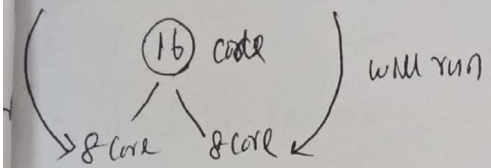
→ Job tracker is over burdened. It has a single component. Job tracker to perform many activities like resource mang, Job scheduling, Job monitoring, maintain all the history of Jobs.

It runs only Map/Reduce Jobs but not non-MR Application like spark, storm.

map	reduce
phase-1	phase 2

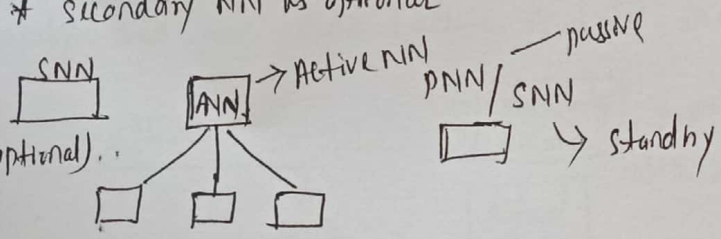
→ fixed no of map slots and fixed no of reduce slots

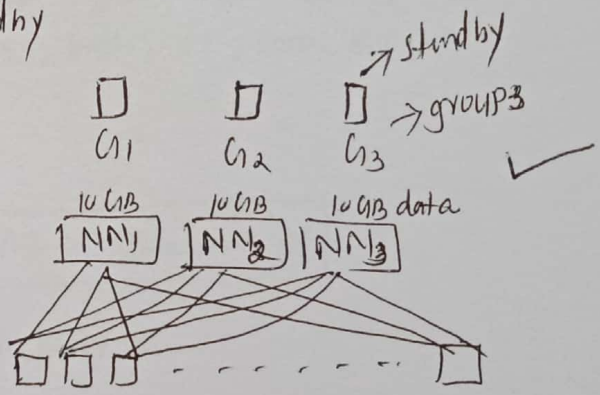
mapper.java reduce.java → It tasks more it can't use more than 8 core.



2.x Hadoop generation

* secondary NN is optional





① NN High availability

② User segregation for multiple Namenode.

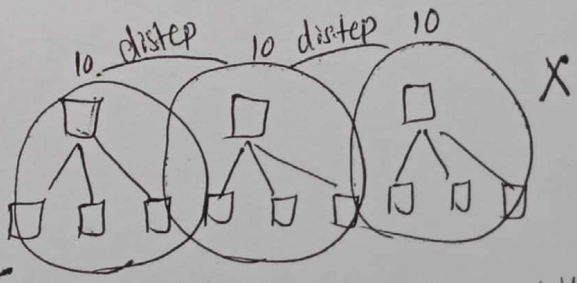
→ Name node horizontal scaling.

③ JT
- RM → Resource manager
- AM → Application master
- JHS → Job History server

YARN

NN HA
NN + HS

If anyone fails, it has high availability.

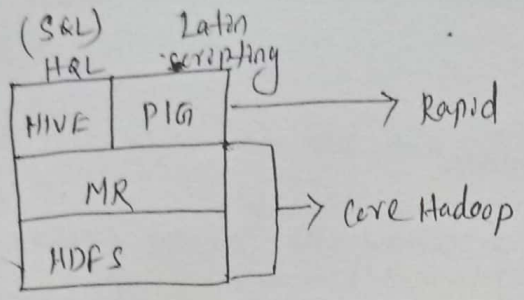


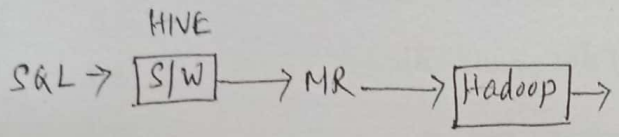
→ Under utilization of cluster resources → max 10 available

→ max 30 resources will be available at any time.

distcp → from one cluster to another cluster.

HIVE



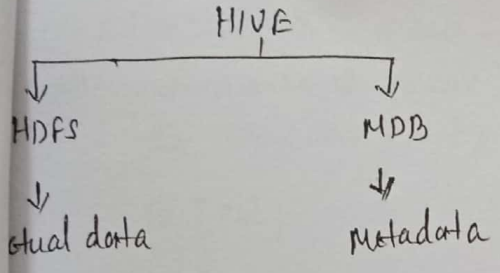


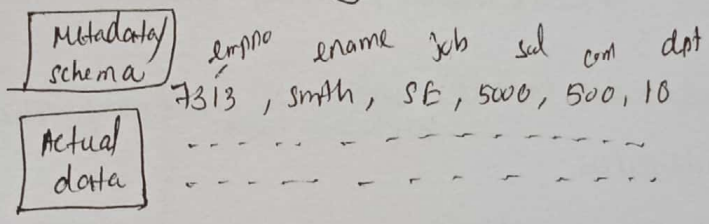
Developed by facebook.

It gives SQL kind of interface on top of data stored in HDFS.

Represents data in the form of tables.

Mainly used for OLAP (Online Analytics processing)





MDB → Metastore DB

* Apache Derby DB
* Session or user oriented.
* To overcome above problem → MYSQL or other DB
* Traditional DB is suitable for well organised Metadata.

→ Create database dvs-nov18

① An empty dir is created in HDFS under default path
(/user/hive/warehouse)
dvs-nov18.db - directory

② Metadata is stored in MDB (MySQL)

dvs-nov18.db → Database

emp → table belongs to default database without under any DB.

hive> describe database dvs-nov18;

→ coming from mysql. ← Metadata

hive> set hive.cli.print.current.db; → setting the current directory.

~~hive>~~ set hive.metastore.warehouse.dir = /user/hive/warehouse

→ $ hadoop fs -ls /user/hive/warehouse/dvs_nov18.db

drwxr-xr-x . . . . . dvs-nov18.db/emp →

→ select * from emp;

dvs_nov18.db

↓

emp (directory)

↓

emp.csv (file)

emp1.csv

emp2.csv

all the data coming from hadoop.

→ hive-site.xml

↳ All the default properties are set.

→ Hadoop will provide write once, read many. This version is not supporting the write access.

→ All files should have

i) same delimeter

ii) same no of columns

iii) same column sequences

→ it will 1st search the metadata then display the details.

① Managed or internal (if table is dropped both metadata & actual data is deleted)

② External → if table is dropped, only metadata is deleted & data in hdfs is untouched.

③ Temporary table

HIVE DATATYPES

primitive/scalar

- tinyint - 1B
- smallint - 2B
- int - 4B
- bigint - 8B
- float - 4B
- double - 8B
- string - 2GB
- date dd-mm-yyyy
- timestamp dd-mm-yyyy hh:mm:ss
- boolean (true/false)

char - can hold char values with variable precision

varchar - can hold char + int + special char with variable precision.

complex

- array
- map
- struct

map datatype

map
key value
sal #5000 , comm #500 , 10

key - value
sal #7000, comm #600 , 20

C2 map<string, int>

C2 [sal]

C3 [comm]

→ sid, sname, Address
BTM $ Banglore $ 560010

struct → collection of heterogeneous data types / dissimilar datatypes.

address struct< street : string , city : string , pin : int >

select * from abc where

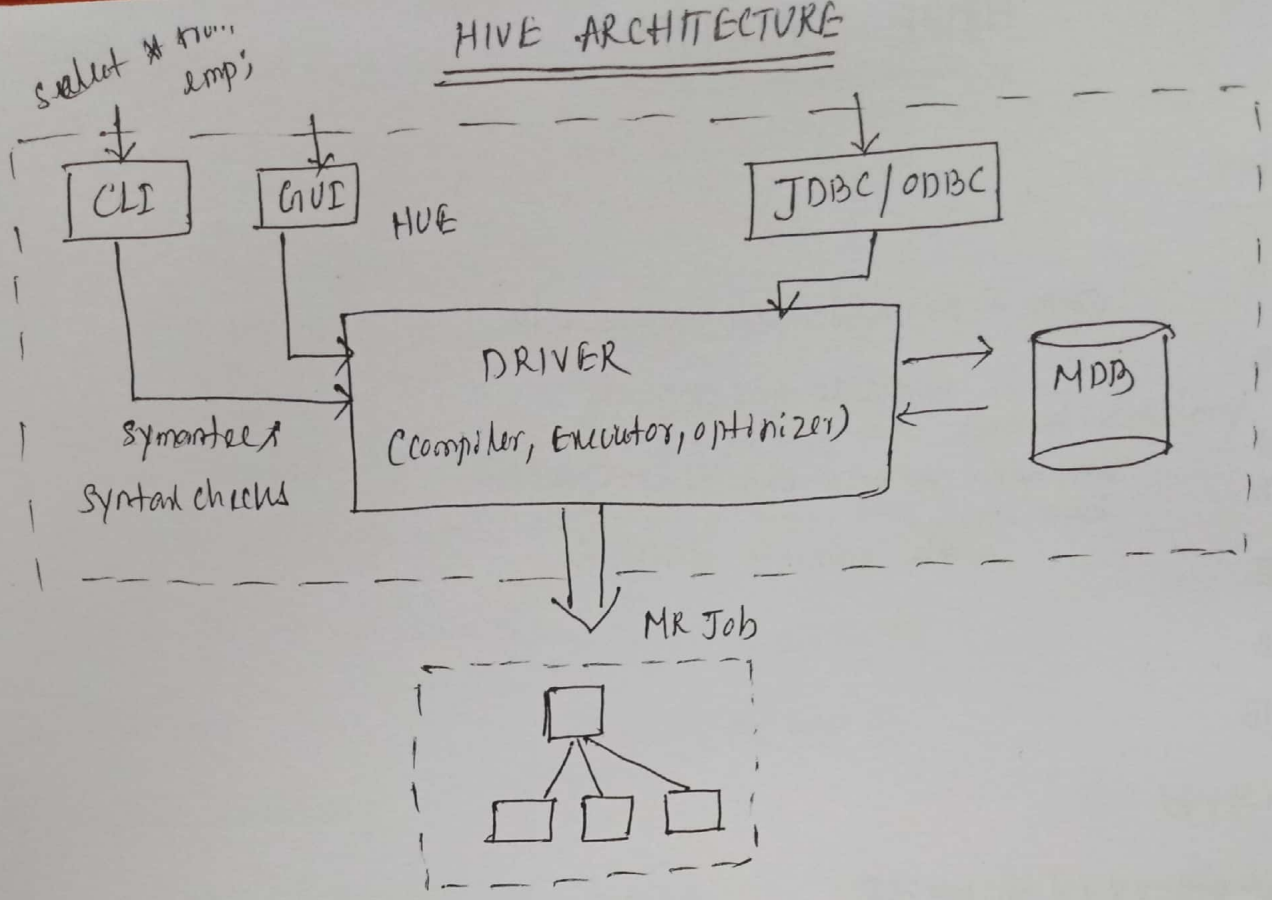
select sid, address.city, address.pin from abc.

on query - while creating use below query.

collection items terminated by '$';

BTM $ Banglore # 560010 — X Not possible with more delimeters same.

HIVE ARCHITECTURE



HUE → GUI (like) → Hadoop User Experience → User friendly

JDBC/ODBC → from frontend like .net/ Java to run the MR Job, but most not used.

NOTE → * If hadoop is not running the metastore DB commands will only not all the commands for getting actual data.

* Always table mapped to the Hadoop directory.
* N number of files can be there in the directory. So WRAE once concept came into picture and more data can be stored.

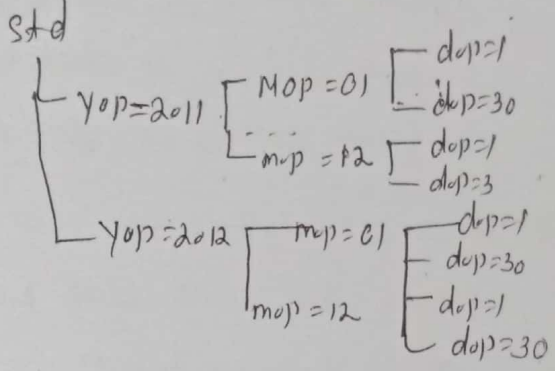
{ 17.12.2018 Monday }

Nested partitions

sid, sname, yop, mop, dop

/std-dp1/year=2011/month=2/day=20

```
cat stud.txt
Ravi, 123, 31.12.2011
```



querying metadata don't need services start/run, but data retrieval will not work.

* select * from std where yop=2011;

360 folders searching

* select * from std where yop=2011 and mop=1 and dop=1;

359 subfolders will be avoided.

Hive scripts

create database if not exists hivecont: db3;

use if not exists hivecont:db3;

create table if not exists hivecont:db3 (sname string, sid int.

JOINS

1) Inner or Equi
2) Left outer → all the rows from left, remain NULL vale.
3) Right outer → all the records from right, matching from left, remains NULL value.
4) full outer
5) cross → 100 records, 10 records → 1000 Records

create view v1 as select * from emp;

> show tables.

VIEWS

Create view v_1 as select * from emp;

Create view v_2 as select $c_1, c_2, \ldots c_{50}$ from emp;

Create view v_3 as select $c_1, c_2, \ldots c_{10}$ from v_2;

* Stores in metastore DB.

Create view v_4 as select A.c1, A.c2, B.c12, B.c13 from emp A join dept B on (A.deptno = B.deptno);

Location with partition table

1) Data in HDFS
2) Data should be well partitioned.

std
- year=2011
  - std-2011.csv
- year=2012
  - std-2012.csv
- year=2013
  - std-2013.csv

-mkdir dvs-HDFS / std / year=2011
=2012
=2013

hadoop fs -put std-2011.csv dvs_hdfs/std/year=2011
/year=2012
/year=2013

hadoop

create table std_et (sname, ...)

load data loc 'std-2011.csv' into std — std directory created in HDFS after copied.

metadata of partitions is created in MDB.

After creating location with partition;
You may not able to fetch the metadata details.

HIVE> alter table std_et add partition (year=2011);
(year=2012);
(year=2013);

hive> msck repair table std_et;

↳ metastore check.

* The way we are fetching and adding data as partitions.

HDFS	MDB
3	0 → 3
0	3 → 0
2	3 → 2
3	2 → 3

you can't delete/update/insert applied on hive structures.

hive> alter table std_pp.drop partition (yop=2014)

→ metastore db data & actual data from HDFS will be dropped.

hive> insert into table std1 partition (year=2017) select * from std where year=2017;
(static)

or//

hive> insert into table std1 partition (year) select * from std where year=2017;
(dynamic)

std

year=1911

2017 TB

1018 TB

std1

empty

↳ hive> alter table std1 exchange partition (yop=2015) with table std;

hive> alter table std1 exchange partition (yop=2016) with table std;

Nao

std1

2015

2016

std_et

hive> Alter table std partition (year=2012) ENABLE NO_DROP;

hive> alter table std_et drop partition (year=2012);

hive> alter table std

INDEXING

hive > select * from emp where empno = 1111;

→ will scan all the files in the directory.

emp (dir)
- emp1.csv
- emp2.csv
- :
- emp100.csv

hive > create index emp_index on table emp(empno)
as 'COMPACT' with deferred REBUILD;

↳ unique columns. ↳ partone/delay

when ever you are loading data need to rebuild the index.

Index table is created - dbname__tblname__index name.

empno	location (user/)	Byte offset (physical addr of the record)
7839	/user/hive/warehouse/dvs_nov18.db/emp/	[0]
7369	emp1.csv;	[32]
8869	- - - - - - - - - - - -	[62]

> create index emp_index on table emp(empno) as 'COMPACT' with deferred re
> show index on emp;

> show tables;

dvs_nov18_emp_emp_index

> select * from
nvdata; as no rebuild;

> alter index emp_index on emp rebuild.
> select * from dvs_nov18_emp_emp_table.
> select * emp where empno = 7782;

1111
1112
1113
:
7782
:

* it will goto index table, get the address and find 7782.
* Auto ascending implements.
* so at very first time it will be taking more time to build the index.

→ When a main table dropped automatically index table is dropped.

BUCKETING

→ for faster joins (join optimization)

→ Table sampling.

Clustered by (deptno)

→ we have to consider amount of joining with table.

→ If we have deptno, our joins will be efficient.

→ 3 buckets = 3 files.

hive > create table emp_buc (.....);

emp_buc
000000_0
000001_0
000002_0

→ check the default value before assigning any value.

Hash function = $\left\lfloor \frac{\text{deptno}}{\text{no of buckets}} \right\rfloor$ = $\lfloor 10/3 \rfloor = 1$

$\lfloor 20/3 \rfloor = 2$

$\lfloor 11/3 \rfloor = 2$

$\lfloor 30/3 \rfloor = 0$

→ hadoop fs -ls ... dis-nov18/emp_buc

hadoop fs -cat ..... /emp_buc/00001_0

→ for non-integers it will consider the ascii value.

select * from emp_buc tablesample (bucket 1 out of 3); union All (bucket 3 out of 3)

both 1 & 3 data will come.

Table sampling

hive > select * from emp_bk tablesample (10 percent);

select * from emp limit 100;

1) Buckets inside partitioning ✓ — possible — files inside folders

2) partitioning inside buckets ✗ — Not possible — folders inside files.

* friday

partition inside bucket ✗

Bucket inside partition. ✓

partition assignments	order-id	order-status	order-date	cust-id
	1	PENDING	08-08-2018	111111

orders

ord-status = PENDING
↳ 000000-0
ord-status = INPROGR
↳ 000000-

orders

ord-year = 2016
↳ 000000
ord-year = 2015
↳ 000000

Assignment-

olympics

1) type = summer — partition based on type
↳ 000000-0
type = winter
↳ 000000-0

2) OLP
country US
↳ 000000-0
country = UK
↳ 000000-0

3) OLP
year = 2000 ↳ 000000-0
year = 2004
↳ 000000-0

4) OLP
country = US → year = 2000
country = UK → year = 2000

5) OLP
year = 2000 → type = summer
year = 2004 → type = winter

6) OLP
country = US → type = summer

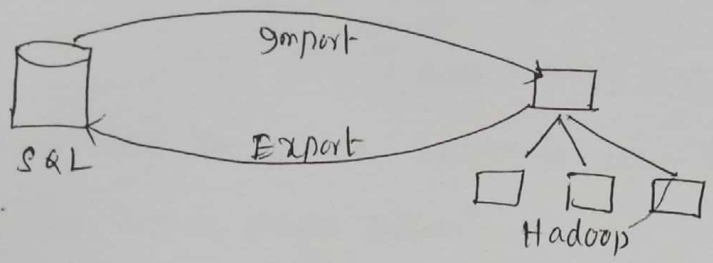
abc
country US → 000000-0, 000000-0, 000000-0
country = UK → 000000-0, 000000-0, 000000-0

* cluster by state - bucket
* sort by city

DDL
UDFS
DT
MR

partitioned by (country) clustered by (state) into 4 buckets sorted by (city).

SQOOP



SQ + OOP
= SQOOP

↳ Data transfer / Integration tool from RDBMS → Hadoop or Hadoop → RDBMS only.

MYSQL
DB → DVS-DB
tbl → Emp

HDFS
sqoop-nov

Import
1) from RDBMS (table) to HDFS file
2) from RDBMS (table) to hive tables

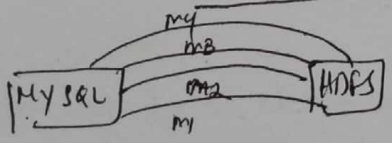
Export
1) from HDFS (folder) to RDBMS (table)
2) from hive (table) to RDBMS (table)

```
sqoop-import \> --connect jdbc:mysql://localhost/DVS-DB \
            > --username hadoop \
            > --password hadoop \
            > --table emp \
            > --target-dir sqoop-nov18 \
            > -m 1;
```

can be oracle, Any address.

No. of mappers

for any sqoop job -
Any sqoop job running with map=100%.
reduce = 0%.



→ m stands for mappers.

1) Depends on sourceDB no. of connections allowed.
2) It should not lead to small files.

reduce 0%.
map 100%.

mysql · fs -ls sqoop2_nav18

hadoop fs -cat /user/training/sqoop-nav18/part-m-00000

part-m-00001
part m.- 0000002
part-m-00003

mysql --u hadoop -p
mysql> use dus_dB;
> select * from emp;

1) restricting rows — where clause
2) restricting columns — columns clause
3) changing field delimeter.

sqoop-import --connect jdbc ..... --username hadoop --password hadoop --t
--target-dir sqoop1-nav18 -m 1 --columns "empno, ename, job"
--where "deptno = 20" --fields-terminated-by '|';

cat > sqoop-perm.txt

--comment
--username hadoop
-- password ~~password~~ hadoop

sqoop-import --options-file sqoop-perm.txt --table emp --target-dir. sqo
-m 1 --columns "empno, ename, job" --where 'deptno=20"
--fields-terminated-by '|';

* --username hadoop -p --table emp

→ prompt for password will come.

* passwd-file /etc/sqoop/conf/passwords/mysql-pass.txt

* sqoop y-nav18 · m 5;

if m=1 - it is sequential input
if m>1 - it is parallel input.
if m=2

part-m-00000
part-m-00001

* it will check the primary key
1
2
1000 - 1000/2 = 500 each

if no primary key for a table we have to either do import with m=1 (or) use split-by "some unique column".

-- split-by empno;

→ hadoop fs -ls sqoop_nov18;
→ if no primary then use split
→ cat /home/training/dvs/./emp.java | wc -l

HIVE IMPORT

MySQL
DB : retail_db
TB : orders

sqoop-import --options-file sqoop_param.txt --table orders --hive-import
--hive-database dvs_nov18
-- hive-table orders
-m 10
--split-by 'order_id';

→ if skipped default DB
→ if skipped same name as mysql
→ default mapper 4.
→ distribution according to primary key if no PK then give composite key
→ split by empid/orderid/.....
→ order is not important, but all the parameters should be satisfied.
→ max mappers depends on the max DB connection sessions per user.

* if m=1
no need for splitby

hue - hadoop user environment

su - switch user

password :-

JPS

mysql -u root -p

password - cloudera

mysql > show tables

hive > show databases;

hive > create database dvs-nov18;

hive > show tables;

cloudera @ quickstart ~] $ sqoop-import --connect jdbc:mysql://localhost/~~dvs~~ retail_db

--username root password cloudera --table orders --hive-import

--hive-database dvs-nov18 --hive-table orders.

HDFS

user / cloudera

orders

--delete-target-dir

HDFS-HIVE loc

/user/hive/warehouse/dvs-nov18.db/orders

hive > show tables

-- orders (sqoop created)

hadoop - root@quickstart ~# hadoop fs -ls /user/hive/warehouse/dvs-nov18.db

part-m-00000

00001

00002

00003

hadoop fs -ls ... orders/part-m-00001/wc-L — count

part-m-00002/wc-L -

part-m-00003/wc-L -

NOTE > delimiter should be defined while creating the table itself.

25.10.2018
Thursday

Types of Loads

1) initial / full / one-time / first load
2) incremental / delta load → Last value method
                            → Last modified date method.

MYSQL (orders)

ord-id	ord-date	ord-status	cust-id
1 2 3 100			
1001 1002 ⋮ 2000			
2001 2002 ⋮ 2050			

(day 1) (day 2) (day 3)

HIVE (orders)

ord-id	ord-date	ord-status	cust-id
1 2 ⋮ 1000			
1001 1002 ⋮ 2000			
⋮ 2050			

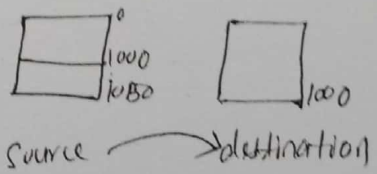
-- hive-overwrite

Last value Method

```
sqoop-import --cs
             --hive-import
             --hive-database
             --hive-table orders
             --incremental append
             --check-column order-id
             --last-value 1000
             -m 10;
```

x = hive -e 'select max(ord-id) from ';

last value from the hive table



Last modified date method

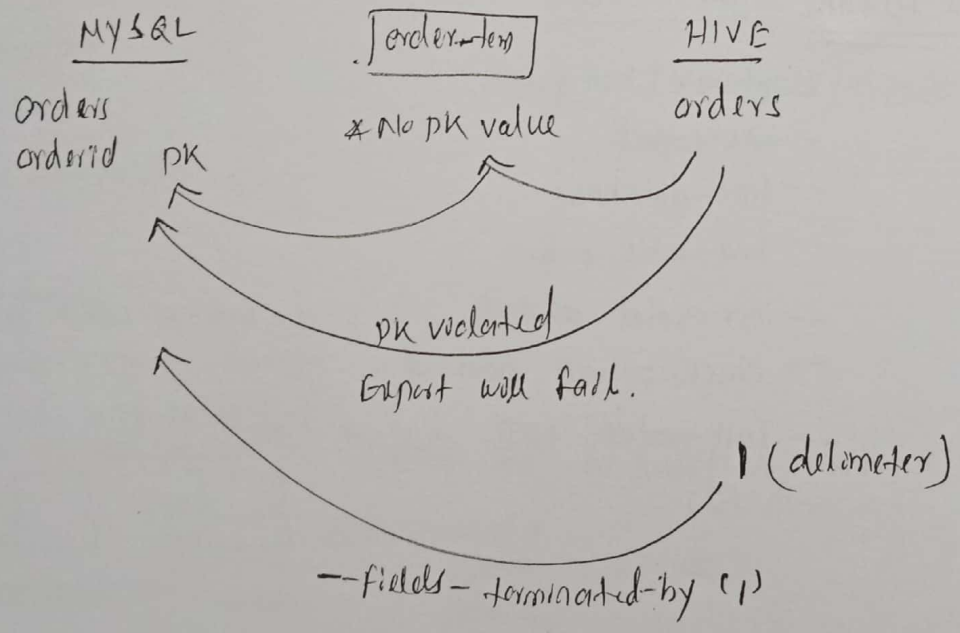
```
--incremental append
--check-column ord-date
--last-value '24-12-2018 23:59:59'
-m 10;
```

ord-id	ord-date	ord-status	cust-id
1	23-12	pending	1111
2	23-12	inprogress	1112
1	24-12	complet	1111
2	24-12	on-hold	1112
1050			

Sqoop Export

Sqoop-export --connect username root --password cloudera --table emp
--export-dir '/user/hive/warehouse/dus.db/emp-bk' --input-f
string 'NN' --input-null-non-string 'NN';

→ MySQL is not part of hadoop ecosystem
→ emp table u have to create in mysql then do export.



mysql> create table orders1 as select * from orders where orderid >=100
Cloudera> hive import from sql → Hive
hive> select * from orders1

mysql> insert into orders1 select * from orders where orderdate between 10
hadoop> hive import from mysql → Hive → lower bound, upper bound, sqoop will find out
hive > select * from orders1;

sqoop job --create dvs_nov_incr --import --connect ....

↓ space

--last-value 1050;

1) If new table ? last-value = 0

2) If existing table ? last-value = max (ord-id) from hive

ex) sqoop metastore - to store job name, details.

sqoop job-list;

dvs-nov-incr

cloudera > sqoop job --exec dvs-nov-incr

No new records deleted since last import;

mysql > insert extra 200 records → 1300 records

cloudera > sqoop job --exec dvs-nov-incr

hive > select * from orders1

1300 → 1300 records

cloudera > sqoop job --show dvs-nov-incr

cloudera > sqoop job --delete dvs-nov-incr

order-date ↓

> 00-00-0000 00:00:00

> 00-00-1900 00:00:00

sqoop metastore

dvs-nov-incr	~~1200~~ ↓ ~~1300~~ ↓ 1400
dvs-new-incr	

sqoop codegen

* To see the corresponding Java class files etc.

eval

sqoop eval --connect

HIVE Metastore

mysql > show databases;

mysql > select * from hive-metastore;

mysql > use hive-metastore;

hive> create database dvs-nov18;

mysql> show databases;

mysql> select * from DBS;

- - - -

dvs-nov18

Mhastore tables → DBS → Database names

TBLS → Table names

COLUMNS-V2 → column names & their datatypes.

mysql > select * from COLUMNS-V2;

TableId	Column name	type-name	Integer-idx
16	cat-id		
16	ord-date		
16	⋮	⋮	⋮
16			

HUE (Hadoop User Environment)

default path → /user/cloudera

hadoop fs -mkdir dvs

select order_status, count(*) from orders group by order_status;

→ su -

→ su - cloudera

→ jps

vi sample.sh

$

esc

:wq

chmod 777 sample.sh

./sample.sh

PIG

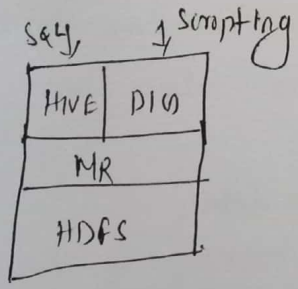
26.12.2018
Wednesday

→ High level data flow language
  session oriented
→ can work without hadoop
→ No Java required
→ 200-300 lines of Java code ≅ 10-15 lines of PIG script.
→ for transformation.

```
A = load 'emp.csv ... )
B = filter A by deptno == 20;
C = order B by sal desc;
D = limit C 2;
```

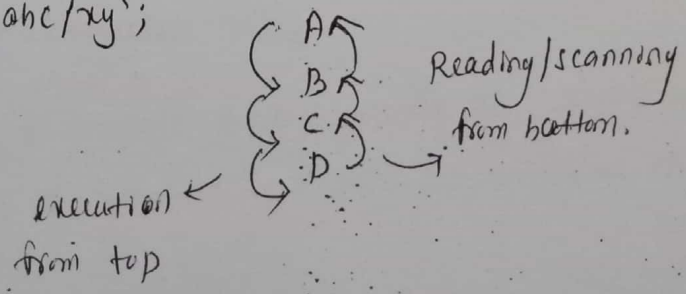
* Bags of data to hold data.

1) one by one flow of data A → B → C → D
2) pig >
   grunt >
   ctrl+c
   & pig
   grunt :> then the actions will not work
   → pig stores nothing
   → pig can be ~~stored~~ work on top of hadoop in simple Linux environment.
3)

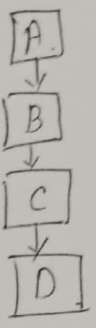


6) Lazy evaluation
7) pig doesn't store data permanently, give data then process, task is done.

→ grunt > store D into 'abc/xy';
  grunt > dump D;



DAG → Direct Acyclic Graph



* By reading from ... top pig prepares ...
* At any time u can ... to the command and ...

$ Pig -exectype local
grunt > pwd;
file:/home/cloudera

Apache
grunt > A = load "/home/training/dvs/emp.csv' using PigStorage(',') as
(empno: int, ename: chararray, job: chararray, sal: double; comm: fl...

grunt > describe A;
grunt > B = filter A by deptno=20;
grunt > describe B;
grunt > C = order B by sal desc;
grunt > D = limit C 2;
grunt > ctrl+L

* filter = space
* PigStorage - p.s -...

$ pig -x local
grunt > describe A;
--error----

] → session oriented No data.

grunt > D = limit C2;
--only check syntaxes.
grunt > dump D;

→ Lazy evaluation./advantages of pig
actual evaluation when give store/dump

Modes of execution of PIG

Local mode (uses LFS for i/p and o/p)

pig -exectype local (or) pig -x local

Mapreduce mode (uses HDFS for i/p and o/p)

pig -exectype mapreduce (or) pig -x mapreduce (or) pig (default)

pig.properties

Default mode is mapreduce in real time., but local in Apache setup.

home/training/pig/pig.properties

grunt > store D into "/home/training/dvs/pig-nov"

cd /home/training/dvs/pig-nov

$ ls

---- part-r-00000

$ cat part-r-00000

empno | name | | | |

⋮

* default delimeter is tab
* loading also

grunt > store D into "/home/training/dvs/pig-nov
using PigStorage(',');

$ cat part-r-00000

7788, aab, analyst, 3000.0, 20

⋮

Foreach --- generate

- used to restrict columns from a bag.
- used to derive new columns from existing columns.
- used to apply column level functions.

grunt > x = foreach A generate empno; empnm; job; → if u know the column name

x = foreach A generate $0, $1, $2; → if u know the column positions.

2) A (6) B (8) 6+2

Bonus = salary * 0.005

incentive = salary * 0.006

grunt > foreach A generate *, sal*0.005 as bnus, comm*0.06 as

warning for comm. as float

grunt > describe x;

3) grunt > y = for each A generates ename, SUBSTRING(ename, 1, 3);

warning Implicit cast to double...

grunt >

starting point → length to capt

CASE SENSITIVE

→ All bagnames are C.S

→ All column names are C.S.

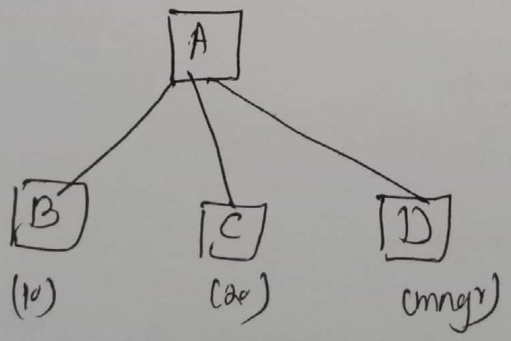
→ function like SUM, MIN, AVG, COUNT, SUBSTRING, LENGTH, TOKENIZE, FLATT

→ keywords like store, load, dump, using, as, join, cross, union, filter, or limit etc are not C.S.

9) B = filter A by sal > 2500;

9) B = foreach A generate substring(ename, 1, 3); ✗

B = foreach A generate SUBSTRING(ename, 1, 3); ✓



B = filter A by deptn

C = " " " deptn

D = " " " job

SPLIT

SPLIT A into B if deptno=10;
deptno==20;
desn='mngr';

→ scans only 1 but the filter will scans 3 times, so best to avoid multiple scans, to improve performance.

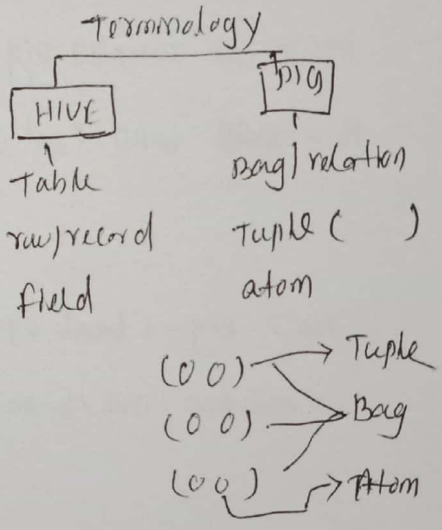
JOINS

g> B = load 'home/cloudra/DVS/dept.csv' using PigStorage(',') as (deptno:int, dname:chararray, dloc:chararray);

g> describe A;

g> describe B;

→ EQUI (INNER)
→ LEFT OUTER
→ RIGHT OUTER
→ FULL OUTER
→ CROSS



g> dump A;

- - - - - tuples
- - - - -

g> dump B;

g> C = join A by deptno, B by deptno;

g> dump C;

g> D = foreach C generate $0, $1, $6, $7;

g> describe C;

g> D = foreach C generate A::empno, A::ename, B::deptno, B::dname;

g> dump D;

g> C = join A by deptno right outer, B by deptno;

g> dump C;

9) C = cross A,B;

PIG SCRIPT

cloudera@ DVS > cat sample1.pig

ll *.pig
ll *.hql

SPLIT A into B if deptno==10, C if deptno==20, D if job=='MANAGER';

store B into '/home/training/dvs/dept10-data' using PigStorage(',');
" C " " " "/dept30-data' " " (';');
" D " /home/cloudera/DVS/mngr-data';

> pig -f sample.pig → default mapreduce

> pig -x local -f sample1.pig → data in Local so change to local

DVS > cat sample2.pig

A = LOAD 'sa' using PigStorage(',') as (eno:int, ename:chararray

DVS > pig -x local -param a=/home/cloudera/DVS/emp.csv -param b=/home/cloudera/DVS/b-out -param c=/home/cloudera/DVS/c-out -f sample2.pig.

x - execute
p - param
m - param_file

DVS > cat pig param.txt

DVS > pig -x local (-param_file) pigparam.txt -f sample2.pig

-m

pig -x local -m pigparam.txt -f sample2.pig

erence between Local & Mapreduce mode

Group and aggregate function

deptno	min(sal)	max(sal)
10 20 30		

g> B = group A by deptno;

g> C = foreach B generate group, MIN(A.sal), MAX(A.sal);

g> C = foreach B generate $0, MIN(A.sal), MAX(A.sal);

Grouping

1> Based on one-column

2> Based on multiple-column

3> all data in one single group.

B = group A by (deptno, job);

dump B;

A = emp.csv

-> mn, mx,

B = group A all;

g> dump B;

g> describe B;

g> C = foreach B generate MIN(A.sal), MAX(A.sal), AVG(A.sal), SUM(A.sal),

COUNT($1);

g> dump C;

$0 = group All

$1 = {} - - - }

1) No schema (A= load 'emp.tsv';

2) full schema (A: . . . .);

3) semischema (A=load 'emp.csv' using pigstorage(',') as (emp, ename,)

1) data will come at the comma separated or tab separated file

> A = load '/home/cloudera/DVS/emp.tsv';

* dump always represent in form of comma delimeter.

* storage will be tab separated.

3) A = load ('/home/...' using pigstorage (',') as
(empno : int, ename : chararray, job, sal, comm, deptno);

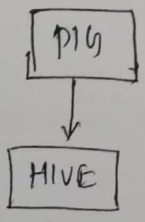
> describe A;

> ename : chararray, job : bytearray, sal : bytearray...

> B = filter A by sal desc;

* bytearray starts operation with ASCII values, so the ordering will not come properly.

* transformation / Arithmetic / mathematical formula will not work with bytearray.



→ No storage

→ HIVE stores data in hdfs.

* Any Reporting tool talk with HIVE, but not with PIG, as file can be used but table on HIVE can be accessed by any tools.

Unstructured data

> cat test.txt

O/P (word occurance)

This, 2
Is, 4
DVS, 1
. . .
. . .

grunt > A = load 'home/cloudera/DVS/Test.txt' as (line:chararray);

> describe A;

A: {line:chararray}

> dump A;

* textfile default delimeter as space.

TOKENIZE

TOKENIZE (col-name, 'delimeter')

Default is 'space'. Apart from space, we need to specify delimeter.

NOTE
- Excel – Transpose
- SQL – PIVOT
- HIVE – EXPLODE
- PIG – FLATTEN

* converting row into the columns

({(this),(is),(dup)}) → nested tuples.

Steps

1) A = load 'home/cloudera/DVS/Test.txt' as (line:chararray);
2) B = foreach A generate TOKENIZE(line) as Token;
3) C = foreach B generate FLATTEN (token) as flat;
4) D = group C by flat;
5) E = foreach D generate $0 as word, $1 as word_occurance;
6) F = store E into '/b' using PigStorage(',');
7) G = order F by $1 desc;
8) H = filter G by $0 != 'is';
9) I = limit G 1;

rating < -1

grant > explain sampler.pig

> run sampler.pig

* real-time implementation-to query huge as not string data for retrieval.

> cat emp.csv emp.csv > empp.csv

> A = load empp.csv

1) distinct records
2) distinct jobs

g> B = distinct A;

g> C = foreach B generate job;

g> D = distinct C;

g> dump D;

or > group by job (Analyst, {(3)}

(manager, {( )3)}

NOTE → distinct is better than grouping for getting data.

→ distinct can be applied on the bag not a column or subqueries.

=>

A = load 'emp.csv' ... ;

B = filter A by deptno == 20;

C = order B by sal desc;

D = limit C 10;

→ D = limit (order (filter A by deptno==20) by sal desc) 10;

* Innermost query gets executed first followed by outer queries.

=> JSON DATA

cat sample.json

grunt> A= load '..... /sample.json'
(food: chararray)

JSONLoader → JSON to CSV.
JSON Storage → CSV to JSON

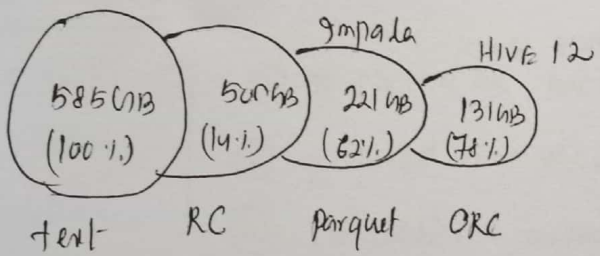
A = load emp.csv
convert from CSV to json.

{"empno": "7369", "ename": "smith", "job": "clerk" ... }

Store A into ' ' using JsonStorage

RC - Record columnar format
ORC - optimized Record columnar format

{31.12.2018 Monday}

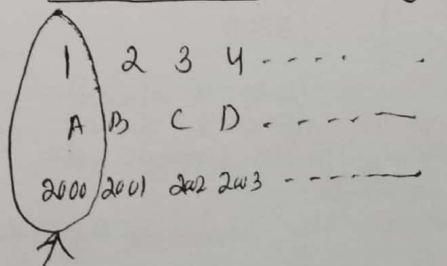


encoded with RC / ORC / parquet / text

Raw based Storage

Sid	Sname	Yop
1	A	2000
2	B	2001
3	C	2002

columnar based storage



HDFS Storage →
1, A, 2000
2, B, 2001
3, C, 2002
4, D, 2003

RC file format

DVS > jps

su -

jps

HIVE

→ create table emp( )

row format delimited fields terminated by ',';

H> load data local inpath '/home/cloudera/DVS/emp.csv' into table emp;

Hadoop fs -ls /user/hive/warehouse/dvs-revised.db/emp

hadoop fs -cat /emp/emp.csv

H> create table emp-rc(

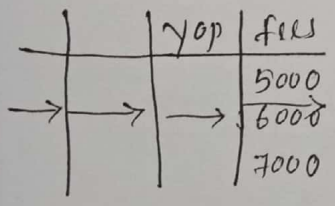
terminated by ',' stored as rcfile;

H> insert into emp-rc select * from emp;

1) How data stored in backend

2) How data retrieval happens

RB



* H will traverse row by row with each records in row based

RCFile

X

X A

X 2000

✓ 5000

* Aggregation data retrieval RC is faster.

Hadoop fs -cat /..... /emp-rc/000000-0

... only understood by HIVE.

Static partition

→ from csv to table

→ LFS format <=> hadoop format

* HIVE doesn't read the data only take and put in the required directory

hive uses schedim on read not write.

Traditional DB schema on 'read'
schema on 'write'

cat emp.tsv

null null null null null

hadoop fs -du emp_rc;

- dynamic partition
- buckets
- fileFormat

temporary table insertion.

- LFS format # hadoop format
- some manipulation is done while storing, you have data for some intelligence to do with temporary table.

DML in HIVE

Insert
Update
delete

① It should be bucketed table
② It should be orc file format
③ Transaction manager locked

emp

000000 - 0
00000 - 1
- - -
00000 - 10

small no of large files ✓ designed for

large no of small files →
- metadata increases
- NN carries all the metadata.
- 1KB metadata get generated for 1 block of data in Datanode.

* Metadata should be less

Update

delete original → new bucket

- copy of file will be taken, modify, delete, & replace.

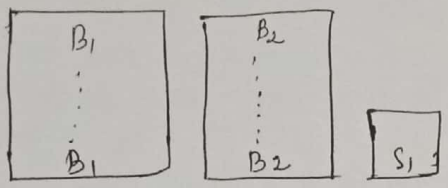
- No History maintained only updation/deletion done.

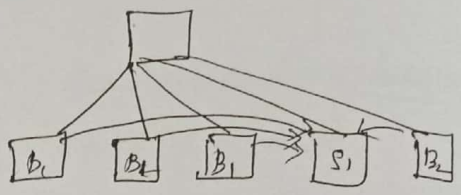
HIVE OPTIMIZATION TECHNIQUES :-

1) partion / Index / Buckets

2) Map Joins (Mapper side joins) complete join happening in mapperside.

set hive.auto.convert.join = true;





→ Replicated over the buffer/cache memory at each data node.

→ To reduce the data transfer over the Network.

→ session oriented set hive.auto - . . . . If changed in hive-site.xml the will be for ever for the whole cluster.

3) vectorization (instead of processing Record by Record, it will process bunch records at once)

Default → 1024

HIVE > set hive.vectorized.execution.enabled = true.

4) fileformats (SEQ, ORC, PARQUET)

5) controlling the no. of Reducers

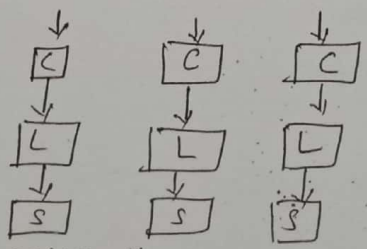
hive > set hive.mapred.reduce.tasks = 4;

6) parallel execution

hive > set hive.exec.parallel = true

Sample.hql

Direct Acyclic Graph



NOTE → By default Reducer in sqoop = 4

8) Data De-normalized data (saves lot of time interms of joins)
- Joins will be faster.

9) final output compressions (compressing final output in HDFS)

SET hive.exec.compress.output=true;

SET mapred.output.compression.type=Block;

SET mapred.output.compression.codec=org.apache.hadoop.io.compress.SnappyCodec

10) Intermediate compression

→ hive.exec.compress.intermidiate=true;

11) CBO (cost based optimization)

set hive.cbo.enable=true;

set hive.compute.query.using.stats=true;

12) using TEZ Execution engine

set hive.execution.engine; → MR
TEZ
SPARK

TEZ - only one read and one write
- get the data from the disk, perform operations and give output, instead of having 5-6 disk read/write like MR.

PIG OPTIMIZATION TECHNIQUES

1) push up filters to avoid unwanted data and add foreach to remove unneeded column.
2) if there is a chance, avoid filter and prefer split by.
3) Limit optimizer - Limit as early as possible.
4) pushDownJoins - Apply join as late as possible.
5) Specialized joins like Replicated, skewed & merge joins.

9) Error handling

pig -x local -f sample.pig.

-f or "stop on failure"

10) prefer DISTINCT over group by / generate

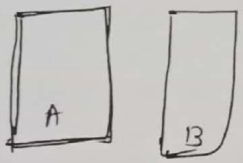
A = foreach A generate deptno;
B = distinct A;

B = group A by deptno;
C = foreach B generate $0;

Specialized joins in pig

1) Replicated – Mapside joins

2) Merge joins



→ when 2 datasets are bigger, more memory.

A1 = order A by deptno
B1 = order B by deptno

→ C = Join A1 by deptno, B1 by deptno using

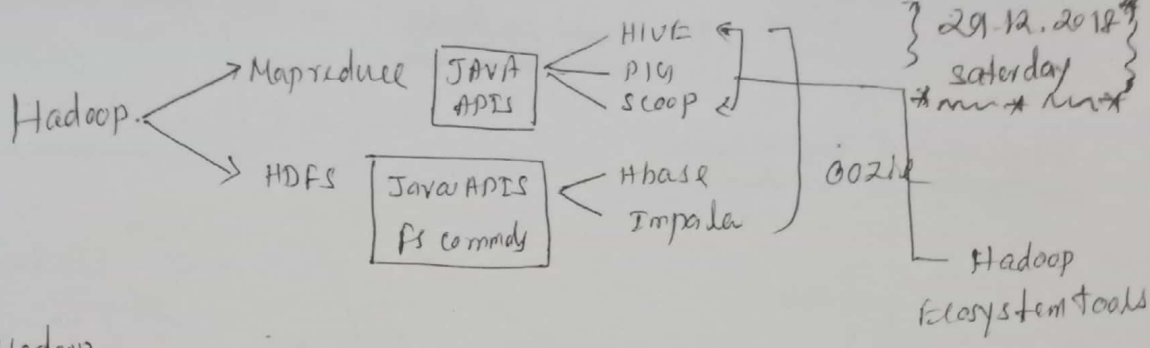
3) SKEWED →

R_1	R_2	R_3	R_4
5	5	10	10 min

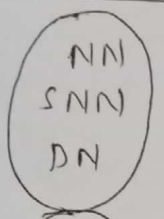
→ Evenly distribute the Reducer task to the free Reducers.

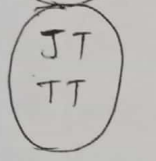
MAP REDUCE

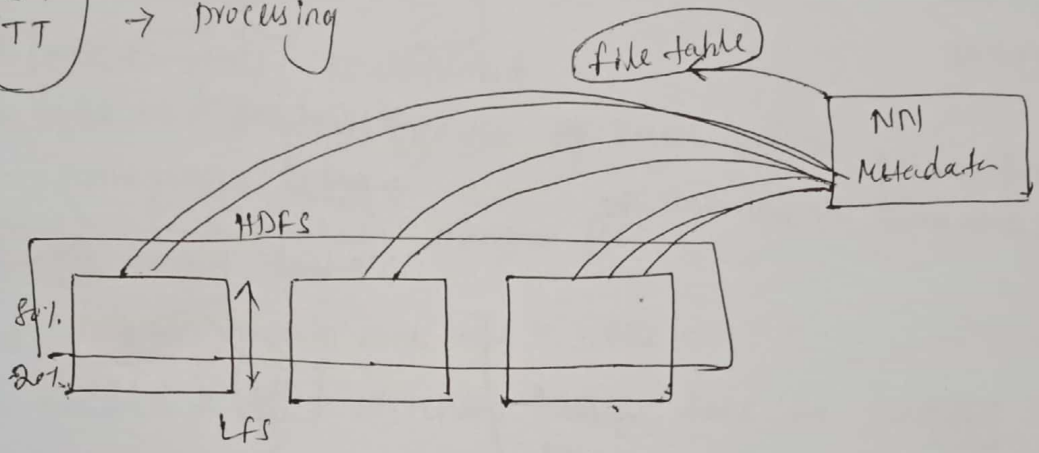
29.12.2018
saterday



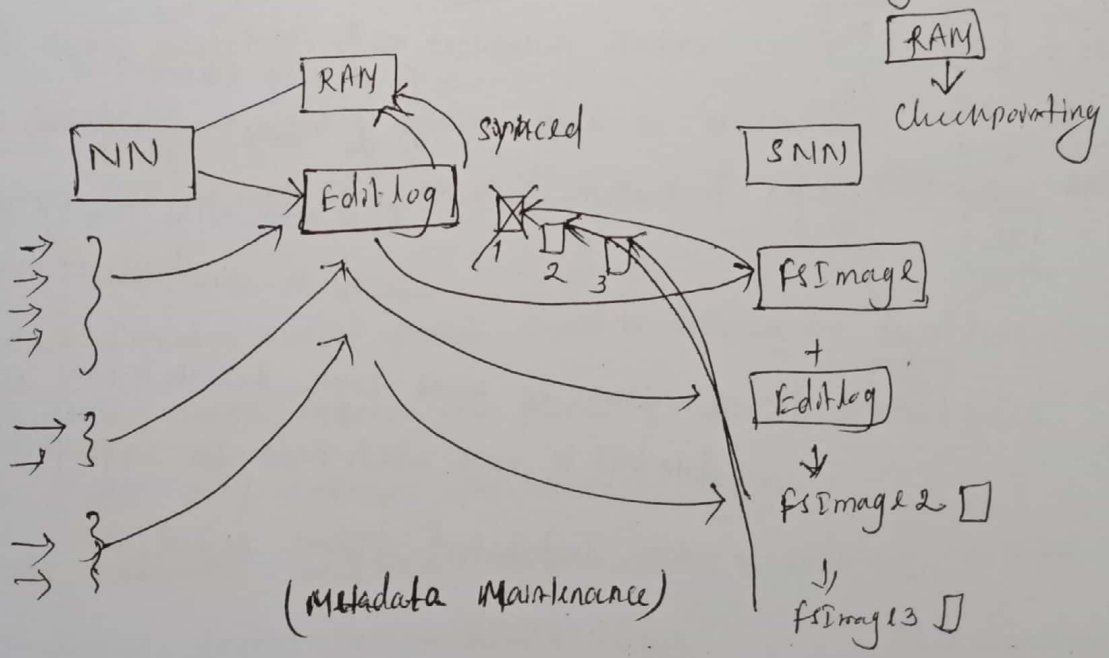
Services for Hadoop







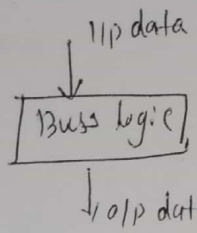
Each LFS shares their space for the HDFS filesystem storage.



* only 2 images will be there.
→ Restart NN the fsImage3 will be go to RAM.

MAP REDUCE

Single system computing model.



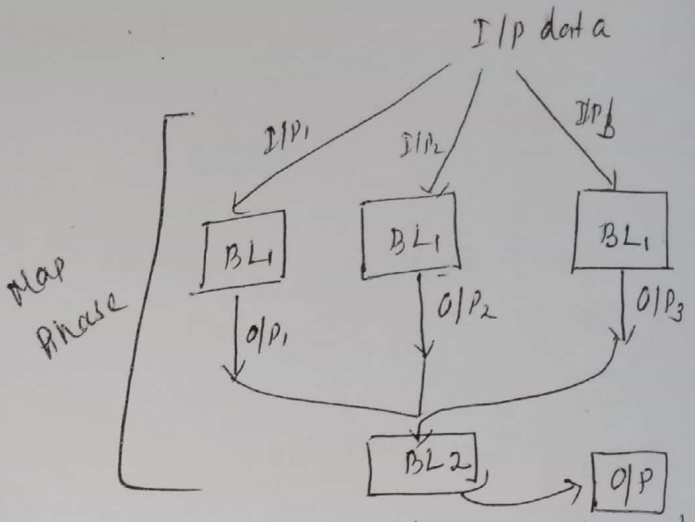
- RAM
- HDD
- CPU cores
- 64 bit OS

performance

→ vertical scaling

→ As the data grows the performance will be less.

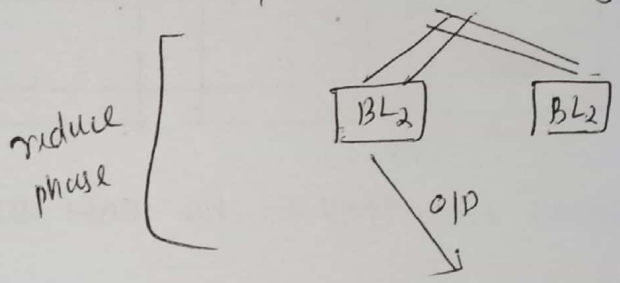
Distributed system . Computing model



→ select prod-id, sum sal bkt groupby prod-id

→ partial aggregation

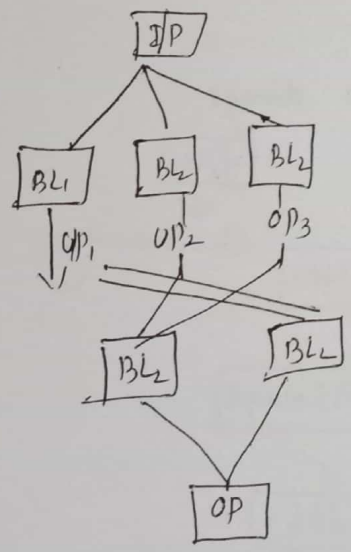
→ 2 level business logic



→ 1) do only selection — product id, salary

2) do only aggregation → sum, group by

SUM, MAX, MIN, group by all aggregation will be in 2nd phase



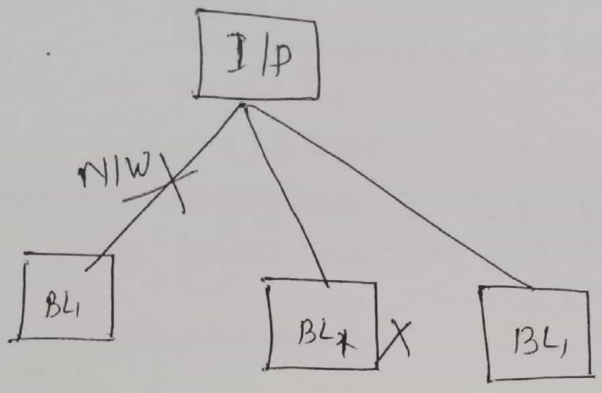
* we have to write logic in such a way that selected data will be go to which node for aggregation function

* This distributed system called as MAP-REDUCE

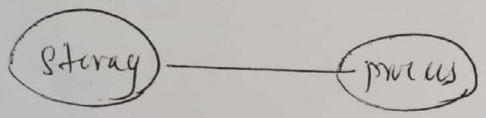
Approach DSC

1) Map-reduce
2) MPP - Massively parallel process
3) CORBA

Map and reduce phase can be n numbers depending query.



> If N/w will fail, how to transfer data to systems.
> After copying the node failed
> How data distributing.
> How to ensure phase I is completed.
> How to ensure final output go the single system.



* data and process in same node - HDFS
* No need to go datawarehouse all the time. for processing.
* processing sent to node where data.
* High availability → replication factor
* Reliability → without failure data in required node.
* same API is used for structed, unstructed same Map reduce API will be used.
* Input format & output format should be followed by Mapreduce.
* Job tracker keeps ensure the processing the job at the nodes.
* As a user only writing the business logic
* The framework will take care of all the activities. so in market Hadoop is highly acceptable.
* write once and read many times, concept.

Overview of MapReduce

1) Distributing computing model - programming model.
2) written Java
3) It provides MR APIs (JAVA APIs) to develop MR applications.

↓

* classes
* interfaces
* abstract classes, * static classes.

↓ * .jar files.

4) Available as .hadoop-core-<v>.jar.
5) By default the MR applications are developed in JAVA.

↓

can be in C/C++/scripting (shell, python, ...

Native support - C, C++ ⇒ hadoop pipes

other language ⇒ hadoop streams

6) Data representation in Mapreduce should be always as a key & value pair.

↓

aggregation, partitioning will be easy.

7) Mp applications generally have 2 phases
   1) Map phase ⟷ select part of query
   2) Reduce phase → group by and aggregate functions.
8) MR Application / Job / Algorithm = [task] = [[Map tasks] + [Reduce task...
9) MR always read/write data from/to FS (filesystem) of Hadoop (HDFS)
10) The Map logics and reducer logic will take / give input / output as key-
11) Though we use Java to develop MR application, the data i.e. key vale pairs represented as hadoop datatypes., because hadoop uses its own serialization to represent key & value objects.
12) Map logic will be written in a mapper class and reduce logic will be written in a reducer class.
13) Set Job properties & Running → Driver class

Mapper class() → Map()
Reducer class → reduce()
driver class → main()
using a Job object
} *.jar

> hadoop jar *.jar Driver_class_name i.p-path o.p-path

Map - structured / semi / unstructured
HIVE - structed
Pig - structed / unstructed

Mapreduce Architecture

Master-slave Architecture
Job Tracker (JT)
Task Tracker (TT)

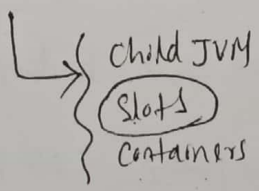
Hadoop 2x
→ Resource Manager (RM)
→ Node manager (NM)

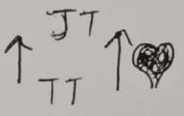
Job tracker:

responsible for resource management
Responsible for job scheduling
Responsible for job execution → job coordination, failure management etc.
Maintaining job history →

Task tracker

executes tasks assigned by Job tracker.
The task can be Map task or the reducer task.





→ task status or progress information

Resource manager

→ Resource management
→ Job scheduling
→ Job execution is deligated to Application Master (AM).
→ only one job can be taken care by the Application Master.
→ Job history will be maintained / handled by job history server. (JHS)

JT & TT are daemon processes, whereas RM & JHS are daemon processes / background process running.

AM & JHS are ...
not ...

* one task one JVM

in 1.x JT have ... tasks of memory as ... to do

Node manager

→ Execute the tasks - either M/R tasks
→ provides container for AM.

Child JVM
Slots
container / yarn child

* assigned with some Java memory.

RM instructs the NM to allocate the memory for the task and application manager as assigned for the job execution.

1.x
4 slots
2 (map) 2 (reduce)

→ reservation for the slots.

2.x
4 slots
4 4

→ No reservable for slots

MyMRApp.jar → MyMapper.class
MyReducer.class
Mydriver.class

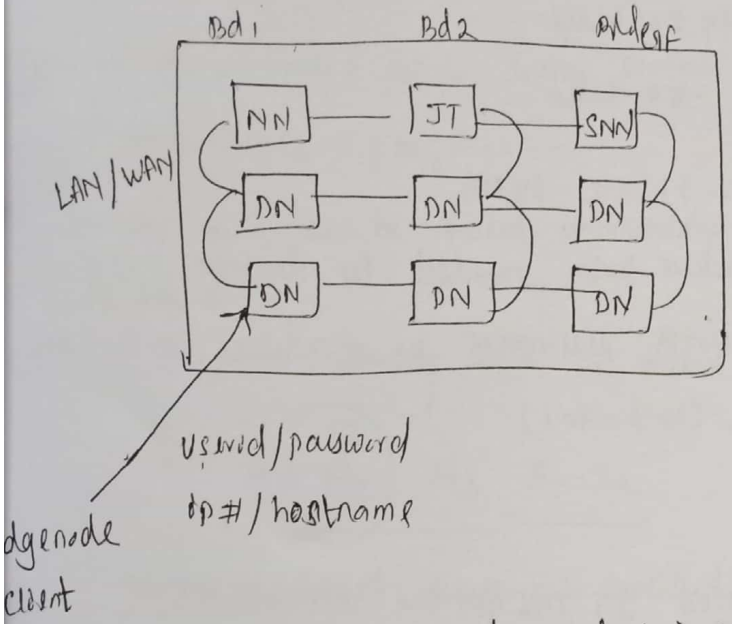
~~To run the job~~ To run the job ⇒ hadoop jar <jar-file-path>
<driver-name> <i/p path> <o/p path>

⇒ hadoop jar /home/hari/MyMRApp.jar org.dus.mr.mydriver i/p o/p
LFS fully qualified class name HDFS HDFS

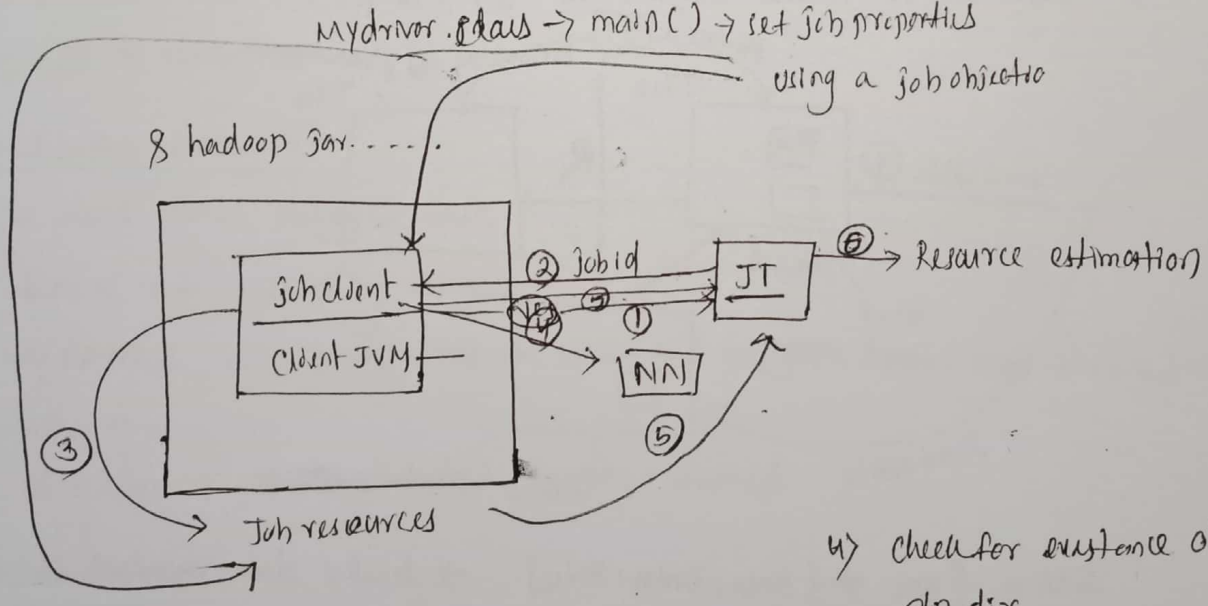
* According to memory slots are decided. 512MB RAM for one containers.

- hadoop jar → JT
- hadoop fs → NN
- hadoop dfs, hadoop dfsadmin → NN with admin
- → shell scripts where we can pass the parameters as fs, dfs, admin

* in hn directory it can be found.



* ssh protocal, they can talk to each other.
* one system access should be there for a developer
* from edge node you can run jar files.



4) check for existence of op dir.

→ Job client will request JT
→ JT provide Jobid to reference future.
→ Job client prepare job resources.
  - jobid info
  - job configuration file (job specific properties)

8

- input splits into - blocks nos.
- make a copy of jar file → MyMRApp.jar
- make a copy of miscellaneous files/supporting files.

→ copy to HDFS → cluster (nodes) shared location.

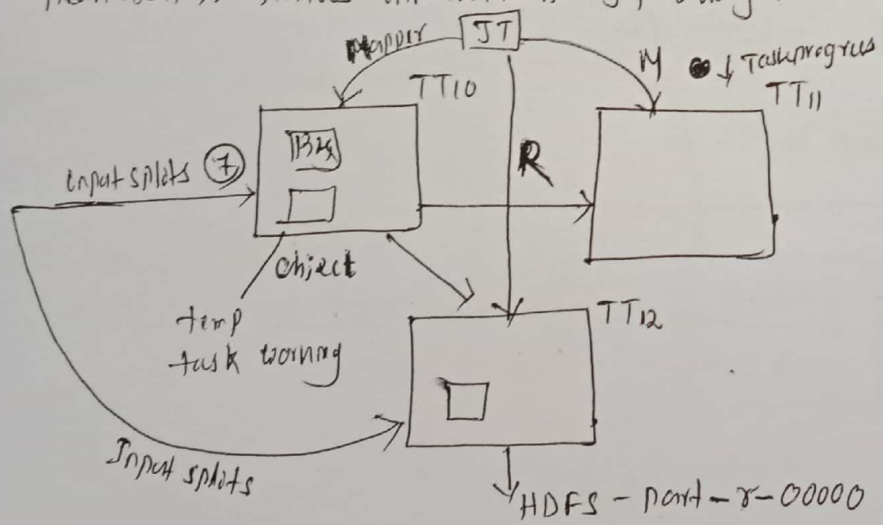
→ check for existence if yes submit to JT, by sharing the job resources.

→ Resource estimation: estimating #M-tasks and/or #R tasks

→ # Mappers = # Input splits - 1 block - 64 MB

Reducers = ① mapred.reduce.tasks provided by developer. e.g.:1
developer: Explicit
② The framework estimates the #reducers, with some algorithm. (10% - 90%)
Implicit

→ Heartbeat & status information JT will get the information.



file = { BK1 → {DN10, DN15, DN4} → locally data available for the file
BK2 → {DN11, DN21, DN22}

→ assign the map task.
→ If DN - not free for BK, it will assign to adjacent node and take copy and assign the map and process & delete temporary data.

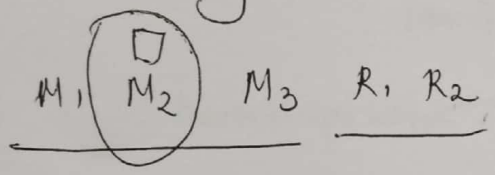
→ Mapper definition will be read
→ create the object in the TTID
→ once the map task done it will assign reducer task.
→ The reducer task doesn't need data availability / locality.
→ After getting all information reduce will do aggregation and gives output to HDFS.

⇒ If the job failed due to wrong status

$DN_{10} \rightarrow DN_{15} \rightarrow DN_{16}$ ------

If any task can be failed continuously 4 time, the entire job will be marked as failed.

⇒ other running task are killed.



If one is failed whole task cancelled.

⇒ All the resources will be released.

2) Speculative execution –

→ to avoid process straggler task.
→ task is not responding 10 mins.
→ who finishes 1st it will take the result and make the other node to cancel the task.
→ at a time 1 speculative task can be started.

3) Hard disk failure

→ will be handled in other node, it will take all the information from the HDFS.

$ hadoop jar *.jar driver i/p o/p → if present job will not process.

↓
Main()

#M = # input splits

#R = mapred.reduce.tasks ← dev

F/W itself will do the estimation.

JT will find TT is free and they have free slots to get the task, Then assign the map to TT. some of the reducer works parallel to mappers. reducer will wait till mapper job done.

10% - 90% - number of reducers as per mappers.

input split resides at the file input format.

Mapper ↔ input splits ↔ file input format

R ↔ mapred.reduce.tasks ↔ job obj.setNumReducer(n)

1.x

2.x (YARN) - Job tracker heavy in 1x./ restriction for number of M or R task

- MR is rewritten.

Master - RM, JHS, AM/APP

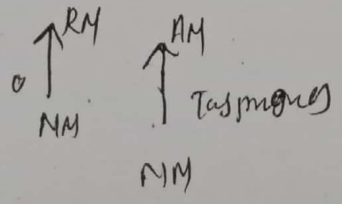
Slave - NM (Map logic, Reduce logic, AM logic)

yarn jar *.jar driver i/p o/p

↓

Yet Another Resource Negotiator

AM estimates the mapper and reducer and request to Resource m



JHS ← AM

- everything done AM handle History on JHS.
- cleans all memory, garbage collector.
- HDFS shared location to hold all the input splits.

→ Temporary working directory present in AM will get the input splits and calculate how many mappers and reducer needed and inform to resource manager to get the free container / JVM in node manager.

1) AM task
2) Map task
3) Reducer task.

JT → JVM
↓ M1
TT → JVM
M1 → JVM ✓
child

JVM ✗ ⇒ if one fails another will continue to work.
child

* once the HDFS written, the AM will maintain history, the signal goto resource manage, then resource manage will inform node manager to release the memory and resources for the application manager.

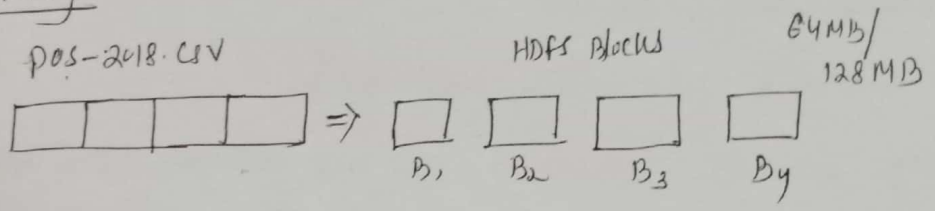
Hadoop 1.x

Spark Libraries
↓
local
client
cluster } resource manager
↓
→ Mesos
⇒ JT

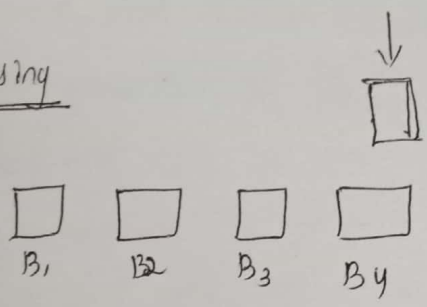
Hadoop 2.x → As spark already did the resource negation, in hadoop 2.x

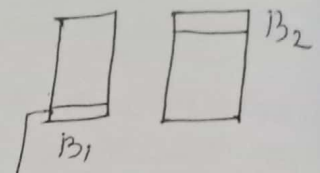
12 named as yet another Resource Negotiator.

Writing



processing





→ Array out of bound
Go records to retrie
where block have two rec

input splits → Logical blocks

IS1 = $B_1 + B_2$* → define data boundaries

$IS_2 = B_2$ → unit of data on which a map can be run logic

$IS_3 = B_3 + B_y$*

$IS_y = B_y$ → #M = #IS.

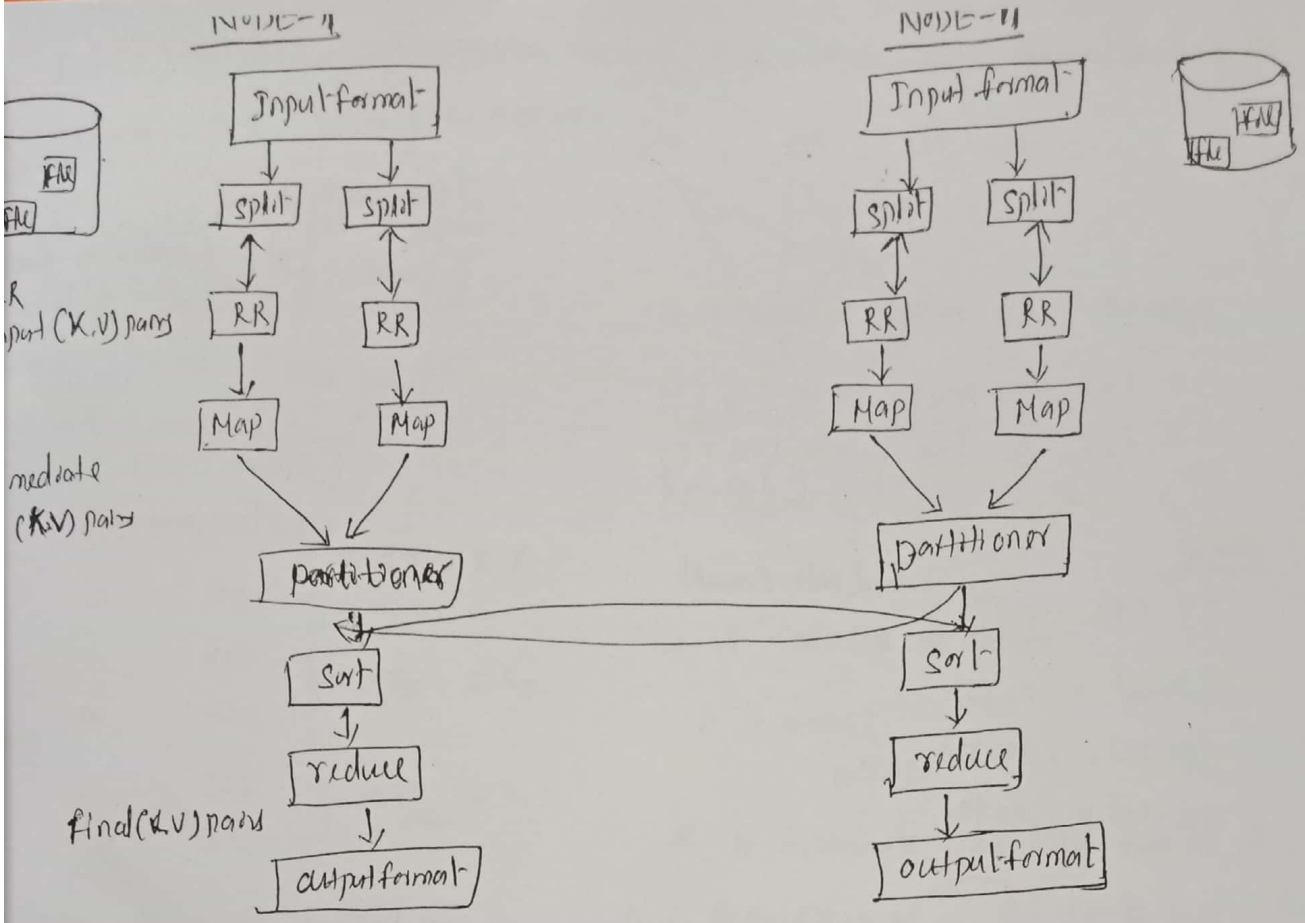
properties

Input split size → Min
→ Max

Input split size ≡ Block size

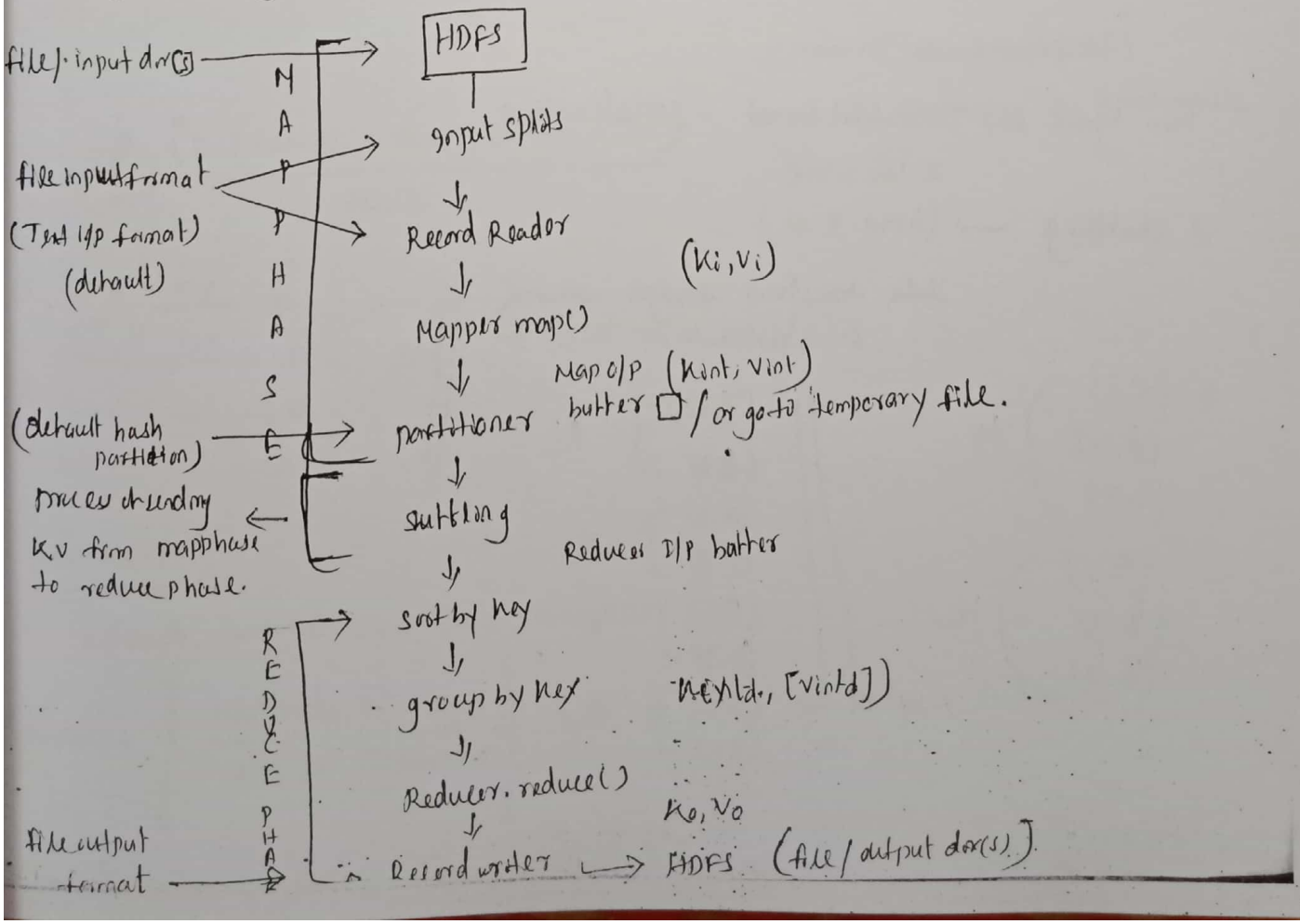
NOTE → Hadoop read is faster than writing.
→ writing is sequential, reading is parallel.
→ pipeline writing / sequential writing.

1) How to see the blocksizes of a particular file after copying to hadoop.
2) Getting the files from the materials folder.

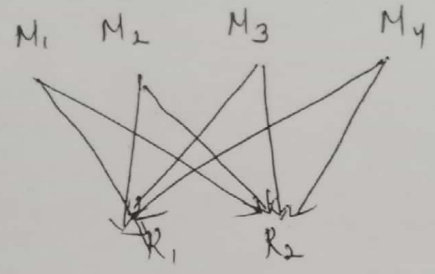


partitioner will decide which pair will go to which reducers.

Sorting, grouping done by the framework, then data will go to outputformat.



→ partitioner will ensure same K,V go to the corresponding Reducer.



(key, value)
(Santoor, amt) → Comes to R

output dir — HDFS
(driver class)
(job properties)

(S, -) (Deors, —)
(D, -) (dave, -)

Output

(S, [ ]) = (S, TS) → total sale amount
(D, [ ]) = (D, TS)
(De, [ ]) = (De, TS)
(d, [ ]) = (d, TS)

→ Semi struct
-structured → Entity Extraction → structured

* It will try to convert as much possible to structed data.

→ describe formatted table

(textinputformat)

HIVE ignore key Text output format - serialization.

→ shuffling → (hindi, virat)

data transform through network

→ (S,X)
(d,Y)
(C,Z) → M1

(D,Y)
(A,X)
(B,Z) → M2

(C,Z)]
(S,X)] R1
(d,Y)]
(d,Y)] R
(A,X)]
(A,X)] R
(B,Z)]
(B,Z)] R

Reducer > 0 → Sorting in mapper

Reduce <= 0 → Sorting in Reducer

Date	Trnsid	prodid	amount
9/2/2016	1000	A	100
9/2/2016	1001	B	200
9/2/2016	1002	C	300

* line number will be the offset value.

(0, 2017-01-01, 1000, A, 100)
(1, 2016-01-01, 1001, B, 200)
(2, 2016-01-01, 1002, C, 300)

A 100] A 100] $65/2 = R_1$

B 200] B 200] $66/2 = R_0$

C 300] C 300] $67/2 = R_1$

* 2 reducers are there

n → map()
n times

* to distribute the load more no of Reducers.

Example

file Input format

RR → 0 → 2018-01-01 1234 santosh 1 26
rd → 121 → 2018-01-01 1234 dettol 2 40

* RR will decide how to read
* one record per one line
* (convert to key, value
* 0, 121 character as ~~input splits~~ offset value

0, ("2018-01-01, 1234, santosh, 1, 26")
key value

0 → 0 120 → n
1 → 121 120 → n
2 → 242 → n
3 → ------

* offset value calculator.

Partitioner

Default partitioner ⇒ Hash partitioner

Modulus (hashcode of key / num of Reducers) = 0 to (n-1) ⇒ Reducer Index

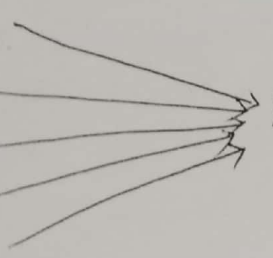
partitioner

↓

int getpartition()

↓

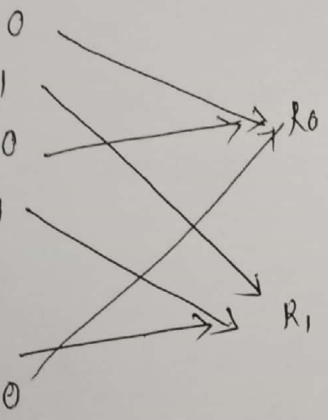
Keys	n=1
10	|10/1| = 0
11	|11/1| = 0
12	|12/1| = 0
13	|13/1| = 0
14	|14/1| = 0
⋮	
20	



n=1

r0 → 11 — part-r-0

Keys	n=2
10	|10/2| = 0
11	|11/2| = 1
12	|12/2| = 0
13	|13/2| = 1
⋮	
19	|19/1| = 1
20	|20/1| = 0

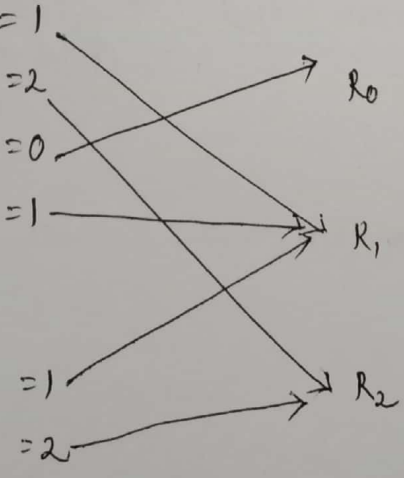


n=2

r0 = 6 part-r-00000

r1 = 5 part-r-00001

Keys	n=3
10	|10/3| = 1
11	|11/3| = 2
12	|12/3| = 0
13	|13/3| = 1
⋮	
19	|19/3| = 1
20	|20/3| = 2



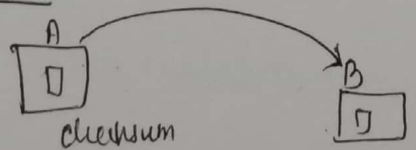
n=3

r0 = 4 part-r-00000

r1 = 4 part-r-00001

r2 = 3 part-r-00002

Checksum



* If checksum same at B then file not corrupted.

Map reduce flow example Hari - Excel

HDFS Reliable

How stored data in HDFS is reliable.

NN
↑ ♡3sec
DN

→ every 3sec ↑ block report informations
→ Checksum of each block is also mentioned.
→ file ; Blocks ; checksum
→ After 30 sec it will match the checksum for corresponding blocks.
→ If down it will replicate in another node & delete the corrupted datablock.
→ Namenod continuously doing the checksum matching.

HIVE

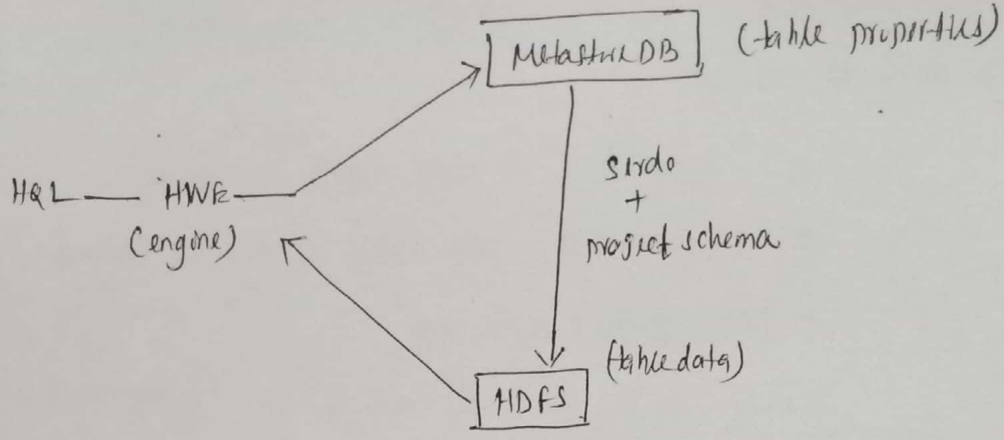
Metastore DB

Tbl properties

→ Table-names-location
→ column names-datatypes
→ I/p O/p format
→ partion info
→ Bucking info
→ function definitions.

HQL → Hive (Engine)

1. syntax check
2. resolve the query
3. logical plan
4. logical generate MR code
5. generate & execute physical plan [HDFS]
6. output → console / HDFS



Skeleton code for Mapper, Reducer, Driver class

04.01.2018
Friday

3 classes for MR Application

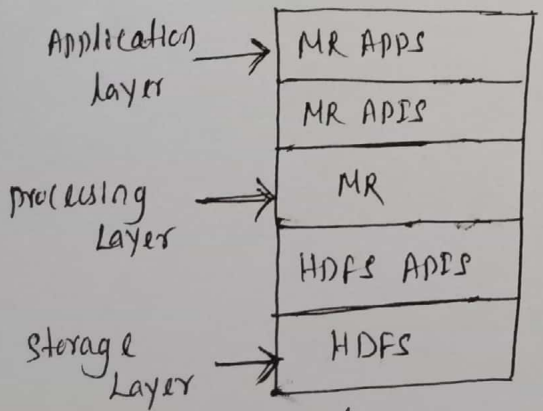
1) Mapper() → Map()
2) Reducer → Reduce()
3) Driver → main()

} compile

*.java ⟶ *.class ⟶ *.jar

→ Eclipse 'export' option
→ Maven build
→ gradle build
→ Ant build

MR APIs to develop MP APPS



Code for Mapper class:-

```
public class MyMapper extends Mapper<Ki, Vi, Kintd, Vintd>
{
    @Override
    void map(Ki, Vi, contextobj)
}
```

```
    contextobj.write(KIntd, VIntd)
  }
}
```

Mapper(ki, vi ....)

API class from MR APIs

↓

*.jar

↓

hadoop-core.jar

↓

import org.apache.hadoop.mapreduce.Mapper;

Skeleton code for Reducer class

```
public class MyReducer extends Reducer<KIntd, VIntd, Ko, Vo>
{  @Override
   void Reduce(KIntd, Iterable<VIntd>, Context obj)
   {
     ---------
     ----Actual reduce logic--
     ---------
     Context obj.write(Ko, Vo);
   }
}
```

Skeleton code for Driver class

```
public class MyDriver
{
  public static void main(String[] args)   ①
  {
    ② // create job object
       Job myjob = new Job();
    ③ // set your job properties through job object
       myjob.setJobName("MyfirstMR");
       myjob.setMapperClass(MyMapper.class);
       myjob.setReducerClass(MyReducer.class);
```

// submit and run the job

myjob.waitforcompletion(true);

}

}

1) first All 3 site.xml file

+

2) Adding job specific properties

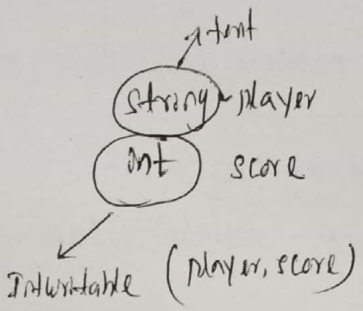
=

Job configuration.

Ex: Data profiling

↓

Data understanding.



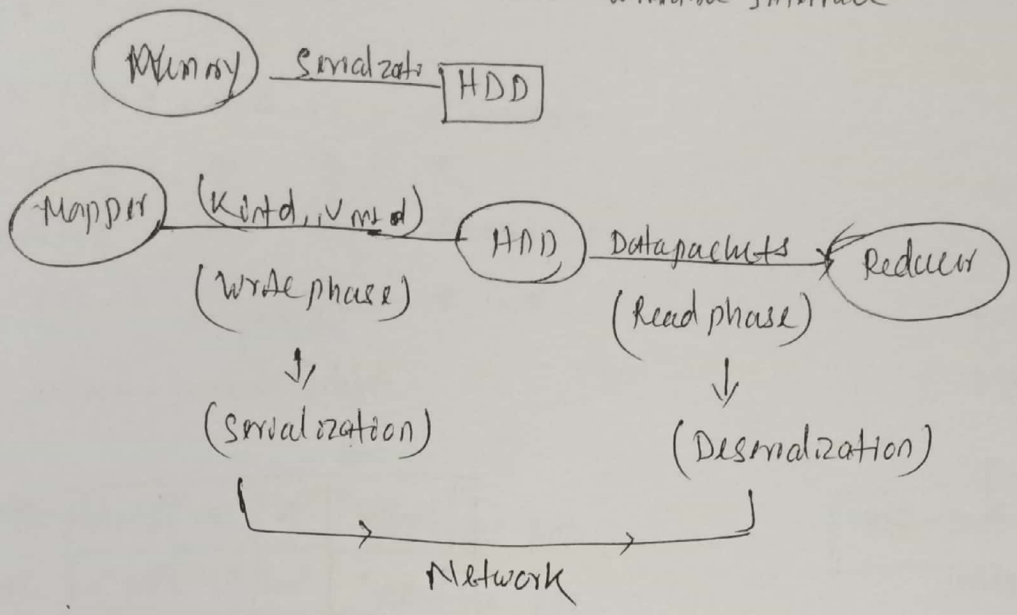
JAVA	HADOOP
Int	IntWritable
long	LongWritable
float	FloatWritable
double	DoubleWritable
String	Text
⋮	⋮

wrapper classes

class implements Writable Comp

* Hadoop have it's own serialization which is different from Java.
* Hadoop serialization is implemented through "Writable" Interface.
* 1) Read fields() || 2 methods of "writable" class
  2) write()
  - serialization
  - Deserialization.
* "Writable Comparable" Interface has addition method-
  1) compareTo() + iset()
  2) Read fields() .get()
  3) write()

Writable comparable class extends from Writable Interface.



Same content can be serialized in different ways - 16 bit, 32 bit .....

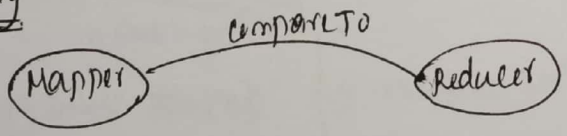
Java serialization not optimized / slow to handle Hadoop serialization / to represt data heap memory Java processing with writable Interface before sending to HDD.

```
{ IntWritable x = new IntWritable()
  x.set(10);
  x.set("DVS");
```

* object writing and Reading will take more memory.

or// `x = new IntWritable(10);`

Shuffling



To compare and sort the keys compareTo needed.

set() ⇒ int → IntWritable ⟶ Boxing

get() ⇒ IntWritable → int ⟶ unboxing

Saturday

Sachin - 98
Kohli - 100
Sachin - 120
Kohli - 150

[K, [V]]
↓
aggregation

Mapper<ki, vi, kintd, vintd>
↑ LongWritable ↑ Text ↑ Text ↑ IntWritable

↓ toString()
Split("\t");
↓

"Sachin"	"98"	"120"	"4"
"Kohli"	"100"	"132"	"150"

← (list of record)

[0] [1]

Context obj. write(kintd, vintd)

* Map() will be done one by one for all the records.

public class MyReducer extends Reducer<kintd, vidtd, Ko, Vo>
↑ Text ↑ IntWritable ↑ Text ↑ IntWritable

@ override
void reduce(kintd, Iterable<vintd>, reduce Context obj) {

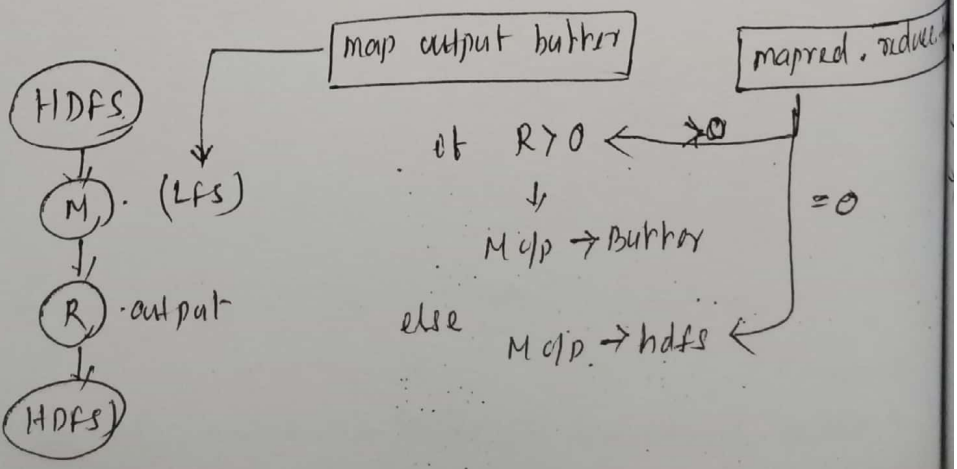
[K, [V]] ... per key one one reducer

Reducer logic
int sum=0
foreach value in list
{ sum=sum+value.get()
}

reducerContextobj.write(Ko, Vo)
reducecontextobj.write(kintd, new IW(sum));
}

Contentobject:

Context inner class



> which hadoop
ed lib
cd hadoop
ls *core*
cd ..
find . | grep -i *hadoop-core*.jar*

vat $ ls -l hadoop-common*.jar

1) Client

MR old API — MR New API
MR 1.x API — MR 2.x API

NOTE - Don't mix the APIs

old MapReduce baseclass → extends Mapper (abstract class)
org.apache.hadoop.mapred → org.apache.hadoop.mapreduce

InputFormat → input split logic
→ Record Reader logic

TextInputFormat → data boundaries are based on "\n"
→ take linecount as key and
linecontent as value;

outputformat → TextOutputFormat

RecordWriter logic — str(k) '\t' str(v) "\n"

i.p [0] o.p [1] - main(args[])

add*
set*

FileInputFormat.addInputFormat(myJob, new Path(args[0]));

→ fully qualified class name - class with package name.
→ export : option in eclipse ← devphase
→ buildtools ← dev QA/* production

24 N

pwd

cd ~/desktop //

pwd

ls -lrt ; CD · MR-Progs/

jar -tf TotalScores.jar

TotalScoreMapper.class

Reducer.class

Driver.class

Meta-INF/MANIFEST.MF

→ Reducer task will start simultaneously, else the task will be go to the indefi loop and it may not get the reducer ~~map~~ slots.

C> WC -l cricketscores.txt

103

Mapreduce.job : the url to track the job: http://localhost:8088/

copy in HUE

Jobhistory server the details will come.

filesystem fs = filesystem.get(new configuration)

fs.delete("path", true);

→ // delete output directory if exist — core.xml

file system hdfs = filesystem.get(new configuration());

hdfs.delete(new Path(args[1]), true);

// submit the job

→ filesystem hdfs = filesystem.get(new configuration());

hdfs.delete(new path(args[1], true);

- Reducer can be set directly

inputsplit size setting will change the mappers().

myJob.setNumReduceTasks(4);

hadoop jar ABC.jar Driver i.p o.p → Directory

(1) directory: All files in the dir
→ Do not consider subdirectories

2) directory location file name

3) directly file names with wild chars

/dvs/data/*.csv

/*2018*.csv

4) multiple directories → inputfile.addpath()

MR1
↓
R2
↓
MR3

Maponly Jobs → where no reducers are there
→ simple transformation without aggregation
→ sorting and ordering.

Identity Reducer → when reducer class commented, but no of reducer mentioned
→ sorted output
→ After map → shuffling → sort by, group by done.
→ input <=> output (same output)
→ Default Reducer = Identity Reducer.

→ No mapper – no Reducer

key	value
1	Sachin 13 9 2
1	Kohli 1 2 3
3100	Lyon 4 5 5

job.setNumReduceTasks()

mapred.reduce.tasks

default → MyMapper → Mapper obj → map

Mapred.reduce.class → MyReducer.class → Reducer obj → reduce

default

r=0 → No reduce phase

→ The o/p will be of Map phase o/p.

→ X partitioner, shuffling, sort by key, group by key *

r>0 → Identity Reducer

*

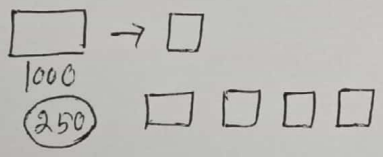
No custom Reducer → The o/p will be of Map phase o/p but in a sorted order.

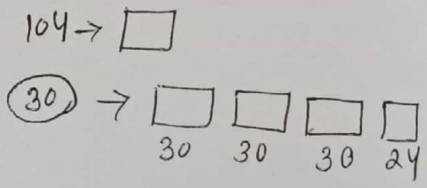
No custom mapper → Identity mapper

→ The o/p will be Record Reader o/p.

⇒ # Mappers = # Input splits

① NLine Input format ≡ Text Input format + No. of lines per split





② By setting the max. Input split size

<Input format>.setMaxInputSplitSize()

80MB = 64MB + 16MB

* good for large volume of data, else for small split also the allocated with 512MB, so not for small splits.

* overhead for creating container will be negligible for large file.> block size

⇒ KeyValueInputFormat

(sachin, 0, 32, 100 ...)

(0, "sachin, 0, 32 ...") → TextInputFileFormat

(key, value)

("sachin", "102, 30 ...") tab as default delimeter

(key, value)

DB Input format

→ sqoop replaces DBInputformat

TableInputformat (HBASE)

→ faster lookup of data

→ unstructed data processing

stream Inputformat

→ Json, xml format

→

SequenceFileInputformat

→ sequence file outputformat

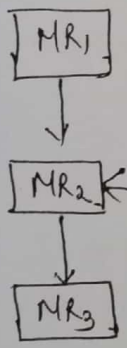
Reduce

Str (K_o) \t Str (V_o) \n — TextOutputformat writer

key value

SequenceFOF → K_o * V_o *

* Internal key value delimeter and delimeters



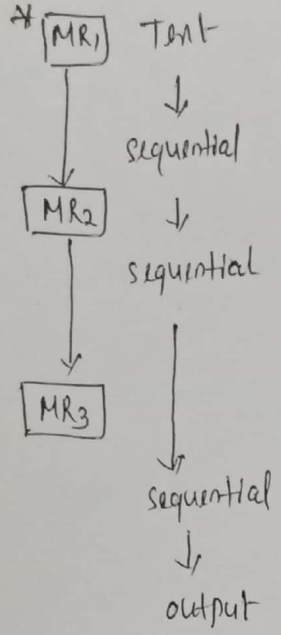
Java Datatype
↓
Map reduce Datatype
] Record Reader will convert

↓
Java Datatype
↓
Map reduce Datatype
↓
Java Datatype
↓
MRDT (K_o, V_o)
↓
JDT

Record writer
output

- MRDT will be directly input to then next MR2. No need to convert MRDT → JDT and JDT → MRDT

* Performance will be increased.



* outside HDFS ecosystem sequential file format can't be read.
* Hive, hbase etc can read the file and converts to human readable format.
* convert Text output format to sequential output format.

HIVE

Input → TextInputFormat

output → HiveIgnoreKeyTextOutputFormat

Yarn

* yarn jar Totalscores.jar

// yarn logs -applicationId job ... > jobs.log

yarn application -list

Generic options parser

Replication factor change setrep ds after writing the file.

hadoop fs -copyfromLocal -D dfs.replication=5 <LFS-path> <hdfs-path>

Java program

→ Internal client reads the xml files from config files

→ hadoop jar Driver -D mapred.reduce.tasks=0 i.p o.p

-conf ABC-properties.xml

* copy fromlocal ✗ put * hadoop.fs
* HIVE IgnoreKey Text output format

```
<configuration>
  <property>
    <name> dfs replication </name>
    <value> 3 </value>
    <description>
  </property>
</configuration>
```

`-p mapred.reduce.task=10` ✗ will not work as it don't know the -p.

```
public class MyDriver
{
  p.s.v.m(String [] args)
  {
    ① create job obj
    ② set job properties through job object
    ③ Run the job through job object
  }
}
```

3.site.xml + job specific properties (5) = Jobconf file

⇓

① 3.site.xml
② command line
③ Job → Jobconffile
↳ run the job

→ ToolRunner.x() ↗ run
↓
④ GOP.parse()
↓ config obj with cmd file properties or set
5 ↓ getconf()
y()
↳ run
↳ All the job task will be done by y().

public class MyGrep Driver

① extends configured

② implements Tool

{

p.s.v.m() ③

④ { ToolRunner.run()

}

⑤ run()

{

// create job obj with Grep configuration

⑥ Job. myjob = new Job (getConf());

⑦ // set job static properties

⑧ // run the job.

}

}

ToolRunner (new configuration,

Tool.obj,

cmd-line args)

Grep parser()

↳ Tool run () ← rest of the

run()

{

getConf();

}

⇒ Grep parser will get the cmd-line args and then read the configuration with new configuration and replaces the cmd-line values and create the new configuration file.

→ But it will not read the Input and output file values. Those values have to set in the Toolrun();

→ So on Run() with getconf() we will get all the configuration set values and with Input/output format set the files names as arguments with static properties

→ Then run the job. Grep creates the configuration object.

-1)

- config
- fs
- jt
- libjars
- files
- archives

Utils.ToolRunner.run()

↳ 1) invoke Grep

2) invoke Tool.run

Tool.Run

↑

Interface

ToolRunner.Run (new configuration, Tool obj, args

driver class obj

args [-D x1=y1 -D x2=y2 i/p o/p]

rest of arguments

Tool.Run(

{ Job myJob = new Job(getConf);

}

Yarn jar Total scores.jar org/bigdata/MR/MyMapDriver -D mapred.reduce.tasks=4 /dvs/data ------.

return (myJob.waitforcompletion

NOTE →

→ LFS we can check and run at local

→ -fs, -jt local read to local and write to local.

Yarn

-fs local -jt local ./home/cloudera ... @cricketscores.txt /tmp/op1/

hdfs://nn:port/

file:///

s3:// — cloud

→ Warning - "local" is a deprecated filesystem name. use "file:///" instead.

mapreduce.task.io.sort.mb=100

MapTask & MapperTaskBuffer

→ cat /temp/op1/

Local execution is faster than HDFS process, as the whole process needs to be involved as assigning container, namenode process, resourcemanagement etc etc.

-libjars
- files
- archives

submit → job-id created / Application-Id

→ It will take the jar file and ships to working node, and

Jar file location added to class path of task tracker

→ to temporary directory

→ doing join operation on MapReduce.

Distributed cache → -files

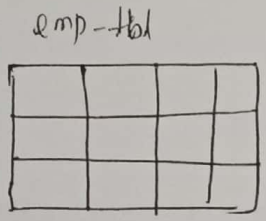
- archieves → compress file distribution
  → working node will uncompress the files and do their job.
  → default value - 10GB

* Nodemanager will allocate memory, it will take the distributed cache files and process them.

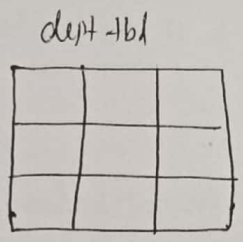
* Not used for Non-Jar concept / Non-Java platform.

JOINS IN MAPREDUCE

* Meant for structed data table.



A



B

Select A.*, B.* from emp A Join dept B on A.dept-id = B.dept-id

Map - 1 map logic - n mappers() → Map side joins

Reduce - 1 Reduce logic - n Reducers() → Reducer side joins

MultiInputFormat

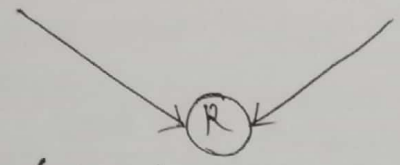
MultipleInputFormat.addFormat("fileA", "MapperA", TextInputFormat.class)
-path .class

MultipleInputFormat.addFormat(file-B-path, MapperB.class, keyvalueInputFo

* Hive operation doing joins do the multipleInputFormat in HDFs backend to do the operation.

(deptid, row+flag(A)) → to know which records are coming from. which t
Kontd vintd

(deptid, row+flag-B) (deptid, row-flag-A)

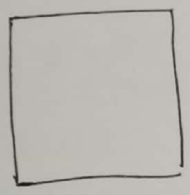


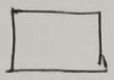
1) (deptid, [row+flag-A, row+flag-B])

2) (deptid, [row+flag-A])

3) (deptid, [row+flag-B])

Exam:-





cricketdata.csv playercountry.csv

(player, Totalscore) → (player, countryname, totalscore)

As the files is very small we can't keep in HDFS to involve a lot of process. so we can keep in the distributed cache to process our job.

Mapside join	DC	
Reducerside join	DC	Multiple Input Format

Adding a distributed cache file(s)

1) Crop => -files option => Refer files from LFS

2) In driver class => Distributed cache.addCacheFile() → mapred.cache.files
↳ => Refer files from HDFS

e.g → hadoop jar ABC.jar Driver -files /dss/lookup files/playercountry.csv
input path output path

e.g → Distributed.addcache file (new URI (lookup-file-HFs-path), jobobj.getconfe

Refer DC files in mapper/reducer

Distributed cache.getLocal cacheFiles (contextobj.getconf());

↳ mapred.cache.files

* will get this property through getLocalcacheFiles

1) get cache file name

2) Read file and create a lookup obj => Map obj List Array

player country . csv

Sachin, India
Dhoni, India
Warnin, Australia

split(',')

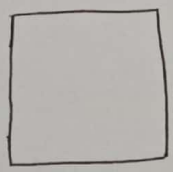
Map playercountry Map = new Hashmap <String, String>
(key, value)

"Sachin"	"India"
[0]	[1]

playercountry Mpp . put (split[0], split[1]);

("player #\t" + country ", totalscore)
(key, value) → Output

(player, list scores)
↓
sum
↓
(total score) → Map playercountry map



(FACT table) (Dimension table)

(200x100) (200) → Lookup table

↳ Transaction table

Distributed cache

→ files are needed are added to the job resources. and pass to the local node manager / worker node, the files will be distributed.

* Create the lookup object, and picked by mappers.
  - → No of mapper reducers
  - → Reducing I/O operation, as DC will be faster
  - → one time transfer of data through network.
  - → Mapside Joins are preferable compared to Reduce side Joins

EX:- select /* MAP JOIN */
a.*, b.* from tableAa, table-B b;

→ C = Join A by x1, B by x2 using replicated

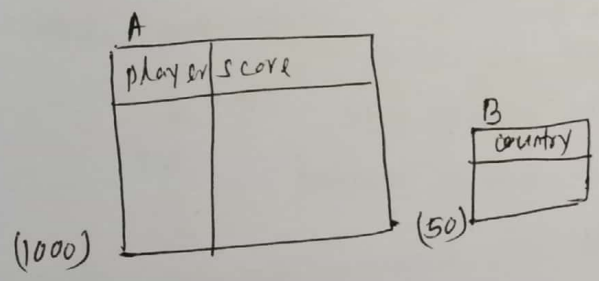
Left side	Right side
Large table	small table

→ scan operations will be costlier if big table will be in right table.

Reducer side Joins

Join → aggregation

aggregation → join → acceptable → still depends on the situation.



player, country, total score

case-2

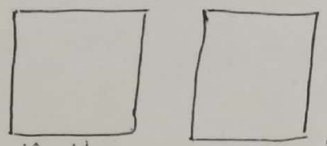
1000 → Join
records
45 → Avg → Join
records

→ Reduces joining cost operation.
→ Avg will be done in Reducer.

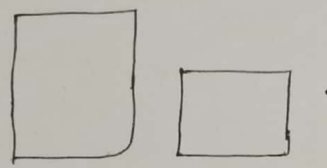
case-1

* If the aggregation depends on both the table, the Join is then aggregation and is acceptable

→ If both the files are bigger use MultipleInput format for joins.



→ Multiple Input format



Mapside → Distributed cache

Reduce side → Join → aggregation

aggregation → Joins

Default Methods

Mapper()

Reducer()

setup() → → 1st method to call
→ at beginning of the task
→ only once it will be called ← setup()

map() → → for n times
→ # records in the input splits

→ for m times
→ # m where m is the distinct keys that reducer received ← Reduce()

cleanup() → → Last method to call
→ At the end of the task
→ called only once ← cleanup()

* Distributed cache will be read at setup() method for one time read.
* HMapobj. will be created in the class
* Call in the setup() method, so it will be build
* Map() method it can be used for join operations.

sort by key
↓
group by key
↓
groupkey, [list value]
↓
Reducer (distinct keys)

- mkdir -p /lookup-data/
- hadoop fs - put playcountry.txt /lookupdata
→ ssh -o vi
→ jar ... TotalScoreMYDCDriver
/dvs /data/cricket-data/ /dvs/op1/

10.01.2018
Thursday

COMPRESSION

→ space
→ faster to transfer through N/w / Harddisk.

HDFS - Compressed file can be stored.
- No overhead for unzip of files in HDFS MapReduce.

cricketscores.txt
↓
cricketscore.txt.gz → Based on file extension it will invoke corresponding Decompression.
↓ HDFS
MR
↓
o/p → compressed o/p

TYPES

CODecs → compressor, decompressor

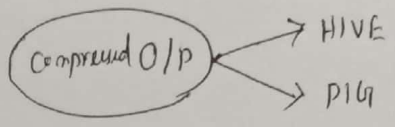
- default
- gzip
- snappy
- Lzip
- bzip2

codec → org.apache.hadoop.IO.compress.GzipCodec.class

1) through command line
2) on Job definition

hadoop fs -text <compressed file in hdfs loc> | head -10

on top of compressed o/p we can define the hive table.



.csv | .gz
.tsv | .bz2/p2
↓ | ↓
fileformat | file compression for

① C> file cricketscore.txt

ASCII text with CRLF line terminators.

C> gzip cricketscore.txt

desktop > file cricketscore.txt.gz

gzip compressed data

desktop> cat cricketscore.txt.gz

... symbolic representation

> gunzip cricketscore.txt.gz

② > gzip cricketscore.txt.

> hadoop fs -mkdir -p /dvs/data/gz/

> hadoop fs -put cricketscores.txt.gz /dvs/data/gz/

desktop> hadoop jar

> hadoop jar org/bigdata/mr/myDCDriver

/dvs/data/gz/ /dvs/op3/

> hadoop fs -cat /dvs/op3/

↳ player country scores

> hadoop fs -ls /dvs/op3/

③ MyDCDriver.java

FileOutputFormat.setCompressOutput(myjob, true);

FileOutputFormat.setOutputCompressorClass(myjob, ~~GzipCodec.class~~ GzipCodec.class

> hadoop jar Totalscores.jar

> hadoop fs -ls /dvs/opy/part-r-00000.gz

> hadoop fs -text /dvs/opy/part-r-00000.gz | head -10

player country scores ← output

-D mapred.output.compress=true -D mapred.output.compression.codec=
org.apache.hadoop.io.compress.GzipCodec /dvs/gz/
cricketscores.txt.gz

→ Generic options parser.

- download to local the configuration file
- search the compress
- Read default value
- set Generic options parser value

io.compression.codec

org.apache.hadoop.io.compress.DefaultCode.
- do - . GzipCodec
- do - . Bzip2Codec
- do - . SnappyCodec

mapred.output.compress — false
mapred.output.compression.codec — org.apache.hadoop.io.compress.DefaultCodec
→ Default values

File 1 GB
- A — 2min — 300MB
- B — 30sec — 750MB

Tradeoff between compression ratio & time to compress
1) depends on time and size availability
2) Command options parser availability for gzip.

⑧ create table in hive – load data gzip – Read the file.

PARTITIONER

values associated with same key will go to the same reducer.

Hashpartitioner

By Default

↓

partitioner → Hashpartitioner ⇒ modulus $\left| \frac{\text{hashcode(key)}}{\#\text{Reducers } (n)} \right| = 0$ to

Custom partion can be implemented through partitioner class

partitioner <Kintd, Vintd>

↳ getpartition (↓ , ↓ , no of ~~arguments~~ Reducers)

{

- - - - - - - - - - - -

- - Actual partioner Logic

- Return <int val> ⟶ from 0 to (n-1)

}

Example :- group by Jan year - Jan 19, Jan 18, Jan 17 - - - - -

- key → YYYY-MM-dd

- And modify your partitioner code to getthe data.

or

(Hashcode)

Code:- Mypartitioner.java

public class mypartitioner extends

-MyGroupDriver.java

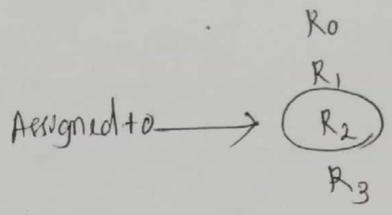
yarn jar

-D mapred.reduce.tasks=4

* set → myjob.setpartitionerclass (mypartitioner.class);

Return 2 partition will go to the R_3 → partition Return will be the partition number.

→ partitioner will tag to partion reducer



Case →

3. R=2, partition=2

R_0
R_1

* After 1 execution when trying to tag it will show error.

* Illegal partion for D.A. warner (2)

* No of mapper - 1 time
  maptask - 4 time
  ] it will attempt for 4 times

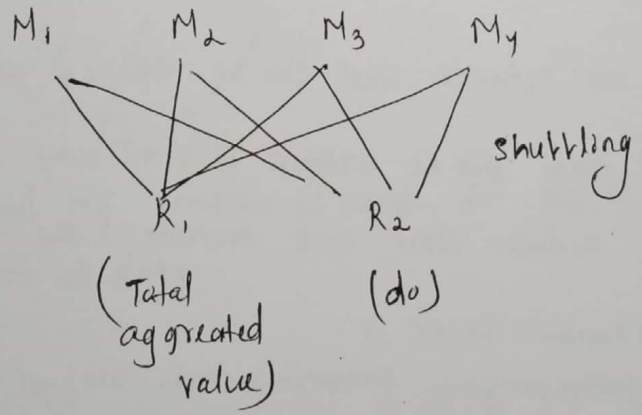
partitioner comes under mapper()

COMBINER LOGIC

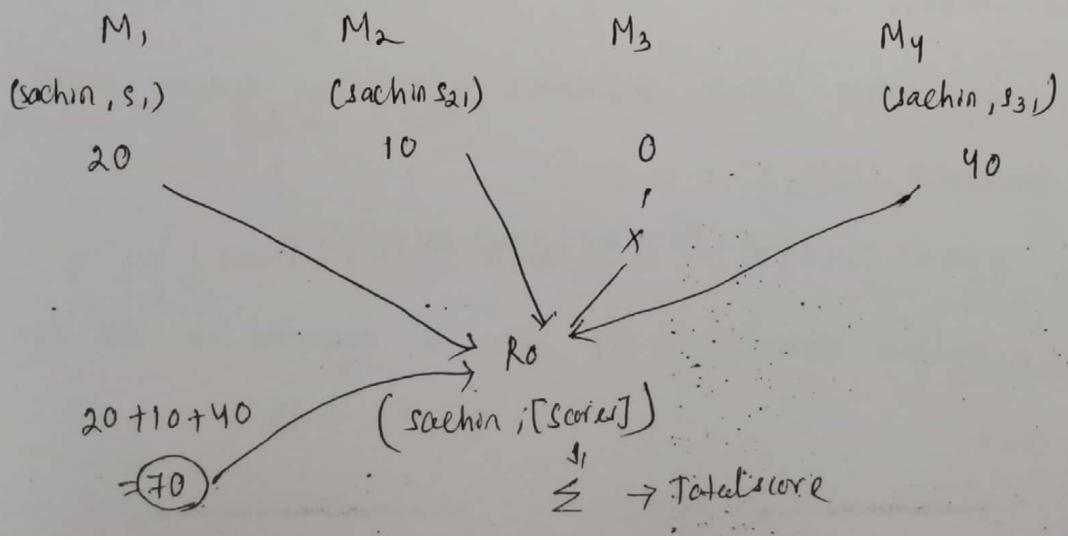
* 11.01.2018 *
* friday *

selection or transformation filter)

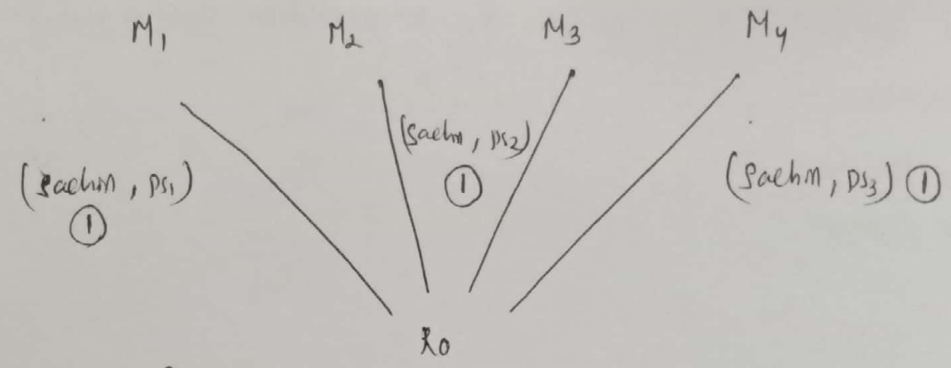
aggregation



Case-1



Case-2



1+1+1 = (3)

(sachin, [PS1, PS2, PS3])

↓

Σ

Total score

* while passing data through the network will be faster to send 3 (keys, va instead of 70 (keys, values).

* Combiner logic ≡ Reducer logic

* Map() ⟶ Combiner() ⟶ partition() ⟶ Reducer()

(FLOW OF DATA)

Keypoints

1) Combiner is a mini Reducer that can be invoked at Mapper class.

2) The combiner class will be written in same way we write Reducer class, by extending Reducer class and override() the Reduce() method.

3) Jobobject.setCombinerClass();

or

Mapred.combiner.name = " ";

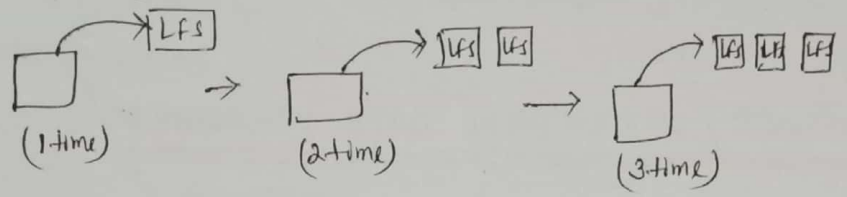
4) If any aggregation follows associative rule then Reducer itself can set as combiner class.

$$A+B+C \equiv (A+B)+C \equiv A+(B+C) \equiv (A+C)+B$$

5) Though a combiner class is a set it is not gurranted to run for all mapper class.

t any mapper's - mapper output buffer spills minimum 3 times then the any mbiner will be invoked for that mapper.

→ framework counter will be increased by one, when the buffer spills. while exceeding the buffer size, it will send to LFS.

→ 

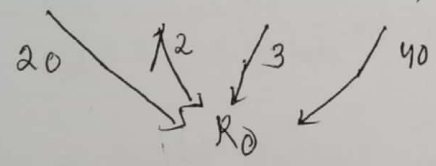
→ for less data no combiner needed.

→ creating the combiners & running for small amount of data itself it is the overhead of running the reducer class for all the mapper class.

→ one combiner will be invoked for only one mapper.

→ M + C ⇒ (Kintd, Vintd)
or
M ⇒ (Kintd, Vintd)

] for datatype value should be always same

→ 

→ for M_1 and M_y combiner is need, for M_2, M_3 combiner not needed for less volume of data.

How to use combiner in case non-associative aggregation.

e.g: avg = $\frac{\Sigma v_n}{n}$

Combiner → (Kintd, Vintd)

↓ ↓

((player) (Psum, Pcount)) ⇒ (ps, Eint) (s1, 1) (s2, 2) (psy, cnty)

↓

(player, (s, 1))

(Reducer logic)

mapred.io.sort.mb = 100MB → Map output buffer size.

Jobobject.set (P1, V1)

.set ("mapred.io.sort.mb", 5)

.set ("mapred.map.class", "org.dvs.bigdata.mr.MyMapper.

SEMISTRUCTURED / UNSTRUCTURED DATA PROCESSING

Semistructed data
Unstructed data ⇒ structured data

Unstructured → semistructured → structured data

(conversion)

Entity Extraction

(senderemail, date, amount) → (mobileno → xxx,
accountno → xxx,
emailid → xxx,
date → xxx)

(key) (value)

→ Data probiling
→ Data sufficiency (missed Entity to get the Business requirements)

EX:-(Task2)

phone[3][4]## I have ....

Battery[-1]##

(Longwritable) [.d]

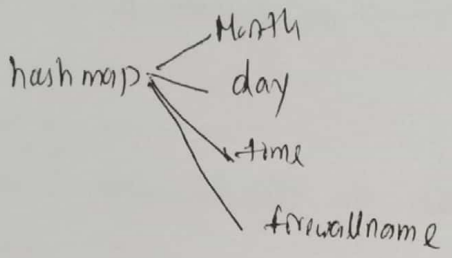
phone, [3][4]....
[0] [1]

* Unstructured data processing tools — NLP
Regular expression

Semistructured data

Trans-port
Trans-port

1) No Reducer
2) Split
3) split [5] → property



i>y → map — date
time
dvname

Header — source, destination, ...

map ← At each mapper()

- hive external table — load data → structured data

UDFs (User Defined Function)

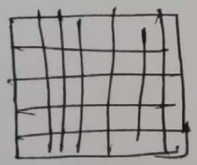
19.01.2018
Friday

HIVE — PIG — MR ⇒ preference
① ② ③

Hive → Java → Builtin functions → UDF extends evaluate()

pig → Java → Builtin functions → EvalFunc extend exec() → *.jar added to classpath

Ex:- (M, lg) ⇒ Google APIs



→ determine the nearby location for particular object

*.jar
↓
Classpath → hive → add jar <jar-file-path>; ①
→ create temporary function getLocation as .... org.dvs
→ pig → Register <jar-file-path>;

(2)

hive → create function getlocation as org.dvs.udfs.hive.GetLocation

→ extends UDFs
→ evaluate

select vehicle id, getlocation (lt-col, lg-col)
from gprs-data-tbl;

→ obj.evaluate()

PIG

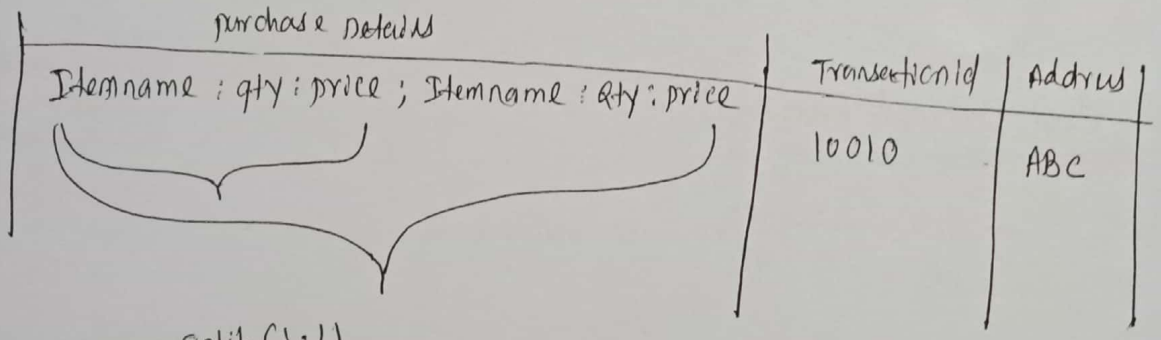
define get-location org.dvs.udfs.pig.GetLocation();

A = load ... .

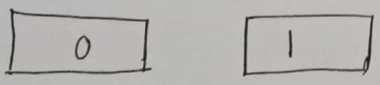
B = foreach A generate $0, GetLocation($1, $2);

* Check for
- keyword
- function

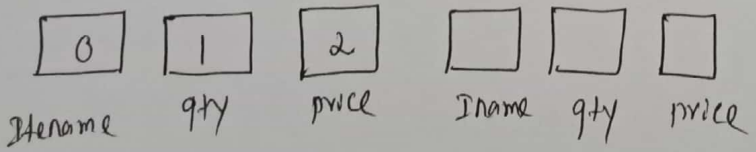
Old	New
- tempory functions	- functions are available in DB levels.
- session oriented	- reusable.
	- permanent functions.



Split (';')



Split (':') Split (':')



Error : —
Log : — Different actions accordingly
Infor : —

HIVE

ive-exec.jar ⇒ <hive-install-lib-path>

unlic class MyUDF extends UDF { ①

* evaluate (x1, x2, x3) ② ↓ hadoop/Java datatype

{

- - - - - - - - .

Actual hive UDF logic

- - - - - - - -

return <hadoop data type>

}

* evaluate (x1, x2, x3, x4, x5)

{

. - - - - - - .

. - - - - - - .

}

}

Add jar HIVE env

↳ local path

create the .hql for future reference

and also .pig

PIG

→ pig.jar ⇒ <pig-install-path>

→ public class MyUDF extends ①

EvalFunc < Return data type >

↓

Java data type

{

@override ② ↓ pig data type t

exec(tuple t)

{

- - - - - - - .

- - - - - - - .

Actual pig UDF logic

return < >

Java data type

}

}

Example

date string	date
22/10/2018	↓
	yyyy-MM-dd
	2018-12-10

datetype in hive.

date — yyyy-MM-dd

timestamp — yyyy-MM-dd HH:mm:ss....

→ month()
day()
year()
dt1, dt2

where dt between '2018-01-01' and '2018-01-31'.

* select store_id, sum(sale_amount) from pos_tbl group by month(sale_date)

group by day

group by year

SimpleDateFormat

Raw date string → parse() — input pattern()

→ format() — output pattern()

NOTE - showfunctions() in hive to see all the existing UDFs.

pig → Register → grunt shell.

Hive → add jar <path to jar file>;

↓

*.jar → classpath

↓

JVM — [-cp]

B = foreach A generate $0, $1;

org.dvs.pig.udfs.formatDate($3, "YYYY-MM-dd")

formatDate() → obj → exec() → tuple

> Define myDateFormat org.dvs.... formatDate();

- function name
- Alias name
- short name

> B = ...$0, $1, myDateFormat($3, "YYYY-MM-dd");

* string fun^, date fun^, Regular Expression

we should create a function, because Hive stores function details separately in Metastore DB.

> create [temporary] function <FUNC-NAME> AS org.dvs.---.formatDate class;

add jar path-to-jar;

session level only — at database level

> select x, y, z, <udf-fn-name>(parameters) from table ----;

↓ ex

MyformatDate(W, "yyyy-MM-dd")

or// hive-site.xml

<name> hive.aux.jar.paths </name>

<value> jar1-path, jar2-path, ... </value>

Simple Dateformat

↳ parse() ← to read the dt string

↳ format() ← to write to the dt string

→ Raw date string ← column

→ dd/MM/YYYY ← arg1

→ yyyy-MM-dd ← arg2

Unix Time | 00:00 00:00:00 Jan 1st 1970

System Time → 0000 0000

Epoch Time → 98 45 12 435

PIG

pig> pig -x local

grunt> register /home/training/hari/UDFs/Pig/pig-udf-format-date.jar;

grunt> Load A ..... sample.tsv

9> B = foreach A Generate $0, org.bigdata.pig.udfs.formatDateUDF1($0);

(25/12/2015, 2015-12-25)

(26/01/2015, 2015-01-26)

9> define fmtdate org.bigdata.pig.udfs.formatDateUDF();

Pig> which pig

cd contrib/piggybank/java/lib

HIVE

localhive> show databases;

> add JAR /home/training/hari/udfs/hive/hive-udf-format-date.jar

> list jars;

> Create temporary function fmt_date As 'org.bigdata.hive.udfs.

> select *, fmt_date(sale_date) as formatted date from .....;

> show function like 'fmt*';

> tab key ---> yes

> describe-function sum;

> describe-function max;

> describe function substring;

> describe function extended substring;

('facebook', -5) → ebook
5932 ?

('facebook', 5) → book

> describe function fmt_date;

> describe function extended fmt_date;

⇓

Explicitly code to maintain this

@ Description (

name = " ",

value = "-FUNC- takes string as parameter and return int";

) extended = " example- - - "

import org.apache.hadoop.hive.ql.exec.Description;

@Description (name = " ") → getDisplayName()

factory class

return " "

Adding the files Automatically

<home-dir> | <user> | .hivere

| home | hari | .profile / .bashrc

| home | Nagendra | .profile / .bashrc

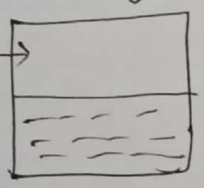
→ all hive related setting can be done for the user.

.hivere

```
- use dvs-db;
- set hive.exec.dynamic.partition=true
- add jar *.jar
- create temp table .... .
```

hive ↵

hive.hql



~]pwd

/home/training/

]ls -l

mc .hive

mc .hivercm1 .hivere

cat .hivere

UDFs - one row → 1 row output

UDAFs → n row → 1 row output

UDTFs → 1 row → n row output

Aggregate

Table Generations

e.g - Ucase, Lcase, substr.

e.g - sum, avg, min, max

e.g - explode <=> flatten (pig)
(hive)

Map — sachin [98, 22, 45, 36]

Array

Struct

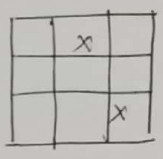
sachin 98
sachin 22
sachin 45
sachin 36

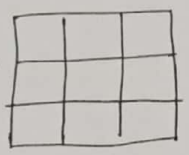
pivoting
transpose
→ 4 rows.

HBASE

OLTP → RDBMS → supports transactionality

OLAP → Data warehouse

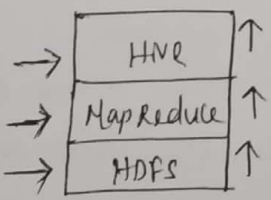


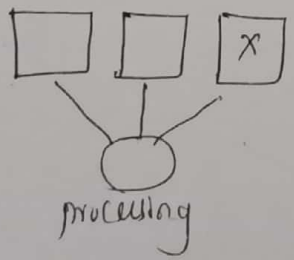


xxx not allowed updates

* Live data
* Transaction purpose

* past data
* Analysis purpose
* hive supports OLAP
* hive is the data warehousing of hadoop
* offline / batch processing can be only done
* No transactionality
* Not good for faster lookup of data.
* doesn't support random read / write operations, as HDFS doesn't support the same.
* All the blocks needs to be read parallelly then sequentially to get a particular data.





HBase

faster lookup data

CRUD operations supports – create, Read, Update & delete a Record.

Huge volume of data.

can store unstructred data / semistructured / structured data.

Web tables . Ex:- Table storing a Recommendations

Huge catalog

Googles Big Table.

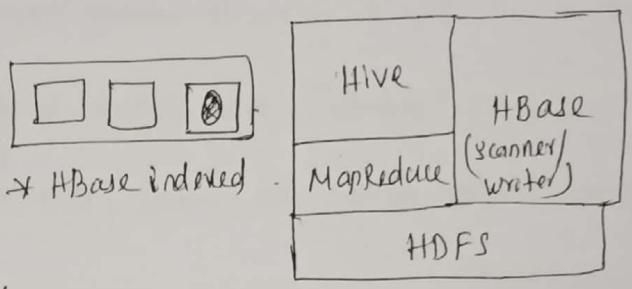
Database of hadoop

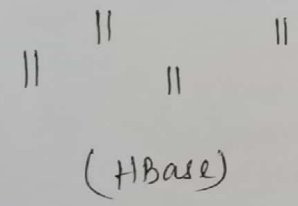
stores Read the data, Not

meant for Analysis

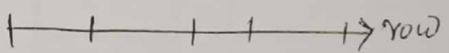
Doesnt supports joins and aggregations.

It stores data as columns / columnar database.





(HBase)



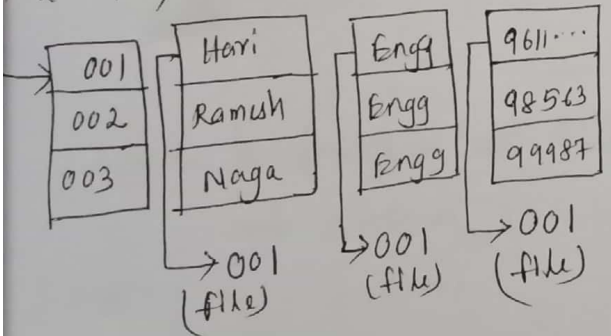
(HIVE)

→ primary key / unique key Retrieval

→ 001 | Hari | Engg | 20000

(id) (name) (prob) (mobile)

→ Row key



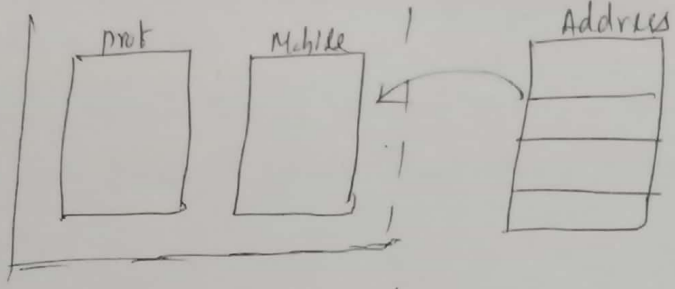
001 → Hari (Name)

Engg (prob)

99987 (mobile)

→ HBase stores data as sorted key, value pair.

→ group of columns is called the Column family



→ data insertion will be column by column.

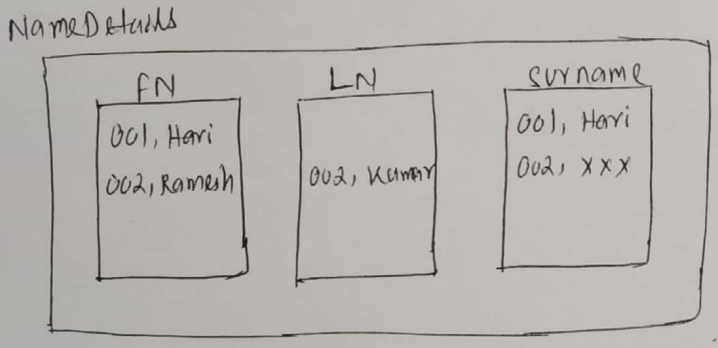
→ Dynamic schema / schema less

→ At the time of insertion we decides the column family, Row key, column name and value.

→ schema of HBase table is column family names.

EX:- Create 'Std-tbl', NameDetails, Batchdetails, Contact Details, Identity Det

→ column family names are virtual / Directory in HDFs only.



(NameDetails column family) (HBase Serde)

→ Key based access, same as JSON format.

→ Sparsed Table = No Reservation of columns for a Record in advance. So we can store unstructured / semi structured / structured data.

JSON — { ["name" : "Hari",
"id" : "001",
"key" : value],
["name" : "Ramesh",
"id" : "002",
"prof" : "Engg",
"no" : "9861"] }

There is no ordering or limitation of colums so it is dynamic schema, but in hive ordering and no of columns is important.

001	Hari	"Engg"	96115
002	Ramesh	prof	9999

(Hive table Representation)

(CSV serde)

Sorted key value pairs => Metadata + Actual data

key value

{ Table Name, [Regions] }

{ Region, [Row keys] }

{ RowKey, [column families] }

{ columnfamily, [columns] }

{ column, [versions] }

{ version, Actual data value }

Region 1 { 001 : 1000 } DN10

Region 2 { 1001 : 2000 } DN12

scanner

get std-table, 105, contactDetails : homeNumber

Table name | Row key | column family | column

105 belongs to Region 1

scanner

put std-tbl, 105, contact-detail : home number, 1234

* If not present create.
* If present overwrite with new version.

.sql ⟶ Phoenix — wrapper Tool
.hql

↓

HBase

24.01.2018
Friday

HMaster

HRegion server

zookeeper

Memstore (write cache)

Hfiles

WAL (write ahead log)

Block cache (Read cache)

HBase scanner

HBase storage handler – Read / write operations

Hregions

.META.

→ Read / write operation

HBase clients

– CLI → HBase shell

– REST API

– JAVA APIs

→ connecting clients

COMPACTION

– Minor compaction

– Major compaction

TTL

→ Data storage

HBase and Hive Integration. → IMPORTANT

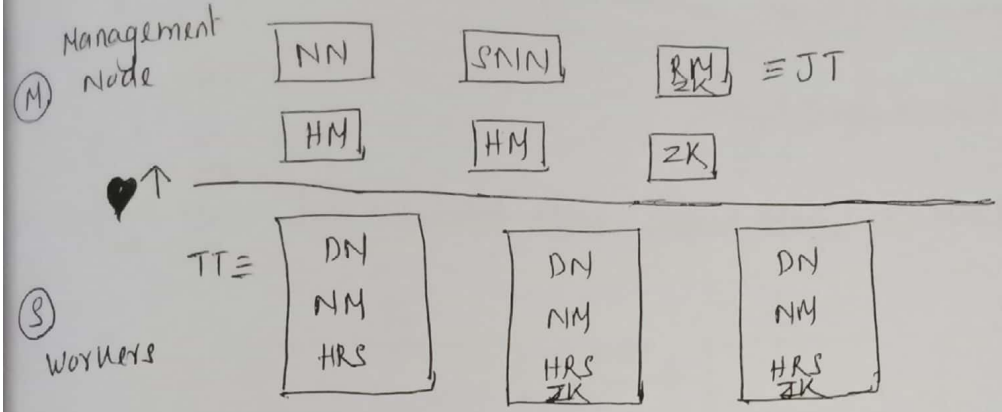
* hadoop cluster should up and running. → Name node
* don't need Mapreduce secondary name node
* Hbase engine to Read/write
* start-hbase.sh & stop-hbase.sh needs to run.

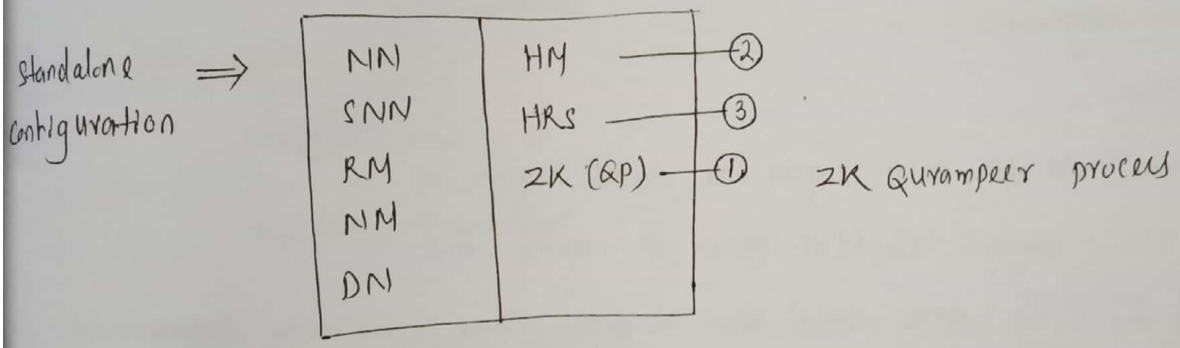
Master slave Architecture

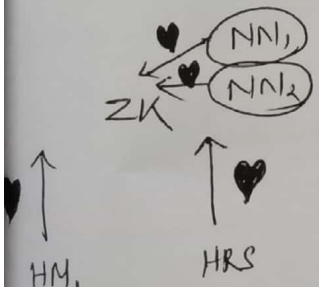
↓ ↓

→ HMaster → HRegion servers

→ Zookeeper



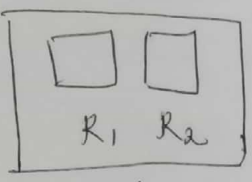




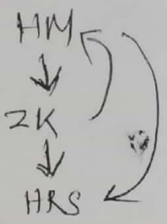
* Minimum 3 HM have to be started.
* they will go and register to ZK that they are alive with corresponding ipaddress.
* one will be the lead HM.
* signals will be send to all the HM.
* polling will be happend between HMs and one will be selected.
* In config file hadoop master will have setups.
* ZK will do the Notification services.
* If HM1 down, it will inform other 2 and one of them will behave as HM.
* No need to do anything manually.

* HRS if down ZK will inform HM that HRS is down.
* Hbase table hbase regions allocated to HRM.
* Regions is set of buffers.

→ as HM assign HRegion to HRS
- activities known as **Region assignment.**



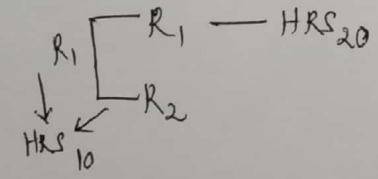
- ZK (Distributed ~~coordinated~~ Resource co-ordinator)



* If NN_1 is down NN_2 will become active by the ZK.

HM

→ Responsible for DDL operation
→ Region assignment.
→ .META.
→ Stored in the file, after region and column region is assigned.
→ All DLL changes should be stored in .META. file.
→ only one region will be created when 1st time HBase Runs, with Region Name.
→ 40MB Region size
→ in the middle of region it will be splited across the key values.



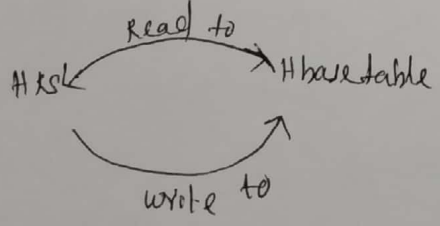


Table → CF_1 CF_2 CF_3 CF_4

0 ⋮ 1000 R_1 → 4 buffer ≡ CF_1 ⋮ CF_4 → HFile

1001 ⋮ 2000 R_2 → 4 buffer ≡ CF_1 CF_2 ⋮ CF_4

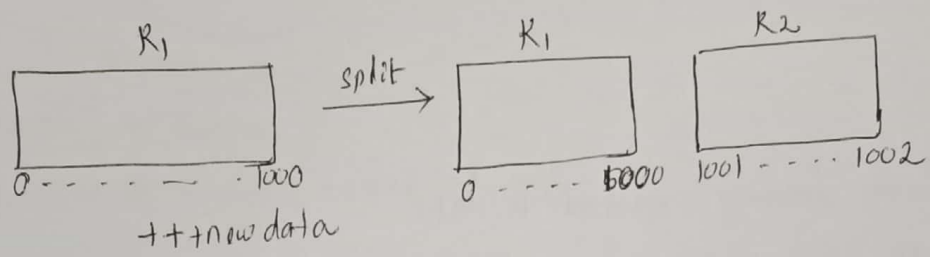
* when the threshold will happens the data will be saved to HFile.

* when data will be more, region split will be happens, with required buffer.

2000 : R_3 → 4 buffers ≡ Cf1, Cf2, : , Cf4 : 3000

(organisation of data)

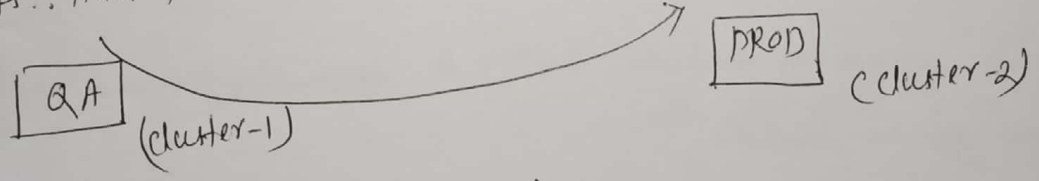
* Regions make data Readability easier as reading 1 region is easy than 4 regions.

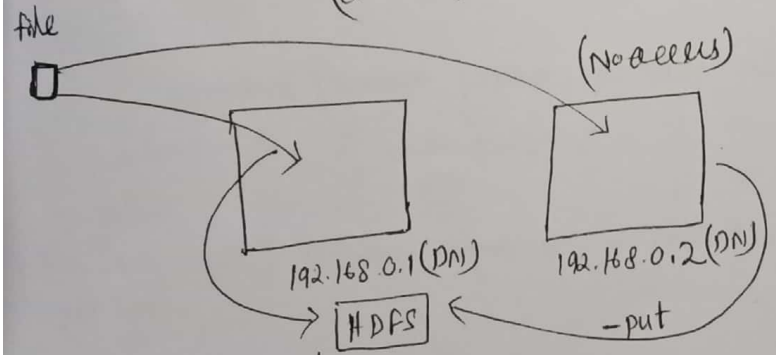


put & copy from local

- put - Explicitly Remote IP
- put - Implicitly → Local host

distcp hdfs://nn1:port1/dvs/data/* hdfs://nn2:port2/production/data





- copy from Local
- put 192.168.0.2/Abc.csv → HDFS.

NOTE:→ - copy from Local → Local
- put → Different systems / datanode
- distcp → cluster to cluster.

25.01.20
→ friday

100 | 100
200 | 175
175 ↓ 200

→ 150 ↓ 100
150
175
200

→ Threshold
→ Idle write timeout

write → [Buffer] → [Memstore] → [Hfiles]

4 column family ≡ [4 buffers] ← [Region] ← [HRegion server]

assigned to
assigned by [HMaster]

* An empty directory created in HDFS
* .META directory created in HDFS → HBase root directory → zookeeper
* 1 Region created 1st time
* RAM of datanode will be dedicated for buffers.
* zookeeper will download the .META to the datanode and read the RAM and do the operations.
* When Request for particular RowKey, it merge Memstore and Hfile then sort and get the details.
* Range of RowKey considered for point of making regions / partitions.
* Default Region 4GB.
* [Region-1] [Region-2] → zookeeper → ~~Region~~ HMaster (check overloading over Regionserv

(Autosharding process)

↓

(splitting of regions)

Region server 2

* [Region ≡ Memstore + Hfile size]

{ RegionID : (start key, HRS Name) }

key — value

$R_1 \rightarrow 0$
HSR10

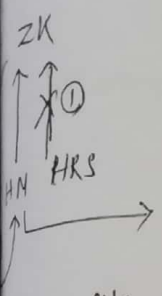
$R_2 \rightarrow 1001$
HSR15

[br1 br2 br3 bry] – Buffers

[cf1 cf2 cf3 cfy] – columnfamilies

case1 → 500 → key → > R_2 key
compare and assign to the region.

put std-tbl, 100, CF2 : CN1, XXXX



→ will check which regions assigned to HRS.

WAL file will have the delta values for the regions.

EN → NN
Bob → JT
Client → ZK

* Meta cache will be changed if META is changed with new timestamp.

Block cache
Memstore
HFile

① ② || Read operation

MemStore
HFile

|| write operation

WAL ≡ HFile
↓ ↓
Not sorted file — sorted file
↓
Write ahead log on disk
↓
Used for Recovery

* Deleted cells marked will be removed from the buffer and put in a large file – hard delete.

* Collect files → merge → sort → remove delisted file → one large file.

Read Amplification

Write Amplification → maintainance time frame → done by the hadoop. Admin

26.01.2019 Saturday

Apache hadoop

- Cloudera, Hortonworks, MapR] commercial distribution of hadoop
- → All ecosystem tools like hive, pig have to download and install manually.

→ ls -p

- hbase-0.94.10.tar.gz
- cd hbase
- cd lib
- ls
- Jars for hbase
- cd docs
- ls
- hbase conf
- vi hbase-site.xml

- - - - - - -

- cat hbase-policy.xml

hbase-manages-zk=true - Internal

Conf > ls

conf > cat regionservers

localhost

hbase > which hadoop

hadoop > ls -p

hadoop > pwd

hadoop > cd conf

conf > ls

core-site.xml

conf > cat masters

conf > cat slaves.

pseudo distributed mode - one system

= fully distributed mode

- jps - 3280 JPS

Conf > - hadoop dfsadmin -safemode get

safe mode is on.

Conf > hadoop dfsadmin - safemode get

safe mode is off.

> start-hbase.sh

- sleep 5

bin > jps

4335 HQuorumPeer

bin > hbase shell

hbase(main):001:0 > help

hbase(main):002:0 >

003:0 > status

- 1 servers

:004:0 > version

- 0.94.10

> whoami

- trial training

HBase → Log structured file system

HDFS → Block structured file system

bin > hadoop fs -ls /hbase/

bin> hadoop fs -ls /hbase/student/

hbase > describe 'std1o'

TTL - Time to leave - the record will mark as deleted when time elapsed.

hbase > count 'aadhar-tbl'

hbase> scan 'aadhar-tbl'

> scan 'std-tbl'

Row	column+cell
1	Colum=cf1:loc ...
1	column=cf1:name...
1	column=cf1:no...
2	- - - -
2	- - - -
2	- - - - -

> put 'std-tbl', 9 'cf1:nam', "Nagendra"

> get 'std-tbl', 9

Column	cell
cf1:name	timestamp = 15...., value = Nag

> get 'std-tbl', 9, 'cf1:loc'

Column	cell
BLANK	

> get 'std-tbl', 9, {column => 'cf1:name', versions => 3}

Column	cell
cf1:name	timstamp = ... , value = Nag
cf1:name	timstamp = ... , value = Nagendra

> put 'std-tbl', 9, 'cf1:name', "Nage"

> put 'std-tbl', 9, 'cf1:name', 'Nag'

cf1 : name timestamp=..... value = 'Nage1'
cf1 : name timestamp=..... value = 'Nage'
cf1 : name timestamp=..... value = 'Nag'

* 3 versions will be kept and old 1st version will be set-deleted

hbase > delete 'std-tbl', 9, 'cf1: name'

hbase> deleted 'std-tbl', 9

hbase > create 'Nov-2018-emp-tbl', 'nameDetails', 'parentDetails', 'IdDetails', dup

hbase > exists 'Nov-2018-emp-tbl'

→ table 'nov-2018-emp-tbl' exist

hbase > scan 'nov-2018-emp-tbl'

hbase > count 'nov-2018-emp-tbl'

hbase> describe 'nov-2018-emp-tbl'

'nov-2018-emp-tbl', {Name => "IdDetails', Data block.....

.....},

{Name => "parent Details', ...}

{Name => "Name Details", ...}

{Name => "Dept Details", ...}

↳ column family

hbase > create 'Nov-2018-emp2-tbl', {olame => 'nameDetails', versions=>5}, 'parentDetails'

hbase > describe 'nov-2018-emp2tbl'

'nov-2018-emp2-tbl', {Name => 'nameDetails', VERSIONS => '5', ...}

properties can be changed while creating tables/attributes

hbase > drop 'nov-2018-emp-tbl'

Table must be disabled, to delete the table — error.

hbase > truncate 'nov-2018-emp-tbl'

→ Disabling table
→ droping table
→ creating table

hbase > describe 'nov-2018-emp-tbl'

HBase & Hive Integration

29.01.2018 Thesday

HBASE	HIVE
> Joins and aggregation (for unstructured data, may not be common columns)	X → CRUD → flexible - can use any serialization
Key based Access - No many where conditions and all.	
Rigid - it uses it's own serialization.	

HBase storage handler

project to ↑

project to ↑

↕

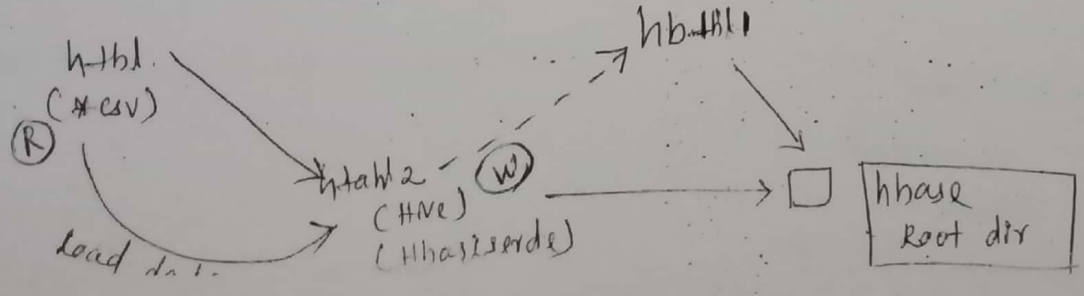
HBase SerDe

(serialization / Deserialization)

Scenario

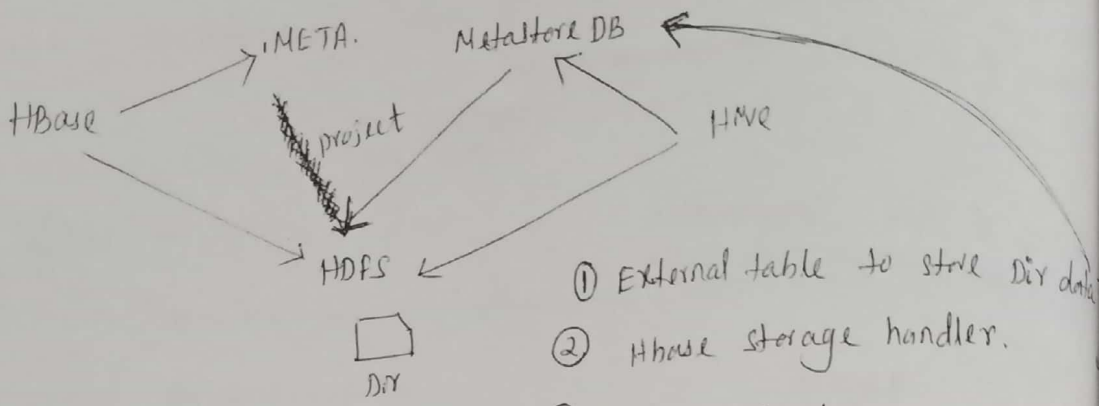
1) HBase table —(view as / read as)→ Hive table. → Deserialization

2) Hive table —(Load into / Bulk load)→ HBase table → serialization



1) HBase → Hive

(CRUD opn) (Aggregation / joins)



① External table to store Dir data

② Hbase storage handler.

③ column Mapping

④ hbase table name should match

Schema on Read → Hive uses

Schema on write

* Blank directory will be created in Hive data warehouse, but read/write is mapped to the hbase HDFS directory. Later you can drop that directory.

zookeeper-*.jar	hive/lib
hive-hbase-handler-*.jar	hbase/lib
guava-*.jar	
hbase-*.jar	

hive > add jar /home/..../zookeeper-3.4.3.jar

⋮

or// hive-site.xml

<name> hive.aux.jars.path </name>

<value> file:///home...jar, ... jar, ... jar </value>

---

hbase > scan 'std10'

Row column+cell

column = cf1: loc

cf2: name

Hive > add jar .....

hive > lst jars

hive > create external table

hive > show tables

hive > Select * from hbl_stdio_in_Hive;

stdid	name	number	loc
1	Ram	123	Hyd
2	Anu	234	null
3	kum	456	MYS
4	Ashok	555	Bang

hbase > put, 'stdio', 9; 'CF:name', "Naga"

hive > select * from hbl_stdio_in_hive;

stdid	name	number	loc
1	Ram	123	Hyd
2	Anu	234	NULL
:			
9	Naga	NULL	NULL

hive > describe hbl_stdio_in_hive;

Table parameters:

External TRUE

hbase.table.name stdio

storagehandler org.apache.....

storage description params:

hadoop >

/ hbl_stdio_in_hive/

hadoop > hadoop fs -ls /hbase/stdio/

② Hive to hbase

```
hive > select * from ds-db.emp-test-tbl;
     > create table tmp-hive-tbl-emp-to-hbase (.....)
       stored by 'org.apache.....'
       with serdeproperties ("hbase.column mapping"=...)
       TBLPROPERTIES ("Jan-2018-emp-tbl");

     > show tables

hbase > exists 'jan-2018-emp-tbl'
hbase > scan 'jan-2018-emp-tbl'
hive > insert overwrite table tmp-hive-tbl-emp-to-hbase select *
       from ds-db.emp-test-tbl;
     --- maponly job
hive > select * tmp-hive-tbl-emp-to-hbase
     ---- Data -----
     ---- Data -----

hbase> scan 'Jan-2018-emp-tbl'
```

⇒ Webtable
- Recommendation Results

Huge Dimensional table
- customer information.

Streaming data Injection
- hbase as sink
- Any Networking data
- Any web apps data

⇒ Mode of users

1) CLI → hbase shell → cluster database
2) REST API → Any application / framework
   Any lang supports REST APIs
3) JAVA API → JDBC like

OOZIE

Automation for Hadoop Ecosystem.

Workflow / Job flow

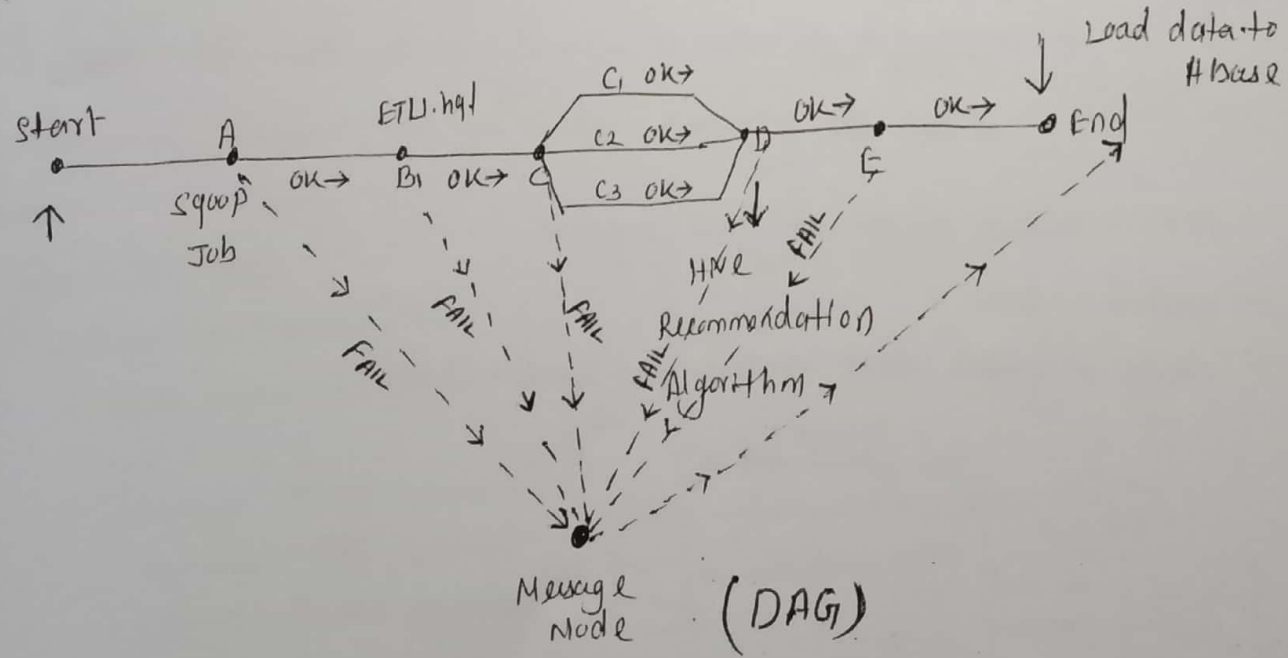
DAG – Directed Acyclic Graph

Scheduling → creating oozie co-ordinators

- automation → creating oozie workflows
- scheduling → creating oozie co-ordinators

> oozie handles
→ combination of automation and scheduling

FS	MR
sqoop	HIVE
scripts	Pig
Java code	Java code
	script
(Data movements)	(Data manipulation & processing)



Workflow.xml → Expression language

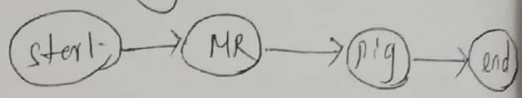
HPDL → Hadoop process Definition Language

communication between the user and Oozie job definition to understand the workflow with XML Language.

```
<workflow-app>
    <start to="Action-1">
    <action name="Action-1">
      <hive>
        <script name=...>
          <args...>
          <---->
        </script>
      </hive>
      <ok to="Action-m">
      <fail to="Action-n">
    </Action>
    </end>
</workflow-app>
```

* At each step the action is defined for ok and fail scenario.

* We can see the diagram while running the job itself



(DAG)

Oozie Architecture

30.01.2018 Wednesday

→ client-server Architecture

→ Oozie server

→ Oozie client → CLI *

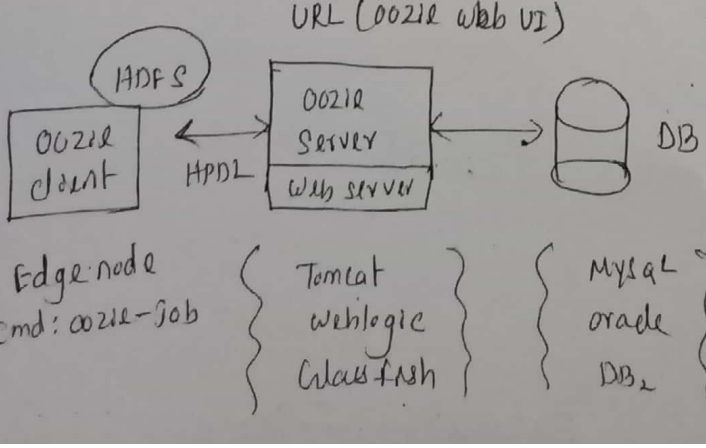
→ Java API

→ Rest API

→ HPDL - Hadoop process Definition Language

→ written in Java

→ closely integrates with Hadoop and ecosystem tools.



Ex:- Set OOZIE_URL = http://<hn>:11000/oozie

> oozie -job -config <job.properties> run

<hn> → <host name>

11000 → port number server

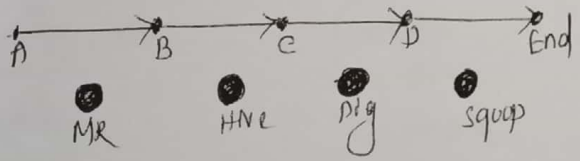
LFS

HDFS
- .../workflow.xml
- .../scripts
- .../lib/*.jar

job.properties
- oozie.wf.application.path = HDFS DIR PATH
- namenode = ...
- jobTracker = ...

HDFS DIR path

1) Connected to server with 11000 port.
2) Job properties file read.
3) go to mentioned directory, submitted to DB
4) jobid created - oozie.job.id : wf-01234
5) process the job.
6) Transition information (from one node to another) stored in Database.
7) JDBC to DB is stored in oozie xml file.



> oozie job -info wf-id → can be access the details for the job id.

workflow.xml - contains the xml tags containing the HPDL to process the data.

```
<workflow-app>
  <start to="Action-1"/>
  <Action name="ACTION-1">
    <pig>
      <script> eli.pig </script>
      <param> 1 date </param>
      <param> $outdir </param>
    </pig>
    <ok to="Action-2"/>
    <fail to="Error-Node"/>
  </Action>
  <Action name=...>
  </Action>
  <Action name=....>
  ....
  </Action>
  <kill name="Error-node">
    <message>
    ....
    ....
    </message>
  </kill>
  <email> ... </email>
  <end name="end"/>
</workflow-app>
```

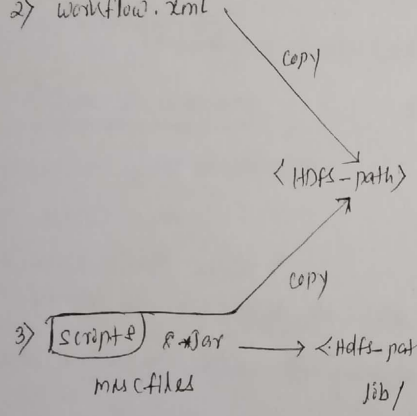
* it will start from the last returned / failure / suc
* we can define the skip failed node.
* current running status for the node is stored in DB, so at any time you can resume the job.

31.01.20
Thursday

1) job properties → LFS

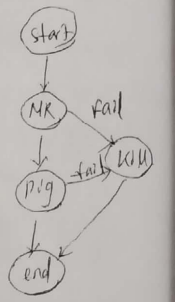
1) Namenode = hdfs://nn:port/
Job tracker = host-name:port
oozie.wf.application.path = <hdfs-path>
input dir =
output dir =

2) workflow.xml



2) <workflow-app--->
<start-to=...>
<action name=-->
<hive>--

</action>
<kill.--->
<end/>
</workflow-app>



* Queue name - Dev, Testing, prod
Choose one queue under which you want to run the job, within the cluster.
* prepare and configuration tags are optional.

#set or export
OOZIE_URL = ----
oozie job -config job.properties -run

oozieexample> oozie job ---job.properties -run
oozieexample> oozie job -info 0000000 ----

01.02.2019
friday

Oozie Co-ordinator

Automation → workflow.xml → oozie-job -conf job.properties -run

Scheduling → Running in a future time

- cron
- control M
- autosys
- Tivoli WF manager

+ Oozie Co-ordinator tool
  - → frequency
  - → window
  - 01-02-2019 → Data dependency

  - Co-ordinator.xml
  - Co-ordinator.properties / job.properties

...../year/month/day/hr/

2019|01|31|12|
2019|02|01|00|
|01|
|02|
|03|

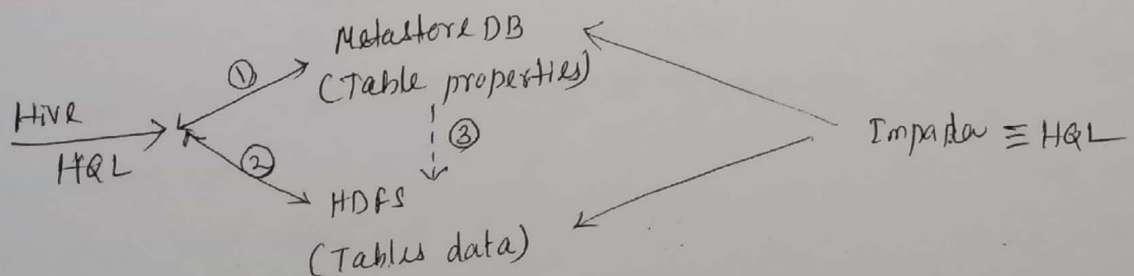
-date
time
date -u

<input-events>
<work-flow>

oozie -job -config co-ordinator.properties ~~-start~~ -run

02.02.2019
Saturday

IMPALA



1) Syntax check
2) Query Resolution
3) Query planning
4) MR code → MR Architecture
5) Physical execution
5) Result — console
written HDFS

Metastore DB project details to HDFS to fetch the data.

Difference – ① MPP Architecture

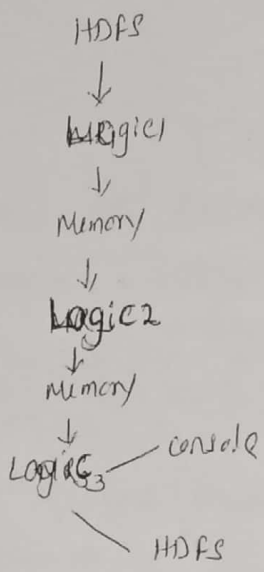
Massively parallel processing

HIVE

HDFS
↓
MR_1
↓
HDFS
↓
MR_2
↓
HDFS
↓
MR_3
↓
HDFS

hive ↵
hive >

Impala



impala-shell ↵
impala >

* Table created in hive can be seen in impala.
* developed by Facebook
* More HDFS process, I/O cycle, serialization needed, so it is slower than Impala.
* Good for large dataset
* Long running batch processes.
* Slow compared to impala for aggrigated data.

* Created in Impala can be seen in hive, as they are sharing the common Metastore DB
* developed by cloudera
* intermidiate memory are there, so impala is faster than hive.
* Good for small dataset, as memory limi
* good for quick & short queries.
* Ideal for aggreated data, like reporting &

HCATALOG

Hcatalog ⟶ Metastore DB
interface (HDFS)

→ DDL operations it supports not DML operations.

→ HcatLoader

→ pig

A = Load Table <tbl-name> using HcatLoader();

→ HcatLoader will get the tbl-name → goto metastore DB → Load data to Reddition A. → convert hive datatype to pig datatype.

B = for each A Generate Trans-id, Trans-data.

result = ...

store result into table <db.table> using HcatStorer();

- Hcatlog input format
- Hcatalog output Format
- HCat storer will convert pig datatypes to hive datatypes.

→ 1) writing to HDFS
2) converting datatype
3) metastore DB updated with corresponding tables.

Cloudera@quickstart ~] hcat

] hive -e "show databases;"

] hcat -e "show tables;"

] hcat -e "create table ....;"

Show create table - used for transformation/automation. ⟶ HQL for running

Create table like

hive > show create table emp_tbl;

hive > create table emp_dup_tbl like emp_tbl;

hive > desc emp_dup_tbl;

BEELINE

hive ↵

hive >

hive based on [Thrift server protocol] not providing any security. anyone can connect to the edgenode. it will not ask for authentication.

[Thrift server2 protocol]

→ hive JDBC url

→ userid
→ password] LDAP authentication

① → beeline -connect "jdbc:hive//<hostname>:<port-1 ...."
-n <userid>
-p <password>

→ beeline > --- hql
> -- all hive query u can run

③ > beeline -file ABC.hql -hivevar x=x1
-hivevar y=y1

Thrift server2 protocol developed in [sqllite].

beeline ↵

② beeline> ! connect <hive-jdbc-url>
> enter userid
> enter password

* Either 1 or 2 or 3 you can choose to work with beeline

* -i <filename> - initialization session for hive.

cloudera] which beeline.

/usr/bin/beeline

] beeline --help

* beeline provides the security feature to hive.

NameNode High Availability

1.x

2.x → hadoop MR written again

→ under utilization of resources mapper/reducers

→ container can do map/reduce both.

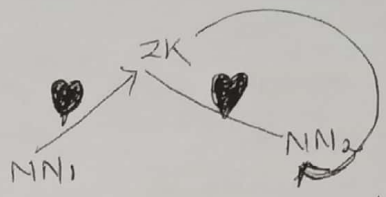
→ TT supposed to do all the processes, if any process down all the process go down.

→ SNN - replica of fsImage, metastore.

1.x

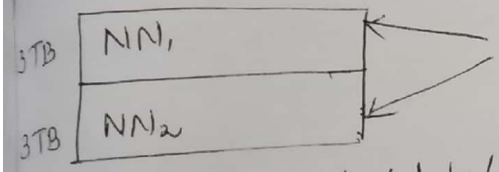
NN SNN

2.x



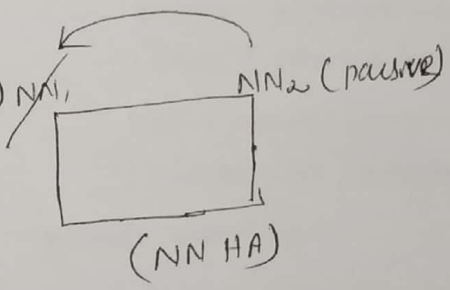
* when NN1 not sending heart beat it will inform NN2 to become active.

NameNode Federation



ABC.txt (1TB)

hdfs://nn:port/dvs/data/



* headache over hadoop admin.
* More space will be consumed as 6 replica will be done with single file.
* So NN HA is preferrable in real time.

hadoop fs -put <LFS> hdfs://nn2:port/dvs/data/

* you have to write in NN2 to read the data.
* NN Fed → Having more than one Name Node in the same cluster.

Scheduling

Job scheduling

FIFO : Default

dev - Job1
QA - Job2
prod - Job3

* one after one
* prod have to wait for dev and QA to complete the job
* so accepted in real time.

1) fair scheduling → facebook
2) capacity scheduling → yahoo

Queues → To assign the resources the groups as per the task. Admin suggests qu

default - 5%.
batch - 35%.
task - 25%.
I/O - 35%.

1) cluster
- About
- nodes
- nod label
- Application
  - ... New
  - ... Submitted, accepted, finished, killed
- scheduler

2) Application queues ←

→ queue root
- queue: default - green - means job is running
- queue: batch
- queue: IO

3) Queue state: running
used capacity: 0.0%
Configured capacity: 35%.
Configured max capacity:
↳ can borrow capacity
Absolute used capacity:
Absolute configured capacity:
Absolute configured max capacity:
used resources: (memory: 35532, vCores: 13)
Configured max application master limit: 20.0
Max application master resources:
used application master resources:

- Submit the job
- Accepted the job
- Application master allocation
- running state
- finished
- success / killed

Num container : 13
Max Applications : 3000
Max application per user : 3000
Configured minimum user limit percent : 100%.
Configured user limit factor : 1.

Hadoop Distributions

NASSCOM – Journals

Choosing a hadoop distribution

- company philosophy
- current Relationships
- Acceptable Risk
- specialized functionality

Hadoop
1) Hortonworks
2) Cloudera
3) Map R
4) packages and online solutions
   - IBM Big insights
   - Oracle big data Appliance
   - HDInsight
   - Many others!

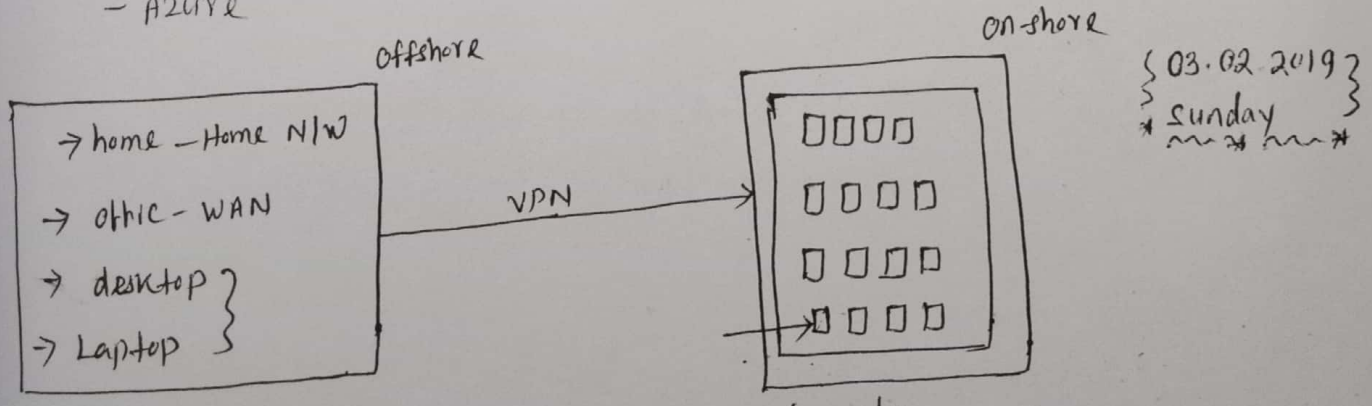
1) Apache projects only – Hortonwork
2) Apache project + proprietary Add-ons – Cloudera
3) Map R – proprietary value adds & Redivelopment new filesystem.
   ↳ provides high performance.

— Ambari same as Cloudera manager to provide Admin job easier.
↓
+ HUE, IMPALA

* Hortonwork started YARN : Data operating system.

Cluster Types

→ Own cluster
→ Cloud cluster → difficulty while re-migration from cloud
- AWS
- GCP
- Azure



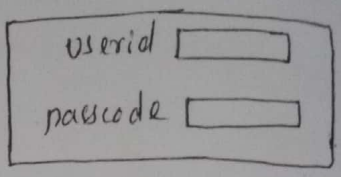
03.02.2019
Sunday

1) VPN - Virtual private Network

2) cosco Anyconnect - client

3) RSA App ⇒ pin number ⇒ passcode

http://tcs.in/vpn.conn



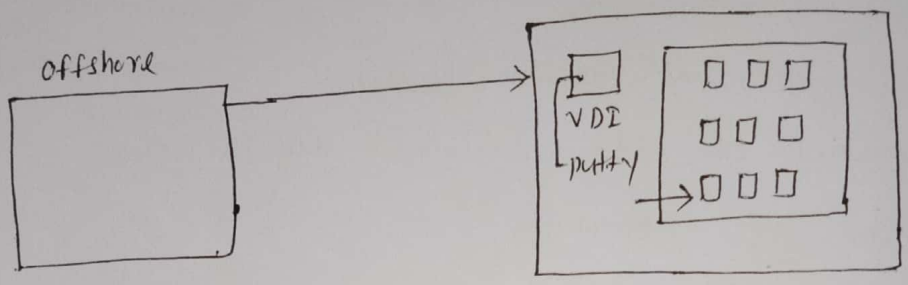
option-1) Edge Node : IP#
host name

Unix userid : Nagendra
sg80012AB

password : * * * * *

SSH to connect the edgenode through putty terminal.

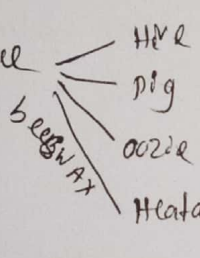
option-2) VDI

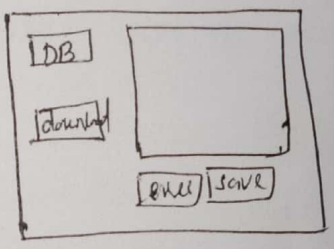


VPN → VDI → putty → EdgeNode

* same sequence for onprime cluster or the cloud cluster.

option-3) VDI → HUE Interface





HUE → Query Editors
- Hive
- pig
- Impala
- DB Query - backend for MySQL
  - Manually added through JDBC connection.

Data browsers
- Metastore manager
- HBase

Workflows
- oozie Dashboards — Resume, suspend, kill the jobs.

Search
- solar search