Introduction to NoSQL :

NoSQL :-

↳ open source, non-relational, distributed data base.
↳ It can handle all the 3 forms of data
↳ It does not support ACID properties
↳ There is no proper fixed schema for the table.

Types of NoSQL database :-

↳ Key value (key value pair) Eg :- Fname : Anagha
Lname : D
↳ Schemaless
↳ Document form of data.

```
{
    book title : ———
    book author : ———
    publisher : ———
}
```

↳ column (data is stored in a column)
↳ graph (it consists of nodes & links).

Need of NoSQL :-

↳ Relational db have monolithic but the NoSQL architecture have scale out architecture.
↳ It can store large amount of structured, unstructured & semi-structured data.
↳ Dynamic schema (insert / update / delete are easy)
(changes to appl in real-time is easy)
↳ Auto sharding
(The available data is distributed into different arbitary servers so that there is no downtime if one server crashes)
↳ Replication
(because of replication of data into all the arbitary servers that are present downtime will not be present)

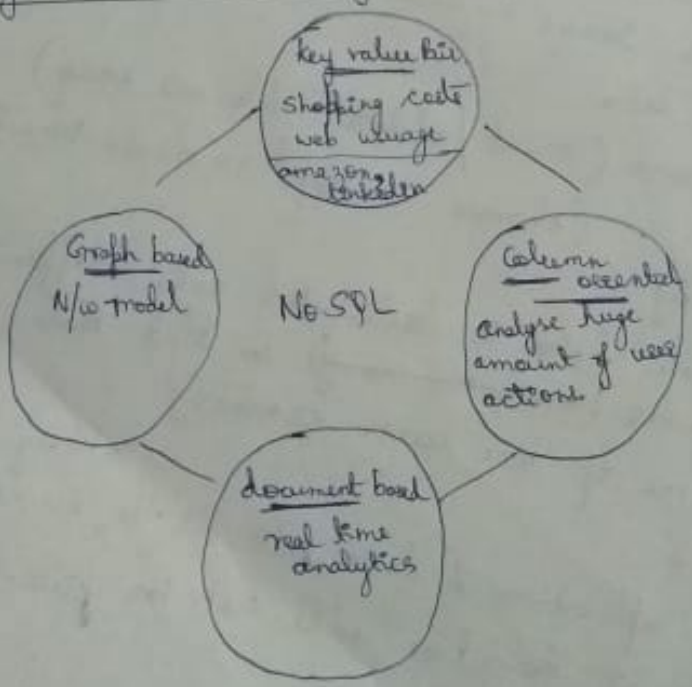
Advantages :-

- Can easily scale up & down.
  - cluster scale
  - performance scale
  - data scale.
- do not require fixed schema
- cheap & easy to implement
- Relaxes the consistency of data
- Data can be replicated & partitioned.
  - sharding
  - replication.

Disadvantages :-

- There are no joins (inner join, outer join, etc) & group by clause.
- No ACID properties
- integration of the appl with other appl which support SQL is not possible.

Uses of NoSQL in Industry :- (Applications)



NewSQL

↳ supports ACID properties

↳ Online Analytical processing is also supported (OLAP)

~~SQL~~	SQL	NoSQL	NewSQL
1) ACID properties	Yes	No	Yes
2) OLAP/OLTP	Yes	No	Yes
3) Schema rigidity	Yes	No	Maybe
4) Data format flexibility	No	Yes	Maybe
5) Scalability	Vertical	Horizontal	Scalable
6) Distributed Computing	Yes	Yes	Yes
7) Community support	Huge	Growing	Slowly growing

Mongodb

↳ cross platform, open source, non-relational, Distributed, document oriented data store, NoSQL

Characteristics :-

1. ↳ using JSON format (Java script object Notation)
   ↳ complex structure & open standard

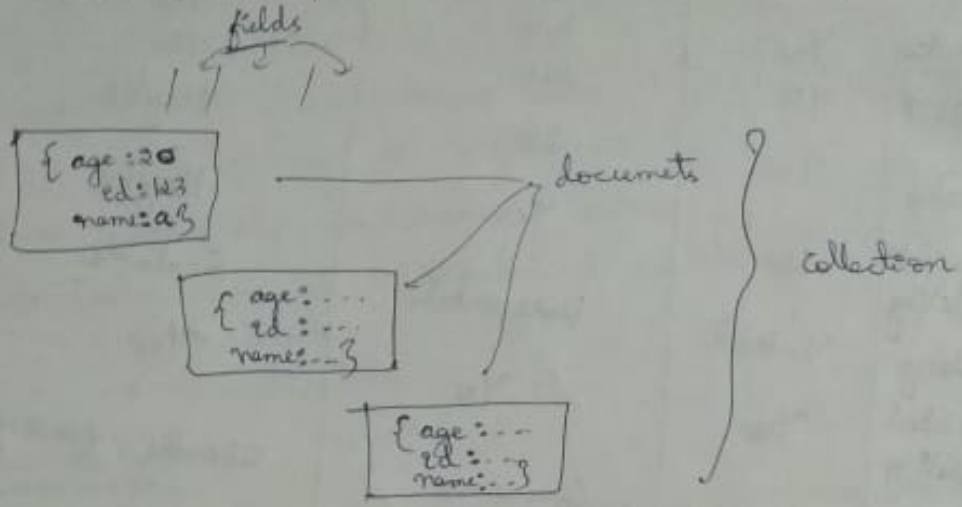
```
.CSV
Fname, lname, contact
abc , k , 1234567---
xyz , l , ----
```

```
{
  fname : abc
  lname : ...
  contact : ...
}
{ ...
```

XML

(2) Creating & Generating unique key.

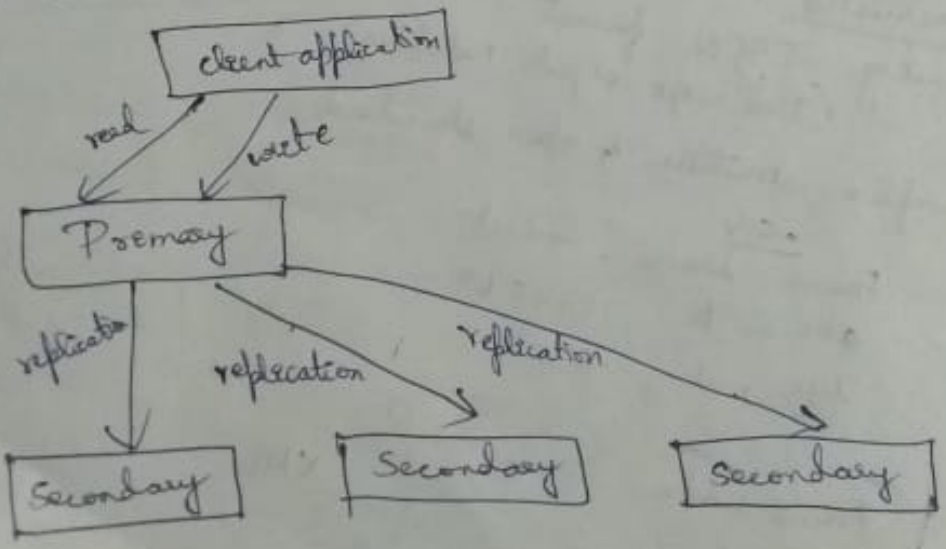
→ database → same as table
* collections (where the data is being stored)
* One MongoDB server can have multiple databases.
→ can have multiple documents



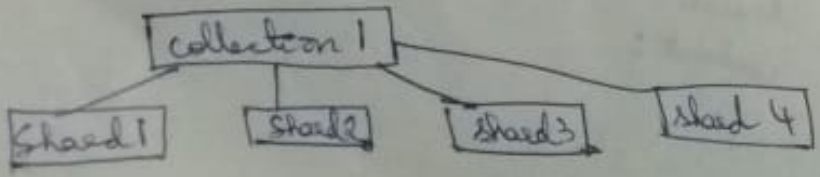
(3) Support for dynamic Queries

(4) Support for Binary data by using GridFS (for small amount of data)
for large amount of data log form (metadata)
is devided into chunks & then stored in a file.

(5) Replication
→ There are two levels of replication based. (primary & secondary)



Sharding



* Each shard contains some amount of data which needs to be processed.
* Because of shards the storing & processing of data is made easier.

(4) Updating data in place

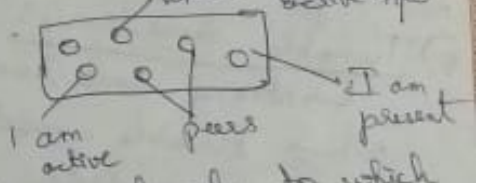
Working of Apache Cassandra

* Scalable
* Distributed Interface
- Born with Facebook
- Twitter, Netflix use Apache

Feature :-

1) Peer to peer network (It has multiple peers, work is distributed among them).
If a node fails, then completion is given to other peers, so that performance is same.
* Mem table : Data stored in datastructure, if data is full in memtable, then data is pushed to another table i.e. SS table. In SS table data is stored & sorted.

2) Gossip & failure Detection :-



3) Partitioner :- work is usually distributed to multiple nodes, partitioner decides to which node to distribute or copy the data.

4) Replication factor :- It tells how many copies of a particular row is present in nodes. Useful when a data is lost if other copies of that data is present.

Replication
- Simple strategy
- n/w topology strategy (Here expansion is easy, adding node to present n/w).

5) Anti-Entropy & Read repair:

Anti-entropy: It compares the updated data with all the replicas which are present & all replicas should be updated & kept.

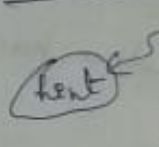
If updation is not done & client reads out the old values.

6) Write in Cassandra:

Sstable - Sorted String table.

client requests data writes → logfile → memtable → full → sstable

7) Hinted handoff:



Hint

↳ Location of node to which data is to be copied.

↳ Version metadata (data about data)

↳ Actual data.

Node A stores a hint to show that node C is down, when C is up, the replication factor is also stored in node C.

8) Tunable consistency:

↳ strong consistency

↳ tunable consistency

When there are multiple nodes & there is client requesting data:

- In strong consistency, when client requests the data which is present, it should be up to date. When the data is sent, the data is updated before getting acknowledgement back.
- In case of tunable, once a data is sent to a client and acknowledgement is received, then the data is copied to other nodes.

Tunable consistency

- Read consistency
- Write consistency.

Hadoop :-

① Everyday facebook produces 2.7 billion of data & Google 24 petabyte

② Every minute FB 2.5 million, twitter 3,00,000, instagram 2,20,000 pics are posted/min, youtube 72 hrs/min emails 20,00,00 messages r made/min, Amazon ₹ 8,00,000 sold, google 4 million

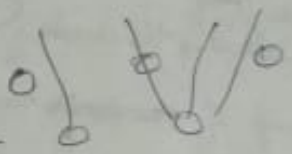
Why Hadoop

→ Massive amount of data } Fastly & Quickly } → it can handle.
→ Categories of data

Features :-

① ~~Redu~~ low cost :- Open source
↳ Different commodities of h/w to store large amount of data

② Computing power :-
↳ uses destributed computing
(if more no of nodes are present then computing power increases).
No. of nodes ↑ ⟶ Computing power ↑



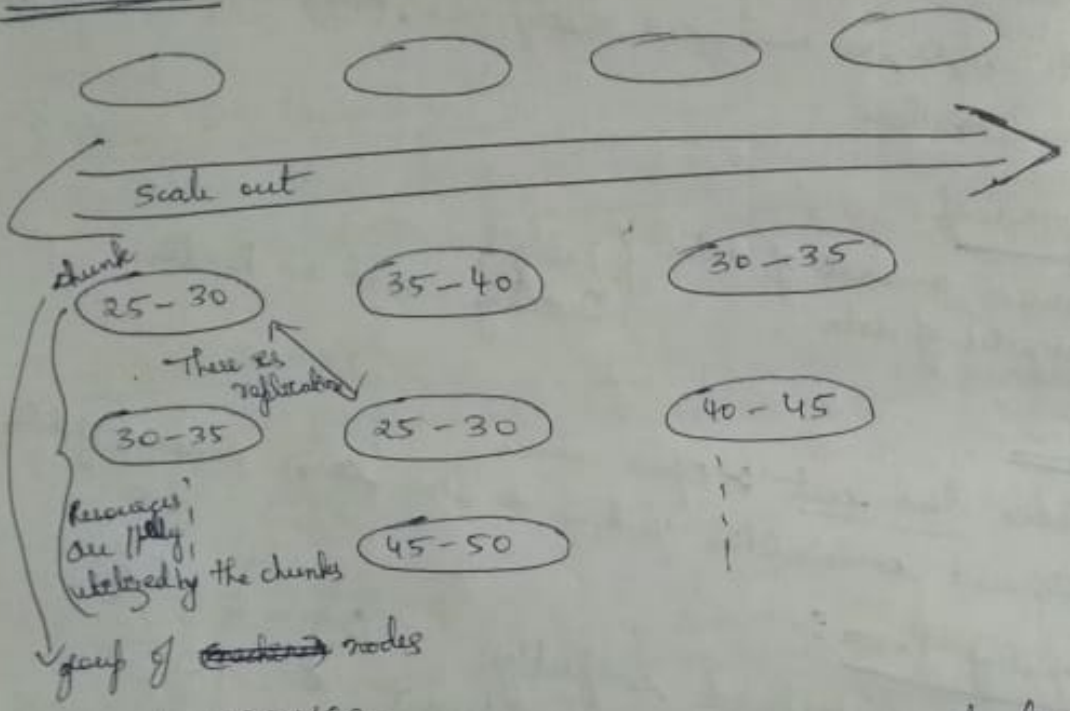
③ Scalability :-
↳ to the existing n/w we can add extra nodes to improve the performance.
↳ not much administration privileges are needed to add new nodes.

④ Flexibility on storage :-
↳ Traditional DB's } → structured data.
↓
After preprocessing the data is stored.
↳ Hadoop → unstructured data {image, audio, video} can be stored on hadoop without any preprocessing.

⑤ Inherent data protection :-

↳ when there are multiple nodes if there is a failure in one node then the data of job is directly assigned to some other node in the n/w.

Framework :-



Why not RDBMS :-

↳ These days the large amount of data that is produced is unstructured data so it is not suitable for process
↳ advanced analytics.
↳ Cost is very high to scale out the data.

Differences	RDBMS	Hadoop
Parameters		
① s/y	Relational database management s/y	Node based flat structure.
② I/p data	Structured data	Unstructured & structured data (files with different formats) (XML, JSON)
③ processing	OLTP — online transaction processing	Big Data processing, Analytical processing

(4) Choice	R/p data which needs consistency & relationship b/w the data	Big data & there is no consistency with the R/p data.
(5) Processor	High end processors & they are costly	Cluster (consisting of nodes & each node consists of hard drive, NIC cards & a processor)
(6) Cost	$10,000 → $14,000 to the terabyte of data	$4,000 / TB data

Distributed Computing Challenges :-

(1) H/w failure : if h/w fails how can we access the data.

(2) How to process gigantic data : How to process large amount of data which is spread around multiple s/y's.

Replication factor = 2 (2 copies of data is stored in a particular s/y)

Map-reduce programming model is used to process large amount of data.

Overview of Hadoop :-

(1) Faster data processing } 2 basic tasks done by hadoop.
(2) Massive data processing }

Key aspects of Hadoop :-

(1) Open Source : s/w. : free to use.

(2) Framework : tools & programs & APIs are already provided by hadoop.

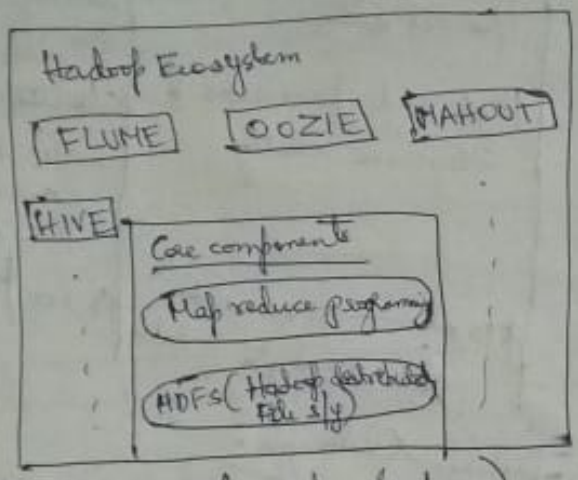
(3) Distributed : processing is distributed among multiple nodes.

} parallel processing.

(4) Massive Storage : vast of variety of data.

(5) Faster Processing → provides quick response.

Components of (~~Hey~~) Hadoop:-

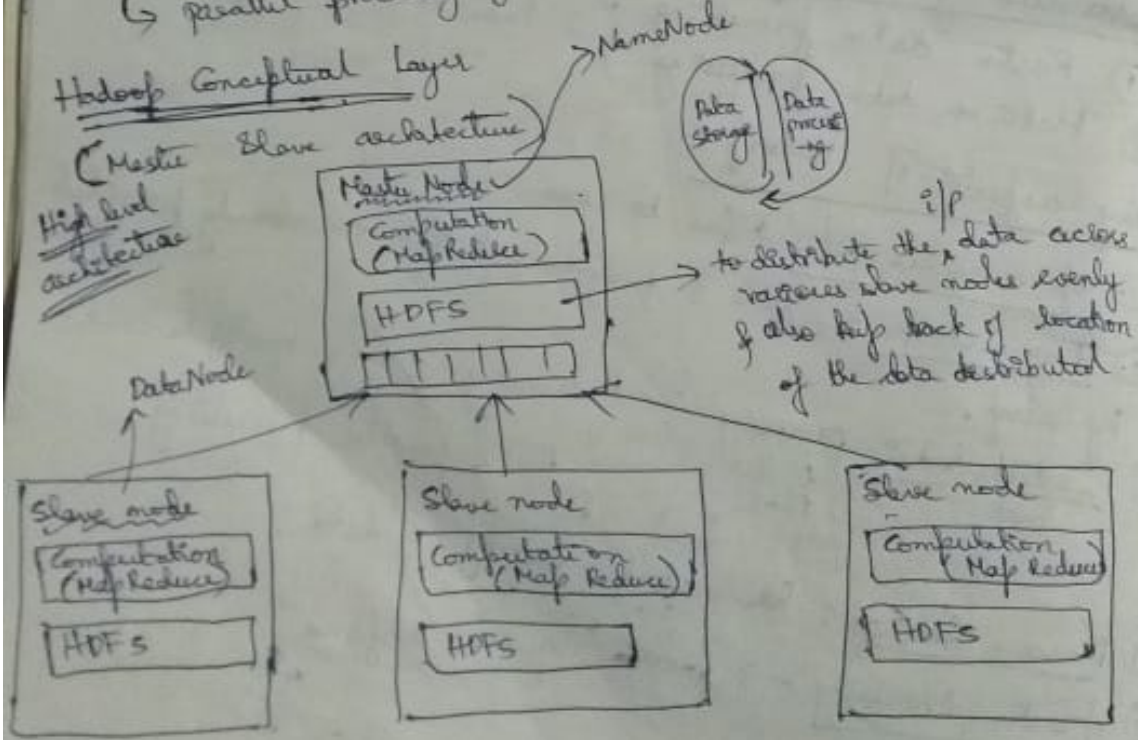


① HDFS (Hadoop distributed File System)
- ↳ storage component
- ↳ i/p data is distributed across several nodes.
- ↳ naturally redundant.

② Map Reduce Programming
- ↳ used for programming purpose.
- ↳ computational Framework.
- ↳ Task is distributed across several nodes:.
- ↳ parallel processing of the data.

Hadoop Conceptual Layer
(Master Slave architecture)

High level architecture



Master Hdp Reduce :-

↳ Computation is distributed evenly across various slave nodes present in the s/y.

• Master Node is called as NameNode & slave nodes are called as DataNodes.

Use case of Hadoop :-

↳ clickstream data analysis and CRM.

↳ an analysis is done on the users based on his clicks. (analyzes diff data of URL clicked by the user).

↳ In-business they come to know about the particular customer based on the history of his clicks.

↳ Hadoop supports clickstream analysis & CRM (customer relationship management)

↳ Scalability.

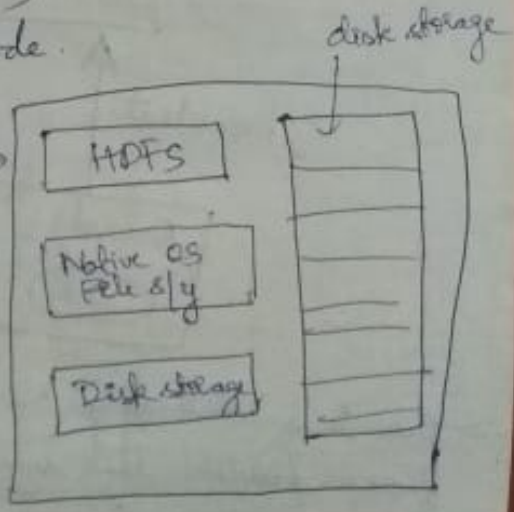
↳ helps hadoop to store large amounts of data.

↳ years of data can be stored without any delay.

↳ Business Analysis → Apache Hive.

HDFS (Hadoop Distributed File S/y)

① Storage components
② Distributed File S/y.
③ Optimized for high throughput.
④ Modelled after Google file s/y.
⑤ Replicate a file (a any nbr of times)
⑥ Re-replicate the data on failed node.
⑦ Read / write



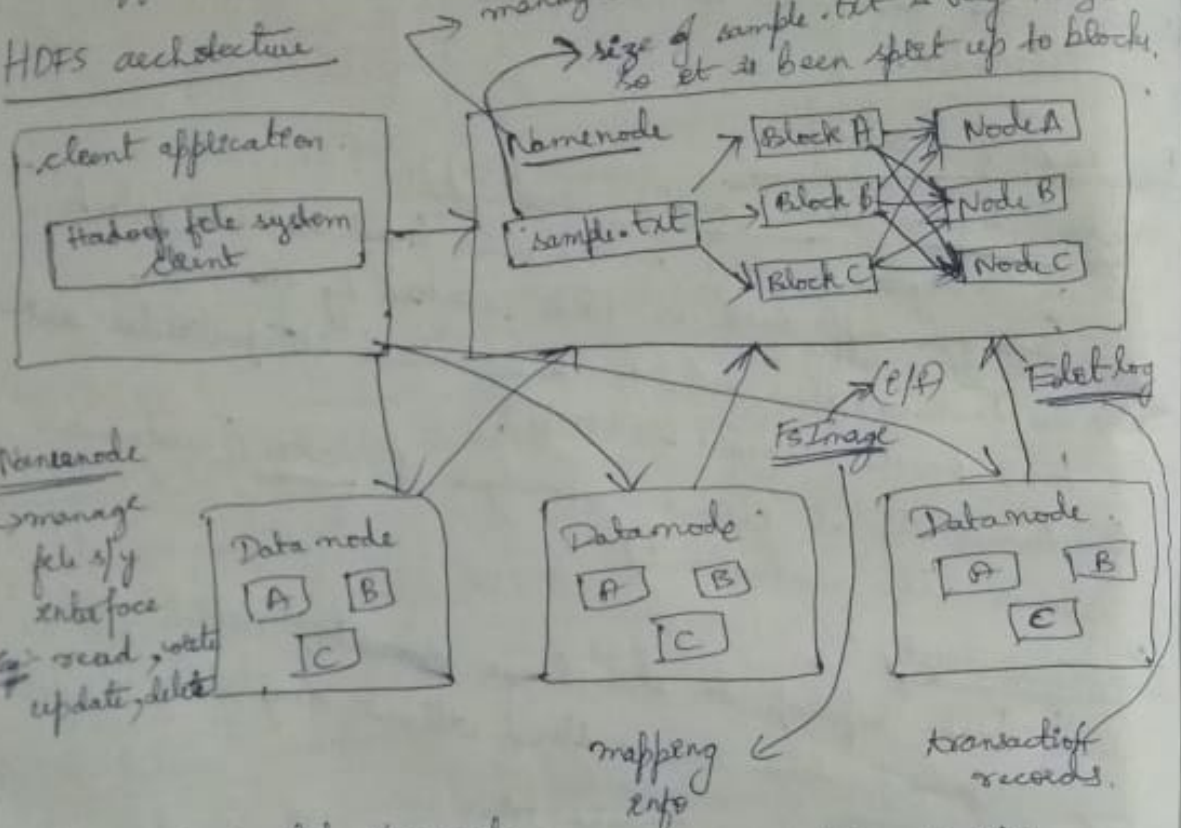
HDFS Daemons

Hadoop distributed file s/y :-

→ store the data in the form of blocks.
→ Replication factor is set as 3.
→ block size 64 Mb

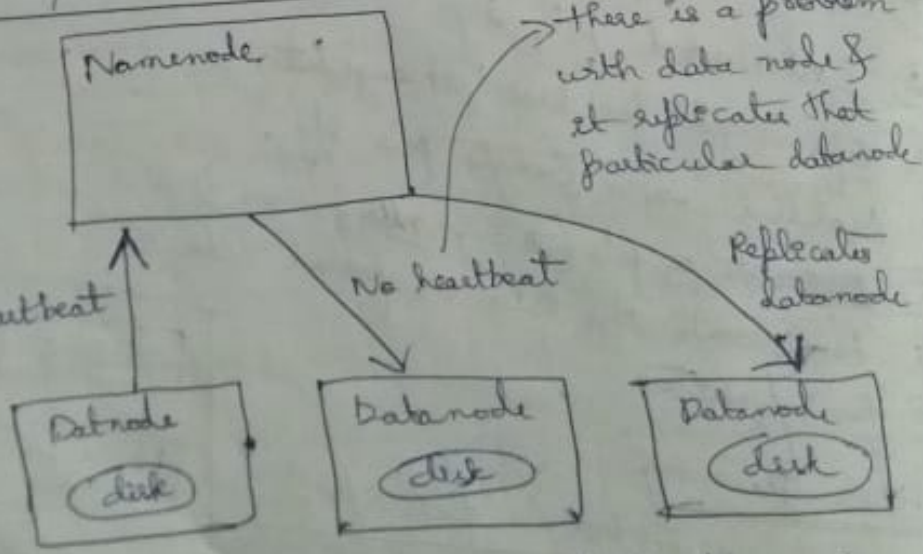
→ Large file is being split up to diff blocks.
→ store data in diff rack & each rack has ID to identify where data is stored.

HDFS architecture



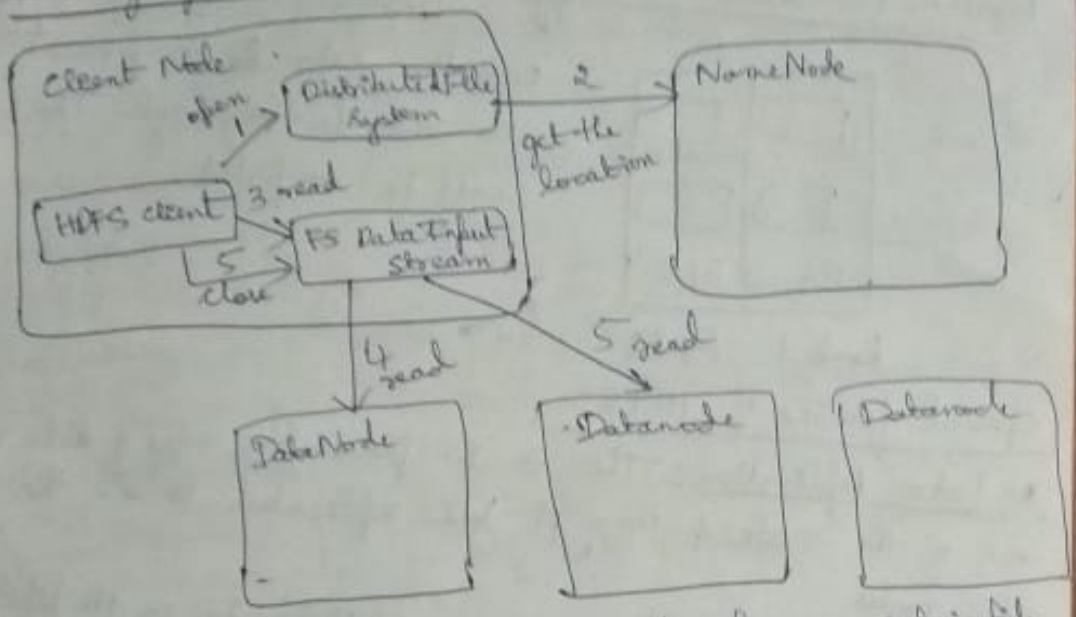
→ In the initial step Namenode will read the contents of FsImage & Edit log of then starts working.

Communication b/w Namenode & Datanode :-



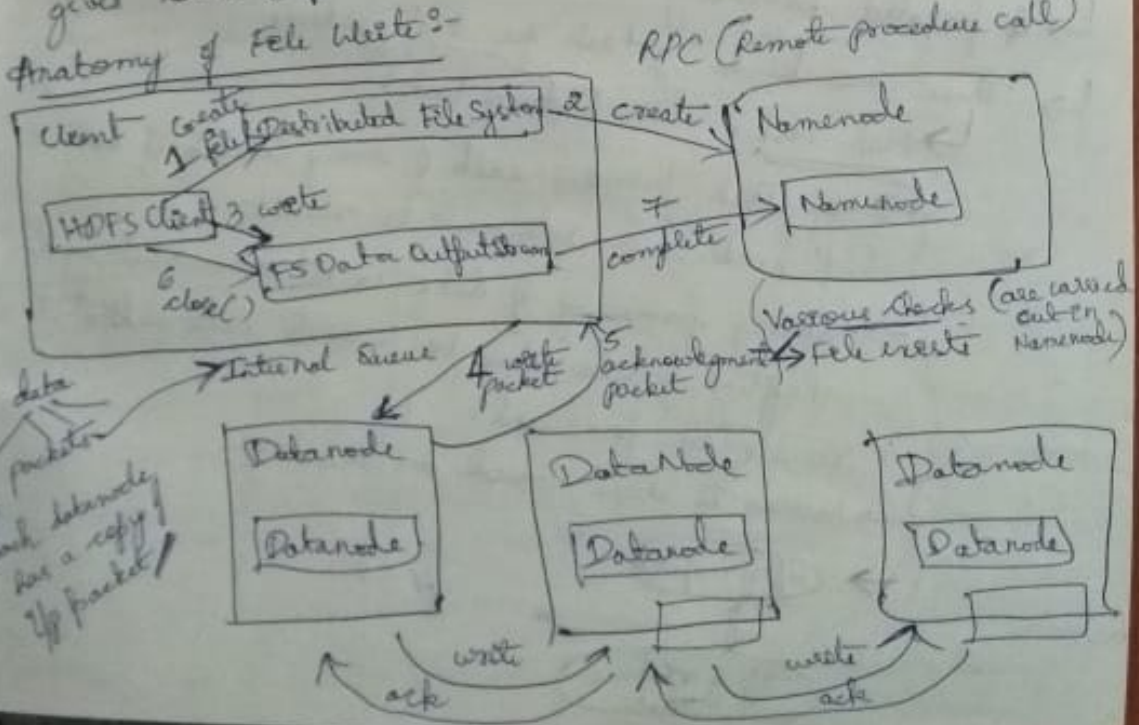
→ there is a secondary Namenode, so that if the Namenode faces a problem then secondary Namenode can be used.
has same resources as Namenode but runs on other system.

Anatomy of File read :-

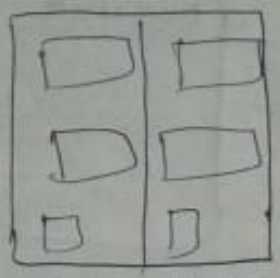


① Initially "open" s/y call is given to open a certain file.
② DFS gives request to NameNode & it gives location of diffunt data blocks present in the s/y.
③ The HDFS client gives a "read" request to FS DataInputStream → gives "read" request to diff datanode
④ Once read request is processed then HDFS client gives "close" s/y call to FS DataInputStream.

Anatomy of File Write :-



Replica Placement Strategy:



replica of
Once the data is present in each of every node. The task will be completed.

Special features of HDFS:

* Data Replication:- The client places the copy of data in one of the node & then it gets replicated to all the other nodes.
* Data pipeline:- Data is put to all the nodes in the pipeline

Processing Data with Hadoop:

↳ Map-reduce is used to process the data.
↳ Large file to be processed is divided into chunks.
↳ & parallel processing is done for each of every chunk
↳ There are kinds of task for Map-reduce:-
  ↳ Map Task:-
    → Map Task processes each of every chunk of data & produces an intermediate result.
    → during processing of data will have sorting & shuffling of data due to which intermediate results are produced.
    → Sorting is done based on the keys.
    → o/p of map task
    ↓
    I/p of reduce task
  ↳ Reduce Task:-

↳ Reduce Task

→ Combine all the o/p's which have been produced by diff Map func's.

→ Data required for processing is available locally due to which performance will improve.

HDFS (Data Storage) } available locally
Map Reduce (Data processing) }

→ There are 2 kinds of tracker to check if processing is done locally or not.

* Jobtracker
  ↳ present in namenode
  ↳ responsibility of jobtracker is to assign diff tasks to tasktracker.

* Tasktracker.
  ↳ present in different data nodes.

Map Reduce phases & daemons

* phases are Map & Reduce

  Map → Use mapping operation, to produce key-value pair as result on i/p

  Reduce → Combine the o/p from diff maps & at the end it gives some result.

* Daemons are Jobtracker & Tasktracker

  Jobtracker → Acts as a Master
  ↳ Used to schedule the diff task.

  Tasktracker → acts as a slave.
  ↳ Execute the assigned task.

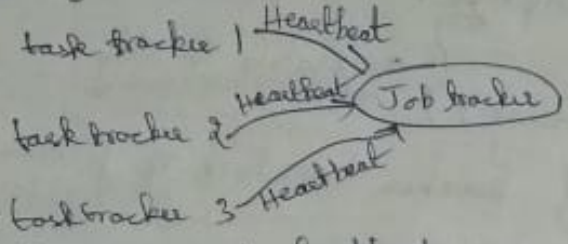
Daemons

* Jobtracker
  ↳ provides connectivity b/w hadoop & application.
  ↳ Execution plan i.e assign diff tasks to diff nodes
  ↳ If there is a failure with the data nodes then it will try again by giving resources but even after many tries there is a failure then it gives tasks to other data nodes that is active in the n/w.
  ↳ The no of jobtracker for a s/y = 1 (usually)

* Task tracker :-

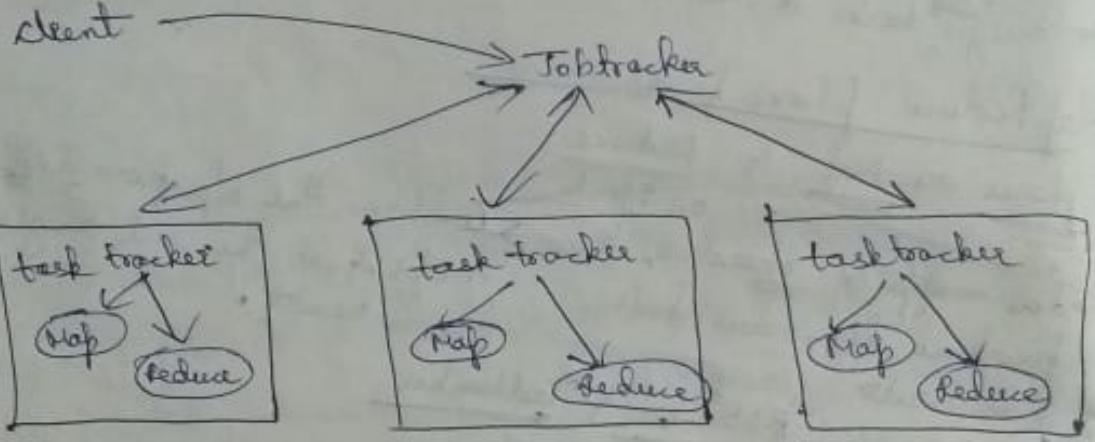
↳ Responsibility — execute the task which is assigned by Jobtracker.

↳ Each node has single task tracker but it may have multiple Java Virtual machine support.

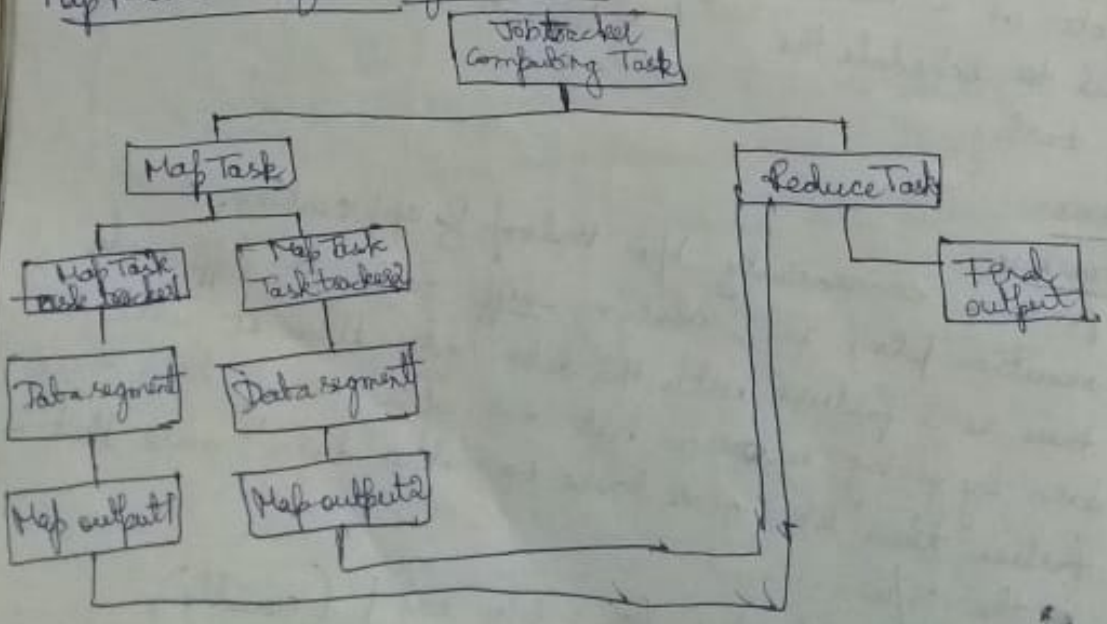
↳ 

Task tracker sends heartbeat msg to Job tracker to specify that it is running. If it doesn't send heartbeat msg then Jobtracker will come to know that there is a failure with task tracker.

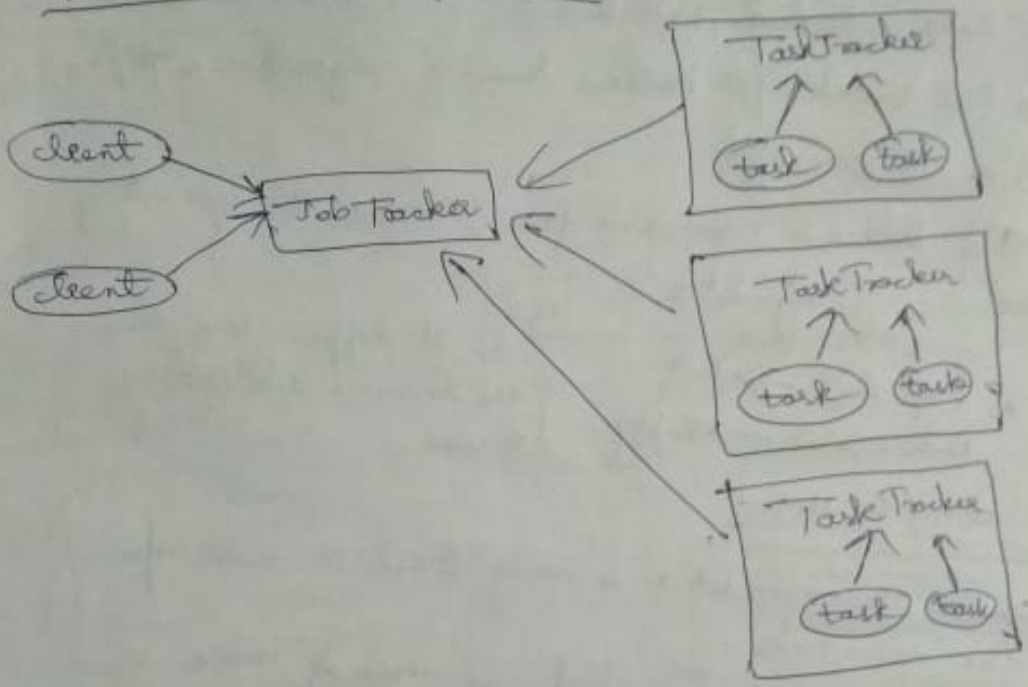
Task tracker & Job tracker Interaction :- (Basic)



Map Reduce Programming Workflow :-



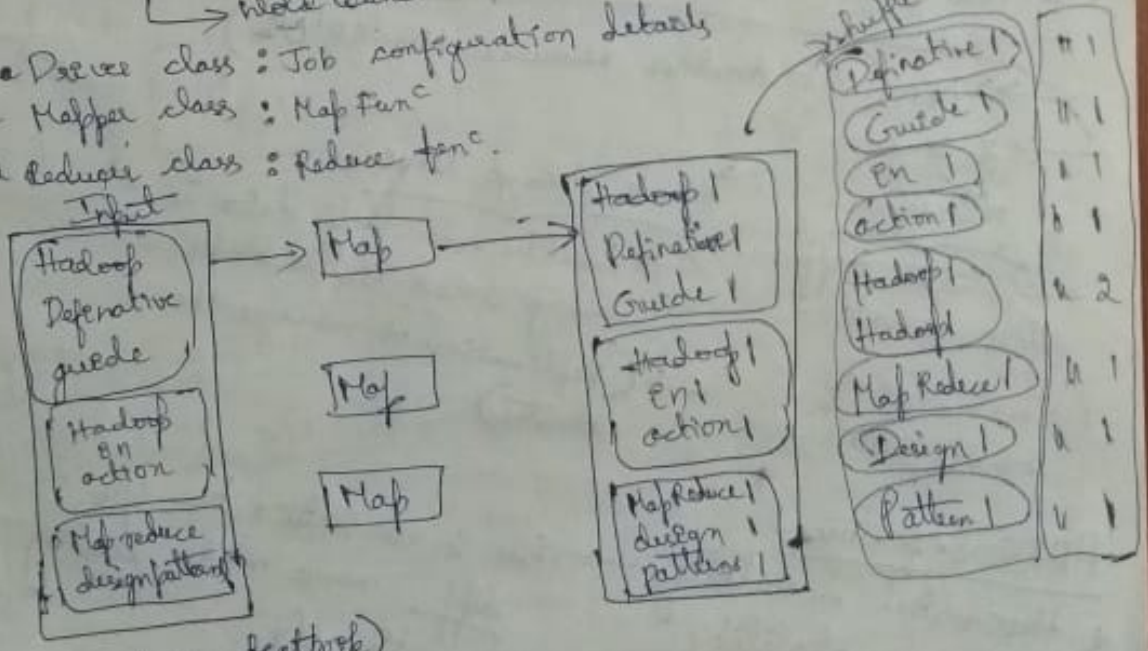
MapReduce Programming Architecture :-



MapReduce Example

↳ Word Count

- Driver class : Job configuration details
- Mapper class : Map func
- Reducer class : Reduce func.



2 programs (Refer textbook)

Limitations of Hadoop 1.0

↳ Single Namenode which takes all the responsibilities.

↳ Restricted processing (useful for batch oriented jobs)

↳ Not suitable for interactive processing or analysis of data. (Hadoop MapReduce)

↳ Not suitable for Machine learning algorithms, Graphs, M/m intensive appl.

↳ Map Reduce is responsible for resource management & processing of s/g data.

- Map slots → Full
- Reduce slots → Empty

↳ If it happens then there are resource utilization issues.

Limitations on HDFS

↳ All the metadata is being stored in main m/m i.e namenode.

↳ Performance is degraded as more & more objects are being stored.

↳ This problem is solved by Hadoop 2.0

→ YARN (Yet Another Resource Negotiator)

→ 2 components

- Namespace → handles diff file operation (Create, update, modify, delete)
- Block storage service → handles when diff data is being stored in the blocks. (Replication & manage the cluster)

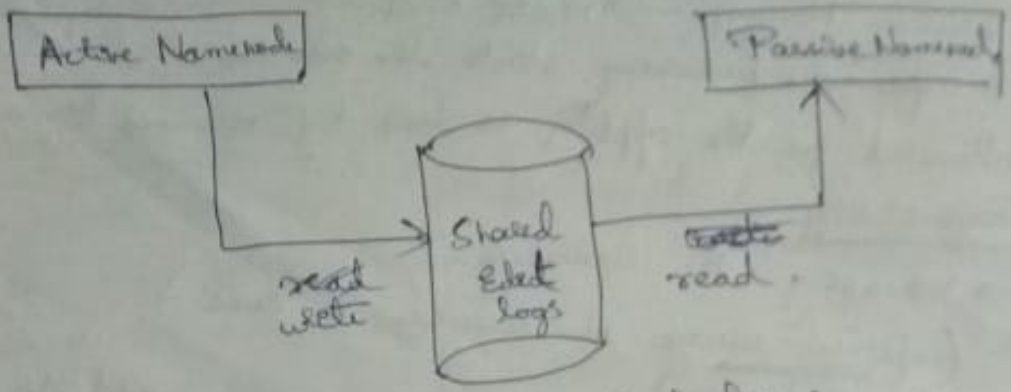
- HDFS 2 features:

& Horizontal scalability → There is no interaction b/w diff name nodes existing in the s/g

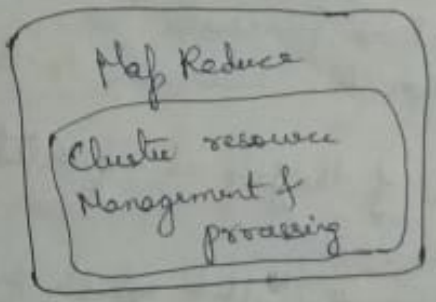
& High availability.

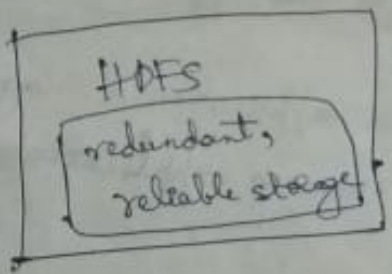
↳ Even if there is a problem with particular name node other name node can continue with the problematic name node's job.

Active Namenode — Passive (Active)

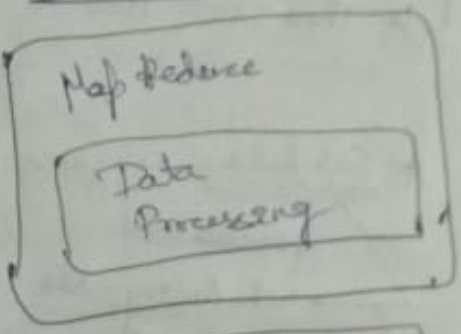


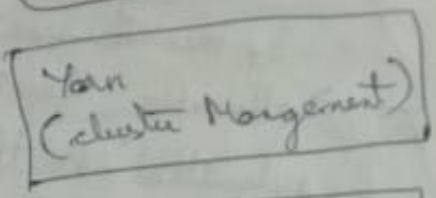
Hadoop 1.0

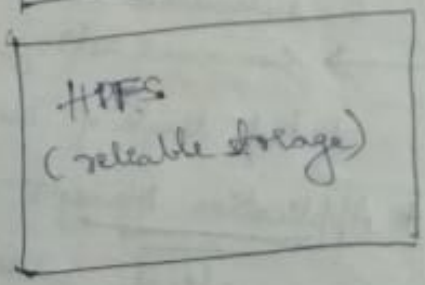




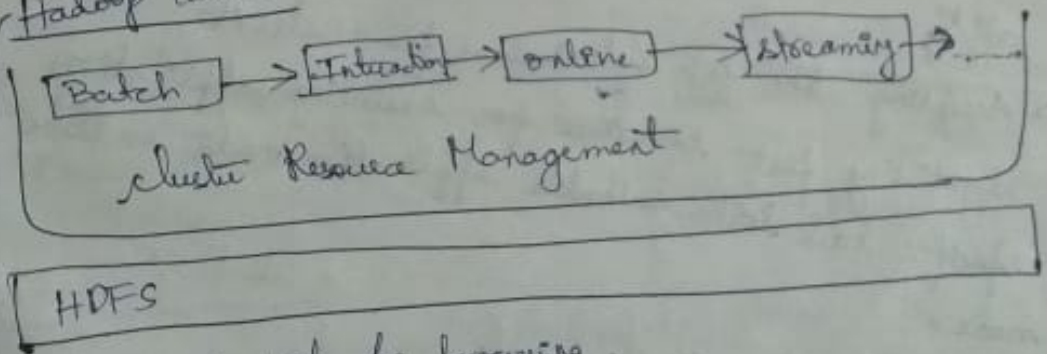
Hadoop 2.0







Hadoop Yarn:



→ Framework used for processing.

Yarn has 3 Major components

- ① Resource Manager
- ② Node Manager
- ③ Application Master → Slave Nodes

① Resource Manager → Master Node.

↳ Before the processing starts the resources need to be allocated for the appl^n it is done by resource Manager

Responsibilities:-

↳ managing the resources.
(optimal usage of resources is done)
(ideal case)

↳ There are two processes, Scheduler & Appl Manager responsible to co-ordinate b/w the two processes.

* Scheduler is going to assign the resources to the processes.
  - ↳ scheduling the execution of the job as requested by the resource manager.
  - ↳ allocating the resources for the appl which are submitted to the cluster.
  - ↳ communicating with your application manager, & keep track of resources of running applications.

* Application Manager
  - ↳ co-ordinate with scheduler to keep track of running appl's.
  - ↳ Accepting the job submissions from client.
  - ↳ negotiating first container for executing application specific task with suitable application master on slave node.

② Node Manager:- → slave node.

Responsibilities:-

↳ Managing & executing the containers.

↳ Monitoring usage of the resources (Memory, CPU..) & reporting it to the resource manager.

↳ Sending heart beat messages regarding status update to Resource manager.

③ Application Master → slave node

↳ specific for the particular appl.

↳ per appl specific library which works with node manager to execute task.

↳ If multiple jobs are submitted on cluster, more than one instance of appl master on slave node.

Responsibilities:-

↳ Negotiating the suitable resource containers.

↳ working with one or more Node managers to check the proper working of slave nodes.

Basic diff b/w Yarn & MapReduce:-

Yarn	MapReduce
→ Supports variety of processing engines & applications.	→ Supports its own applications (supports its own batch processing appl)
→ Work is distributed among 3 major components (separate its duties across multiple components)	→ considerated most of its work in single component.
→ Can dynamically allocate pools of resources to appl.	→ static allocation of resources for the designated task.

Needs of Yarn:-

↳ opened up new uses for Apache HBase, Apache Hive.

↳ offers scalability, resource utilization, high availability & performance is improved as compared to map reduce.

↳ Scalable for real-time processing because it separates HDFS from Map Reduce & also other applications

need not wait for batch jobs to finish.
↳ Manages the resources on clustered environment.
↳ interacts with computer resources & then assigns the resources because it will come to know the scarcity of the resources.

Hive

(Data Warehousing tool)

- On top of hadoop framework structured data is present & to query this Hive tool is used.
- Reduces HDFS for storage purpose.
- For executing it uses MapReduce.
- Stores metadata in RDBMS.

Reasons for Hive to be called as Data Warehousing tool:-

- Suitable for data warehousing appl.
- It processes batch jobs.

Eg: Web logs, Appl logs.

History of Hive

↳ 2007 – used by Facebook (to analyze incoming log data in FB).

↳ 2008 – Apache Hadoop subproject.

Recent release of Hive:-

Hive 0.10
- batch data proxy
- read-only data.
- HiveQL (Query language)

Hive 0.13
- To process interactive data.
- read only data
- substantial QL

Hive 0.14
- For processing transactions with ACID properties.
- Cost based optimizer
- SQL temporary tables

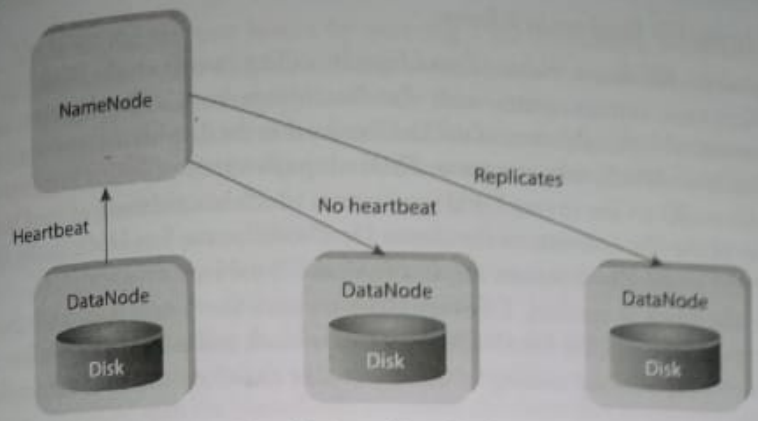


Figure 5.17 NameNode and DataNode Communication.

5.10.1.3 Secondary NameNode

The Secondary NameNode takes a snapshot of HDFS metadata at intervals specified in the Hadoop configuration. Since the memory requirements of Secondary NameNode are the same as NameNode, it is better to run NameNode and Secondary NameNode on different machines. In case of failure of the NameNode, the Secondary NameNode can be configured manually to bring up the cluster. However, the Secondary NameNode does not record any real-time changes that happen to the HDFS metadata.

5.10.2 Anatomy of File Read

Figure 5.18 describes the anatomy of File Read.

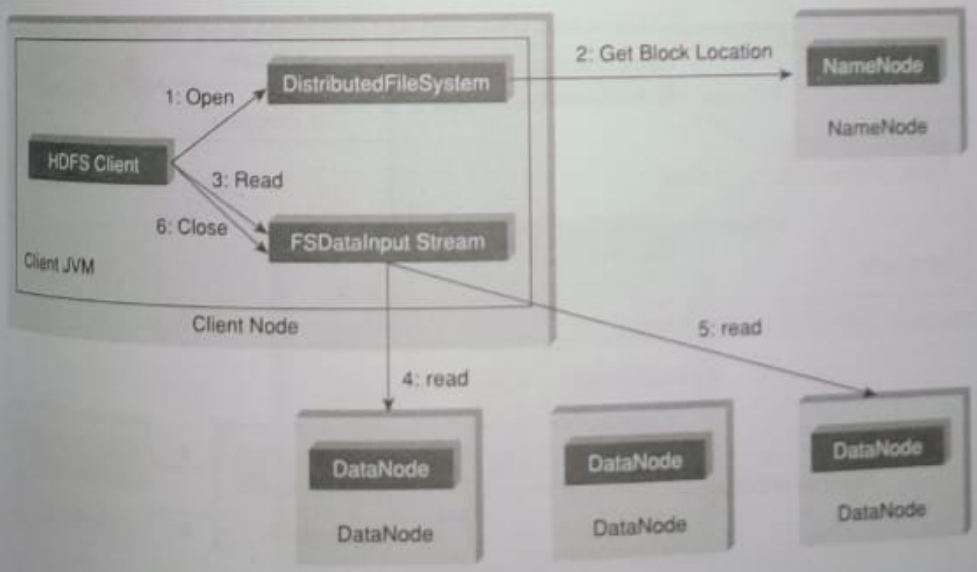


Figure 5.18 File Read.

The steps involved in the File Read are as follows:

1. The client opens the file that it wishes to read from by calling open() on the DistributedFileSystem.
2. DistributedFileSystem communicates with the NameNode to get the location of data blocks. NameNode returns with the addresses of the DataNodes that the data blocks are stored on. Subsequent to this, the DistributedFileSystem returns an FSDataInputStream to client to read from the file.
3. Client then calls read() on the stream DFSInputStream, which has addresses of the DataNodes for the first few blocks of the file, connects to the closest DataNode for the first block in the file.
4. Client calls read() repeatedly to stream the data from the DataNode.
5. When end of the block is reached, DFSInputStream closes the connection with the DataNode. It repeats the steps to find the best DataNode for the next block and subsequent blocks.
6. When the client completes the reading of the file, it calls close() on the FSDataInputStream to close the connection.

Reference: Hadoop, The Definitive Guide, 3rd Edition, O'Reilly Publication.

5.10.3 Anatomy of File Write

Figure 5.19 describes the anatomy of File Write. The steps involved in anatomy of File Write are as follows:

1. The client calls create() on DistributedFileSystem to create a file.
2. An RPC call to the NameNode happens through the DistributedFileSystem to create a new file. The NameNode performs various checks to create a new file (checks whether such a file exists or not). Initially, the NameNode creates a file without associating any data blocks to the file. The DistributedFileSystem returns an FSDataOutputStream to the client to perform write.
3. As the client writes data, data is split into packets by DFSOutputStream, which is then written to an internal queue, called *data queue*. DataStreamer consumes the data queue. The DataStreamer requests

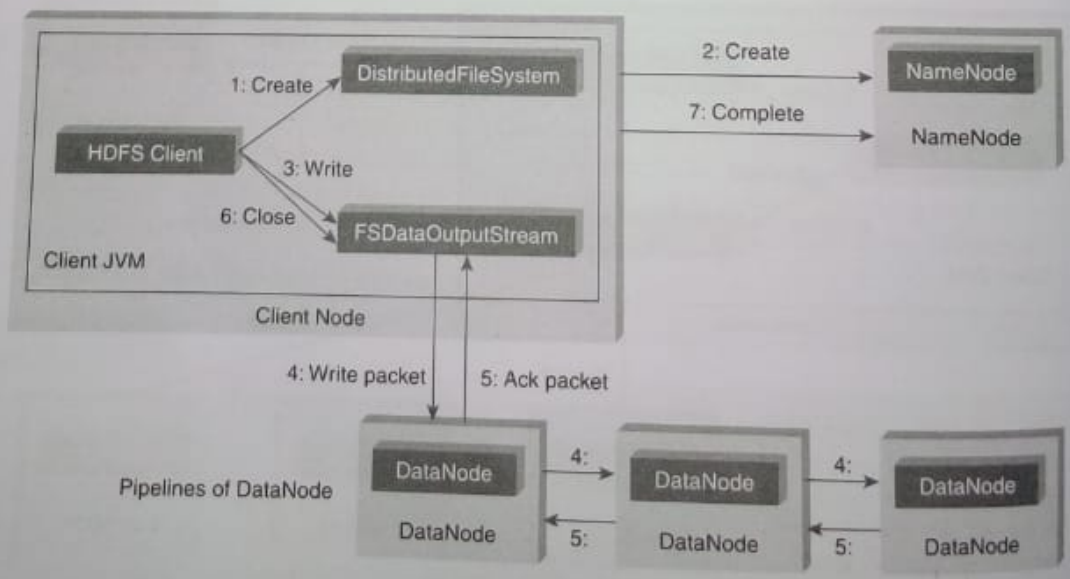
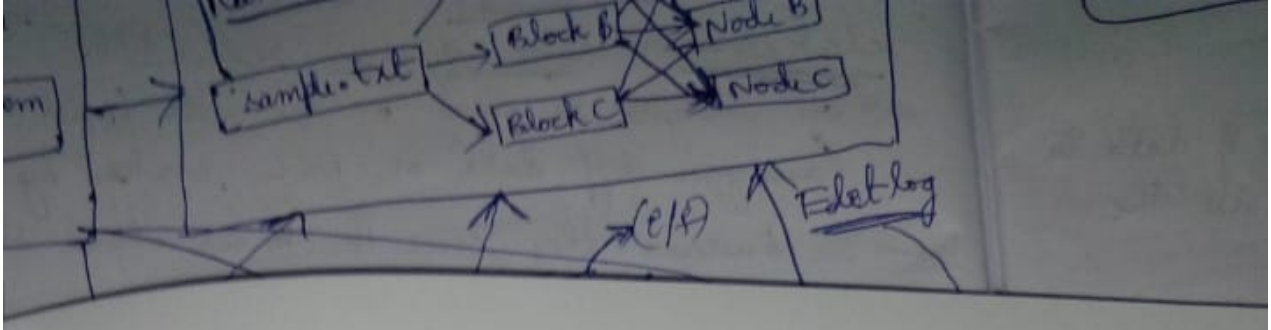


Figure 5.19 File Write.



the NameNode to allocate new blocks by selecting a list of suitable DataNodes to store replicas. This list of DataNodes makes a pipeline. Here, we will go with the default replication factor of three, so there will be three nodes in the pipeline for the first block.

4. DataStreamer streams the packets to the first DataNode in the pipeline. It stores packet and forwards it to the second DataNode in the pipeline. In the same way, the second DataNode stores the packet and forwards it to the third DataNode in the pipeline.
5. In addition to the internal queue, DFSOutputStream also manages an "Ack queue" of packets that are waiting for the acknowledgement by DataNodes. A packet is removed from the "Ack queue" only if it is acknowledged by all the DataNodes in the pipeline.
6. When the client finishes writing the file, it calls close() on the stream.
7. This flushes all the remaining packets to the DataNode pipeline and waits for relevant acknowledgments before communicating with the NameNode to inform the client that the creation of the file is complete.

Reference: Hadoop, The Definitive Guide, 3rd Edition, O'Reilly Publication.

5.10.4 Replica Placement Strategy

5.10.4.1 *Hadoop Default Replica Placement Strategy*

As per the Hadoop Replica Placement Strategy, first replica is placed on the same node as the client. Then it places second replica on a node that is present on different rack. It places the third replica on the same rack as second, but on a different node in the rack. Once replica locations have been set, a pipeline is built. This strategy provides good reliability. Figure 5.20 describes the typical replica pipeline.

Reference: Hadoop, the Definite Guide, 3rd Edition, O'Reilly Publication.

5.10.5 Working with HDFS Commands

Objective: To get the list of directories and files at the root of HDFS.
Act:
hadoop fs -ls /

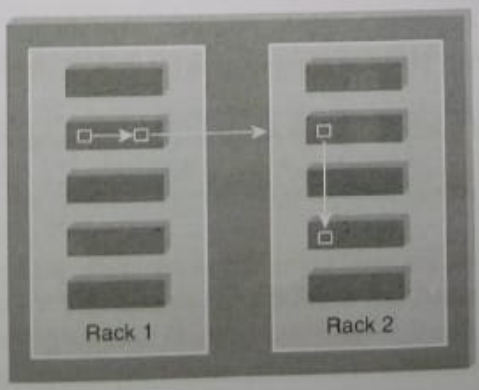


Figure 5.20 Replica Placement Strategy.