UNIT - II

Introduction to Hadoop

Introducing Hadoop :-

→ Big Data seems to be the buzz word!

→ Enterprises, the world over, are beginning to realize that there is a huge volume of untapped information before them in the form of structured, semi-structured, and unstructured data.

→ This varied variety of idea is spread across the networks.

→ Few statistics to get an idea of the amount of data which generated every day, every minute, and every second.

1. Every day :-
   - NYSE (New York Stock Exchange) generate 1.5 billion shares and trade data.
   - Facebook stores 2.7 billion comments and Likes.
   - Google processes about 24 peta bytes of data.

2. Every minute :-
   - Facebook users share nearly 2.5 million pieces of content.
   - Instagram users post nearly 2,20,000 new photos
   - YouTube users upload 72 hours of new video content.
   - Google receives over 4 million search queries.

3. Every Second :-
   - Banking applications process more than 10,000 credit card transactions.

Data : The Treasure Trove :-

1. provides business advantages such as generating product recommendations, inventing new products, analyzing the market, and many more.
2. provides few early key indicators that can turn the fortune of business.
3. provides room for precise analysis, If we have more data for analysis, then we have greater precision of analysis.

"I am inundated with data" How to store terabytes of mounting data?

"I need this data to be processed quickly. My decision is pending." How to access the information quickly?

"I have data in varied sources. I have data that is rich in variety - structured, semi-structured, unstructured." How to work with data that is so very different?

Fig:- challenges with big volume, variety & velocity of data.

Why Hadoop?:-

→ It's capability to handle massive amount of data, different categories of data - fairly quickly.

Key considerations of Hadoop:-

1) Low cost:- Hadoop is an open-source framework and uses commodity hardware to store enormous quantities of data.

2) Computing power:- Hadoop is based on distributed computing model which processes very large volume of data fairly quickly. The more the number of computing nodes, the more the processing power at hand.

3) Scalability :- Simply adding nodes as the system grows and requires much less administration.

4) Storing flexibility:- unlike the traditional databases, in Hadoop data need not to be pre-processed before storing it. Hadoop provides the convenience of storing as much data as one needs and also, the added flexibility of deciding later as to how to use the stored data.

5) Inherent data protection:- Hadoop protects data and executing application against hardware failure. If a node fails, it automatically redirects the jobs that had been assigned to this node to the other functional and available nodes and ensures that distributed computing does not fail.

→ Hadoop makes use of commodity hardware, distributed file system, and distributed computing, as shown below.

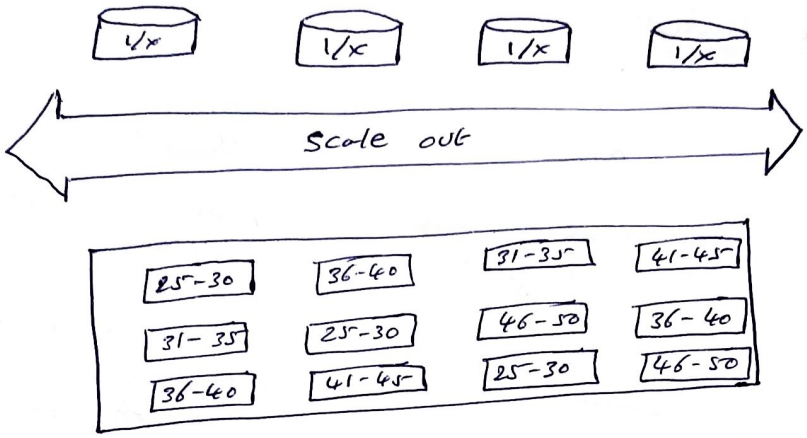


Fig:- Hadoop framework (distributed file system)

→ In this new design, groups of machine are gathered together, it is known as a cluster.

→ The data can be managed with Hadoop as follows:-

1) Distributes the data and duplicate chunks of each data file across several nodes, for example 25-30 is one chunk of data.

2) Locally available compute resource is used to process each chunk of data in parallel.

3) Hadoop framework handles failover smartly and automatically.

Why Not RDBMS:-

→ RDBMS is not suitable for storing and processing large files, images and videos.

→ RDBMS is not a good choice when it comes to advanced analytics involving machine learning. with respect to cost & storage.

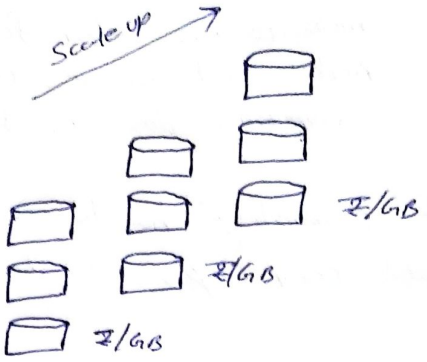


Fig:- RDBMS with respect to cost/GB of storage.

RDBMS Versus Hadoop:-

Parameters	RDBMS	Hadoop
1) System	Relational database Management system	Node based flat structure
2) Data	Suitable for Structured Data	Suitable for Structured, unstructured data. Supports variety of data formats in real time such as XML, JSON, text based flat file etc.
3) Processing	OLTP	Analytical, Big Data processing
4) choice	The data needs consistent relationship	Big Data processing, which does not require any consistent relationship b/w data.
5) processor	Need expensive hardware or high-end processors to store huge volumes of data	In a hadoop cluster, a node require only a processor, a network card, and few hard drives.
6) cost	cost around $10,000 to $40,000 per terabytes of storage	cost around $4,000 per terabytes of storage.

History of Hadoop:

→ Hadoop was created by "Doug cutting", the creator of Apache Lucene (a commonly used text search library).

→ Hadoop is a part of Apache Nutch (Yahoo) project (an open-source web search engine), and also a part of the Lucene project.

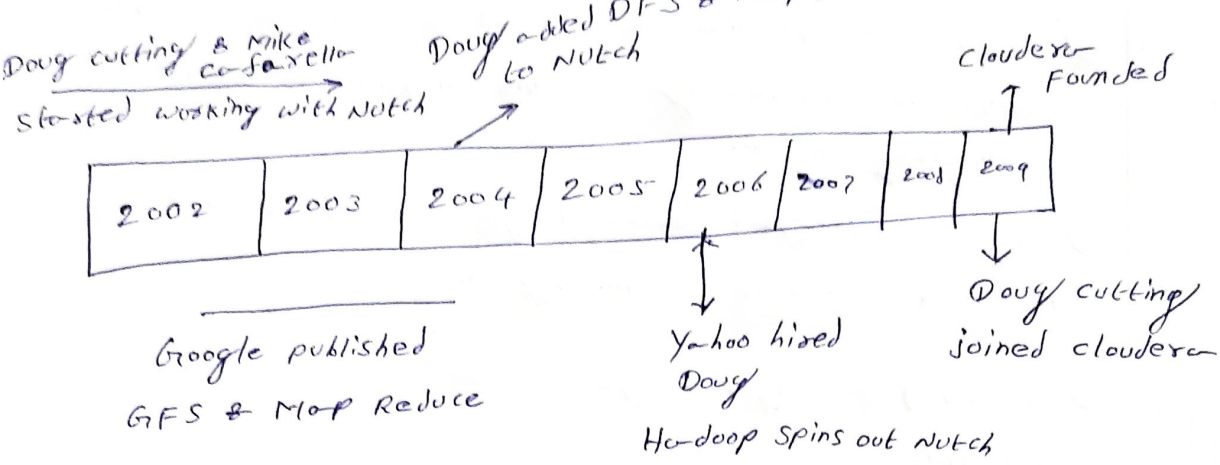


Fig: History of Hadoop

→ The "Name Hadoop" is not an acronym; it's made-up name.

→ How the name came about: "The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell & pronounce, meaningless, & not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term".

Hadoop overview:

→ open-source software framework to store and process massive amount of data in a distributed fashion on large clusters of commodity hardware.

→ Basically, Hadoop accomplishes two tasks:

1) Massive data storage
2) Faster data processing.

Key aspects of Hadoop:-

1) Open source software : It is free to download, use and contribute to.

2) Framework : Means everything that you will need to develop & execute and application is provided - programs, tools, etc,

3) Distributed :- Divides and stores data across multiple computers. Processing is done in parallel across multiple connected nodes.

4) Massive Storage : Stores large amount of data across nodes of low-cost commodity hardware.

5) Faster processing: Large amounts of data is processed in parallel.

Hadoop Components:-

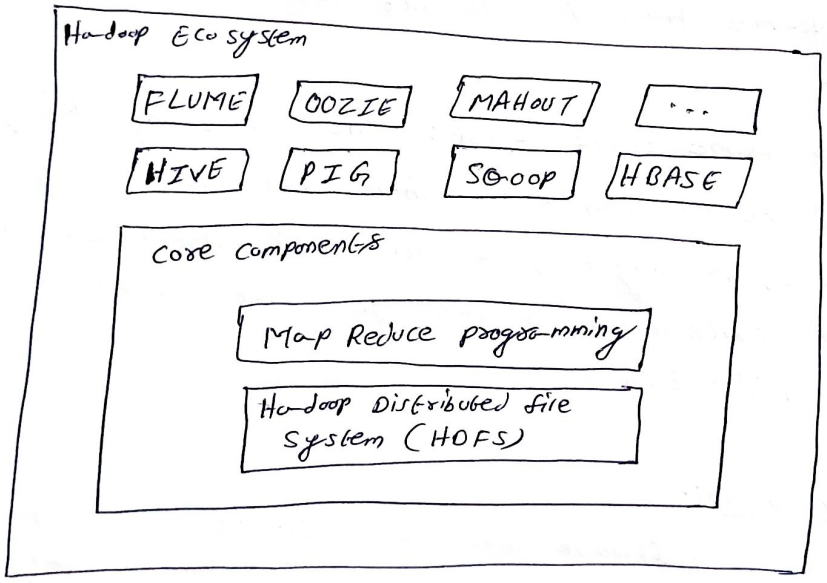


Fig :- Hadoop Components

1) Hadoop core components:-

a) HDFS:-
- Storage components
- Distributes data across several nodes
- Natively redundant.

(b) Map Reduce :-

- Computational framework
- Splits a task across multiple nodes
- processes data in parallel.

Hadoop Eco System :-

→ Hadoop Eco system are support projects to enhance the functionality of Hadoop core components.

→ The Eco projects are as follows :-

1) HIVE
2) PIG
3) Sqoop
4) HBASE
5) FLUME
6) OOZIE
7) MAHOUT

Hadoop Conceptual Layer :-

→ It is conceptually divided into :-

1) Data Storage Layer
2) Data processing Layer.

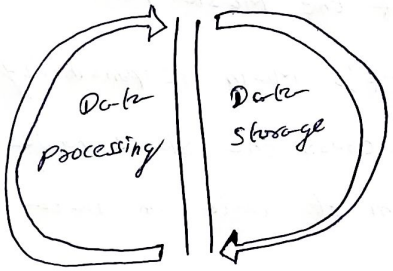


Fig :- Hadoop conceptual Layer

→ Data Storage Layer : stores huge volume of data

→ Data processing Layer : processes data in parallel to extract richer & meaningful insights from data.

High - Level Architecture of Hadoop :-

→ Hadoop is a distributed "Master - slave Architecture."

→ Master node is known as "Name Node".

→ Slave node is known as "Data Node".

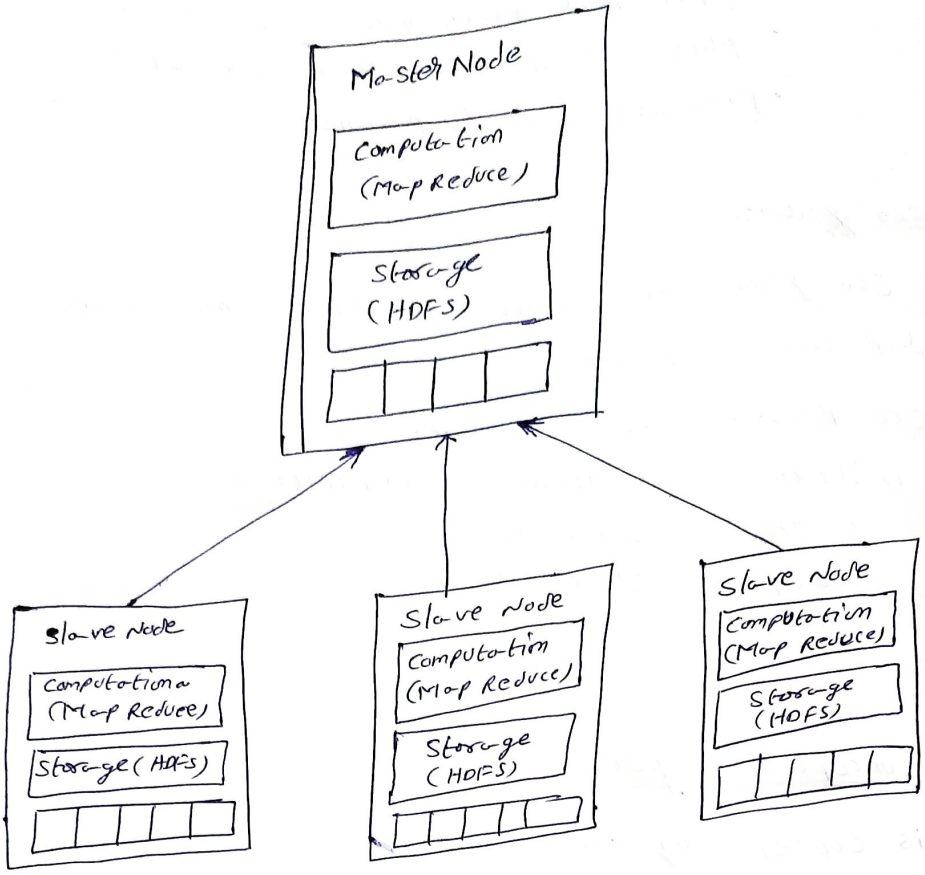


Fig:- Hadoop high-level architecture

→ The key components of the Master Node:-

1) Master HDFS:- It's main responsibility is partitioning the data storage across the slave nodes. It also keeps track of location of data on Data Nodes

2) Master Map Reduce:- It decides and schedules computation task on slave nodes.

Hadoop Distributors:-

→ It provides products that include Apache Hadoop, Commercial support, and/or tools and utilities related to Hadoop.

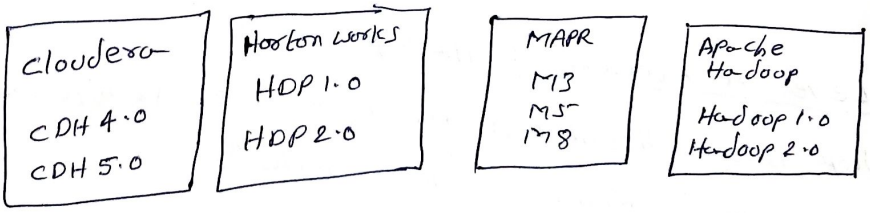


Fig:- Hadoop Distributors.

Features of Hadoop:-

→ It is optimized to handle massive quantities of structured, semi-structured, & unstructured data, using commodity hardware, that is, relatively inexpensive computers.

→ Hadoop has a shared nothing architecture.

→ It replicates its data across multiple computers so that, if one goes down, the data can still be processed from another machine that stores its replica.

→ Hadoop is for high throughput rather than low latency. It is a batch operation handling massive quantities of data. therefore the response time is not immediate.

→ It complements "OLTP" and "OLAP". However, it is not a replacement for a "RDBMS".

→ It is not good when work cannot be parallelized (or) when there are dependencies within the data.

→ It is not good for processing small files. It works best with huge data files & datasets.

Use case of Hadoop:-

→ click stream data helps you to understand the purchasing behaviour of customers.

→ click stream analysis helps online marketers to optimize their product web pages, promotional content, etc, to improve their business.

click stream data analysis using Hadoop - key benefits		
Join clickstream data with CRM & Sales data.	Stores years of data without much incremental cost.	Hive (or) Pig Script to analyze data.

Fig:- click stream data Analysis.

Key benefits :-

→ Click stream data analysis using Hadoop.

1) It helps to join clickstream data with other data sources such as Customer Relationship Management data. (customer demographics data, sales data, & Information on Advertising Campaigns). This additional data often provides much information to understand customer behaviour.

2) It's Scalability property helps you to store years of data without much incremental cost. This helps you to perform temporal (or) year over year analysis on click stream data which your competitors may miss.

3) Business analysts can use "Hive" or "pig" for website analysis. With these tools, you can organize Click stream data by user session, refine it, and feed it to visualization or analytics tools.

Hadoop Distributed File System (HDFS) :-

Key Points of HDFS :-

1) Storage Component
2) Distributed File System
3) Modeled after google file system
4) Optimized for high throughput
5) You can replicate a file for a configured number of times, which is tolerant in terms of both s/w & h/w.
6) Re-Replicate data blocks automatically on nodes that have failed.
7) You can realize the power of HDFS when you perform read (or) write on large files.
8) Sits on top of native file system such as ext3 & ext4.

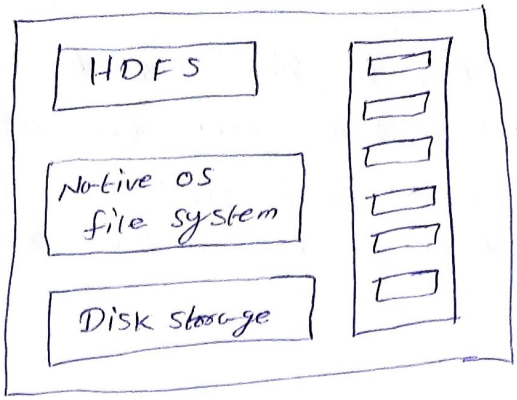


Fig:- Hadoop Distributed file system.

Hadoop distributed file system - key points		
Block structured file	Default Replication factor : 3	Default Block size : 64 MB

Fig:- HDFS key points

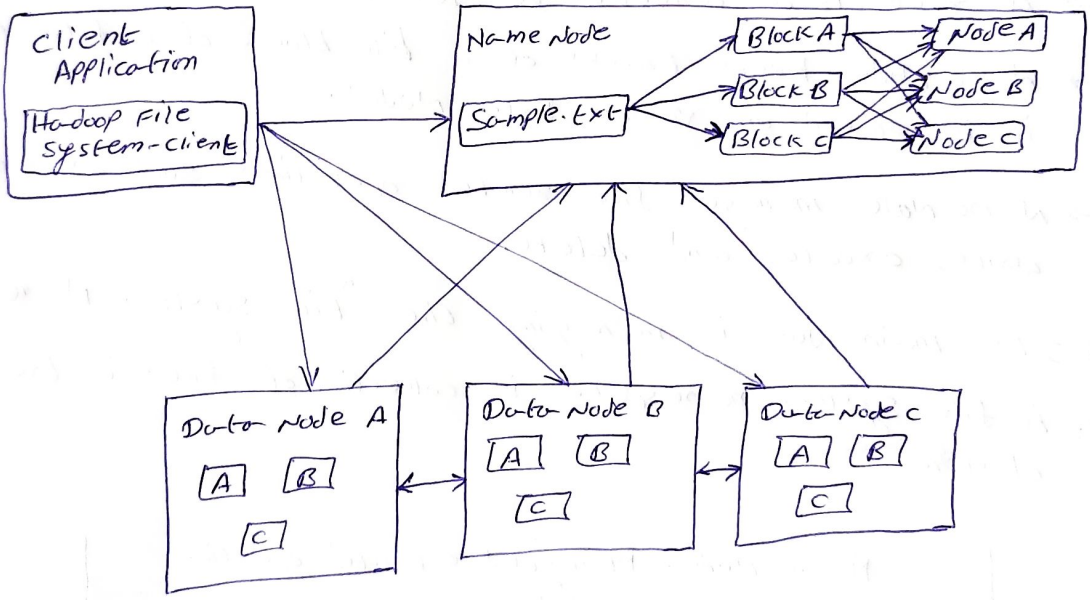


Fig:- HDFS Architecture

→ Client Application interacts with Name Node for meta data related Activities.

→ Client Application communicates with Data Node to read & write files.

→ Data Node converse with each other for pipeline reads & writes.

For example:-

Let us assume that the file "Sample.txt" is size of "192 MB". As per the default Block size "(64 MB)", it will be split into three blocks and replicated across the nodes on the cluster based on the default replication factor.

HDFS Daemons:-

→ "Daemon", A process which runs in the background & is not interactive.

Name Node:-

→ HDFS breaks a large file into smaller pices called blocks.

→ Name Node uses "rack ID" to identify "Data Nodes" in the rack.

→ A rack is a collection of Data Nodes within the cluster.

→ Name Node keeps tracks of a ~~file~~ block of a file as it is placed on various Data Nodes.

→ Name Node manages file related operations such as read, write, create, and delete.

→ Its main job is managing the "File System Name Space".

→ A file system namespace is collection of files in the cluster.

Name Node – Manages file Related operations	
FSImage - File, in which entire file system is stored	Edit log - Records every transaction that occurs to file system meta data.

Fig:- Name Node

→ Name Node ~~uses~~ stores "HDFS name space".

→ File system name space includes mapping of blocks to file, file properties and is stored in a file called "FS Image".

→ Name Node uses a EditLog to record every transaction that happens to the file system meta data.

→ Name Node starts up, it reads FSImage and EditLog from disk & applies all transactions from the EditLog to in-memory representation of the FSImage.

→ Then it flushes out new version of FSImage on disk & truncates the old EditLog because the changes are updated in the "FSImage".

Data Node:-

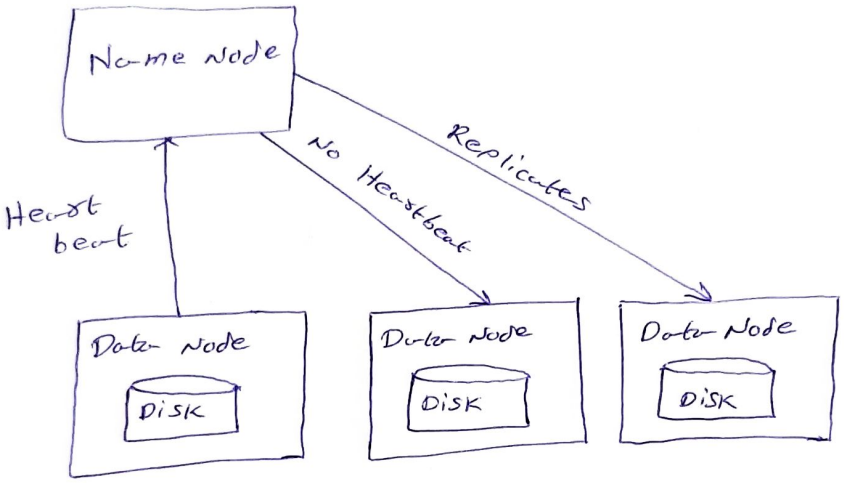


Fig:- Communication with Name Node & Data Node.

→ There are multiple Data Node per cluster.

→ During Pipeline read & write Data Nodes communicate with each other.

→ A Data Node also continuously sends "heartbeat" message to Name Node to ensure connectivity b/w the Name Node & Data Node.

→ In case there is no "heartbeat" message from data Node, the Name Node replicates that data node within the cluster & keeps on running as if nothing had happened.

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 & Secondary NameNode on different machines.

→ In case of failure, of the Name Node, the Secondary Name Node can be configured manually to bring up the cluster.

→ the secondary Name Node does not record any real-time changes that happens to the HDFS metadata.

Anatomy of file Read:-

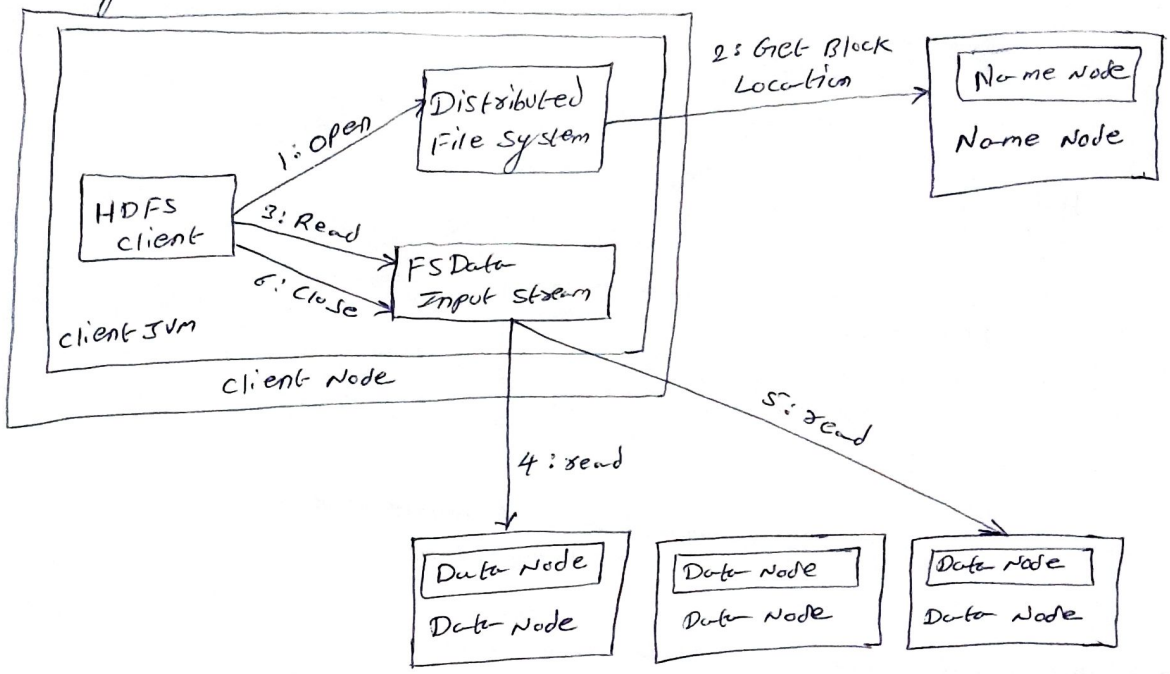


Fig: File Read

1) Client opens the file to read from by "calling open()" on the Distributed file system.

2) Distributed file system communicates with the NameNode to get the location of data blocks, Name Node return with the address of data nodes that the data blocks are stored on. Subsequent to this, the "DFS" returns an FSDataInputStream to client to read from the file.

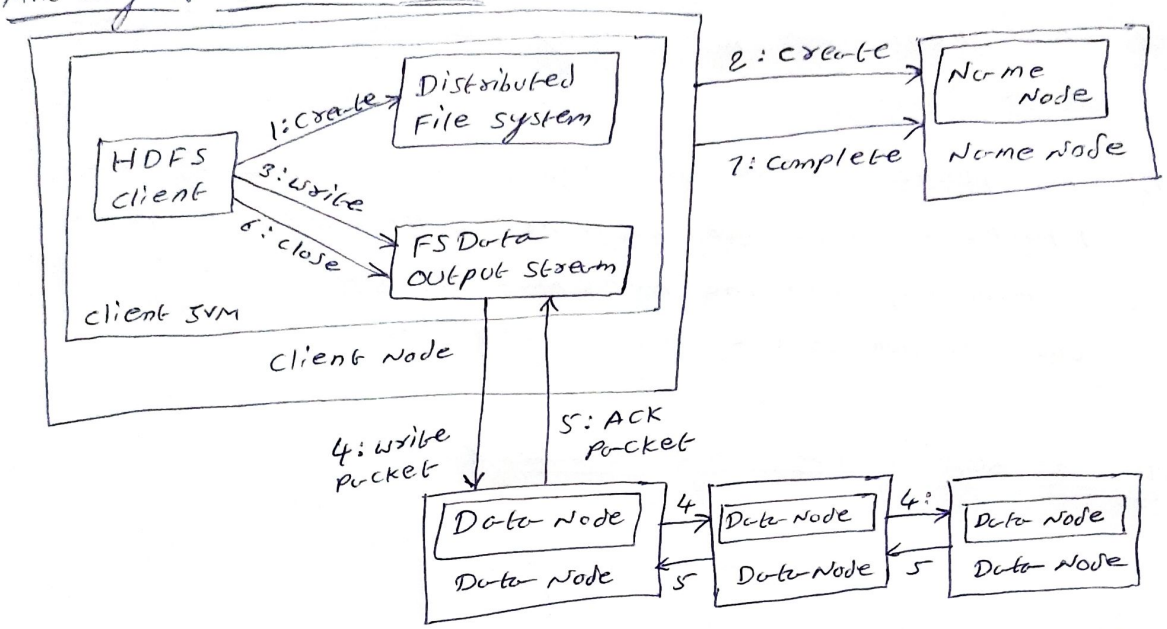
3) client then calls read() on the stream DFSInputstream, which has address of the Data Node for the first few blocks of the file, connects to the closest Data Node for the first block in the file.

4) client calls read() repeatedly to stream the data from the Data Node

5) When end of the block is reached, DFSInputStream closes the connection with the Data Node. It repeats the steps to find the best Data Node for the next block & subsequent blocks.

6) The client completes the reading of the file, it calls close() on the FSDataInputStream to close the connection.

Anatomy of file write():-



1) The client calls create() on Distributed file system.
2) An Remote Procedure calls (RPC) to the Name Node happens through the Distributed File system to create a new file. The Name Node performs various checks to create a new file. Initially, the Name Node creates a file without associating any data block to the file. The DFS returns an FSData-OutputStream to the client to perform write.
3) The client writes data, data is splits into packets by DFS OutputStream, which is then written to an internal queue called data queue. Data Streamer consumes the data queue. The data streamer request the name node to allocate new block by selecting a list of suitable Data nodes to store replicas. The list of Data Nodes makes a pipeline.

4) DataStreamer streams the packets to the first DataNode in the pipeline. It stores packet & forwards it to the second data node in the pipeline. In the same way, the second data node stores the packet & forwards it to the third DataNode in the pipeline.

5) 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 acknowledge by all the datanodes in the pipeline.

6) The client finishes writing the file, it calls closes on the stream.

7) This flushes all the remaining packets to the DataNode pipeline & waits for relevant acknowledgment before communicating with the NameNode to inform the client that the creation of the file is complete.

Hadoop Replica Placement Strategy:-

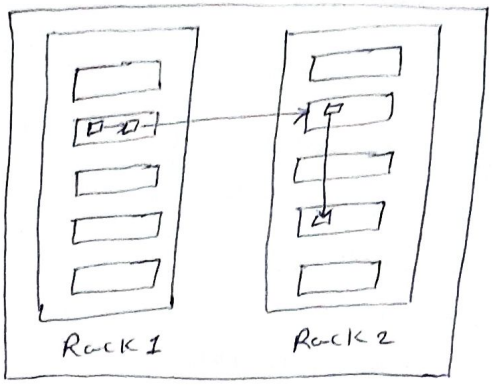


Fig:- Replica placement strategy

→ 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 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.

Working with HDFS Commands:-

→ To get the list of directories & files at the root of HDFS

```
hadoop fs -ls /
```

→ To get the list of complete directories and files of HDFS

```
hadoop fs -ls -R /
```

→ To create a directory in HDFS (Sample)

```
hadoop fs -mkdir /Sample
```

→ To display the contents of an HDFS file on console

```
hadoop fs -cat /Sample/file.txt
```

→ To remove a directory from HDFS

```
hadoop fs -rm -r /Sample
```

→ To copy a file from one directory to another on HDFS

```
hadoop fs -cp /Sample/file.txt /Sample1
```

→ To copy a file from local file system to HDFS

```
hadoop fs -put /root/sample/test.txt /sample/test.txt
```

→ To copy a file from HDFS to local file system

```
hadoop fs -get /Sample/test.txt /root/sample/test.txt
```

Special features of HDFS:-

⇒ Data Replication is absolutely no need for a client application to track all blocks. It directs the client to the nearest replica to ensure high performance.

→ Data pipeline:- client application writes a block to the first DataNode in the pipeline. Then this datanode takes over & forwards the data to the next node in the pipeline. This process continues for all the data blocks, & subsequently all the replicas are written to the disk.

Processing Data with Hadoop:-

→ Map Reduce programming is a software framework.

→ Map Reduce Programming helps you to process massive (or large amount of data in parallel.

→ In Map Reduce Programming, the input data-set is split into independent chunks.

→ "Map task" process these independent chunks completely in a parallel manner.

→ The output of the mappers are automatically shuffled and stored by the framework.

→ Map Reduce framework sorts the output based on keys.

→ This sorted output becomes the input to the "reduce tasks"

→ Reduce task provides reduced output by combining the output of the various mappers. Job input and output are stored in a file system.

→ Map Reduce framework also takes care of other tasks such as scheduling, monitoring, re-executing failed tasks etc.

→ HDFS & Map Reduce framework run on the same set of nodes. This configuration allows effective scheduling of tasks on the nodes where data is present (Data Locality).

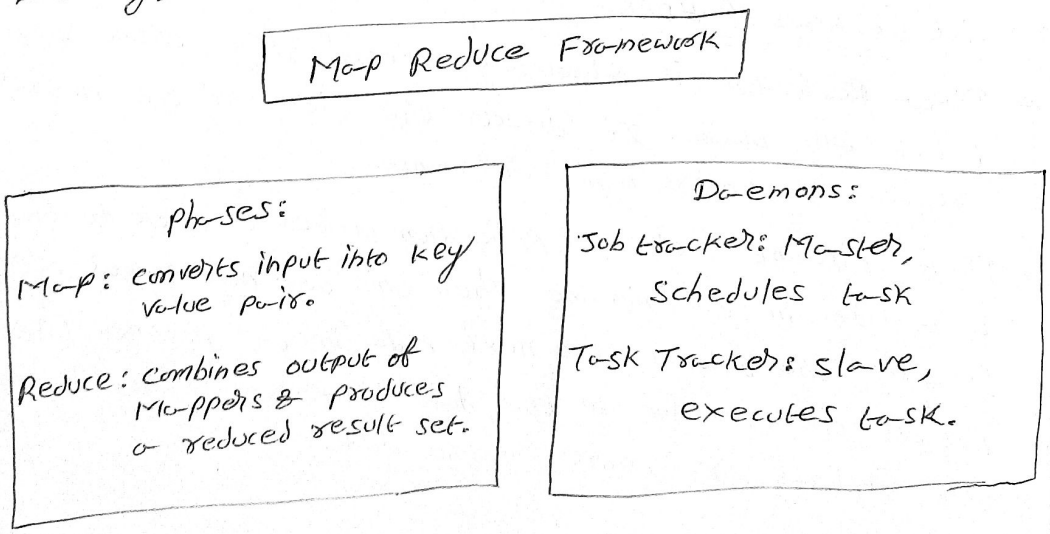


Fig: Map Reduce programming phases & daemons

→ There are two daemons associated with MapReduce Programming.

1) A single master "Job tracker" per cluster & one slave "Task tracker" per cluster node.

2) The "Job tracker" is responsible for scheduling tasks to the "Task trackers", monitoring the task, & re-executing the task just in case the "Task Tracker" fails.

Map Reduce Daemons:-

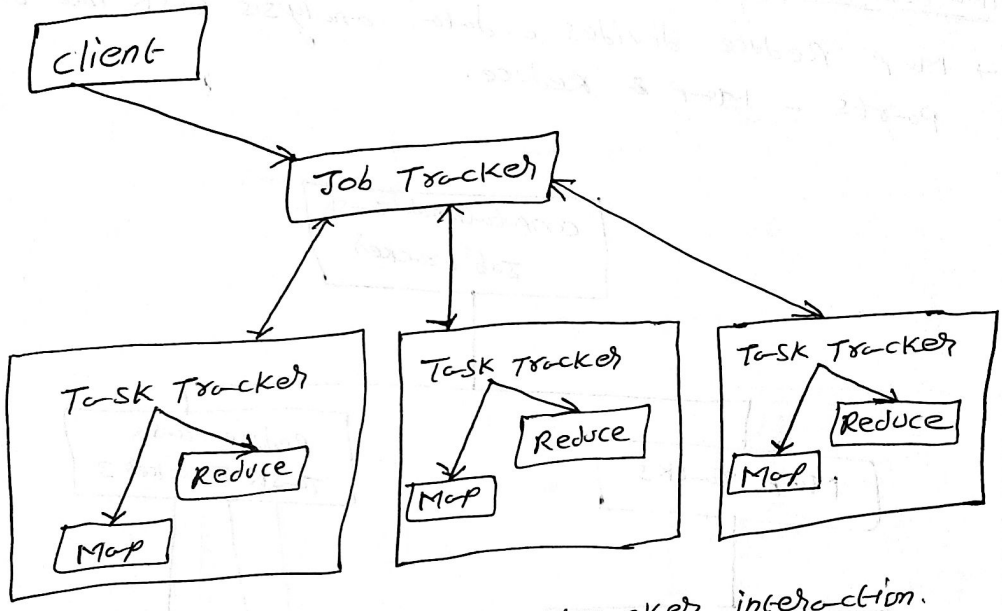


Fig:- Job tracker & Task tracker interaction.

→ once client submits a job to the Job tracker, it partitions & assigns diverse Map Reduce tasks for each task tracker in the cluster.

→ "Job tracker":- provides connectivity b/w Hadoop & application. When you submit code to cluster, Job tracker creates the execution plan by deciding which task to assign to which node. It also monitors all the running tasks. When a task fails, automatically re-schedules the task to a different node after a predefined number of retries. Job Tracker is a master daemon responsible for executing overall Map Reduce Job.

→ Task Tracker :- responsible for executing individual task that is assigned by Job tracker. There is a single Task Tracker per slave and spawns multiple (JVMs) to handle multiple map or reduce tasks in parallel. "Task Tracker" continuously sends "heart beat" message to Job tracker. When the Job tracker has failed to receive "heart beat" message from a Task Tracker, the Job tracker assumes that the task tracker has failed & resubmits the task to another available node in the cluster.

How does map Reduce work?:-

→ Map Reduce divides a data analysis task into two parts - Map & Reduce.

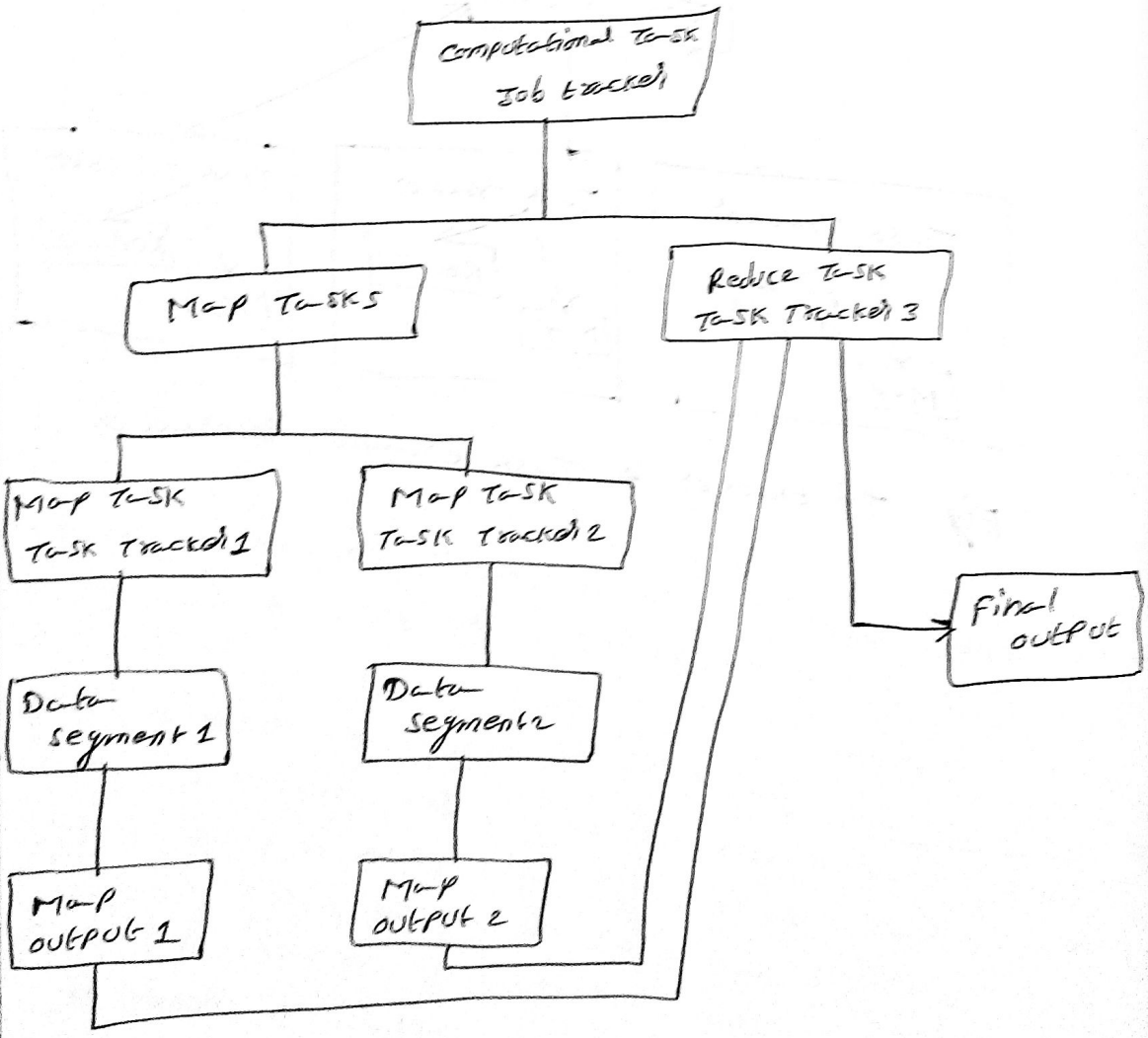


Fig:- Map Reduce programming flow

Example:- Mapper

There are two mappers and one reducer. Each mapper works on the partial dataset that is stored on that node & the reducer combines the output from the mappers to produce the reduced result set.

Working model of MapReduce Programming:-

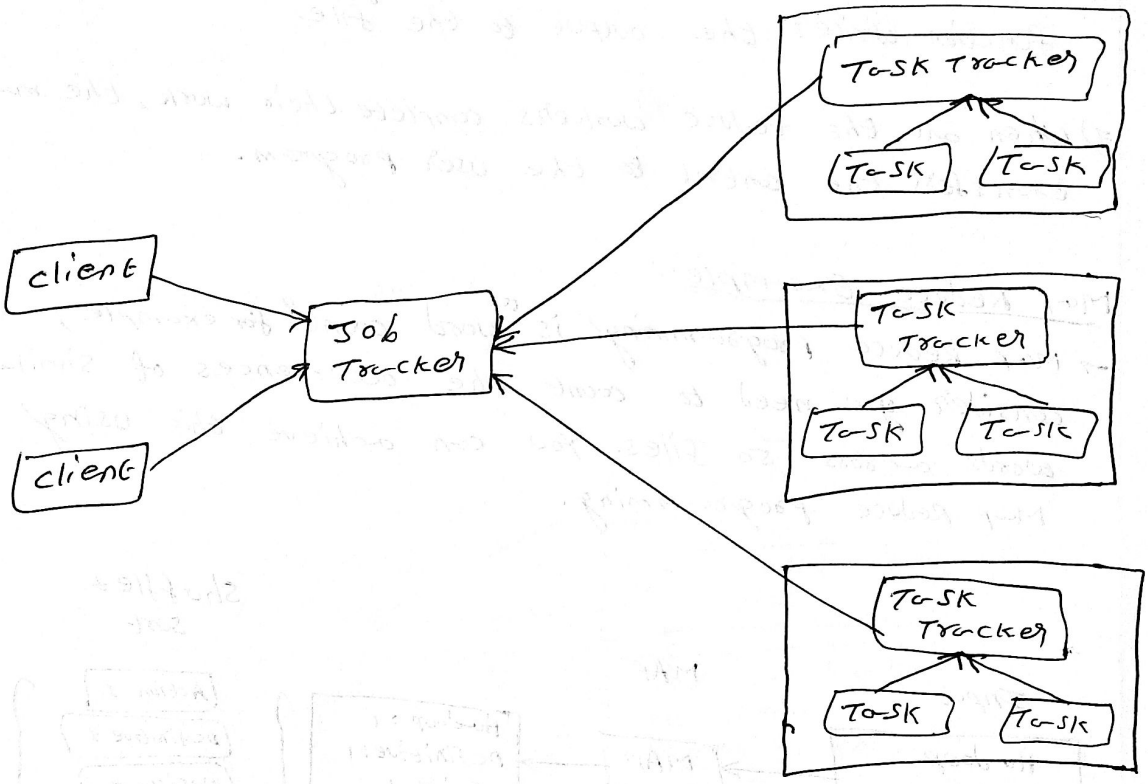


Fig:- MapReduce programming architecture

1) The input dataset is split into multiple pieces of data. (small subset).

2) The framework creates a master & several workers processes and executes the worker processes remotely.

3) Several map tasks work simultaneously and read pieces of data that are assigned to each map task. The map worker uses the map function to extract only those data that are present on their server & generates key/value pair for the extracted data.

4) Map worker uses partitioner function to divide the data into regions. partitioner decides which reducer should get the output of the specified mapper.

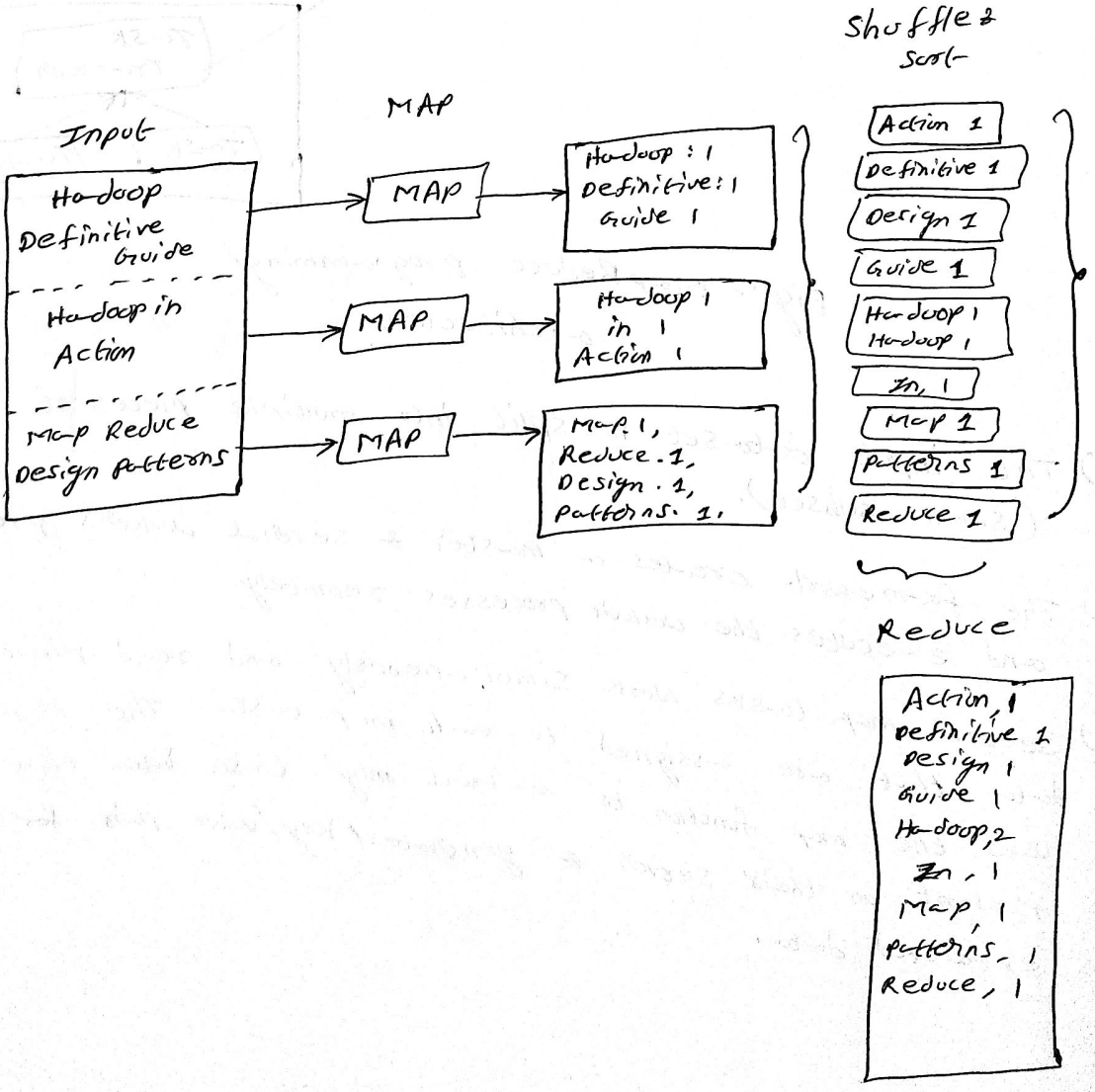
5) When the map workers complete their work, the master instructs the reduce workers to begin their work. The reduce workers in turn contact the map workers to get the key/value data for their partition.

6) Then it calls reduce function for every unique key. This function writes the output to the file.

7) When all the reduce workers complete their work, the master transfers the control to the user program.

Map Reduce Example:-

→ Map Reduce programming is "word count" for example, consider you need to count the occurrences of similar words across 50 files. You can achieve this using Map Reduce programming.



NOSQL:-

→ NoSQL stands for Not only SQL
→ NoSQL is developed in the year 1998 by Carlo Strozzi.
→ NoSQL is a light-weight, open-source, non-relational database.

Features of NoSQL database are follows:-

1) They are open-source
2) They are non-relational
3) They are distributed
4) They are schema-less
5) They are cluster-friendly
6) They are born out of 21st century web application.

where is it used:-

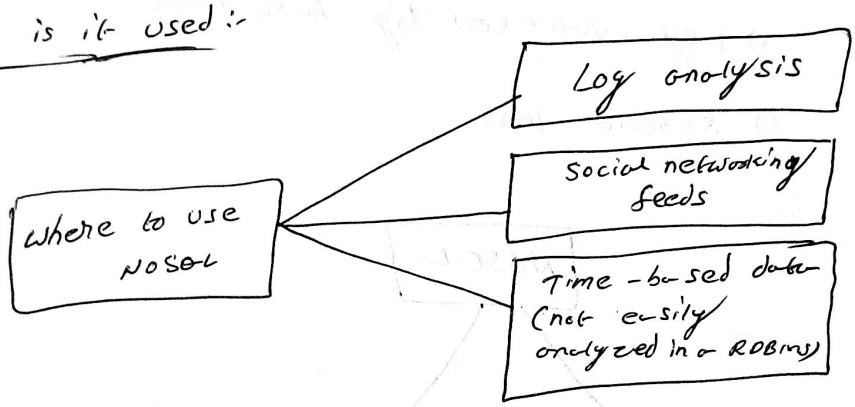


Fig:- where to use NoSQL

→ NoSQL databases are widely used in big data & other real-time web applications.
→ NoSQL databases is used to stock log data which can be pulled for analysis.

What is it?:-

→ NoSQL stands for Not only SQL.
→ They are hugely popular today owing to their ability to scale out or scale horizontally & the adeptness at dealing with a rich variety of data: structured, semi-structured & unstructured data.

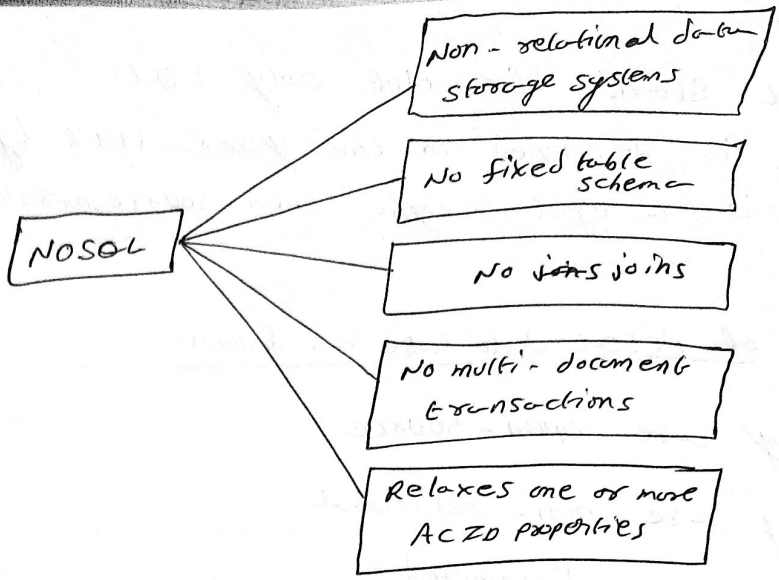


Fig:- NOSQL : what it is

Types of NOSQL databases:-

→ NOSQL databases are non-relational.

→ They are broadly classified into the following:-

1) key-value (or) big hash table

2) schema-less.

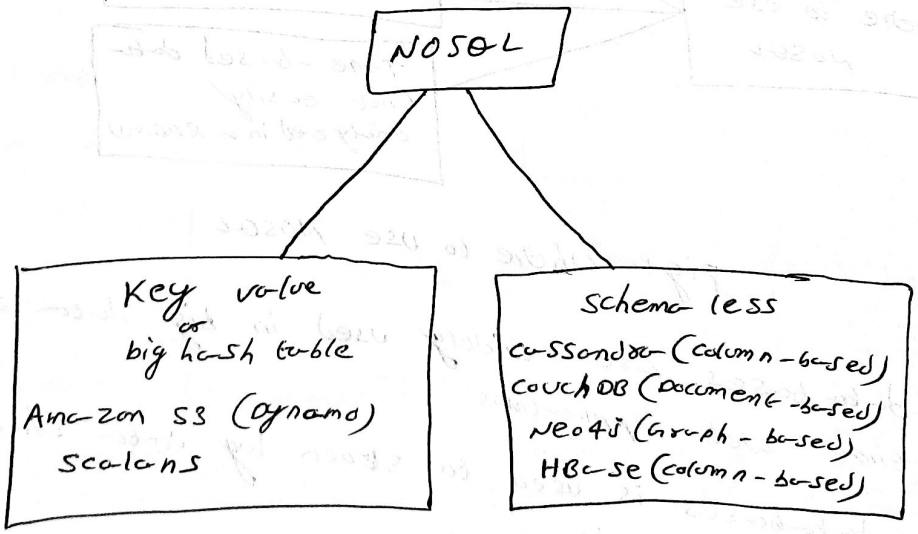


Fig:- Types of NOSQL databases

1) Key-value:-

It maintains a big hash table of key & values.

For example:- Dynamo, Redis, etc.,

Sample key-value pair in key-value Databases

Key	Value
First Name	John
Last Name	Adam

2) Document :-
It maintains data in collections constituted of documents.
For example :- MongoDB, Couchbase, Apache couchdb etc,

```
{
  "BookName" : "Big data"
  "Year" : "2007"
  "publisher" : "wiley"
}
```

3) Column :-
Each storage block has data from only one column.
for example :- Cassandra, HBase, etc.

4) Graph :-
They are also called network database. A graph stores data in nodes.
For example :- Neo4J, Hyper Graph DB, etc.

Why NOSQL ?:-
→ It has scale out architecture instead of the monolithic architecture of relational databases.
→ It can house large volumes of structured, semi-structured, & unstructured data.
→ NOSQL database allows insertion of data without pre-defined schema.
→ It automatically spread data across an arbitrary number of servers.
→ It offers good support of data replication which in turns guarantees high availability, fault tolerance, & disaster recovery.

Advantages of NoSQL:-

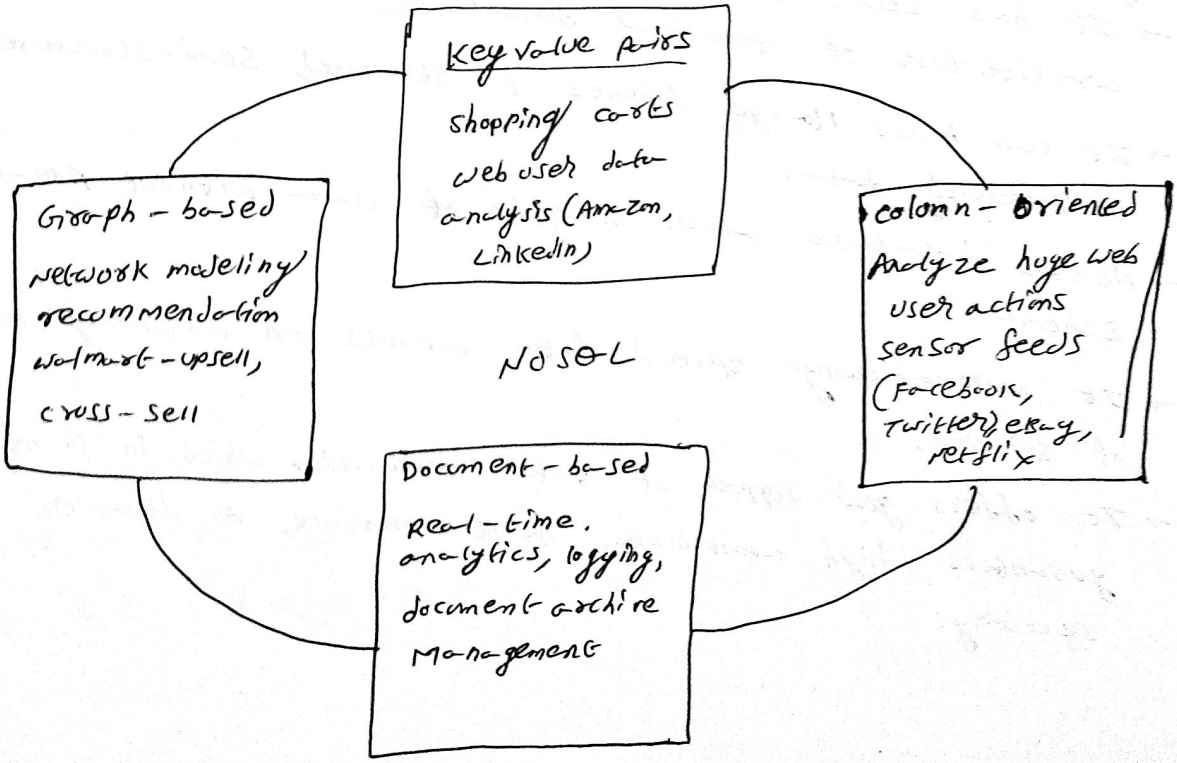
1) cheap, easy to implement
2) Easy to distribute
3) can easily scale up & down
4) Relaxes the data consistency requirement
5) Doesn't requires a pre-defined schema
6) Data can be replicated to multiple nodes & can be partitioned.

What we miss with NoSQL:-

1) NoSQL does not support joins.
2) It does not have provision for ACID properties of transactions.
3) NoSQL does not have a standard SQL interface but NoSQL databases such as MongoDB & ~~CouchDB~~ Cassandra have their own rich query Language to compensate for the lack of it.
4) one thing which is dearly missed is the easy integration with other applications that support SQL.

Use of NoSQL in industry:-

Fig:- Use of NoSQL in industry.



NOSQL vendors:-

company	product	Most widely used by
Amazon	DynamoDB	LinkedIn, Mozilla
Facebook	Cassandra	Netflix, Twitter, eBay
Google	Big Table	Adobe photoshop

SQL vs NOSQL :-

SQL:-

1) Relational database model
2) pre-defined schema, table based databases
3) vertically scalable (by increasing system resources)
4) Not preferred for large datasets, a best fit for hierarchical data.
5) Emphasis on ACID properties
6) Excellent support from vendors
7) Supports complex querying & data keeping needs, can be configured for strong consistency
8) Ex:- MySQL, oracle, postgreSQL, DB2, etc.

NOSQL:-

1) Non-relational, distributed database, & model-less approach
2) Dynamic schema for unstructured data
3) Document-based, graph-based, wide-column store & key-value pair datasets, & it's horizontally scalable.
4) uses unstructured query Language (UnQL), it's preferred for large datasets.
5) Best fit for hierarchical storage as it follows the key-value pair of storing data similar to JSON.
6) Follows Brewer's CAP theorem & relies heavily on community support.
7) doesn't have good support for complex querying.
8) Few support strong consistency (e.g:- Mongo DB), some others can be configured for eventual consistency (e.g. cassandra).
9) Ex:- Mongo DB, HBase, cassandra, Neo4J, couch DB, couchbase, etc.