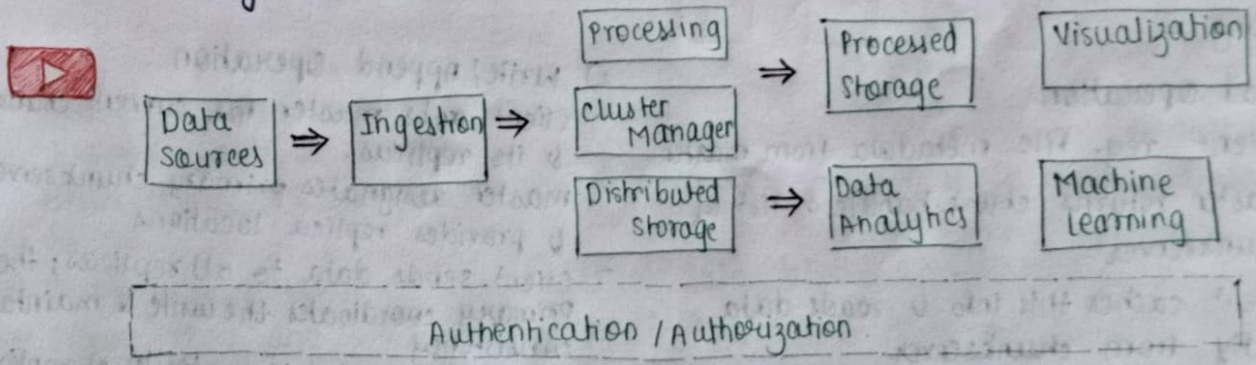
UNIT 3 - BIG DATA PROCESSING

> BIG DATA ECOSYSTEM

Refers to the collection of frameworks, tools, and technologies used to store, process, analyze and visualize massive volumes of data.
Enables organizations to derive insights from structured, semi-structured & unstructured data.

Key Components :—

1) Data Sources - Database, DW; XML & JSON files; Social media, Emails, Video data
2) Data Ingestion - consume the data from various sources (Kafka, Flume, Sqoop, NiFi [Apache])
3) Data Storage - Distributed File System (HDFS, GFS, Amazon S3)
4) Data Processing - Clean data (Apache Spark, A Storm, Hadoop Map Reduce)
5) Data Analysis - Take insights & analyze the data (Apache Hive, Apache Pig)
6) Machine Learning & Advanced Analytics - (MLlib, Apache Mahout)
7) Processed Data Storage - (Cassandra, MongoDB, HBase) storing NoSQL database
8) Visualization - Tableau, PowerBI, QlikView
9) Cluster Management - Allocates resources, scheduling Jobs, etc (YARN)



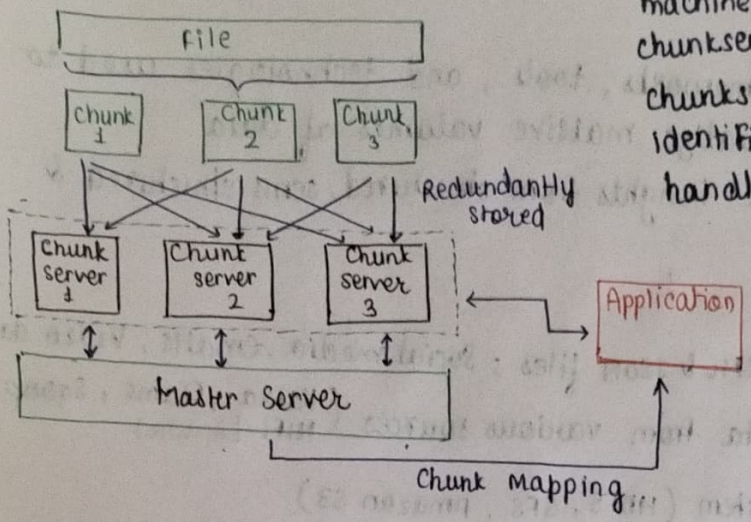
> INTRODUCTION TO GOOGLE FILE SYSTEM (GFS)

- GFS is scalable distributed file system for large distributed data-intensive appl"
- GFS is designed to handle batch workloads with lots of data.
- The system is distributed : multiple machine store copies of every file, and multiple machines try to read/write the same file.
- GFS was originally designed for Google's use case of searching & indexing the web. It was developed around 2003, new version released in 2010 (GFS2)

Characteristics :—

1) Fault tolerant - even if disk is corrupted, data in them can be used & restored.
2) Designed for Large files - Big data size.
3) Master-Slave Architecture - Uses single master to manage metadata & chunk servers to store data. Each file is divided into fixed size chunks (64 MB)
4) Write Once read many workloads
5) Internal usage - GFS only used within Google Infrastructure; It is not open-source or available for public use.

Architecture :-



The system consist of a sigle master machine and many chunkservers. These chunkservers store parts of files called chunks (64MB data blocks), which are identified ~~using~~ by a 64 bit chunk handler.

Main components :-

1) Master server -
- maintains all file systems metadata
- does not handle actual data; only metadata (file namespace, access control, file mapping, chunk location) and coordination.

2) Chunk server -
- stores actual file data as fixed chunks (64MB by default).
- Each chunk is replicated (default: 3) across different chunkserver for fault tolerance.
- Serve read & write req. from client directly.
- Send & updates status to the master server.

3) Client/Application -
- Appln or users that access GFS.
- Interacts with the master server to obtain metadata & chunk location
- Communicates directly with chunk-servers for actual data transfer, minimizing the master's load.

Working :-

1) Read Operation
- Client req. file metadata from master
- Master returns chunk handle & list of chunkserver.
- Client caches this info & reads data directly from chunkserver.

2) Write/Append Operation
- Client asks master for current chunk & its replicas.
- Master designates primary chunkserver & provides replica locations.
- Client sends data to all replicas; the primary coordinates the write & maintains consistency
- Write is successful if majority of replicas acknowledges

❤ Heartbeat Mechanism - Master regularly checks chunkserver health.

Advantages :-
1) scalable 2) Fault Tolerance 3) Cost Effective 4) Efficient for large files
5) Distributed Read/Write

Disadvantages :-
1) single point failure (Master) 2) Internal fragmentation (lazy space allocation)

> HADOOP ARCITECTURE

What is Hadoop? Apache Hadoop is an open source software library that is used to manage data processing & storage in big data applications. Hadoop facilitates analyzing large amount of data parallely & more quickly. Apache Hadop was introduced in 2012 by ASF ie Apache Software foundation.

Core Components :- 1) HDFS 2) YARN 3) MapReduce 4) Hadoop Common.

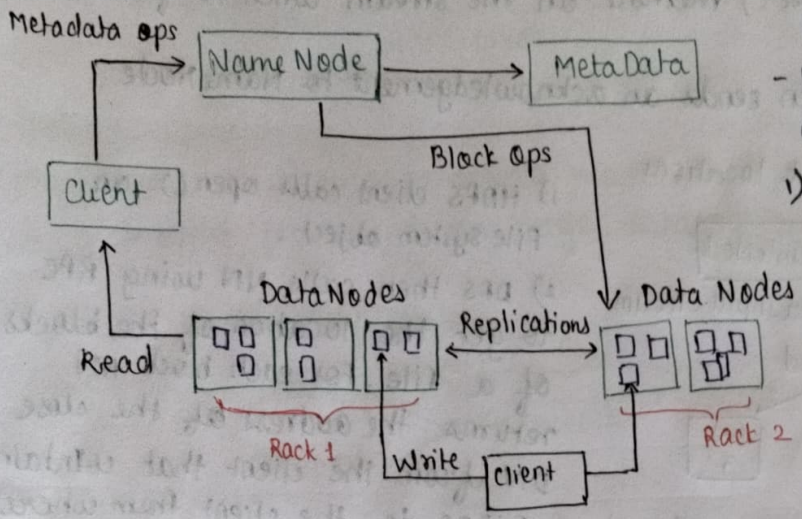
- YARN (Yet Another Resource Negotiator) - responsible for Resource mangement and job scheduling in HDFS.
- Hadoop Common - Java libraries & utilities used across modules.

> HADOOP STORAGE

HDFS (Hadoop Distributed file System)

HDFS is a distributed file system for storing & retriving large files with streaming data in record file.

Architecture :-



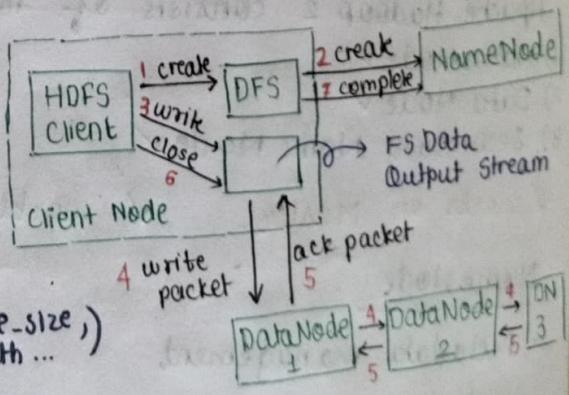
- It is a core component of Hadoop ecosystem.
- HDFS has Master-Slave architecture.

1) NameNode (Master)
   - Central server that manages the file system namespace and metadata.
   - Keeps metadata in memory for fast access.
   - Handles client request for file operations (open, close, rename, delete)
   - Monitors DataNode via heartbeats & block reports.
   - Ensures data block replication & manages recovery in case of DataNode failure.

2) Data Node (Slave) - stores actual data blocks on local disks.
   - Serve read and write req. from client as directed by the NameNode.
   - Report back to NameNode with block information & health status.

Files and Blocks - the file is the data which we want to store, when we store a file into HDFS it's broken to blocks, the default size of each one is 128/256 MB or 64 MB.
Each Block is replicated for providing fault tolerance & availability, the default number is 3.

ANATOMY OF FILE WRITE AND READ



1) HDFS client calls create() on DFS (Distributed file System) to create a file.
2) DFS interacts with Namenode through RPC (Remote Procedure Call) to create a new file in filesystem namespace (with file_size, dest_path ...) associated with it.

3) The Namenode makes sure that the file doesn't already exist & client has the right permissions to create file. If all these checks pass, the Namenode makes a record of the new file. The DFS returns an FSDataOutputStream for client to start writing data to Datanode.

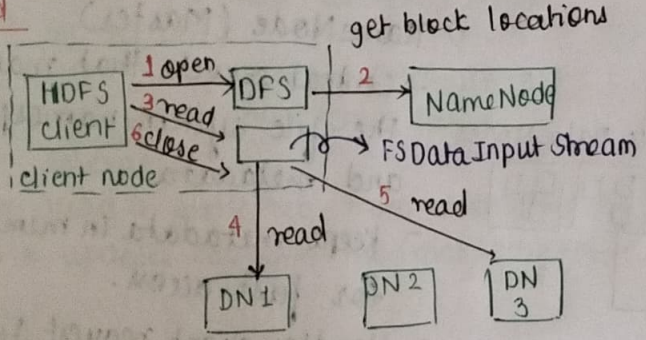
4) As client starts writing data, the DFSOutputStream (wrapped in FSDOS) splits the client's data into packets & writes it to an internal queue called data queue. This dataqueue is used by DataStreamer, which tells NameNode to allocate new blocks. The NameNode returns a list of suitable DataNodes for replication.

5) The DFSOutputStream also maintains another queue of packets, called ack queue, which is waiting for the acknowledgement from DataNodes.

6) The HDFS client calls the close() method on the stream when it finishes writing data.

7) The FSDataOutputStream then sends an acknowledgement to NameNode.

Read



1) HDFS client calls open() on File system object.

2) DFS then calls NN using RPC to get the location of the blocks of a file. For each block NN returns the address of the close DN from the client that contain a copy of that block. Then DFS returns an FSDOS to the client from where the client can read the data.

3) Then the client calls the read() method on FSDataInputStream object.

4) The DFSInputStream (wrapped in FSDIS) which contains the addresses of the blocks of the file, connects to the closest DN to read the first block in the file.

5) Upon reaching the end of the file, DFSIS closes the connection with that DataNode and finds the best suited DN for the next block.

6) When the client has finished reading the data, it calls close() on the FSDIS.

NameNode, Secondary NameNode, and DataNode.

Apache Hadoop 2 consists of foll^{n} Daemons [Daemons = Process]

1) NameNode ✓
2) Data Node ✓
3) Secondary Name Node ✓
4) Resource Manager ✓
5) Node Manager ✓

✓ works on Master ✓ works on Slave machine

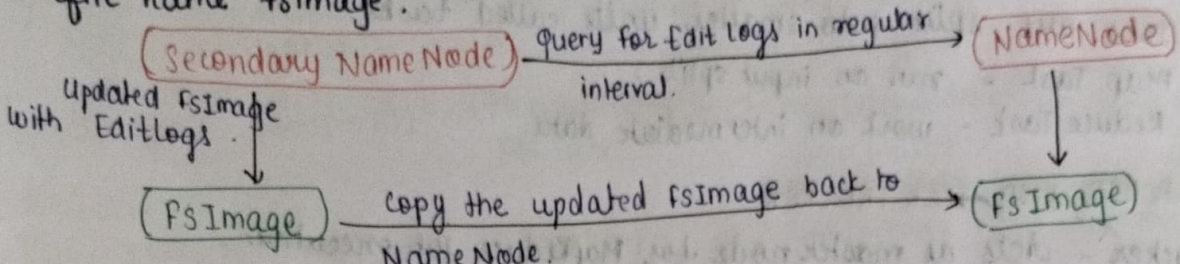
NameNode

- Metadata management

Start name node : hadoop-daemon.sh start namenode
Stop name node : hadoop-daemon.sh stop namenode

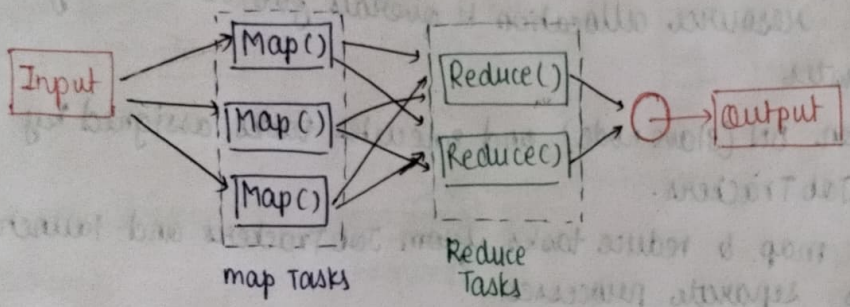
DataNode Start : hadoop-daemon.sh start datanode
Stop : hadoop-daemon.sh stop datanode.

Secondary Name Node - used for taking the hourly backup of the data. Incase, the Hadoop cluster fails, or crashes, the secondary NameNode will take hourly backup of data & store this data into a file name fsimage.



HADOOP MAPREDUCE PARADIGM

- MapReduce is a programming model that allows us to perform parallel & distributed processing on huge datasets.

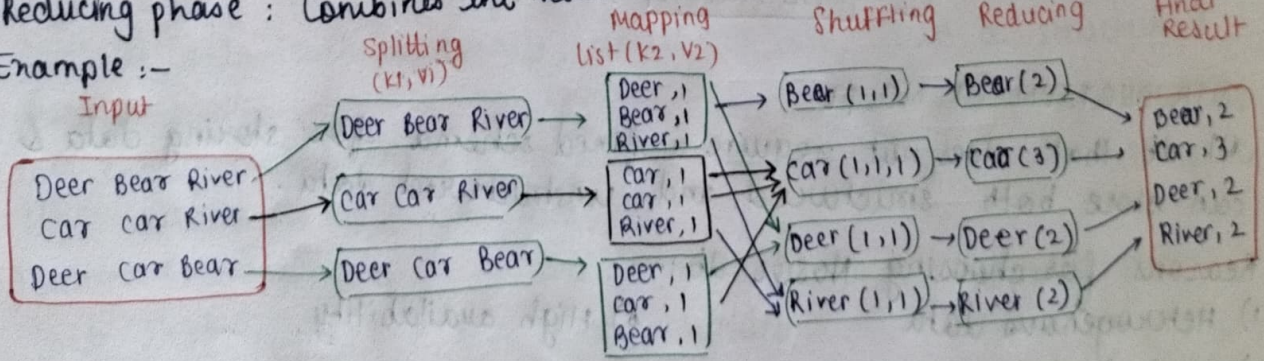


- MapReduce consist of 2 tasks -
  1) Map
  2) Reduce.
- Reduce phase takes place after the mapper phase has been completed.

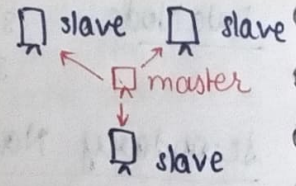
Mapper is responsible for splitting & mapping the data, while Reducer is responsible for shuffling and reducing the data.

1) Splitting phase : Input data is split into smaller chunks.
2) Mapping phase : Chunks are converted into <key, value> pairs with their occurence frequency. Here if a word occurs multiple times, it will be stored as single value.
3) Shuffling & Sorting phase : System performs sort & sends the output to the Reducer. sorting is done based on keys, not value.
4) Reducing phase : Combines the result & outputs, which is stored in HDFS.

Example :-



Advantages of Mapreduce :-



1) Parallel Processing :- Job is divided among multiple nodes
2) Data locality : Instead of moving data to the processing unit, we are moving the processing unit to the data.

MAP-REDUCE TASKS

Job - The entire Mapreduce appln, submitted by user.
Task - Jobs are split into smaller units called tasks.
Map Task - run on input splits.
Reduce Task - work on intermediate data.

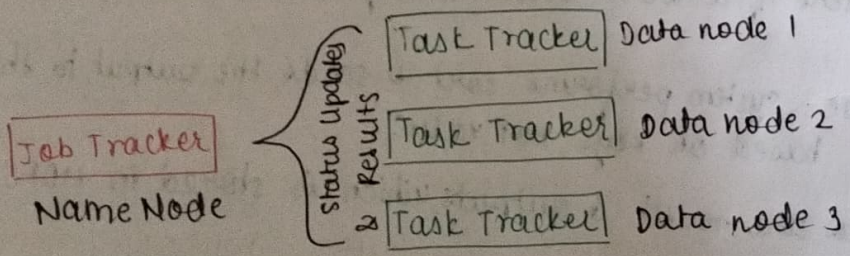
JOB TRACKER AND TASK TRACKER

1) Job Tracker - Acts as master node for Mapreduce job execution.

Responsibilities - Receive MapReduce jobs from client.
- Communicates with NN to determine location of data Blocks.
- Monitors the progress & status of all tasks through regular updates (heartbeats ♡) from Task Trackers.
- Manages resource allocation & overall job scheduling in the cluster

2) Task Tracker - Runs on DN (Slave node) and executes tasks assigned by the Job Trackers.

Responsibilities - Receives map & reduce tasks from Job Trackers and launches them as separate processes
- Monitors & manages execution of these tasks, capturing output and exit codes.
- Periodically sends heartbeat signals & status updates to Job Tracker to confirm it is alive and report task progress



> INTRODUCTION TO NOSQL

All db that does not require a fixed schema for storing data & can store both structured and unstructured data.

Reasons for choosing NoSQL db :-

1) Heterogenous data
2) High scalability
3) High performance
4) High availability
5) High functionality

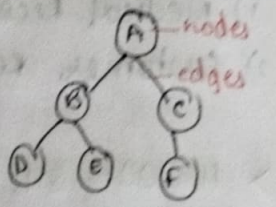
Types of NoSQL DB.

Features	Key-Value	Document	Column	Graph
Data format	key → value	JSON / BSON / XML	Rows with columns	Nodes + Edges
Schema	Schema less	schema flexible	Schema flexible	schema less
Use Case	Caching, lookup	CMS, e-commerce	Time-series, analytics	Social networks, fraud detection
Query language	Simple API	MongoDB Query, SQL-like	CQL (Cassandra), HQL (Hive)	Cypher (Neo4j)
Horizontal Scaling	Excellent	Excellent	Excellent	Good (depends)
Examples	Redis, Dynamo DB	MongoDB, Couch DB	Cassandra, HBase	Neo4j, Amazon Neptune

Key	value
Apple	Red, 100
Mango	yellow, 80

```
{
"ID" : '1',
"Product" : "Book"
}
```

Row Key	Name	
	F. Name	L. Name
1	Homi	Dhurmal



Comparision	Relational DB (Eg. DB2, MySQL, Oracle, SQL Server, Postgress)	NoSQL DB. (Eg. CouchDB, MongoDB, HBase, Cassandra)
1) Schema	Required	Not Required
2) Stores	Structured Data	Structured & Unstructured data
3) Scalability	low	High
4) complex Queries	Easy to write	Hard to write
5) Performance	low	High
6) Data Model	Tables (Row & Col.)	4 types
7) Data Size	Limited	Big Data.
8) Property	ACID	CAP (consistency, availability, partition tolerance

> TEXTUAL ETL PROCESSING

- Textual ETL is the process of transforming unstructured or semi-structured textual data - such as emails, documents, etc - into a structured format.
- This enables organizations to analyze the data using standard tools like relational db, Excel, and DW.
- Textual ETL enables organizations to unlock valuable insights from text that would otherwise remain hidden.

Why Needed ? To extract valuable insights from text.

Working :-

Extraction : Raw text is extracted from various sources such as docs, emails, logs and web pages.

Transform : Once extracted, the data goes through several transformation steps -
Data Cleaning, Data Validation, etc.

Loading : The final, structured data is then loaded into a target system like Relational DB, Data Lake or Data Warehouse (DW)

Benefit :-

1) Improved Data Integration
2) Enhance Data Analysis
3) Better Decision Making.

Examples :-

1) Medical Records
2) Customer Reviews
3) Social Media Posts
4) Emails
5) Web Page contents
6) Marketing & Ads

> HADOOP SHELL COMMANDS.