Big Data Technology notes

(For CCEE and Theory)

Prepared by : B.Dheeraj Chandan
Course : DBDA(Big Data Analytics)
Batch : March-August 2024
PRN number : 240350125017

LinkedIn : https://www.linkedin.com/in/balivada-dheeraj-chandan-022b91223/

Big Data Technologies

Big Data - Technology that handles large data sets, data storages.
- Skills : Data analysis, Programming(Python, R ,SQL)
- Tools : BigData tools(Hadoop, Spark),Data Visualization,ML and Statistics
- Sources : IOT, Transactional ,Web, govt..

Big Data adoption -
- Drivers (Adoption of high big data)
- Barriers (high cost, lack of skilled personnel ,data privacy)

Research and changing nature of data repositoriesn - Trend and Focus
- Trends : Advamcements in cloud computing , distributed data bases ,real time databse systems.
- Focus : High data storage, retrieval speed ensuring data integrity and security.

Hadoop versions - V1.0 / V2.0 / V3.0
- Version 1.0 : Introduced HDFS and MapReduce
  Limitation in terms of scalability and resource managament
- Version 2.0 : Introduce YARN (),Separates resource management from processing layer.
  Enhancing scalability and flexibility
  MapReduce + Graph processing , Interactive and Querying
- Version 3.0 : Supports erasure coding
  Improved storage efficiency,
  Addition of docker container ,
  Enhancement to HDFS, YARN, MapReduce

Hadoop & Components :-
- HDFS - High throughput access to application data.
  Splits files into large blocks & nodes, distribute across nodes
- YARN:- Schedules jobs and manages clusters resources,
  Allows multiple data processing engines to run on Single hadoop cluster
- MapReduce:- 2 main tasks Map (Processes & filter data) and Reduce (Pygogates & Sammarises)

Hadoop vs Traditional DB:-
RDBMS handle

DFS (Distributed File Systers) :-
Data stored accross multiple Servers & locations. High scalability.
Availability and reliability.
Distributed environment enabling efficient data processing using tools like MapReduce, YARN & Hadoop Components.

Components of HDFS: Name Node & Data Mode
Namenode : Master Node manages files system namespace, Regulates access to files
Datanode : Worker/Slave nodes that stores & retrieche block when they are told. Report backs to Name node with block that they are storing. Periodically.

HDFS daemons:-
Name Node - Manages metadata and file system namespace
It keeps on tracking.
Data Node - Daemons that mange storage on individual nodes.
Handles read and write requests from a client.
Secondary Namenode - Merging namespace image & edit logs to reduce load on Name Node.

HDFS follows master-slave architecture where master is referred to Name node and Slave referred to Data node.

Scaling & Rebalancing :
Scaling : Horizontally adds data nodes to cluster.
High storage capacity.
Enhances parallel procating cupabilities.

Rebalancing : Distribution of data Node beome inbalane due to date growth or node additions/removals [HDFS balancer & redistributes]

Replications: Replicates data blocks across multiple data nodes

Rack Awarmers :- It uses a rack awarness algo to improvedata readability and bandwidth utilization.

Data Pipline :- Client writes to Datanode in sequence, 1st data node and lastdata node

Nade failure Management : It handles node failures automatically. If data node fails the name node automatically initiates the replication of the block.

Zookeeper- ZooKeeper is used to manage and coordinate distributed processes, maintain configuration and state, and handle tasks like leader election and failure recovery.

HDFS architecture : Manages & Process data sets
Design principles: Replications, rack awarness, High availability
Features can build big data solutions.

Hadoop Operation Mode:-
Standalone Local Mode (HDFS on a single Machine)
Pseudo distributed (HDFS runs on single machine simulated distributed environment)
Fully Distributed (Runs on multiple machines separate nodes: Namenode, DataNode, Resouuce node and Secondary Name Node)HDFS and YARN daemons [Prodution environment]

Core-site.xml : Configuration Hadoop proposties
HDFS : Configuration for Data Node Nane Node
MapRed : MapReduce job ; history (location and framework)
Yarn : Configuration for YARN ; Resource Manager & Node Manager
Resource Manager : Manages resources & schedules jobs
Node manager : Manages individual nodes resources & monitors resource usage
Data Node : Stores actual data block
Name Node : Manages file system namespace and metadata
Secondary NameNode : Periodically merges mamespace image & edit logs.
Job Tracker(MRVI) : Manages job scheduling & task trauking.
Résource Manager: handles resource management
Application Master: Takes care of Job scheduling & task Monitoring
Task Tracker(MRVI)→ Executes individual tasks assigned by job tracker.
Resource Manger (YARN) : Manages resources & application
Node Manager (YARN) : Manages resource & containers on a single node
Application Manager (YARN) : Manages life cycle of a single application

DataFlow : Client requests a name node and then name node accepts and gives address of data node so that client can directly access Data node directly for read and write operation.

Scaling : HDFS is horizonatal scaling, High storage capacity.
More Data Nodes are added to cluster for high storage capacity.
Horizontal scalability: Adding date nodes to cluster & handle
Load Balancing :

NN federation :

Rebalancing in HDFS-
Ensures even distribution of data across Data Node.
HDFS balancer Tool: To redistribute accross DataNode via cmd
Threshold Based Balancing & Threshold parameter to determine When rebalancing is necessary.

Incremental Balancing: Rebdancing is performed incremenatally to minimize disruption to the system.
Optimal performance & resource utilization HDFS a robust solution

MapReduce for parallel processing

HDFS commands:-
Data split into blocks & Blokes are replicated 3 ,128 MB each. Data Nodel & ensures fault, Tolerance & high availability.

hadoop fs -chmod /path - To change permissions
hadoop dfsadmin -safemod - To enter into safe mode
hadoop dfsadmin -safenode - To leave
hadoop dfsadmin - To refresh the list of Data Nodes

NameNode : 9870 and ResourceMaanager : 8088 port numbers.

Hadoop MapReduce : Processing large datasets in a parallel across a datasets. Map - Process input data and generates key value pairs. Reduce - It aggregatesintermediate key-value pairs to produce final output.

Map Reduce Daemons :
Resource manager - manages resources
Node Manager - Monitors resources
Application Master - Manages execution of a single map reduce

Amatomy of a MapReduce Job Run
Job Submission - Client submit job to Resource Manager
Input splits - Splits into logical chunks possessed by separate Map tasks
Map phase - Proceses input & generates key-alue pairs
Shuffle and sort - Key value pairs shuffled and sorted by keys.
Reduce Phase - Process grouped key-value pairsto generate final output.
Output - final output written to HDFS or another storge System

Postioners & Combiners
Partioner - It determines which reducers a particular key-value pair will go to (to control data distribution)
Conbiner - Performs local aggregation of intermediate key value pairs that they sent to reducers.

Input Formats

Input splits & Records - Logical chunks to Map tasks.Generates key-value pairs.
Text input - Default input format . Each line treated as a record
Binary input - Sequence file input format to read binary key value pairs stored in hadoop sequence file
Multiple input - Allows input for different data sources in the same job

Output Formats :-
Text output - Key-value pairs written as text lines default output format
Binary Output - Sequence output format (Writing binaries key value pairs to hadoop sequence files.
Multiple output - Allows multiple data to multiple output from a single job

Distributed Cache -
Distributes read only data needed by MapReduce jobs to all nodes in the cluster.
add CacheFile(): Adds a file to a distributed cache
add Cache Archieve() : Adds an archive to the distilate cache

Hadoop MapReduce paradigsm - large scale data processing by dividing tasks into smaller subtasks .Distributed Cache performed by making read only data readily available to all nodes.

Hadoop DataTypes:
[6 Primitive] - Int, Long, Float, Double, Boolean, Text
[3 Compositée] - Array, Map ,Null

Java & MapReduce
Mapreduce written in Java.
Robust API for implementing Mapper, Reducer
Map Reduce components:
- Mapper : Define map function
- Reducer : Define reduce function,
- Drive class: configures job & specifies input output paths, Mapper & reducer etc

Map Only - No reductionor aggregate is needed. Reducer is omnited or set to null Reduce only
Reducer only - Preprocessing has already done only aggregation is needed

Combiner & Partitioner :
Combiner - Acts as mini Reducer to reduce data transfer
Partitioner - Controls the partitioning of intermediate map outputs
job.setCombinerClass(),
Job.setPartitioningClass()
Counters - Tracks the progress or count the events by .getCounter()

Schedulers :- Its task for job scheduling
FIFO scheduler - Jobs executed in order they arrieved
Fair scheduler - Jobs are allocated resources that all jobs get an equal share of resource over time

Capacity scheduler - To run a Hadoop jobs in a shared ,mult tenant cluster to ensure certain capacity guantee.

Custom Writables : To define custom data types like
Writable
Writable Comparable

Compression : Reduces amount of data transferred & stored by following methods
.setCompressOutput()
.setCompressorClass()

Complex Map Reduce Programming

Multiple MapReduce jobs can be chained together output of one job becomes input for next which is also called chaining job.

Different stages of processing having multiple mappers and reducers

Map reduce Streaming: Writing MapReduce jobs in any languagethat can read from standard input and standard output

Python & Map Reduce :- Python scripts for both Mapper & Reducer

MapReduce on image dataset

Image processing (resizing, format Conversion, feature Extraction) using Python Libraries : PIL or OpenCV

Implementation:

Mapper - Process each image ,performs nessary image processing tasks

Reducer - Aggregates results if needed.

Hadoop ETL

ETL (Extract Transform Load): Extracting dates from various sources, Transform it into suitable fomat similar storage system. Loading into Hadoop's HDFS

Exctration: Extract/Load data from files

Transform: Cleanup data or arrange data to standard or specific

Load : Transformed data into HDFS, HBASE or Hadoop compatible storage

ETL functions :-

Data Extraction:

Sqoop (data between Hadoop & RDBMS)

Flume (collects, aggregates-Streaming data into HDFS)

Kafka (data streams processed in real the on loaded in HDFS)

Data Transformation:

pig (Piglatin to handle complex data)

Hive (SQL like query for data transfom & analysis

Spark (API for complex date transforms using RDDs and Dataframe)

Data Loading:
HDFS (Storage for large datasets)
HBase (NoSQL DB runs on top of HDFS, read and write access)
Hive (Acts as dataware house storing data in a structured format)

Need for ETL Tools :- Complex data Management, Automation, Scalability, Integration

Advantages of ETL :- Efficiency, data Quality, Compliance, Flexibility ,Performance, User Friendly

HBase-
Open Source, non-relational, distributed database.
Written in Java
Handles data accross commodity servers.
Provides real time read/write access to big data
Features : column oriented storage, Scalability, Consistency, Integration with Hadoop, High availability , Horizontal scaling
Architecture :
Components -
HBase Master - Manages Cluster (Table crectiona deletion & balancing regions)
Region servers - Handles read/write requests. Manages HFiles, & WAL (Write ahead log):
Zoo Keeper - Coordinates distributed process & Provides Configuration & Synchronization services.

Data Model : Tables, Rows, columns, column families ,cells.
Storage : HFiles, WAL (Write-Ahead Log)
Region - Subset of table horizontally partitioned row
key range - Region Split fically splits it Automatically

Apache Hive -
Data warehousing and SQL-like query language tool built on top of Hadoop.
Allows users to query, summarize, and analyze large datasets stored in Hadoop's HDFS (Hadoop Distributed File System).
Converts SQL-like queries into MapReduce jobs that run on Hadoop, making it easier for users to work with big data without needing to write complex MapReduce code.

Key Features:

- SQL-like Interface: Uses HiveQL, a query language similar to SQL.
- Schema on Read: Data is interpreted during query execution,not when stored.
- Supports Large Datasets: Designed to handle and process petabytes of data.
- Extensibility: Supports user-defined functions (UDFs) for custom processing.
- Integration: Works with HDFS, HBase, and other Hadoop ecosystem tools.

Use Cases:

- Data Warehousing: For storing and querying large volumes of structured data.
- Data Analysis: For performing data summarization, querying, and analysis on big data.

Hive is widely used in big data analytics for its simplicity, scalability, and ability to integrate seamlessly with the Hadoop ecosystem.

Big Data Technology - Notes

By - B. Dheeraj Chandan
(2403501250017)

3/7/24

→ Bigdata helps to analyze, systematically extract information deals with large or complex data by data processing applications.
→ Data can be unstructured, semi structured, Quasi structured & Structured data.

Importance :- Cheaper storage, cloud based technologies, improvements in data processing techniques & analytics
penetration of Technology - IOT, smart devices
Ever increasing data, ↑ of unstructured data.
Unstructured data : Text document, PDF, Img, videos
Quasistructed structured data : clickstream
Semi Structured data : Spreadsheet XML ~~B.Dheeraj Chadan~~
Fully Structured data : Database

Industries :- Res, Aerospace, sports, Retail —

Characteristics of big data : Volume, Variety, velocity, Veracity
- volume → Quantity of generated & stored data.
- Variety → Type & nature of data
- Velocity → speed of data generation & processing time
- veracity → Defn. of big data which refers to data quality & the data value.

Big data ∈ [Data size, type of data, speed of data processing, data quality & data value]

Challenges :
→ Scalability, timeline & cost challenge before making a move to big data.
→ Not all data is useful (data contains more noise)
→ Lack of technical talent
→ Data Security cornerstone of all IT infrastructures.
→ lot of organisation and human resistance when it come to making a transition of bigdata.

Scalability : (able to handle the big/large amount of data in the system) Data usability should be reusable.
Lack of technical talent : automated management is ... Organisation is investing in ramping up technology skills of workforce.

Data Security : Security systems need to be used by Hadoop admins.

Platforms : Cloudera, AWS, Google, Azure, Digital Ocean, MapR, IBM, Intel

↳ For Hadoop installation we have to do many configurations.

What is Hadoop :-

↳ open source software framework for storing large amount of data & running appl^n.. parallelly on clusters of commodity hardware.

- Ability to store large amount of data and type & process it quickly (Computing power - Capable of processing data very fast).

Fault Tolerance, Flexibility, Low cost, scalability

FT : Data & appl^n. processing are protected against hardware failure.

Flexibility : Like RDBMS no need to preprocess data before saving. it stores as much of data & decide how to use it later. ex:- It can store unstructured data like text, img, videos.

- Low Cost : Free opensource framework, uses commodity hardware to store large amount of data
- Scalability : Handling large amount of data by adding more nodes.

↳ Created by Doug Cutting

Evolution :-

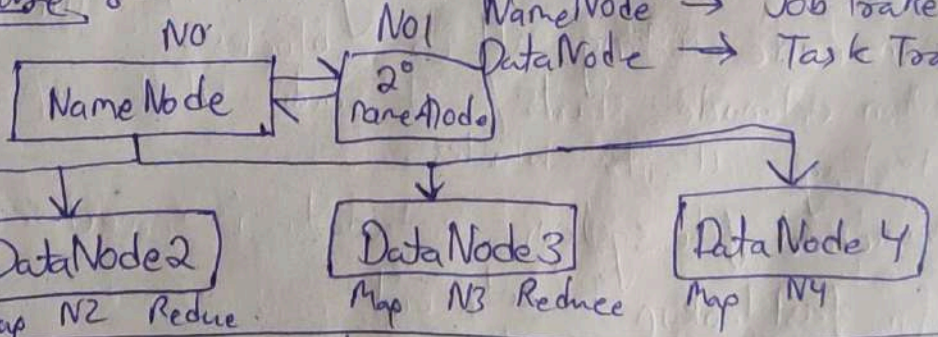
Hadoop evolution → Hadoop1, Hadoop2, Hadoop3

Hadoop1 - Highly distributed, fault tolerance storage, data local parallel processing

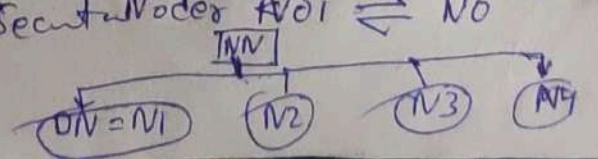
Hadoo2 - Allowing better fault tolerance, ↑ ecosystem now allows more specialized applications & in memory processing

Hadoop3 - It has enhancements around fault tolerance, optimization etc...

Hadoop Architecture :-

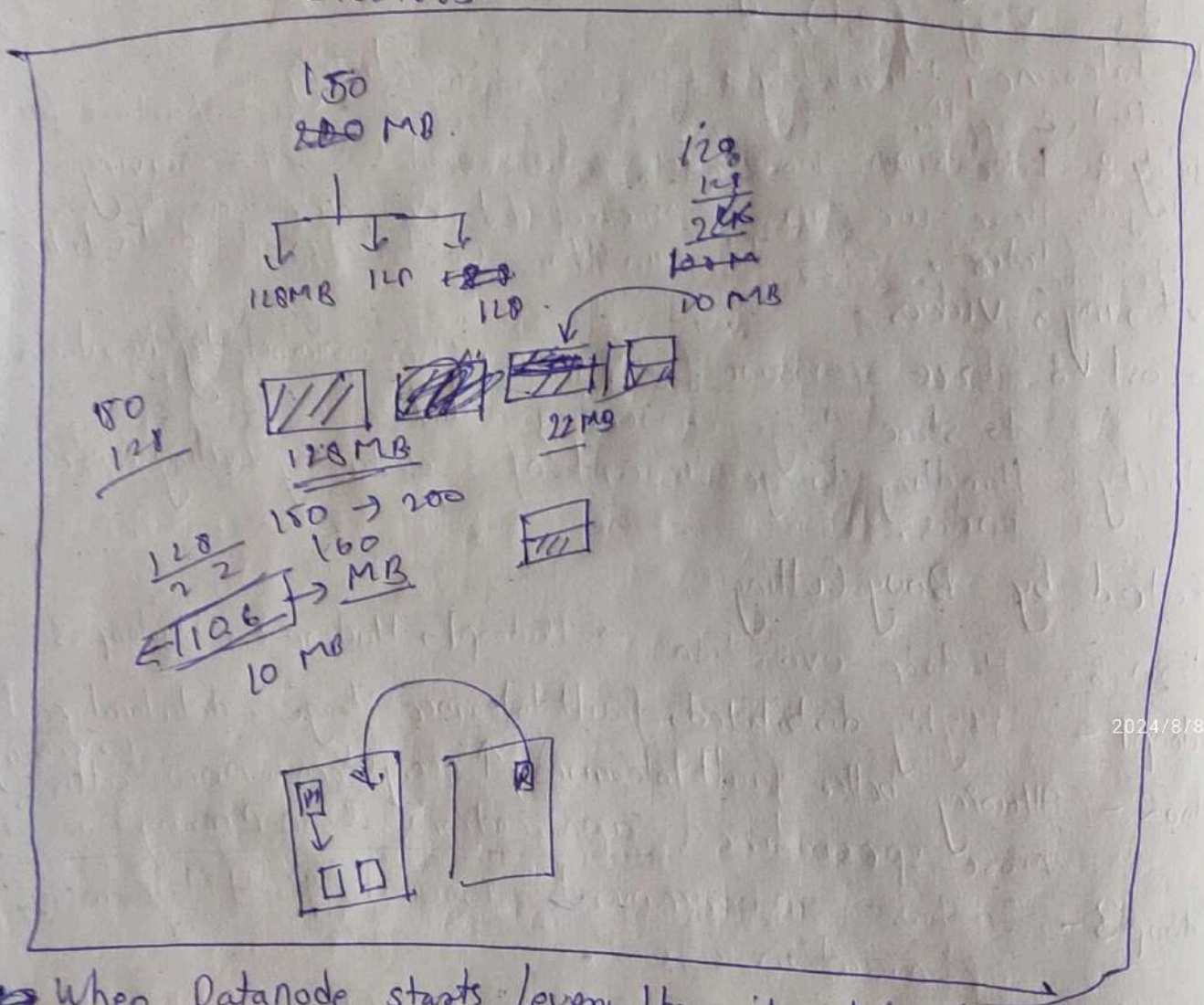


N0 NameNode → Job Tracker
N1 DataNode → Task Tracker
N2 N3 N4

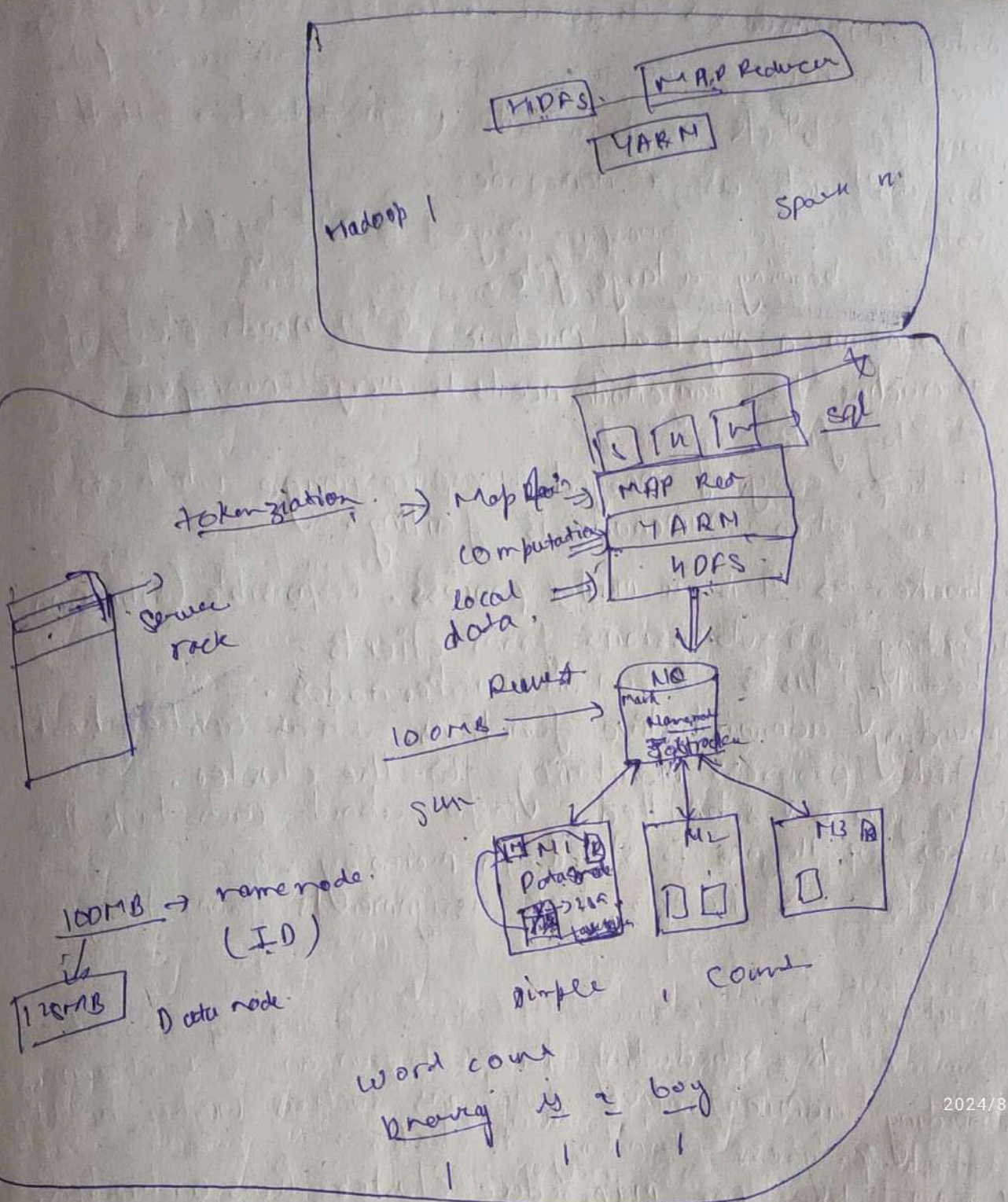


Architecture components: Namenode, datanode, secondary namenode
Job Tracker, Task Tracker

Namenode — Filesystem + Metadata, info stored in local disk & saved in 2 forms: namespace image & edit log.
It knows all datanodes. Info is reconstructed from data node when the system starts. Cli
client connects namenode to perform filesystem operations.
Datanodes always reports their status to the namenode so that namenode has full view of all datnodes in a clusters.



- When Datanode starts / every 1hr it send block report to namenode, listing all its blocks.
- When datanode starts / it sends block reports to namenode, listing all its blocks.
- Namenod file system metadata is served entirely from RAM for fast lookup & retrieval



2024/8/8 18:41

Datanode — Workhorses of file system, store & retrieve data from block when they are told by namenode. They report back to namenode periodically with lists of blocks that they are storing.

Secondary namenode → To avoid loosing of all files of the system since no way of knowing how to reconstruct files from the block on data nodes (namenode to resilient to failure).
This don't acts as a namenode.
*Role : Merge namespace image & edit log file to avoid becoming a large file.
It runs on a physical machine & it needs plenty of CPU as muchas the namenode needs to merge namespace image & edit log file.
It keeps copy of merged namespace image which can be used when namenode is failing.

Job Tracker → It is a master process. Responsible (submission
- accepting job submissions from clients.
- scheduling tasks to run on worker nodes.
- Providing administrative functions such as worker health & task progress monitoring to the cluster.
- When job is submitted, info about each task that makes up the jobs is stored in memory.

Tasktracker → Accepts task assignments from the jobtracker

4/7/24

Job Tracker → It is a master process, accepting job submis
- from clients. Scheduling tasks to run on worker node.
Praiding administrative functions such as worker health & task progress monitoring to cluster.
- 1 jobtracker per map reduce cluster. Runs on hardware since failure of master results in failure of all running jobs.
- clients & task trackers communicate with jobtrackers by RPC (Remote Procedure Calls)
- TaskTracker informs Job Trackers as to their current healt & status by way of regular heartbeats. Each heartbeat contains no. of map & reduce tasks available, no. occupied, detailed info.. about currently executing tasks

- A tasktracker is dead after period of no heartbeat.
- Job tracker uses a thread pool to process heartbeats & client requests in parallel.
- Job submitted & info.. about task that makes up job is stored in memory.

Task Tracker :→
- Accepts tasks from Jobtracker, executes locally, reports progress back to job tracker periodically.
Single tasktracker on each worker node.
- Tasktodcker & datanodes run on same machine, makes each node both a compute node and storage node.
- configured with specific no. of map & reduce tasks slots which runs in parallel
- Tasktracker uses a list of user-specified directories holds intermediat map o/p & reducer i/p. Required bcz of large data to fit in memory for large jobs. Many jobs runby in parallel.

Hadoop Distributed File System :-
- Manages the storage accross network of machines
- They are network based. distributed filesystems more more complex than regular disk filesystems.
- Hadoop comes with distributed filesystem called HDFS
- HDFS stores large files with streaming data platforms running on clusters of commodity hardware.

HDFS features → Can process 100s of TB of data.
It built around idea of data processing pattern write once and read many times.
- Dont require expensively, highly reliable hardware to run on.
- Designed to run on clusters of commodity hardware.

HDFS limitations → Appln. require low-latency access to data. HDFS optimized to delivering high throughput data which may be high expensive. HBase is best for low latency access.

2024/8/8 18:41

- It is not suitable for small sized large no. of files.
- File in HDFS written to by a single writer. Writes are always made at the end of file.

Components of HDFS →

- NameNode (Job Trackers) NO ; DataNode (Task Trackers) N1, 2, 3, 4
- File can be larger than any single disk in the network.
- It simplifies storage of subsystem Storage subsystems deals with blocks, simplify storage managements.
- Blocks fit well with replications for providing fault Tolerance & availability.
- HDFS blocks are larger than normal disk blocks.
- By making the block size large make the transfer of data in less time.
- Files in HDFS broken into chunks and stored as independent units.
- Map tasks work at a block level.

HDFS Architecture :-

↳ Distributed file system, Designed to run on commodity hardware, Highly fault tolerance & high throughput access to application data & suitable for applications that have large datasets

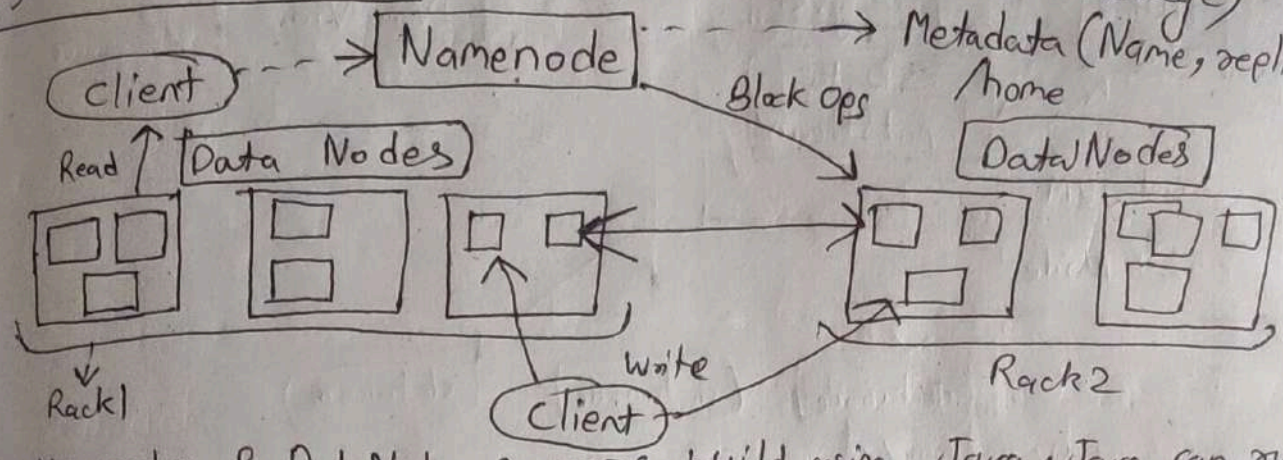
[DFS, Designed to run on commodity hardware, high fault tolerance, high throughput access to application data & suitable for applications that have large datasets.]

HDFS - Assumption & Goals :-

Hardware failure, Streaming data access, Large datasets simple coherency Model, Moving computation is cheaper than moving data, Portability across Heterogeneous Hardware & Software Platforms.

↳ Architectural goal of HDFS : Detection of faults & quick automatic recovery from them.

HDFS Architecture :- (Includes HDFS data storage)



↳ Namenodes & DataNodes : HDFS build using Java, Java can run Namenode or Datanode software.

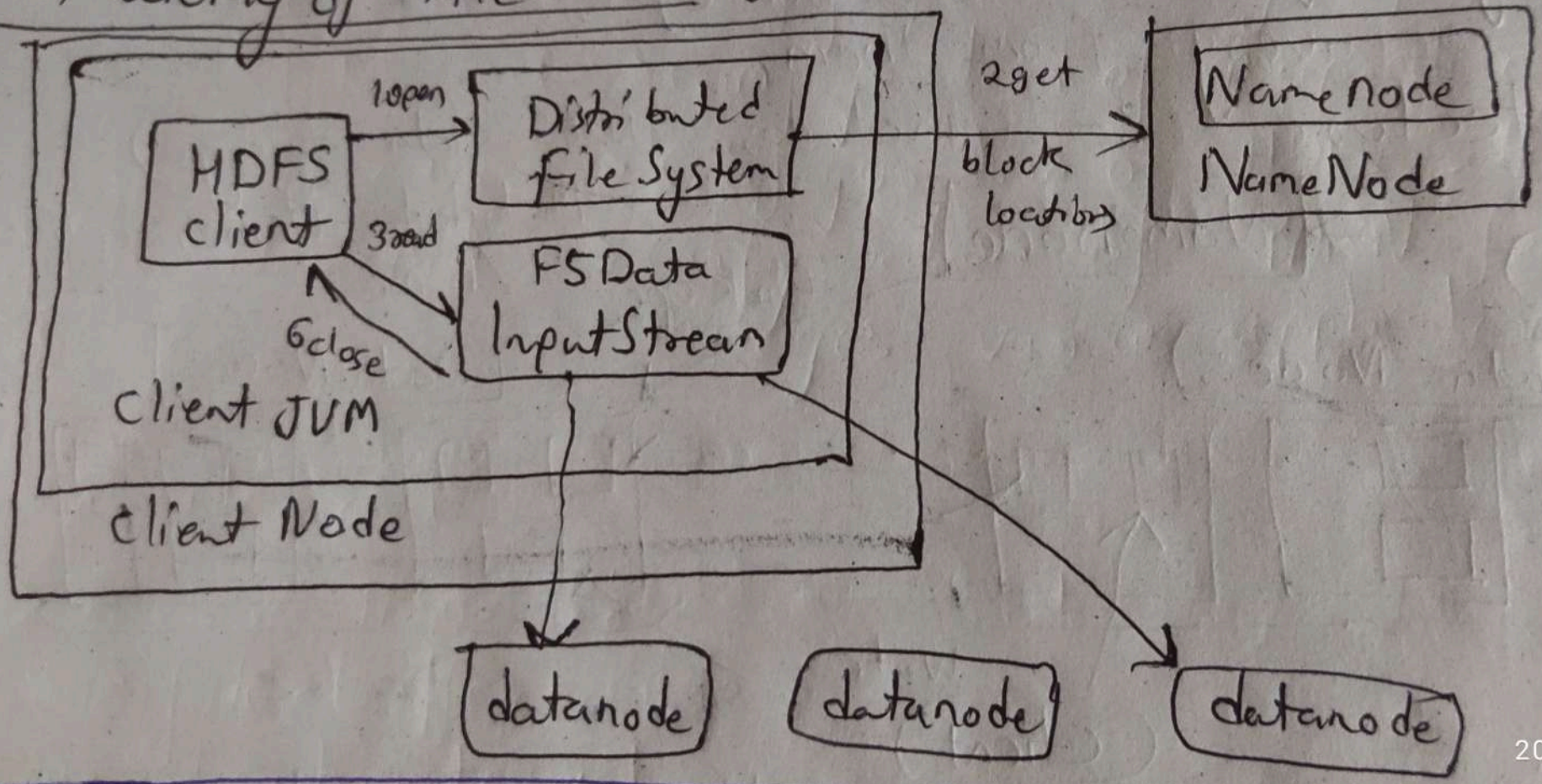
- HDFS has master/slave architecture.
- HDFS cluster consists of single namenode.
- No. of datanode of one namenode.
- HDFS save/stores user data in files. A file split into one or more blocks stored in a set of datanodes.
- Name node executes file system namespace operations. [opening, closing, renaming files & directories]
- Datanodes responsible for read & write requests from file system.
- Mapping of blocks to dataNode.
- DataNode performs block creation, deletion & replication upon instruction from NameNode.

2024/8/8 18:42

HDFS Architecture :- NameNode & DataNode, Data replication Replica placement, The persistance of file system metadata, Robustness [Data disk failure, heartbeats & re-replication, Data Integrity, Metadata disk failure, snapshots] Data Organization [Data blocks, Replication Pipelining]

HDFS Architecture :- NameNodes & DataNodes, File System Namespace, Data Replication [Replica placement, The persistance of file system Metadata, Robustness (Data disk failure, heartbeats & Re-replications), Data Integrity), Meta data disk failure, Snapshots Data Organizations (Data blocks, Replication Pipelining)

Anatomy of File read :-



2024/8/8 18:42

spth

```
import os
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession
```

```
import os
from pyspark import SparkConf, SparkContext
from
```

core-site.xml , hdfs-site.xml, mapred-site.xml, yarn-site.xml

(core, hdfs, mapred & yarn)

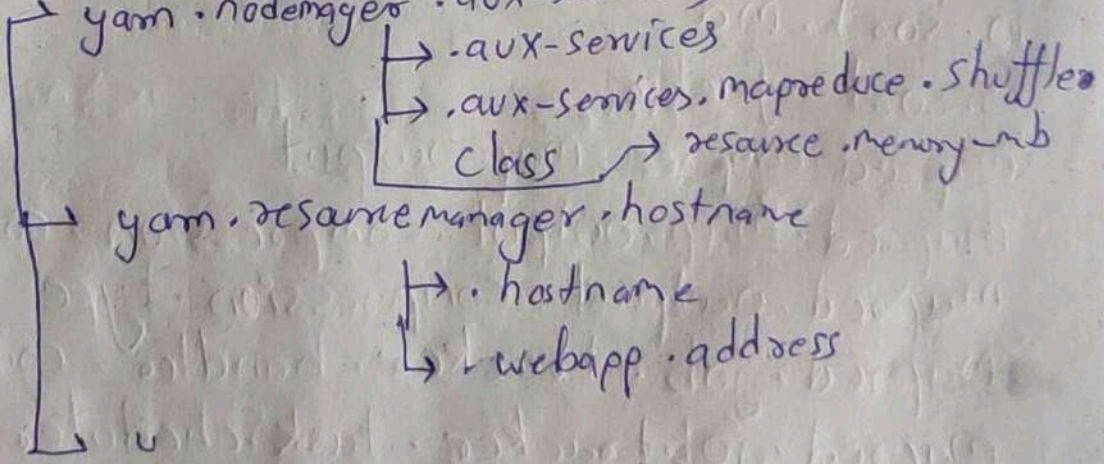
core-site.xml → This file contains configuration setting of Hadoop
- hadoop.tmp.dir
- fs.default.FS

hdfs-site.xml → This file contains configuration setting of HDFS
- dfs.replication
- dfs.permission

mapred-site.xml → The file is for configuration related to MapReduce framework
- mapreduce-framework.name

Yarn-site.xml → This file configuration for the YARN framework which manages for various application
- yarn.nodemanager.aux-services
- yarn.resourcemanager.hostname

2024/8/8 18:43



core-site.xml → Configuration settings of Hadoop
mapred-site.xml → Configuration of mapreduce framework
hdfs-site.xml → Configuration of HDFS settings
yarn-site.xml → Configuration of YARN framework manages various applications.

core-site.xml :

↳ hadoop.tmp.dir ——— 1
io.file.buffer.size
fs.trash.interval
fs.default.name

mapred-site.xml :

↳ mapreduce.map.memory.mb ———
mapred.reduce.tasks ———
mapred.child.java.opts ———
mapred.job.tracker ———
mapred.local.dir ———
mapred.child.ulimit ———
mapred.tasktracker.map.tasks.maximum ———
mapred.tasktracker.reduce.tasks.maximum ———
io.sort.md
io.sort.factor
mapred.compress.map.output
mapred.map.output.compression.codec
mapred.output.compression.type
mapred.job.tracker.handler.count ———
mapred.jobtracker.taskScheduler ———
mapred.reduce.parallel.copies
tasktracker.http.threads
mapred.reduce.slowstart.completed.maps

2024/8/8 18:43

hdfs-site.xml :

↳ dfs.balance.bandwidthPerSec
dfs.block.size
dfs.datanode.data.dir
dfs.namenode.name.dir
dfs.namenode.checkpoint.dir
dfs.permissions.supergroup
dfs.datanode.du.reserved
dfs.namenode.handler.count
dfs.datanode.failed.volumes.tolerated
dfs.hosts
dfs.hosts.exclude

yarn-site.xml :

↳ yarn.nodemanager.aux-services
yarn.nodemanager.auxservices.mapreduce.shuffle.class
yarn.resourcemanager.hostname
yarn.resourcemanager.webapp.address

→ hdfs dfsadmin -repo

2024/8/8 18:43

<c>

<n>
<v>
</n>
</v>

</c>

```
    return(date, open, close)
sdt = spark.SpackContext.textFile("...")
sdt = sdt.map(parseLine)
sdt.take(2)
o_sdt = spark.read.option("InferSchema", "True").
        .load("...")
o_sdt = o_sdt.toDF("","") .rdd.map(lam
                                   row[0], ro
o_sdt = o_sdt.filter(lambda x: x[2] > row
```

---

core-site.xml :

↳ hadoop.tmp.dir

mapred-site.xml :

↳ mapreduce.map.memory.mb
mapred.reduce.tasks
mapred.child.java.opts
mapred.job.tracker
mapred.local.dir
mapred.child.ulimit
mapred.tasktracker.map.tasks.maximum
mapred.tasktracker.reduce.tasks.maximum
mapred.job.tracker.handler.count
mapred.jobtracker.taskscheduler

HDFS-site.xml :

↳ dfs.block.size
dfs.datanode.data.dir
dfs.namenode.data.dir

2024/8/8 18:43

yarn-site.xml :

↳ yarn.nodemanager.aux-services
yarn.nodemanager.auxservices.mapreduce.shuffle.class
yarn.resourcemanager.hostname
yarn.resourcemanager.webapp.address

mapreduce.map.memory.mb, mapred.reduce.tasks, mapred.child.java.opts

Hadoop → open source framework designed to handle large amount of data which ranges between peta to exa bytes.
Data handles : structured, semiStructured & UnStructured data.

Components → HDFS : Provides throughput access to system
YARN : Manages & schedules resources in cluster
MapReduce : Process the large datasets with parallel, distributed algorithm.

- HDFS architecture → Master/Slave architecture :
Namenode (Job Tracker) - Master [Manages file system namespace, control access]
DataNode (Tak Tracker) - Slave [Manage storage attached to nodes. Data is splitted and distributed access each datanode.]

Namenode → HDFS manages the file system & (Metadata & namespace) of file system

Datanode → Data blocks responsible for read & write requests from file system.

Secondary namenode → Not a backup of namenode. Periodically it merges namespace image with edit log to prevent the edit log from becoming too large.

2024/8/8 18:45

- YARN → Resource management layer of Hadoop
Allows multiple data processing engines to handle data stored in a single platform. Providing resource management & job scheduling.

Components → Resource Management : Allocates resources
Node Manager : Monitors resource usage on each node and reports to resource manager.
Application master : Manages the lifecycle of applications running on YARN cluster

MapReduce programming model → It involves 2 steps
2 steps : Map & Reduce
Map – Process i/p data & generates key value pairs.
Reduce – Takes o/p from Map tasks & aggregates it to produce final result.

Combiner in mapreduce → Optional function that can perform a local reduce task on the output of the map tasks. It helps in minimizing amount of data transferred b/w map & reduce tasks.

HDFS → Ensures data reliability through replication. data block is replicated accross multiple datanodes
Maintains 3 copies of each block.
Block is amount of data to read or write

Hadoop performance → Tuning no. of mappers or reducers
Combiners to reduce data transfer
Configuring hadoop parameters

speculative execution → process where duplicate copies of slow tasks are executed on other nodes.
1st task to complete is used & other tasks are killed
It helps in reducing overall job execution time.

Setup Hadoop cluster → - Install hadoop on all nodes
- configure core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site
- Formatting namenode
- Start the HDFS & YARN daemons

Hadoop-env.sh → To set environment variables that effect Hadoop daemons. JAVA_HADOOP.

Hadoop-1.x. → Uses JobTracker/TaskTracker for MapReduce

Hadoop-2.x. → Uses YARN resource management & job scheduling. It supports multiple data processing frameworks other than MapReduce

Issues & troubleshoot Hadoop → Namenode failures, network connectivity problems & data corruptions.
Troubleshooting involves checking log files, using monitoring tools & verifying configuration settings.

↳ Install
update
group add
system configuration
disk format
authorized key
ssh root@localhost
download hadoop
change owner
configuration in .bashrc
check java & hadoop version
xml properties
jps
hdfs input & output
jar file

pwd, cp, variable data files, cat
locate, Paas, chown, FTP
df, HFS+, useradd, .bashrc, ipconfig,
scp, NTFS, kill, /var variable directory,
Saas, apt-get update

/root/.ssh

Cloud computing
↳ Security, to access or transfer data via internet.
example :- Laptop, Network, Tablet Computer, Database, mobile, server, smart phone.
It is easy to access it provide storage to, Public private & Hybrid cloud

11 | 7 | 24

2024/8/8 18:46

↳ yum package
chmod is to change file permission
default interactive shell - Bash
Nano - text editor for model editing capabilities
Systemctl to start-stop & manage services
ps to list currently running process
/bin contains configuration files in Linux
Ubuntu - Often used in enterprise environment
SSH for secure remote login
VMware is native to Linux & provides hardware assisted virtualization

19/7/24

Big Data

Big Data : Handles large data sets, data storages.
- Skills → Data analysis & programming [Python, R, SQL]
- Tools → Bigdata tools [Hadoop, spark], DV, ML & statistics
- Sources → IOT; transactional; web; govt..

Big data Adoption : Drivers & Barriers
(Adoption of Big data ↑) (↑cost, lack of skilled person-nel, data privacy)

Research & changing nature of data repositories :
↳ Trends & Focus

Trends – Advancements in cc, distributed databases, and 2n databases
Focus – ↑ data storage, retrieval speed ensuring data integrity & security

Data sharing & reuse practices & their implications for Repository data curation :
Practices –
Implications –

Hadoop versions :-

2024/8/8 18:47

V1.0 :- HDFS & MapReduce.
- Limitations in terms of scalability & resource management

V2.0 :- YARN, (Separates resource management from processing layer).
enhancing scalability & flexibility
MapReduce + [Graph processing, Interactive Querying]

V3.0 :- Supports erasure coding, improved storage efficiency, addition of docker containers.
enhancements to HDFS, YARN, MapReduce.

Hadoop & Components :-

↳ HDFS :- ↑ throughput access to application data.
- splits files into large blocks & distribute across nodes.

YARN :- Schedules jobs and manages clusters resources,
- Allows multiple data processing engines to run on single hadoop cluster

MapReduce :- 2 main tasks Map (Processes & filter data)
Reduce (Aggregates & Summerises)

Hadoop vs Traditional DB :-

↳ RDBMS handle

DFS (Distributed File System) :-

↳ Data stored across multiple servers & locations, ↑ availability, reliability & scalability.
↳ Distributed environment, enabling efficient data processing using tools like MapReduce, YARN & Hadoop components.

Components of HDFS : NameNode & DataNode

↳ Namenode – Master Node manages files system namespace
– Regulates access to files
↳ Datanode – Worker/slave nodes that stores & retrieve block when they are told.
– Report backs to Namenode with block that they are storing. Periodically.

2024/8/8 18:48

HDFS daemons :-

↳ NameNode → Manages meta data & file system namespace. It keep on tracking.
DataNode → Daemons that runge strage on individual nodes.
& and w request from a client
2nd NM → Merging namespace image & edit log to reduce load on NN.

↳ HDFS follows master-slave architecture
NameNode ← → DataNode

Scaling & Rebalancing :-
Scaling → Horizontally adds datanodes to cluster. ↑
↑ storage capacity
Enhances parallel processing capabilities.
Rebalancing → distribution of dataNode become in balance (data in)
due to data growth or node additions/removals
[HDFS balancer & redistributes]

Replications :- Replicates data blocks across multiple data-Nodes.

Rack Awarness :- It uses a rack awarness algo to improve data reliability & bandwith utilization.

Data pipline :- Client writes to DataNode in sequence 1st DataNode then to next 2nd DataNode

Node failure Management :- Handles Node failuresses automatically. If DN fails the NN automatically initiates the replication of the block ~~allocates~~

Zookeeper
HDFS architecture :- Manages & Processes data sets
Design principles → Replications, rack awarness, high availability
Features can build big data solutions.

↳ Drop → Deletes entire table
Delete → Removes row from table
Truncate → Removes all row data

B B C C A B
B

20/7/24

Hadoop Operation Mode :-

↳ Standalone Local Mode (HDFS on a single Machine) [daughter]
↳ Pseudo distributed (HDFS runs on single machine simulated distributed environment)
↳ Fully distributed (Runs on multiple machines ; separate nodes : NN, DN, RM, NM) HDFS and YARN daemons [Production environment]

↳ core-site.xml : Configures Hadoop properties
* HDFS : " " DataNode & NameNode
mapred : " MapReduce ; job history & (loc and Framework)
* Yarn : " YARN ; ResourceManger & NodeManger

Resource manager → Mangges resources & schedules jobs
Node manager → Manages individual nodes resources & monitors resource usage
Data Node → Stores actual data block
Name Node → Manages file system namespace & Metadata
Secondry NN → Periodically merges namespace image & edit log.

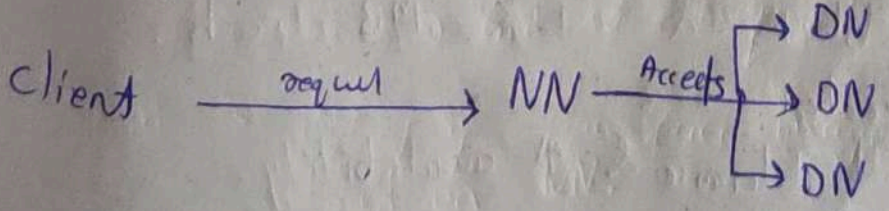
Job Tracker (MRV1) → Manages job scheduling & task tracking
- Resour Manager : handles resource management
- Application Master : Takes care of Job scheduling & task monitoring.

Task Tracker (MRV1) → Executes individual tasks assigned by job tracker.

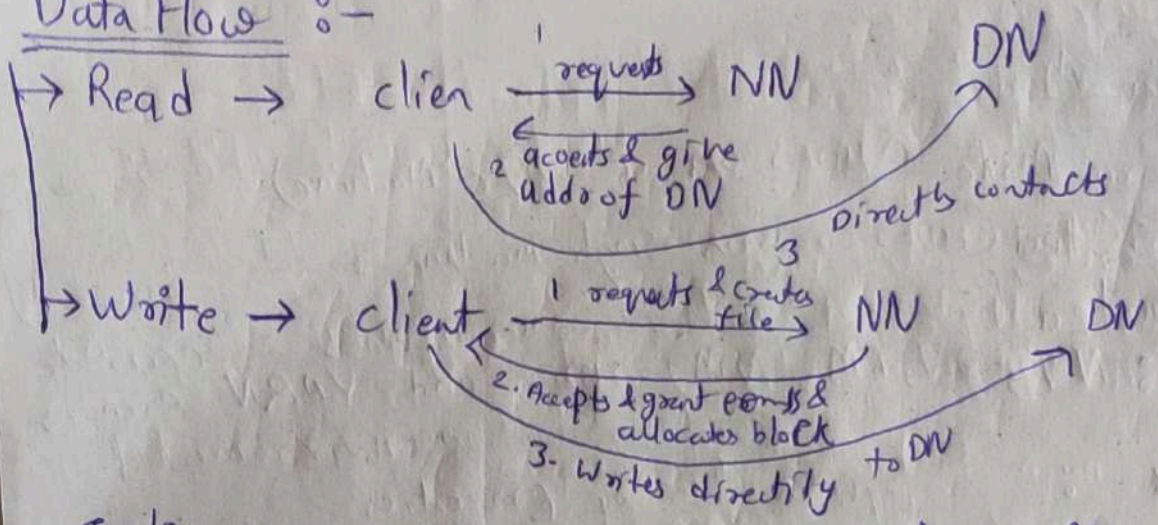
Resource Manager (YARN) → Manages resources & application

Node Manager (YARN) → Manages resource & containers on a single node

Application Master (YARN) → Manages life cycle of a single application



Data Flow :-



Scaling → HDFS is horizonatal scaling. More DN are added to cluster to ↑ storage capacity

- Horizontal scalability : adding data nodes to cluster & handle
- Load Balancing :
- NN federation :

Rebalancing in HDFS :- Ensuring even distribution of data across DN

- HDFS balancer Tool : To redistribute access DN via cmd
- Threshold Based Balancing : Threshold parameter to determine when rebalancing is necessary.
- Incremental Balancing : Rebalancing is performed incrementally to minimize disruption to the system.

↳ Optimal performance & resource utilization
HDFS a robust solution

Map Reduce for parallel processing

↳ HDFS commands :-

→ Data split into blocks & 128 MB each.
Blocks are replicated → 3
DaN & ensures fault Tolerance & ↑ availability.

↳ hadoop fs -chmod /path : A permissions
dfsadmin -safemode enter
dfs admin -safemode leave
dfs admin dfsadmin Refresh the list of DataNodes.

↳ NameNode : 9870
ResourceManager : 8088

↳ Hadoop MapReduce : processing large data sets in a parallel across a dataset

- MapReduce tasks → Map [processes i/p data & generates key-value pair]
- Redue tu Reduce [Aggregates intermediate key value pair to produce final o/p]

MapReduce Daemons :-
↳ Resource manager [manages resources]
↳ Node manager [monitors resources]
↳ Application master [Manages execution of a i/e MapReduce]

Anatomy of a MapReduce Job Run :-
↳ Job Submission - client submit job to RM
i/p splits - splits into logical chunks processed by separate map tasks
Map phase - Processes i/p & generates key-value pairs.
Shuffle & Sort - key-value pair shuffled & sorted by keys
Reduce phase - Processes grouped key-value pair to generate final o/p.
o/p - final o/p written to HDFS or another storage system.

Partioners & Combiners :-
↳ Partioner determines which reducer a particular key-values pair will go to. (To control data distribution)
Combiner performs local aggregation of intermediate key value pairs that they sent to reducers.

↳ i/p formats

Input Formats :-

- i/p splits & Records [Logical chunks to Map task generates key-value pairs]
- Text i/p [Default input format each line treated as a record]
- Binary i/p [Seqn file ile format to read binary key-value pairs stored in Hadoop seqn files]
- Multiple i/p [Allows i/p for different data sources in the same job]

Output Formats :-

- Text output [key-value pairs written as text lines default o/p format]
- Binary output [Sequence o/p format (Writing binary key value pairs to Hadoop seq. files)]
- Multiple output [Allows multiple data to multiple o/p from a single job]

Distributed Cache :-

- Distributes read only data needed by MapReduce jobs to all nodes in the cluster
  - addCacheFile() : Adds a file to a distributed cache
  - addCacheArchive() : Adds an archive to the distributed Cache

- Hadoop MapReduce paradigm & large scale data processing by dividing tasks into smaller subtasks Distributed Cache performed by making read only data readily available to all nodes.

Hadoop Datatypes ⇒ 6 (int long float double bool Text) [Primitive]
⇒ 3 (Array, Map, Null) [Composite]

Java & MapReduce :-

↳ Mapreduce written in Java.
↳ Robust API for implementing Mapper, Reducer
↳ MapReduce components :
- Mapper : define Map function
- Reducer : define reduce function
- Drive class : configures job & specifies i/o paths, Mapper & Reducer etc.

Map Only :
↳ No reduction or aggregate is needed. Reducer is omitted or set to null

Reduce only :
↳ Preprocessing has already done only aggregation is needed.

Combiner & Partioner :
↳ Combiner — Acts as mini Reducer to reduce data transfer
↳ Partitioner — Controls the partitioning of intermediate map outputs
job.setCombinerClass(), job.setPartitionerClass()

Counters : Tracks the progress or count the events
- getCounter()

Schedulers : Job scheduling
↳ FIFO scheduler — Jobs executed in order they arrived
Fair scheduler — Jobs are allocated resources that all jobs get an = share of resource over time.
Capacity scheduler — To run Hadoop jobs in a shared, multi tenant cluster to ensure certain capacity guarantee.

Custom Writables : To define custom data types
- Writable
- WritableComparable

Compression : Reduces amount of data transferred & stored
- .setCompressOutput()
- .setCompressorClass()

2024/8/8 18:49

Complex Map Reduce Programming

↳ multiple MR jobs can be chained together.] chaining job
o/p of one job being i/p for next

↳ different stages of processing using multiple mappers & reducers.

Map Reduce Streaming :- writing MR jobs in any language that can read from standard i/p & write to standard o/p.

Python & Map Reduce :- Python scripts for both Mapper & Reducer

MapReduce on image dataset :-

↳ Image processing (resizing, format conversion, feature Extraction)
- using Py Lib : PIL or OpenCV

↳ Implementation :
Mapper - Process each image, perform necessary image processing task
Reducer - Aggregate results if needed.

Hadoop ETL :-

↳ ETL (Extract Transform Load) : Extracting data from various sources, Transform it into suitable format loading into Hadoop's HDFS or similar storage system.
- Extraction : Extract/Load data from files
- Transform : Clean up or arrange data to standard or specific
- Load : Transformed data into HDFS, HBase or Hadoop compatible storage.

↳ ETL functions :-
- Data Extraction : Sqoop (data b/w Hadoop & RDBMS)
Flume (collects, aggregates - streaming data into HDFS)
Kafka (data streams processed in real time or loaded in HDFS)
- Data Transformation : pig (Pig latin to handle complex data)
Hive (SQL - query for data transform & analysis)
Spark (API for complex data transforms using RDDs & DF)

2024/8/8 18:49

- Data Loading : HDFS (i° storage for large datasets)
  HBase (NoSQL DB runs on top of HDFS, rw access)
  Hive, (Acts as data ware house storing data in a structured format

Need for ETL Tools :- Complex data Management, Automation, Scalability, Integration

Advantages of ETL :- Efficiency, data Quality, Compliance, Flexibility, Performance, User Friendly

---

HBase :-

↳ Open Source, non-relational, distributed database.
↳ Written in Java
↳ Handles data across commodity servers.
↳ Provides real time read/write access to big data

Features : column oriented storage, Scalability, Consistency, Integration with Hadoop, High availability.
↳ Horizontal scaling :-

HBase Architecture :

→ Components →
- HBase Master : Manages Cluster (Table creations, deletion & balancing region)
- Region Server : Handles r/w requests. Manages HFiles & WAL (Write Ahead Log)
- ZooKeeper : Coordinates distributed process & provides configuration & synchronization services.

→ Data Model →
Tables, Rows, columns, col families, cells.

→ Storage → HFiles, WAL (Write-Ahead Log)

→ Regions →
Region - Subset of table horizontally partitioned by row key range
Region Split - Automatically splits it

2024/8/8 18:49

Installation :-

- Prerequisites : Hadoop cluster installed & running
  Zookeeper ensemble configured & running
- Steps :
  - Download & Extract HBase
  - Configure HBase
  - Start HBase

Operations :-

- Creating a Table - create
- Listing Tables - list
- Inserting data - put
- Retrieving data - get
- Scanning data - scan
- Deleting data - delete
- Dropping table - disable
  drop
- Counting rows - count

HBase commands :-

- hbase shell - starting the shell
  listing comands - help

Java Client API for HBase :-

2024/8/8 18:49

- Establishing connection - HBaseConfiguration.create()
  .createConnection()
- CRUD operations -
  - Create : .getTable() , .addColumn()
  - Read : .get() , .getValues()
  - Update : HBase overwrite the value
  - Delete : .addcolumn() , .delete()

→ Admin API :-
- Creating : .getAdmin() , .addFamily() , .createTable()
- Listing tables : .listTables()
- Disabling & Dropping Tables : .disableTable, .deleteTable

↳ HBase – Scan, Count, Truncate
(scan , count , truncate)

↳ HBase security
Authentication : kerberos
Authorization : Access Control (Defining permissions)
Encryption : Data encryption (Encrypt using HDFS encryption zones)
Network encryption (Encrypt data in transit TLS/SSL)

↳ Robust support for storing & processing large datasets with real-time r/w access.

HIVE

Features : SQL like Query Language, schema on Read, Integration with Hadoop, support for diff.. storage types.

Components : UI, MetaStore, HiveQL process Engine, Execution Engine, HDFS

Installation :
- Prerequisite : Java JDK, Hadoop, SSH

Datatypes : string, int, bigint, Boolean, float, double, Timestamp, Date

Complex types : Array, Map, Struct, UnionType

String functions : CONCAT(), Length(); upper(); lower()
Date functions : Current_date(), Date_Add(), date_sub
Hive tables : Managed tables (manages data & metadata), external tables (manages metadata only)

Partions & Buckets :

Partions → Divides tables into parts based o values of particular column.
Buckets → Divided into more managable parts Used for more effiive sapling & joins.

Storage formats :- Text file, sequence file,
- Text files : Default storage format (text_table
- Sequence files : seq_table
- ORC : (Optimized Row Columnar)
- Parquet : parquet_table

Importing data :- LOAD DATA command by creating external tables.

2024/8/8 18:50

Quering Data :-
→ Sorting data : Order by, sort by, Distribyte by
→ Aggregating data : Count, sum, avg, min, max

Map Reduce scripts :- Hive can translate HiveQL into MapReduce jobs.

Joins & Subquiries :- Selet, from, whose, Inner join, subquery

View are virtual tables creating using the result of a query

Map and Reduce Side joins to Optimize Query

Map side and Reduce side joins to optimize Query.

map side join : smaller dataset fit into memory.

Reduce side join : Default join in HIVE, suitable for large datasets.

Data manipulation with HIVE :-

Inserting data, Updating data, deleting data

UDFs (User Defined Functions) :-

↳ Creates custom functions to extend HiveQL capabilities. Can be written in Java & other languages.

- Creates Java class extending UDF.
- Implement the evaluate method
- Compile & add the JAR to HIVE.

Custom mapReduce in HIVE

↳ Scripts can be integrated into HIVE queries (TRANSFORM).

Writing HQL scripts :-

↳ HQL allows for the batch processing of HIVE queries. HQL script is a text file containing HIVEQL commands.

For executing Hive : hive -f script.hql

2024/8/8 18:50

Hadoop :- Open source, to handle large datasets. Allows for storage & processing of big data.

components : HDFS, YARN, MapReduce

HDFS :- Provides scalable & reliable data storage designed to span large cluster of commodity servers

Architecture : NNode, DNote, 2° Node, HDFS daemons (Name Node & Data Node)

HBASE :- Distributed, Scalable, large amount of sparse data Stores and manage

Architecture : HBase master, Region Servers, Zoo Keeper HFile

Hive : Data ware house built on top of Hadoop.
— Data summarises, querying & analysis.

Architecture —
Metastore
Hive server
Hive driver
Execution engine

↳ Hadoop : Provides distributed storage (HDFS) & processing data (Map Reduce)

HDFS : Handles large datasets with high reliability & scalability

HBase : NoSQL DB built on top of HDFS, realtime read/write access to large dataset

Hive : Data warehousing that provides SQL like interface for Querying & analyzing data stored in Hadoop.

Apache Spark :—

↳ open source, distributed computing system. Provides interface for programming entire cluster with implicit data parallelism & fault tolerance.

→ cluster with implicit data parallism & fault tolerance

→ To overcome limitations of Hadoop MapReduce framework and provide faster, more efficient data processing.

2024/8/8 16:50

Linking with Spark →

↳ Hadoop
Apache Hive
Apache HBase
Apache Kafka

Initializing spark :-

↳ SparkSession.builder(

RDD (Resilient Distributed Datasets):

↳ Properties : (Immutable, Fault tolerance, Parallel processing)

. RDDs fundamental data structures of Spark.

creating RDDs :- .parallelize(data)

↳ RDDs : Low Level APIs provide transformatnd actions.

↳ pyspark.sql import SparkSession

sparksession.builder().appName().getOrCreate()

sc = spark.sparkContext

data = [, , , ,]

rdd = sc.parallelise(data)

rdd = sc.textfile(" --- ")

Shuffle Operations :-

↳ reduceBy key & groupBy key

RDD persistence :-

↳ It can be persisted in memory or disk to avoid re computation.

Removing data :- To unpersist data [rdd.unpersist()]

2024/8/8 18:50

Shared Variables :-

- Broadcast variable read only value to all nodes
- Accumulator : variables added to from workes & read only by drivers.

Deploying to a cluster :-

- standalone mode : Spark manages own cluster
- YARN : Spark appl.. on Hadoop cluster
- Mesos : cluster manager used to run spark.

MapReduce with Spark :-

Spark [with hadoop] → Hadoop HDFS for storage
YARN for resource management

[without Hadoop] → Use local file systems or other storage solutions like S3.
Spark's standalone cluster manager.

↳ Storage : with Hadoop — HDFS
without Hadoop — Local file system or cloud st

Resource Management : with Hadoop — YARN
without Hadoop — Standalone or Mesos

Data Preprocessing :- step in data analysis pipeline.

Transforms raw data into a clean & usable format.

Steps → Data cleaning
Data Transformation
Data Reduction [(PCA), Sampling]
Feature Engineering

EDA (Explanatory Data Analysis) :-

2024/8/8 18:50

↳ Analyzing data sets to summarize their main characteristics, often using visual methods.
Understanding the data & uncovering patterns, anomalies, relationships.

- Descriptive Statistics [mean, median, mode, var, SD]
- Data visualization [Plots]
- Correlation Analysis [Relationship b/w variables using correlation coefficients]
- Outlier detection [Identifying the outliers in data

Apache kafka :-

↳ Designed for high throughput, low latency in data streaming. For build a real time data pipelines & streamin applications.

spark streaming Architecture →

- → Discretized Streams (DStreams) [Spark streams represent Streams of data as series of RDDs]
- → Recievers [To ingest data from sources like kafka, Flume & others]
- → Processing [Each batch data processed by Spark jobs, can perform transformations & actions on data]
- → Output Operation [Processed data can be written to various output sinks (HDFS, DB,

↳ Apache spark & kafka for building scalable, real time data processing pipelines.
Combining capabilities of Spark for processing

kafka connect API :-

- Source Connectors (Import data from → Ext.. syst into kafk
- Sink Connectors (kafka export → Ext.. systems)

↳ Apache kafka :- written in java & scala.

- Messaging, Storage, Processing
- Topics, Partitions,
- Producers & Consumers
- Brokers
- Zoo Keeper

Apache Spark

2024/8/8 18:51

21/7/24

Apache kafka :-

↳ Open source stream processing platform. Written in Scala & Java.

Functionalities : Messaging, storage, Processing

Concepts : Topics – Categories where records are stored
Partitions – Segments of topics that allow parallel processing
Producers – Data written to kafka topics
Consumers – Data Read from kafka topics
Brokers – Kafka made of brokers handles storage & retrival of records.
Zookeepers – Manages & coordinats kafka brokers.

Apache Spark :- Open source unified analytics engine for large scale data processing.

Provides high leve API in java, scala, Python, R

Components : Spark Core, Spak SQL, spark Streaming, MLlib, GraphX

Spark SQL :- Spark module for structural data processing.

It provides DataFrame API & SQL queries.

Features : DF & DataSets → DF (Data organized into name colums)
Dataset (Provides benifits of RDD with optimized adv of Spark SQL)

SQL Integration → Allows executing SQL queries along-side complex analytic algorithms.

Catalyst Optimizer → Query Optimizer for executing queries efficiently.

Datasource API → Allows Integration with variety of data source (JSON, Parquet Avro)

Spark MLlib :- ML lib built on top of spark.

Features : Algo – Classification, Regression, Clustering, Collaborative filtering etc.–
Featurization – Feature extraction, ET, dimensionality Reduction

Pipelines – Tools for constructing, evaluating & tuning ML
Utilities – statistics, data handling & other utilities.

Predictive analysis with Spark :-

↳ using historical data to predict future outcome

steps – Data Preprocessing (cleaning & Transforming)
Feature Engineering (Creating new feature from existing data)
Model building (LR, DT)
Model Evaluation (Metrics like accuracy, precision, recall)
Model deployment (Integrating model into production systems to make predictions)

Workflow in Spark :-

↳ Initialize Spark Context → Load & preprocess data → Feature Engineering → Split data → Train Model → evaluate Model → Deploy model

Linking kafka with Spark :-

↳ kafka used as source and sink for Spark streaming application.

steps : – Setting up kafka
– Creating Spark Streaming Context
– Consuming data from kafka
– Processing data
– Producing data to kafka

→ Apache Kafka & Apache Spark powerfull tools for realtime, scalable, data processing pipelines.
→ Spark SQL and MLlib provides robust robust frameworks for data processing & ML, enabling predictive analysis.
→ Integrating kafa and Spark, creates efficient streaming applications that handles large volumes of data with low latency.

2024/8/8 18:51

→ Benefit of dashboard actions : To link visualization and enable user interactivity
→ Automatic layout helps to design dashboards that adapt to different screen sizes.

→

Hadoop

↳ purpose of data replication : Ensure fault tolerance & data availability.
↳ HDFS High Availability (HA) : provides high availability by allowing a standby Namenode.
↳ ACLs (Access Control Lists) in HDFS allows administrators to manage access to files & directories. (User permissions)
↳ Kerberos authentication to authenticate users & services in a secure manner.
↳ YARN → MapReduce jobs
↳ Pig is a Hadoop component which provides a high-level abstraction over MapReduce.
↳ MapReduce Framework is to process the data in parallel across a distributed cluster.
↳ MapReduce Framework provides fault tolerance through data replication.
↳ Hive is a tool in hadoop ecosystem which is used for data warehousing & SQL like queries
↳ Apache Hbase is to provide real time read/write access to large datasets.
↳ Sqoop tool is used to import & export data b/w Hadoop & RDBMS.
↳ Apache Flume is used to data ingestion from various sources.
↳ JobTracker : Coordinates MapReduce jobs
↳ Task Tracker : Execute tasks assigned by JobTracker
↳ Secondary NameNode : Merges namespace image & edit log ; to help in recovering the file systems metadata in case of a failure.
↳ Feature of Hadoop data storage model : Data stored in large, fixed size blocks.

2024/8/8 18:51

↳ Rackawareness: distribution across the different racks to improve fault tolerance & n/w bandwidth usage.
↳ Implementing Kerberos is to authenticate users & services in a secure manner.
↳ Hadoop key management Server (KMS) is to manage & store cryptographic keys for data encryption.
↳ Namenode: Manages distributed File System (DFS) Maintaing metadata of HDFS & manages file system namespace.
↳ JobTracker (+1V) / ResourceManager (+2V) (YARN): Coordinates MapReduce jobs. Manages & scheduling & resource manging, execution of MapReduce jobs
↳ DataNodes: It stores the output data.
↳ Monitoring (Cluster monitoring Hardware) is handled by cluster monitoring tools like, [Ambari, Ganglia or Nagios] (used in hadoop ecosystem for monitoring purpose)
→ Resource Manager is role of managing resources in the cluster in YARN
→ Task Tracker in (Ver 1) or NodeManager in (Ver 2+) responsible for executing tasks assigned by Job Tracker or Resource manager
→ Job Tracker in (Ver 1) or Resource Manger in (ver 2+) is responsible for scheduling & manging execution of jobs in the cluster
→ Hadoop (Key Management Services) KMS is to mange & store cryptographic keys for data encryption It handles generation, management & storage of encryption keys for securing data in HDFS.
→ Monitoring the health of the Hadoop cluster is done by external tools like Ambari, Ganglia or Nagios used for system & application monitoring & including Hadoop clusters.

→ Hadoop : opensource framework, process large datasets in a distributed computing environment, performs parallel computing across the cluster of machines
Components : HDFS, MapReduce, YARN

→ Zookeeper : is a centralized service for maintaining configuration info., naming, providing distributed coordination for distributed application. Often used to manage coordination & configuration of Hadoop cluster and services like HBase & kafka. It manages failures.

→ HBase : Distributed, Scalable, NoSQL database built on top of HDFS. Provides real time r/w access to large datasets. Supports structured data as a key value pairs & designed to handle large tables;
Application : real time analytics & random real time r/w access to big data.

↳ Hive : Data warehouse infrastructure built on top of Hadoop. Provides tools ~~like~~ for data summary, query and analysis & SQL like language called HiveQL. Abstracts complexity of Hadoop's MapReduce programming. Used for batch processing querying large data sets in HDFS.

↳ Apache Spark : Open Source, distributed computing system. In memory data processing engine, faster than hadoop MapReduce. It supports batch processing, real time stream processing, ML, graph processing. offers API in Java, Scala, Python & R

↳ Apache Flume : Centralise data store, Used to ingest data into Hadoop for further processing and analysis. Collects data from various web servers, appl. servers, & other sources & store it in HDFS, HBase

↳ HDFS (Hadoop Distributed File System) : Stores large amt of data sets reliably & stream data sets at high bandwidth splits large files into blocks (128 MB). Providing fault tolerance, ensures data redundancy by replicating each block multiple times across different nodes.

2024/8/8 18:52

↳ DFS (Distributed File System): It refers to HDFS. Manage & store data accross a distributed n/w of a machine. Provides ↑ throughput access to application data & capable of storing v large files.

↳ YARN (Yet Another Resource Manager): Management layer of hadoop. Separates resource mangent & job scheduling functions from the data processing componants. Allowss multiple data processing like MapReduce spark. to run & process data stored in HDFS. It contains Resource Manager (Arbitrates resources among all applicn) Node Manager (Monitors resource usage on each node)

↳ MapReduce: Programming model & implementation for processing large datasets in a distributed fashion Map phase (processes i/p & produces key value pairs) Redue phase (Intermediate results to produce the final o/p - Written in Java.

↳ PIG: High level platform for creating MapReduce scripting language (PIG Latin) Abstracts complex MapReduce programming model. Used for data transformation, aggregation & analysis tasks.

↳ Oozie: Workflow scheduler system to manage Hadoop jobs. Define sequence of actions to be executed. in a workflow (MapReduce jobs, Pig scripts, Hive queries) Supports scheduling & chaining of workflows enabling complex data processing pipelines in Hadoop.

↳ Sqoop: Tool for transfering bulk data between Apache Hadoop & structured data stores such as RDBMS. Imports data from external database into HDFS or HIVE exporting data from hadoop back to database. minimizes amount of manual coding required.

2024/8/8 18:52

↳ Flume : For efficient collecting, aggregating & moving large amount of log data or othe data streams from various sources to a centralized data like HDFS or HBase.

↳ HCatalog: Table & storage managment layer for Hadoop provides schema & storage layer for Hadoop tools. Allows to share data between different tools like Pig, MapReduce & Hive. Provides unified view of data stored in HDFS.

Hadoop Daemons :-

↳ NameNode : Mangges metadata & namespace of HDFS, keeps track of data where data is stroged accross the cluster. Operations (open, close, renaming files/directories)

↳ Secondary NameNode : Periodically (checkpoints of HDFS metadata to prevent data loss). Merges edit logs & namespace images (fsimage). Keep Namenode's memory usage efficient.

↳ Resource Manager : Part of YARN, Manages resources accross the cluster. Schedules and allocates resources to applications based on their needs.

↳ Node manager : Manages resources of a single node in the YARN cluster. Monitors resource usage (CPU, memory, disk) & reports to resource manager.

↳ Job History Server : Track completed MapReduce jobs & Stores information about history for debugging & auditing purposes.

↳ Task Tracker (Depricated in +2v) :
In Hadoop 1, Task Tracker managed task running on individual nodes.
In Hadoop 2, Replaced by NodeManager under the YARN framework.

2024/8/8 18:52

↳ A, B, B, C, B, B, C, C, A, B, A, B, A, D, A, C
C, B, A, B, B

→ Histogram : frequency of data points within specified range within a specified range. [distribution, shape & spread of data

→ Year() to extract year from a date value in many

→ EOMONTH() last day of the month for a given date.

LOD (Level of Detail) : Data analytics & visualization

To define how data should be aggregated in calculations

Concepts of LOD :-

- Granularity : Level of detail or specificity in data.
- Aggregation : Process of summarizing data

Types of expressions : Fixed, Include, Exclude

- Fixed :
- Includ :
- Exclude :

Use cases of LOD :-

- Consistent Benchmarks
- Dynamic Scoping
- Removing unwanted details

Gantt chart : For project management & scheduling. It displays timeline for tasks or activities, showing their start and end dates for how they overlap.

↳ Plots :-

Histogram → Distribution of data points accross a continous variable. Frequency distribution of a continuous variable. Distribution of data points in a dataset. To display distribution of data. Frequency of events.

↳ Scatter plot : Relationship b/w numerical variable. Correlation between 2 variables.

↳ Bar chart : Comparision of sales/Quantity accross category comparision accross categories, compare categorical data,

↳ Gantt chart : The progress of tasks over time in a project

↳ Line chart : Trend over time, changes in a variable over time, trends over time, Time series data

↳ Tree map : Treemap displays (Proportional data in a hierachical data).

↳ Stacked Map chart / Mosaic Plot : Relationship b/w 2 categorical variables

↳ Pie chart : Proportional relationships b/w parts of whole, To illustrate parts of a whole.

↳ Area chart : To show cumalative total over time

Apache Spark :- Open source, distributed computing system provides a fast & general purpose cluster ~~computing~~ computing framework. Large scale data processing and handles batch processing, Interactive queries, Stream processing, ML & graph processing

Speed, ease of use, wide range of workloads.

- Speed → 100x times faster than Hadoop MapReduce
- Ease of use → Java, scala, Pytho, R.
- Unified engine → Cover a wide range of data analytics :
  - Batch processing, stream processing, ML, Graph processing.
- Distributed computing → Operates distributed cluster, Scale horizontally accross multiple machines.
- Resilient Distributed Dataset (RDD) → Data structure of Spark. Distributed collection of object can be processed in parallel
- Data Frame & dataset API, In memory Computing, Integration Hadoop

2024/8/8 18:53

↳ Use cases of Apache spark :-

Data processing & ETL - Extract Transform Load

Real time data processing - Apache, Flume, Kafka, Kinesis. real time analytics and monitoring.

ML -

Interactive data Analysis -

Graph Processing -

Architecture of Apache Spark :-

↳ Driver - Controls the flow of data and execution of tasks.

cluster Manager - Allocates resources accross the cluster. builtin cluster manager, Apache Mesos, Hadoop YARN

Workers - Responsible for executing tasks assigned by the driver. Each worker node hosts one or more executors.

Executors - Executes individual tasks & keep data in its memory. They communicate with the driver & perform the actual computation

---

HBase :- Distributed, scalable, high performance NOSQL database built on (HDFS). Real time r/w access to large datasets, designed to handle large amount of sparse data. 25/7/24

key features → Real time r/w access, Horizontal scalability, Column oriented storage, Integration with hadoop ecosystem, fault tolerance.

Architecture → Region server, Region, HMaster, Zookeeper

Use case → Real time data processing, Large scale data storage

Hive :- Data warehousing & SQL like query language built on top of Hadoop. Data analysis & quering using HiveQL or HQL similiar to SQL like language. Handles large datasets & integrates seamlessly with Hadoop.

Feature → SQL like interface, Data warehouse infra structure, Integration with hadoop, Supports various data formats, extensible

Architecture → Metastore, Driver, Execution engine Client Interface

Use cases → Data Warehousing, Batch processing

HBase	Hive
→ NoSQL DB for real time access to large datasets.	→ SQL like language HiveQL (HQL) for batch processing & quring large datasets.
→ fast read/write capabilities	→ Built on top of Hadoop
→ Built on top of HDFS	

Airflow :- Open source platform to programatically author, schedule, monitor workflows. Used to define DAGs (Directed Acylic Graphs) of tasks. Enabling the orchestration & management of complex data pipelines

2024/8/8 18:53

Features :- DAGs (Directed Acylic Graphs), Tasks, Operators (Python Operator, Bash Operator, SQL operator Sensor Operator) scheduling, Task dependencies, Extendibility, Monitoring logging, Retry & alert Machines, Scalability, Integration with cloud and external services,

Uses : ETL pipeline, Data analytics & ML, Data Integration, Data quality & monitoring, Batach processing

→ from pyspark.sql import

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName().getOrCreate()
df = spark.read.csv("      ", header=True)
df.show()
```

1 → DataSource API → read
2 → SparkContext read
3 → Pandas to RDD

```
from pyspark import SparkContext
sc = SparkContext("  ", "  ")
rdd = sc.textFile("  ")
for line in rdd.take(5)
    print(line)
sc.stop()
```

(1)
```
from pyspark.sql import SparkSession, Row
spark = SparkSession.builder.appName("  ").getOrCreate()
data1 = spark.read.csv("  ", hea=True)
data2 = spark.CreateDataFrame([Row(  ,  )
                               Row(  ,  )]
from pyspark.sql.Funtions import col
unique_ele = spark.groupby("  ").count().filter(col()>1)
```

2024/8/8 18:54

```
from pyspark import SparkContext
sc = SparkCont()
rdd = sc.textfile()
```

```
from pyspark.sql import SparkContext
sc = SparkContext()
```

↳ # DataFrame API

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName().getOrCreate()
from pyspark.sql import Row
data = [Row(id= , name= ),
        Row(id= , name= ),
        Row(id= , name= )]
df1= spark.createDataFrame(data)
df1.show()

df2 = spark.read.csv(" ", header = True)
df2.show()
unique_words = df2.groupby(" ").count().filter(col(" ")>1)
                .dropduplicates
```

↳ ~~from~~ # API

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.AppName().getOrCreate()
from pyspark.sql import Row
data = [Row(ids:1, none="a"), Row( ), Row( )]
df = spark.createDataFrame(data)
df.show()
from pyspark.sql.functions import col
dup_df_1 = df.groupby("name").filter(col("none")>1)
            .count().
drop_df=df.dropDuplicates("IDs?", "none")
```

2024/8/8 18:54

↳ # Text fi—

```
from pyspark import SparkContext
sc = SparkContext("local", " ")
rdd = sc.parallize([" "," "," "]).rdd
for i in rdd.collect():
    print(i)
words = rdd.flatMap(lambda line: line.split(" "))
unique = words.distinct()
words_rdd = words.collects
for i in words_rdd:
    print(i)
```

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName().getOrCreate()
#data = spark.read.csv("  ")
#data.show()
data = pd.DF({'':[],"":[]})
df = spark.createDataFrame(data)
d)
```

```
↳ from pyspark.sql import SparkSession
spark = SparkSession.builder.appName().getOrCreate()
from pyspark.sql import Row
data = [Row(),Row(),Row()]
df_1 = spark.createDataFrame(data)
dup_data = df1.groupby("").count().filter(col()>1)
drop_data = df1.dropDuplicates(["","  "])
drop_data.write.csv("  ")
```

```
↳ from pyspark.sql import SparkSession
import pandas as pd
$park = SparkSession.builder.appName().getOrCreate()
data = pd.DataFrame({})
rdd = Spark.createDataFrame(data).rdd
dup_rdd = rdd.distinct()
df = spark.createDataFrame(data)
dup_df = df.groupBy("").count().filter(col()>1)
drop_dup = df.dropDuplicates(["","  "])
```

2024/8/8 18:55

```
↳ from pyspark import SparkContext
sc = SparkContext()
data = ["  ","  "]
words_rdd = sc.parallizize(data)
words = .flapMap(lambda l:l.split(" "))
dup_words = words.distinct()
words_list = words.collect()
```

Hadoop & Components :-

↳ HDFS :- ↑ throughput access to application data.
- splits files into large blocks & distribute accross nodes.

YARN :- Schedules jobs and manages clusters resources, Allows multiple data processing engines to run on single hadoop cluster

MapReduce :- 2 main tasks Map (Processes & filter data)
Reduce (Aggregates & Summerisses)

Hadoop vs Traditional DB :-

↳ RDBMS handle

DFS (Distributed File Systems) :-

↳ Data stored accross multiple servers & locations, ↑ availability, reliability & scalability.

↳ Distributed environment, enabling efficient data processing using tools like MapReduce, YARN & Hadoop components.

Components of HDFS : NameNode & DataNode

↳ Namenode – Master Node manages files system namespace
– Regulates access to files

↳ Datanode – Worker/slave nodes that stores & retrieve block when they are told.
– Report backs to Namenode with block that they are storing. Periodically.

HDFS daemons :-

↳ NameNode → Manages meta data & file system namespace. It keep on tracking.

DataNode → Daemons that runge strage on individual nodes.
r and w request from a client

2nd NM → Merging namespace image & edit log to reduce load on NN.

2024/8/8 10:44

Installation :-

- Prerequisites : Hadoop cluster installed & running
  Zookeeper ensemble configured & running
- Steps :
  - Download & Extract HBase
  - Configure HBase
  - Start HBase

Operations :-

- Creating a Table - create
- Listing Tables - list
- Inserting data - put
- Retrieving data - get
- Scanning data - scan
- Deleting data - delete
- Dropping table - disable
  drop
- counting rows
  - count

HBase commands :-

- hbase shell - starting the shell
  listing comands - help

Java Client API for HBase :-

- Establishing connection - HBaseConfiguration.create()
  .createConnection()
- CRUD operations -
  - Create : .getTable() , .addColumn()
  - Read : .get() , .getValues()
  - Update : HBase overwrite the value
  - Delete : .addcolumn() , .delete()

Admin API :-
- Creating : - getAdmin() , - addFamily() , - createTable()
- Listing tables : - listTables()
- Disabling & Dropping Tables : - disableTable
- deleteTable

↳ HBase - Scan, Count, Truncate
(scan , count , truncate)

↳ HBase security
Authentication : Kerberos
Authorization : Access Control (Defining permissions)
Encryption : Data encryption (Encrypt using HDFS encryption zones)
Network encryption (Encrypt data in transit TLS/SSL)

↳ Robust support for storing & processing large datasets with real-time r/w access.

HIVE

Features : SQL like Querying Language, schema on Read, Integration with Hadoop, support for diff.. storage types.

Components : UI, MetaStore, HiveQL process Engine, Execution Engine, HDFS

Installation :
- Prerequisite : Java JDK, Hadoop, SSH

Datatypes : string, int, bigint, Boolean, float, double, Timestamp, Date

Complex types : Array, Map, Struct, UnionType

String functions : CONCAT(), Length(), upper(), lower()
Date functions : Current_date(), Date_Add(), date_sub()
Hive tables : Managed tables (manages data & metadata), external tables (manages metadata only)

Partions & Buckets :
Partions → Divides tables into parts based on values of particular column.
Buckets → Divided into more managable parts
Used for more effiecne sapling & joins.

Storage formats :- Text file, sequence file,
- Text files : Default storage format (text_table)
- Sequence files : seq_table
- ORC : (Optimized Row Columnar)
- Parquet : parquet_table

Importing data :- LOAD DATA command by creating external tables.

Querying Data :-
→ Sorting data : Order by, sort by, Distribyte by
↳ Aggregating data : Count, sum, avg, min, max

Map Reduce scripts :- Hive can translate HiveQL into MapReduce jobs.

Joins & Subqueries :- Selet, from, whose, Inner join, subquery

View are virtual tables creating using the result of a query.

2024/8/8 17:08

Map and Reduce Side joins to Optimize Query

Map and Reduce side joins to optimize Query.

map side join : smaller dataset fit into memory.

Reduce side join : Default join in HIVE, suitable for large datasets.

Data manipulation with HIVE :-

Inserting data, Updating data, deleting data

UDFs (User Defined Functions) :-

↳ Creates custom functions to extend HiveQL capabilities.
Can be written in Java & other languages.
- Creates Java class extending UDF.
- Implemat the evaluate method
- Compile & add the JAR to HIVE.

Custom mapReduce in HIVE

↳ Scripts can be integrated into HIVE queries (TRANSFORM).

Writing HQL scripts :-

↳ HQL allows for the batch processing of HIVE queries.
HQL script is a text file containing HIVEQL commands.

For executing Hive : hive -f script.hql

Hadoop :- Open Source, to handle large datasets. Allows for storage & processing of big data.
components : HDFS, YARN, MapReduce

HDFS :- Provides scalable & reliable data storage designed to span large cluster of commodity servers
Architecture : NNode, DNote, 2° Node, HDFS daemons (Name Node & Data Node)

HBASE :- Distributed, scalable, large amount of sparse storeses and manage
Architecture : HBase master, Region Servers, Zoo Keeper
HFile

2024/8/8 17:08

Hive : Data ware house built on top of Hadoop.
— Data summarises, querying & analysis.

Architecture —
Metastore
Hive server
Hive driver
Execution engine

↳ Hadoop : Provides distributed storage (HDFS) & processing data (Map Reduce)

HDFS : Handles large datasets with high reliability & scalability

HBase : NoSQL DB built on top of HDFS, realtime read/write access to large dataset

Hive : Data warehousing that provides SQL like interface for Querying & analyzing data stored in Hadoop.

Apache Spark :—

↳ open source, distributed computing system. Provides interface for programming entire cluster with implicit data parallelism & fault tolerance.

→ cluster with implicit data parallism & fault tolerance

→ To overcome limitations of Hadoop MapReduce framework and provide faster, more efficient data processing.

Linking with Spark →

↳ Hadoop
Apache Hive
Apache HBase
Apache Kafka

Initializing spark :-

↳ sparkSession.builder(

RDD (Resilient Distributed Datasets):

↳ Properties : (Immutable, Fault tolerance, Parallel processing)

. RDDs fundamental data structures of Spark.

creating RDDs :- . parallelize(data)

↳ RDDs : Low Level APIs provide transformatnd actions.

↳ ej spark.sql import SparkSession

sparksession.builde().appName().getOrCreate()

sc = spark.sparkContext

data = [, , , ,]

rdd = sc.parallelise(data)

rdd = sc.textfile(" ___ ")

Shuffle operations :-

↳ reduceBy key & groupBy key

RDD persistence :-

↳ It can be persisted in memory or disk to avoid re computation.

Removing data :- To unpersist data [rdd.unpersist()]

Shared Variables :-

- Broadcast variable : read only value to all nodes
- Accumulator : variables added to from workes & read only by drivers.

Deploying to a cluster :-

- standalone mode : spark manages own cluster
- YARN : Spark appl.. on Hadoop cluster
- Mesos : cluster manager used to run spark.

2024/8/8 17:09

Map Reduce with Spark :-

Spark [with hadoop] → Hadoop HDFS for storage
YARN for resource management

[without Hadoop] → Use local file systems or other storage solutions like S3.
Spark's standalone cluster manager.

↳ Storage : with Hadoop — HDFS
without Hadoop — Local file system or cloud st

Resource Management : with Hadoop — YARN
without Hadoop — Standalone or Mesos

Data Preprocessing :- step in data analysis pipeline.

Transforms raw data into a clean & usable format.

steps → Data cleaning
Data Transformation
Data Reduction [(PCA), Sampling]
Feature Engineering

EDA (Explanatory Data Analysis) :-

↳ Analyzing data sets to summarize their main characteristics, often using visual methods.
Understanding the data & uncovering patterns, anamolies, relationships.

- Descriptive Statistics [mean, median, mode, var, SD]
- Data visualization [Plots]
- Correlation Analysis [Relationship b/w variables using correlation Coefficients]
- Outlier detection [Identifying the outliers in data

Apache kafka :-

↳ Designed for high throughput, low latency in data streaming. For build a real time data pipelines & streamin applications.

Spark Streaming Architecture →

- → Discretized Streams (DStreams) [Spark streams represent Streams of data in series of RDDs]
- → Recievers [To ingest data from sources like kafka, Flume & others]
- → Processing [Each batch data processed by Spark jobs, can perform transformations & actions on data]
- → Output Operation [Processed data can be written to various output sinks (HDFS, DB,

↳ Apache spark & kafka for building scalable, real time data processing piplines.
Combining capabilities of Spark for processing

kafka connect API :-

- Source Connectors (Import data $\xrightarrow{\text{from}}$ Ext.. syst into kafka
- Sink Connectors (kafka $\xrightarrow{\text{export}}$ Ext.. systems)

↳ Apache kafka :- written in java & scala.

- Messaging, Storage, Processing
- Topics, Partitions,
- Producers & Consumers
- Brokers
- Zoo Keeper

Apache Spark

2024/8/8 17:09

21/7/24

Apache kafka :-

↳ Open source stream processing platform. Written in Scala & Java.

Functionalities : Messaging, storage, Processing

Concepts : Topics – Categories where records are stored
Partitions – Segments of topics that allow parallel processing
Producers – Data written to kafka topics
Consumers – Data Read from kafka topics
Brokers – Kafka made of brokers handles storage & retrival of records.
Zookeeper – Manages & coordinats kafka brokers.

Apache Spark :- Open source unified analytics engine

for large scale data processing.
Provides high leve API in java, scala, Python, R
Components : Spark Core, Spak SQL, spark Streaming, MLlib, Graph X

Spark SQL :- Spark module for structural data processing.

It provides DataFrame API & SQL queries.

Features : DF & DataSets → DF (Data organized into name colums)
Dataset (Provides benifits of RDD with optimized adv of Spark SQL)

SQL Integration → Allows executing SQL queries along-side complex analytic algorithms.

Catalyst Optimizer → Query Optimizer for executing queries efficiently.

DataSource API → Allows Integration with variety of data source (JSON, Parquet Avro)

Spark MLlib :- ML lib built on top of spark.

Features : Algo – Classification, Regression, Clustering, Collaborative filtering etc.–
Featurization – Featur extraction, ET, dimensional Reduction

2024/8/8 17:09

Pipelines – Tools for constructing, evaluating & tuning ML
Utilities – statistics, data handling & other utilities.

Predictive analysis with Spark :-

↳ using historical data to predict future outcome

steps – Data Preprocessing (cleaning & Transforming)
Feature Engineering (Creating new feature from existing data)
Model building (LR, DT)
Model Evaluation (Metrics like acuracy, precision, recall)
Model deployment (Integrating model into production systems to make predictions)

Workflow in Spark :-

↳ Initialize Spark Context → Load & preprocess data → Feature Engineering → Split data → Train Model → evaluate Model → Deploy model

Linking kafka with Spark :-

↳ kafka used as source and Sink for Spark streaming application.

Steps : – Setting up kafka
– Creating Spark Streaming Context
– Consuming data from kafka
– Processing data
– Producing data to kafka

→ Apache Kafka & Apache Spark powerfull tools for realtime, scalable, data processing pipelines.
→ Spark SQL and MLlib provides robust robust frameworks for data processing & ML, enabling predictive analysis.
→ Integrating kafa and Spark, creates efficient Streaming applications that handles large volumes of data with low latency.

2024/8/8 17:09