

Hadoop Intro (refer U3)

RDBMS vs Hadoop:

Relational Database Management Systems (RDBMS) and Hadoop are both data storage and management systems, but they serve different purposes and have distinct architectures. Here's a comparative overview of RDBMS and Hadoop:

RDBMS (Relational Database Management System)

Architecture:

- Structured and based on a schema.
- Uses tables with rows and columns.
- SQL (Structured Query Language) for data manipulation and queries.

Data Handling:

- Suitable for transactional data.
- Ensures ACID (Atomicity, Consistency, Isolation, Durability) properties.
- Best for structured data with predefined schemas.

Scalability:

- Vertically scalable (upgrading the existing hardware).
- Limited horizontal scalability.

Performance:

- High performance for complex queries and transactions.
- Optimal for read and write operations in a structured data environment.

Examples:

- MySQL, PostgreSQL, Oracle, SQL Server.

Use Cases:

- Online Transaction Processing (OLTP).
- Financial systems, banking, enterprise resource planning (ERP).

Hadoop

Architecture:

- Distributed file system (HDFS - Hadoop Distributed File System).
- Uses a cluster of commodity hardware.
- MapReduce for processing large data sets.

Data Handling:

- Designed for large-scale data storage and processing.

- Handles structured, semi-structured, and unstructured data.
- Not bound to a schema; schema-on-read approach.

Scalability:

- Horizontally scalable (adding more nodes to the cluster).
- Can scale to handle petabytes of data.

Performance:

- High throughput for batch processing.
- Suitable for big data analytics, but not optimal for real-time queries.

Examples:

- Apache Hadoop, Cloudera, Hortonworks.

Use Cases:

- Big data processing.
- Data warehousing, log analysis, machine learning, large-scale ETL (Extract, Transform, Load) operations.

Key Differences

1. **Data Structure:**
 - **RDBMS:** Structured data with predefined schema.
 - **Hadoop:** Can handle structured, semi-structured, and unstructured data.
2. **Scalability:**
 - **RDBMS:** Vertical scalability, limited horizontal scalability.
 - **Hadoop:** Highly scalable horizontally, can handle massive datasets.
3. **Processing:**
 - **RDBMS:** Real-time processing, ACID compliance.
 - **Hadoop:** Batch processing, not inherently ACID compliant.
4. **Use Case Fit:**
 - **RDBMS:** OLTP, complex queries, transactional systems.
 - **Hadoop:** Big data analytics, large-scale data processing, ETL jobs.
5. **Cost:**
 - **RDBMS:** Can be costly due to licensing fees and hardware requirements.
 - **Hadoop:** Generally lower cost, uses commodity hardware.

Conclusion

RDBMS and Hadoop serve different purposes and choosing between them depends on the specific requirements of the project. For transactional systems requiring real-time processing and structured data, RDBMS is typically the best choice. For large-scale data processing, analytics, and handling diverse data types, Hadoop is more suitable.

Hadoop Overview:

- Hadoop Overview Hadoop is an Apache open source framework written in Java that allows distributed processing of large datasets across clusters of computers using simple programming models.
- The Hadoop framework application works in an environment that provides distributed storage and computation across clusters of computers.
- Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.
- **Hadoop Architecture**
- At its core, Hadoop has two major layers namely:
 - Processing/Computation layer (MapReduce)
 - Storage layer (Hadoop Distributed File System)

(refer U3 for Map reduce and HDFS topics)

Hadoop distributions

Hadoop distributions are versions of the Hadoop ecosystem provided and maintained by various organizations. These distributions often include Hadoop and other related tools, along with enhancements, support, and additional features. Here are some of the major Hadoop distributions:

1. Apache Hadoop

- **Provider:** Apache Software Foundation
- **Description:** The original, open-source distribution of Hadoop. It includes HDFS (Hadoop Distributed File System) and YARN (Yet Another Resource Negotiator), along with the Hadoop MapReduce framework.
- **Features:** Basic components of Hadoop, community support, frequent updates.

2. Cloudera

- **Provider:** Cloudera Inc.
- **Description:** A comprehensive big data platform that integrates Hadoop with various other big data tools and services.
- **Features:**
 - Enterprise-grade security, governance, and management.
 - Cloudera Manager for cluster management.
 - Integration with other Cloudera products like Cloudera Data Science Workbench and Cloudera DataFlow.
 - Offers support, training, and consulting services.

3. Hortonworks Data Platform (HDP)

- **Provider:** Hortonworks (now part of Cloudera)
- **Description:** Another enterprise-grade Hadoop distribution designed for managing large volumes of data across clusters of commodity hardware.
- **Features:**
 - Open-source, with a focus on data governance, security, and integration.
 - Provides tools like Apache Ambari for cluster management.
 - Strong focus on supporting the open-source ecosystem.

4. MapR (Now part of HPE)

- **Provider:** Originally MapR Technologies, now part of Hewlett Packard Enterprise (HPE)
- **Description:** A distribution that enhances Hadoop with its own file system (MapR FS) and database capabilities (MapR DB).
- **Features:**
 - High performance and reliability.
 - Provides additional features like multi-tenancy, high availability, and distributed namespace.
 - Integration with Kubernetes for containerized environments.
 - Includes enterprise-grade security and data governance tools.

5. Amazon EMR (Elastic MapReduce)

- **Provider:** Amazon Web Services (AWS)
- **Description:** A cloud-based Hadoop distribution that allows users to easily process vast amounts of data using Hadoop and other big data frameworks like Apache Spark.
- **Features:**
 - Fully managed, scalable, and cost-effective.
 - Integrated with other AWS services such as S3, EC2, RDS, and DynamoDB.
 - Automatic provisioning, management, and scaling of clusters.
 - Pay-as-you-go pricing model.

6. Google Cloud Dataproc

- **Provider:** Google Cloud
- **Description:** A fast, easy-to-use, fully managed cloud service for running Apache Spark and Apache Hadoop clusters.
- **Features:**
 - Quick cluster startup and shutdown.
 - Integration with other Google Cloud services like BigQuery, Cloud Storage, and Bigtable.
 - Managed and scalable infrastructure.
 - Pay-per-second billing.

7. Microsoft Azure HDInsight

- **Provider:** Microsoft Azure
- **Description:** A cloud service that makes it easier to use the most popular open-source frameworks like Hadoop, Spark, Hive, LLAP, Kafka, and more.
- **Features:**
 - Fully managed, full-spectrum, open-source analytics service for enterprises.
 - Integration with Azure services such as Azure Data Lake Storage, Azure Synapse Analytics, and Power BI.
 - Security features like integration with Azure Active Directory and encryption.
 - Scalable and flexible with various pricing options.

Conclusion

The choice of a Hadoop distribution often depends on your specific needs, such as on-premise versus cloud deployment, the need for enterprise features, support requirements, and the preferred integration with other

tools and services. Each distribution has its own set of features and strengths, making it important to evaluate them based on your project requirements.

HDFS (refer U3)

HDFS Daemons

HDFS (Hadoop Distributed File System) is the primary storage system used by Hadoop applications. It follows a master-slave architecture, where data is distributed across multiple nodes to ensure fault tolerance and high throughput. HDFS operates using several key daemons (services), each with specific roles:

1. NameNode

- **Role:** The master daemon that manages the metadata and namespace of HDFS.
- **Functions:**
 - Maintains the directory tree of all files in the file system.
 - Keeps track of all the files and blocks in the file system, along with their locations.
 - Manages the filesystem namespace and regulates access to files by clients.
 - Handles the block reports sent by DataNodes to ensure data consistency.
- **Components:**
 - **FsImage:** The persistent checkpoint of the file system metadata.
 - **EditLog:** A transaction log that records changes to the file system metadata.

2. Secondary NameNode

- **Role:** A helper daemon for the NameNode.
- **Functions:**
 - Merges the EditLog with the FsImage to create a new checkpoint.
 - Helps in reducing the load on the NameNode by periodically combining the namespace image with the edit log.

3. DataNode

- **Role:** The worker daemons that store and manage the actual data blocks.
- **Functions:**
 - Stores HDFS data blocks on the local file system of the DataNode.
 - Handles read and write requests from HDFS clients.
 - Periodically sends block reports and heartbeats to the NameNode to report on its status and the blocks it holds.
 - Responsible for replicating data blocks to ensure data redundancy.

4. JournalNode

- **Role:** Part of the HDFS high availability (HA) architecture.
- **Functions:**
 - Helps in implementing the QJM (Quorum Journal Manager) for the active and standby NameNodes.

- Stores the edit logs for the NameNode in an HA setup, allowing the standby NameNode to keep up-to-date with the active NameNode.

5. CheckpointNode

- **Role:** Combines the functionalities of both the Secondary NameNode and the NameNode.
- **Functions:**
 - Downloads the current FsImage and EditLog from the NameNode.
 - Merges them and uploads the new checkpoint back to the NameNode.
 - This process helps in minimizing the EditLog size and reducing the startup time for the NameNode.

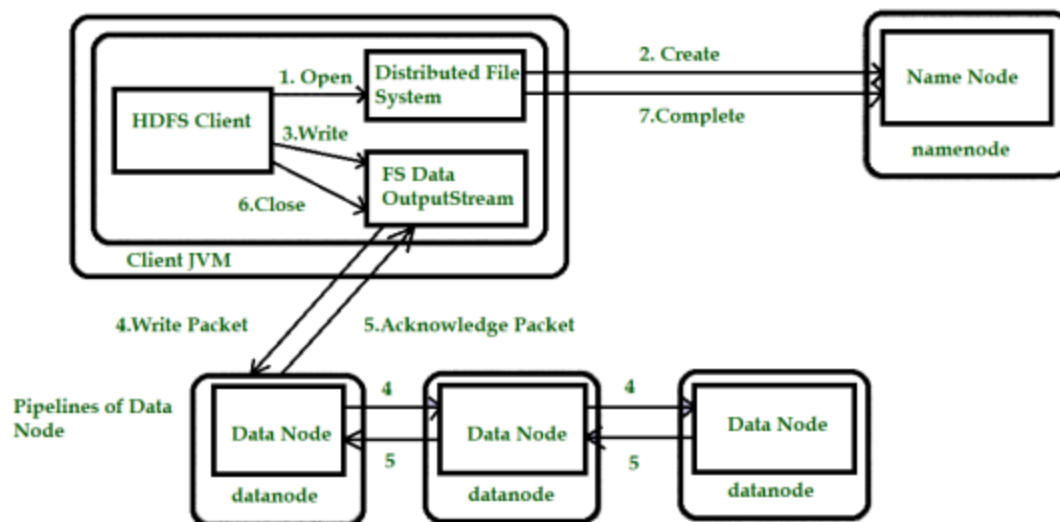
HDFS Daemons Summary:

Daemon Name	Role	Key Functions
NameNode	Master daemon	Manages metadata, directory tree, file system namespace, block locations, handles client access
Secondary NameNode	Helper daemon for NameNode	Merges FsImage and EditLog, creates checkpoints
DataNode	Worker daemon	Stores data blocks, handles read/write requests, reports to NameNode
JournalNode	Part of HA architecture	Stores edit logs, supports Quorum Journal Manager for HA
CheckpointNode	Combines Secondary NN and NameNode	Downloads and merges FsImage and EditLog, uploads new checkpoints to NameNode

Conclusion

HDFS daemons work together to provide a robust, fault-tolerant, and scalable distributed file system. Understanding the roles and functions of these daemons is crucial for managing and maintaining an HDFS cluster effectively.

Anatomy of File Write and Read:



Understanding the anatomy of file write and read operations in HDFS involves detailing how data is written to and read from the Hadoop Distributed File System. Here's a step-by-step breakdown of these processes:

File Write Operation in HDFS

1. **Client Request:**
 - The client initiates a file write request to the HDFS.
 - The file is split into blocks (default size is 128MB or 256MB).
2. **Communication with NameNode:**
 - The client contacts the NameNode to get the metadata information and to determine which DataNodes will store the blocks.
 - The NameNode responds with a list of DataNodes for each block.
3. **Block Allocation:**
 - The client starts writing data to the first block and establishes a pipeline with the selected DataNodes.
 - Typically, each block is replicated to three DataNodes to ensure fault tolerance (default replication factor is 3).
4. **Data Pipeline:**
 - The client writes the first block to the first DataNode in the pipeline.
 - The first DataNode forwards the block to the second DataNode.
 - The second DataNode forwards the block to the third DataNode.
5. **Block Replication:**
 - This process continues until all blocks are written and replicated across the DataNodes.
 - Each DataNode sends an acknowledgment back up the pipeline after successfully writing the block.
6. **Completion:**
 - After all blocks are written and replicated, the client informs the NameNode.
 - The NameNode updates its metadata to reflect the new file and its block locations.

File Read Operation in HDFS

1. **Client Request:**

- The client initiates a file read request to the HDFS.
- 2. **Communication with NameNode:**
 - The client contacts the NameNode to get the block locations for the requested file.
 - The NameNode responds with the list of DataNodes that contain the blocks of the file.
- 3. **Reading Data:**
 - The client contacts the closest DataNode (based on network topology) that has the first block of the file.
 - The DataNode streams the block data to the client.
 - The client reads each block sequentially by contacting the respective DataNodes.
- 4. **Block Read Sequence:**
 - The client continues to read the subsequent blocks from the corresponding DataNodes until the entire file is read.
- 5. **Completion:**
 - The read operation completes when the client has received all the blocks of the file.
 - The client assembles the blocks to reconstruct the file.

Diagram of File Write and Read Operations

File Write Operation

1. **Client -> NameNode:** Requests file write and block locations.
2. **NameNode -> Client:** Provides list of DataNodes for block storage.
3. **Client -> DataNode1 -> DataNode2 -> DataNode3:** Writes blocks through the pipeline.
4. **DataNode3 -> DataNode2 -> DataNode1 -> Client:** Acknowledgment of block writes.
5. **Client -> NameNode:** Confirms block write completion.
6. **NameNode:** Updates metadata.

File Read Operation

1. **Client -> NameNode:** Requests file read and block locations.
2. **NameNode -> Client:** Provides list of DataNodes with block locations.
3. **Client -> DataNode1:** Reads first block.
4. **Client -> DataNode2:** Reads second block.
5. **Client -> DataNode3:** Reads third block.
6. **Client:** Reconstructs file from blocks.

Summary

- **Write Operation:** Involves client requests, block allocation, data pipelining, block replication, and metadata updates.
- **Read Operation:** Involves client requests, retrieving block locations, sequentially reading blocks, and reconstructing the file.

Understanding these processes helps in managing HDFS effectively, optimizing performance, and ensuring data reliability and fault tolerance.

Name Node:

The NameNode is a critical component of the Hadoop Distributed File System (HDFS) that serves as the master server responsible for managing the filesystem namespace and regulating access to files by clients. Here's an in-depth look at its functions, components, and significance:

Functions of the NameNode

- 1. Filesystem Namespace Management:**
 - Maintains the directory tree of all files and directories in the HDFS.
 - Handles operations such as file creation, deletion, renaming, and moving.
- 2. Metadata Management:**
 - Stores metadata for the file system, including the structure of the directories and the mapping of file blocks to DataNodes.
 - Maintains information about the locations of blocks and their replication.
- 3. Block Management:**
 - Keeps track of the status and health of DataNodes.
 - Decides on block placement, replication, and re-replication based on configured policies.
 - Coordinates block creation, deletion, and replication according to the replication factor.
- 4. Client Interaction:**
 - Provides clients with metadata and information about DataNodes where file blocks are stored.
 - Handles client requests for reading and writing files, directing them to the appropriate DataNodes.
- 5. Fault Tolerance:**
 - Monitors DataNodes through heartbeats and block reports.
 - Automatically re-replicates blocks from failed DataNodes to ensure data redundancy and reliability.

Components of the NameNode

- 1. FsImage (Filesystem Image):**
 - A persistent, point-in-time snapshot of the entire file system metadata.
 - Stored on disk and loaded into memory when the NameNode starts.
 - Represents the state of the filesystem at a particular moment.
- 2. EditLog:**
 - A write-ahead log that records every change to the filesystem metadata.
 - Logs operations like file creation, deletion, and block allocation.
 - Changes are appended to the EditLog until a checkpoint is created.
- 3. Namespace Image (NNImage):**
 - In-memory representation of the filesystem namespace and metadata.
 - Combines the FsImage and EditLog to reflect the current state of the filesystem.

Conclusion

The NameNode is the heart of HDFS, responsible for managing metadata, regulating client access, and ensuring data reliability through block replication and fault tolerance mechanisms. Its role in the Hadoop ecosystem is vital for the smooth and efficient functioning of the distributed file system. Understanding the NameNode's functions and components is key to managing and optimizing an HDFS cluster effectively.

Secondary Name Node:

The Secondary NameNode is a crucial component in the Hadoop Distributed File System (HDFS), but it is often misunderstood. Unlike its name might suggest, it is not a backup for the NameNode. Instead, it plays an important role in managing the filesystem metadata.

Functions of the Secondary NameNode

1. **Checkpointing:**
 - The primary role of the Secondary NameNode is to create checkpoints of the filesystem metadata. This involves merging the edits log with the current state of the filesystem (FsImage) to produce a new, updated FsImage.
2. **Metadata Management:**
 - It helps in reducing the size of the edits log by periodically creating a new checkpoint. This helps in ensuring that the NameNode does not run out of memory, as the edits log can grow significantly over time with each change made to the filesystem.
3. **Reducing NameNode Restart Time:**
 - By maintaining an up-to-date FsImage, the Secondary NameNode helps in reducing the time it takes for the NameNode to restart after a failure. Without the Secondary NameNode, the NameNode would have to process a large edits log, which could take a considerable amount of time.

How the Secondary NameNode Works

1. **Fetching Metadata:**
 - Periodically (based on a configured interval), the Secondary NameNode downloads the current FsImage and the edits log from the NameNode.
2. **Merging Edits:**
 - It merges the edits log into the FsImage to create a new FsImage that reflects the latest state of the filesystem.
3. **Uploading the Checkpoint:**
 - The new FsImage is uploaded back to the NameNode. This updated FsImage can replace the older one, reducing the number of edits the NameNode needs to apply during a restart.

Components Involved

- **FsImage (Filesystem Image):** A snapshot of the filesystem metadata at a certain point in time.
- **Edits Log:** A log of all the changes made to the filesystem since the last checkpoint.

Process Workflow

1. **Initiation:**
 - The Secondary NameNode initiates the checkpointing process at regular intervals.
2. **Downloading:**
 - It downloads the current FsImage and edits log from the NameNode.
3. **Merging:**
 - The Secondary NameNode merges the downloaded edits log into the FsImage to create a new, consistent state of the filesystem.

4. **Uploading:**

- The updated FsImage is uploaded back to the NameNode, which can then discard the old FsImage and edits log.

Importance of the Secondary NameNode

1. **Memory Management:**

- Helps manage the memory usage of the NameNode by preventing the edits log from growing too large.

2. **Efficiency:**

- Increases the efficiency of the NameNode by ensuring that it only needs to apply a smaller number of edits during a restart.

3. **Reliability:**

- By regularly creating checkpoints, the Secondary NameNode contributes to the overall reliability and stability of the HDFS.

Misconceptions

● **Not a Backup:**

- The Secondary NameNode is often mistaken as a backup for the NameNode. However, it does not provide high availability or failover capabilities. Its primary role is to assist with checkpointing and not to take over in case of a NameNode failure.

Conclusion

The Secondary NameNode is an essential component in maintaining the efficiency and stability of HDFS by handling the critical task of checkpointing the filesystem metadata. It ensures that the NameNode operates smoothly and can restart quickly, thus contributing to the robustness of the Hadoop ecosystem.

Data Nodes:

The DataNode in the Hadoop Distributed File System (HDFS) is a vital component responsible for storing and managing the actual data blocks. It acts as a worker node in the HDFS architecture, handling the bulk of the data storage and retrieval tasks. Here's a detailed look at the role, functions, and importance of the DataNode:

Functions of the DataNode

1. **Data Storage:**

- DataNodes store HDFS data blocks on the local file system of the node.
- Each block is typically replicated on multiple DataNodes to ensure fault tolerance.

2. **Block Management:**

- Manages the storage of blocks, including the replication of blocks according to the configured replication factor (default is 3).
- Ensures that the data blocks are available and accessible.

3. **Data Retrieval:**

- Handles read requests from clients, retrieving the requested blocks and streaming the data back to the client.
- Also processes write requests, storing new blocks received from clients or other DataNodes.

4. **Heartbeat and Block Reports:**

- Periodically sends heartbeat signals to the NameNode to indicate that it is alive and functioning.
- Sends block reports to the NameNode, detailing the blocks it is storing. This helps the NameNode keep track of the overall block distribution across the cluster.

5. **Replication Management:**

- Participates in the block replication process. When the NameNode determines that a block needs to be replicated, it instructs the DataNodes to create additional copies of the block.

6. **Error Detection and Reporting:**

- Detects any errors in the storage disks and reports these errors to the NameNode.
- If a block becomes corrupted, the DataNode reports this to the NameNode, which can then initiate a re-replication of the corrupted block from other replicas.

DataNode Workflow

1. **Startup:**

- When a DataNode starts up, it registers with the NameNode and provides information about the blocks it stores through an initial block report.

2. **Client Interactions:**

- **Write Operation:**
 - The client contacts the NameNode for block locations.
 - The client then writes data to the designated DataNodes, which store the blocks and replicate them as instructed.
- **Read Operation:**
 - The client requests the block locations from the NameNode.
 - The client reads the required blocks from the corresponding DataNodes.

3. **Ongoing Operations:**

- DataNodes continually send heartbeat signals and block reports to the NameNode.
- Perform block replication, deletion, and re-replication as instructed by the NameNode.
- Handle read and write requests from clients.

Importance of the DataNode

1. **Scalability:**

- By distributing data storage across multiple DataNodes, HDFS achieves horizontal scalability, allowing it to handle vast amounts of data efficiently.

2. **Fault Tolerance:**

- The replication of data blocks across multiple DataNodes ensures that data remains accessible even if some nodes fail.
- This redundancy is crucial for maintaining data integrity and availability.

3. **Performance:**

- DataNodes are designed to handle high-throughput read and write operations, providing efficient data access to clients.
- The distribution of data across nodes helps balance the load and optimize performance.

Conclusion

DataNodes are the backbone of HDFS, providing scalable, reliable, and high-performance storage for the Hadoop ecosystem. They work in coordination with the NameNode to manage data distribution, replication,

and retrieval, ensuring that the system can handle large datasets with fault tolerance and efficiency. Understanding the role and functioning of DataNodes is crucial for managing and optimizing an HDFS cluster.

Hadoop Architecture (Refer U3 and U4)

Hadoop Configuration:

Configuring Hadoop involves setting up various parameters and properties to ensure the Hadoop cluster operates efficiently and meets the requirements of your workloads. Here's an overview of the main configuration files and key parameters you'll need to consider when configuring a Hadoop cluster:

Key Configuration Files

1. core-site.xml

- Configures Hadoop core settings, including I/O settings.
- Important properties:
 - `fs.defaultFS`: Defines the default filesystem URI.
 - `hadoop.tmp.dir`: Specifies the base directory for temporary files.

2. hdfs-site.xml

- Configures HDFS-specific settings.
- Important properties:
 - `dfs.replication`: Sets the default block replication factor.
 - `dfs.namenode.name.dir`: Specifies the directory on the local filesystem where the NameNode stores metadata and the FsImage.
 - `dfs.datanode.data.dir`: Specifies the directory on the local filesystem where the DataNode stores its blocks.
 - `dfs.permissions`: Enables or disables HDFS permission checking.

3. mapred-site.xml

- Configures MapReduce-specific settings.
- Important properties:
 - `mapreduce.framework.name`: Specifies the framework for MapReduce, typically set to "yarn".
 - `mapreduce.job.tracker`: Defines the JobTracker location for non-YARN clusters (deprecated in favor of YARN).
 - `mapreduce.job.reduces`: Sets the default number of reducers for MapReduce jobs.

4. yarn-site.xml

- Configures YARN (Yet Another Resource Negotiator) settings.
- Important properties:
 - `yarn.resourcemanager.hostname`: Specifies the hostname of the ResourceManager.
 - `yarn.nodemanager.resource.memory-mb`: Sets the maximum memory available for containers on a NodeManager.
 - `yarn.scheduler.maximum-allocation-mb`: Defines the maximum memory allocation for a single container.

5. hadoop-env.sh

- Environment variables for Hadoop daemons.

- Important settings:
 - `JAVA_HOME`: Specifies the path to the Java installation.
 - `HADOOP_HEAPSIZE`: Sets the heap size for Hadoop daemons.

Conclusion

Configuring Hadoop involves setting up various parameters to ensure the cluster operates efficiently. Key configuration files include `core-site.xml`, `hdfs-site.xml`, `mapred-site.xml`, `yarn-site.xml`, and `hadoop-env.sh`. Properly configuring these files and understanding their roles are essential for managing and optimizing a Hadoop cluster.

Map Reduce Framework

The MapReduce framework is a core component of the Hadoop ecosystem, designed to process large datasets in a parallel and distributed manner. It follows a specific programming model that consists of two main tasks: Map and Reduce. Here's an in-depth look at the MapReduce framework:

MapReduce Programming Model

1. **Map Phase:**
 - The input data is split into independent chunks, processed in parallel by the mapper functions.
 - Each mapper processes a key-value pair and generates intermediate key-value pairs.
2. **Shuffle and Sort Phase:**
 - The framework sorts the intermediate data by key.
 - Intermediate data is then shuffled so that all values associated with the same key are grouped together.
3. **Reduce Phase:**
 - The reducer processes each group of intermediate key-value pairs.
 - The reducer consolidates the values associated with each key to produce the final output.

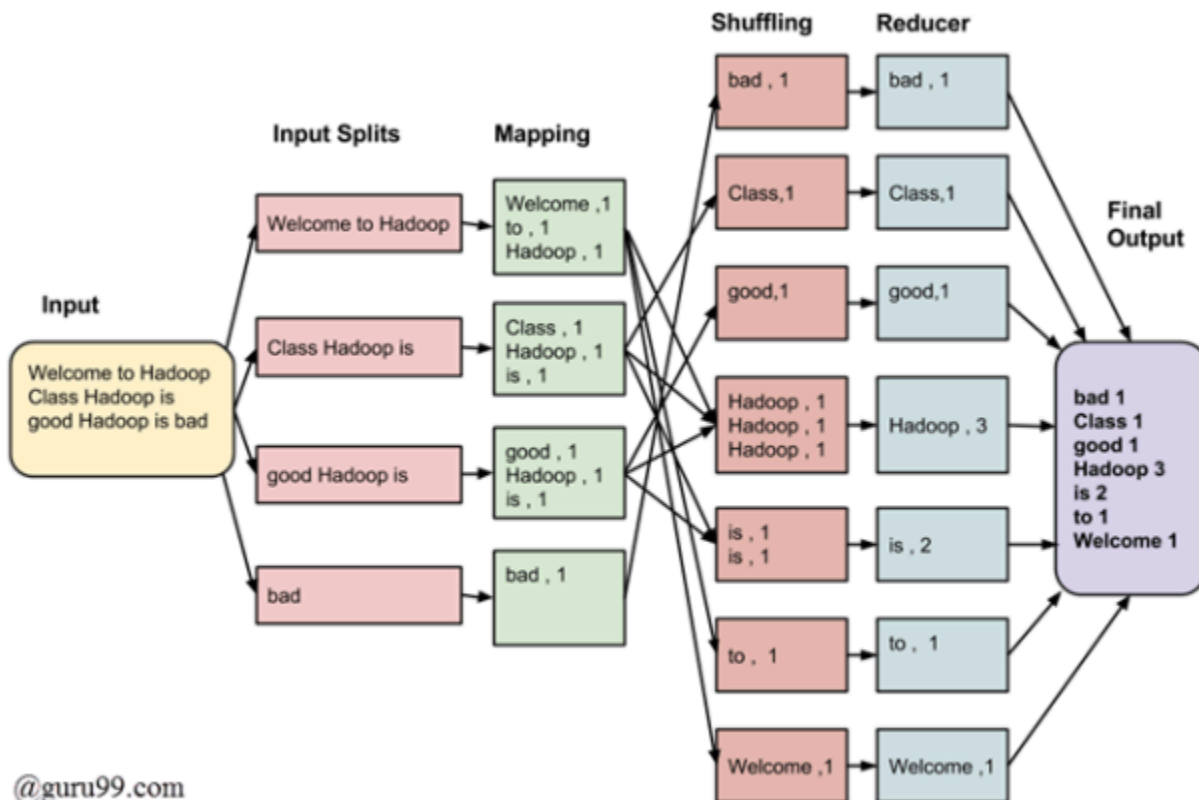
MapReduce Job Workflow

1. **Job Submission:**
 - The client submits a MapReduce job to the cluster.
 - The job configuration includes details about input and output locations, the mapper and reducer classes, and other necessary parameters.
2. **Input Splitting:**
 - The input data is split into fixed-size chunks called input splits.
 - Each split is processed by a separate mapper task.
3. **Mapping:**
 - Each mapper reads its input split and processes it to generate intermediate key-value pairs.
 - These pairs are written to local disk and partitioned for the shuffle phase.
4. **Shuffling and Sorting:**
 - The framework sorts the intermediate data by key and shuffles it across nodes so that all values associated with a particular key are brought together.
5. **Reducing:**
 - Reducer tasks read the sorted intermediate data.

- Each reducer processes the values for a particular key and writes the final output to HDFS.
6. **Output:**
- The final output of the reduce tasks is written to the specified output directory in HDFS.

Key Components of MapReduce

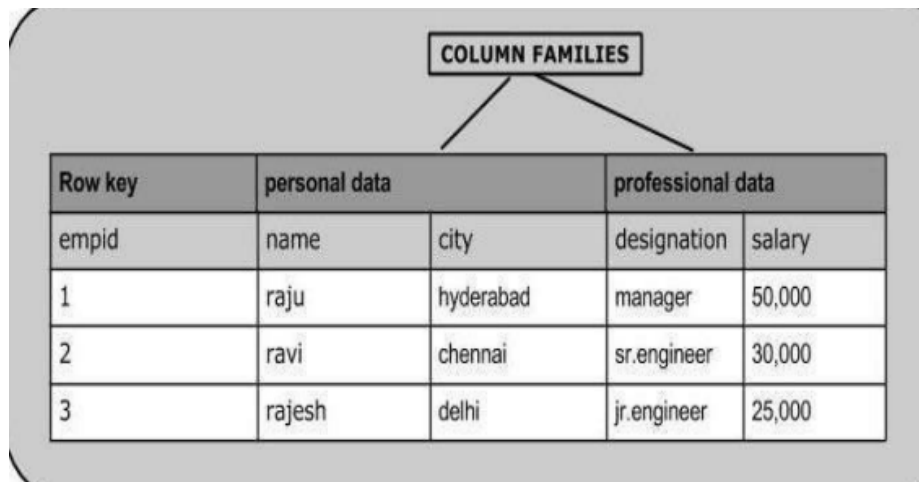
1. **JobTracker (Deprecated in YARN):**
 - Manages the job by scheduling tasks, monitoring them, and re-executing failed tasks.
 - Deprecated and replaced by the ResourceManager in YARN.
2. **TaskTracker (Deprecated in YARN):**
 - Executes individual tasks as directed by the JobTracker.
 - Deprecated and replaced by NodeManager in YARN.
3. **ResourceManager (YARN):**
 - Manages resources across the cluster.
 - Allocates resources to various applications, including MapReduce jobs.
4. **NodeManager (YARN):**
 - Manages resources on a single node.
 - Monitors resource usage (CPU, memory) and reports it to the ResourceManager.
 - Manages the lifecycle of containers, which run the mapper and reducer tasks.
5. **ApplicationMaster (YARN):**
 - Manages the execution of a single application (e.g., a MapReduce job).
 - Coordinates with the ResourceManager to allocate resources and with NodeManagers to execute tasks.



HBase

- HBase is a distributed, scalable, NoSQL database built on top of the Hadoop file system (HDFS). It's designed to handle large amounts of data across many machines.
- Applications such as HBase, Cassandra, couchDB, Dynamo, and MongoDB are some of the databases that store huge amounts of data and access the data in a random manner.
- Hbase is a column-oriented database.
- Column-oriented databases are those that store data tables as sections of columns of data, rather than as rows of data. Shortly, they will have column families.

Ex:



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

- Here's a breakdown of its key features and role in big data processing:

Key Features:

1. **Column-Family-Based Storage:** Unlike traditional relational databases that use rows, HBase stores data in columns. This allows for efficient retrieval and storage of large datasets where queries often involve only a subset of columns.
2. **Scalability:** HBase is designed to scale horizontally. You can add more machines (nodes) to the cluster to handle more data and higher loads.
3. **Random, Real-Time Read/Write Access:** HBase provides random, real-time access to data, making it suitable for applications that need quick access to large datasets.
4. **Integration with Hadoop Ecosystem:** HBase integrates with the Hadoop ecosystem, allowing it to leverage Hadoop's distributed storage and processing capabilities. For example, you can use HBase in conjunction with MapReduce, Hive, and Pig.
5. **Automatic Sharding and Distribution:** HBase automatically distributes data across nodes and handles the complexities of sharding (splitting data across multiple servers).

Role in Big Data Processing:

1. **Handling Large Datasets:** HBase is used to store and manage large datasets that are too big to fit into memory on a single machine. Its distributed nature allows it to handle petabytes of data.

2. **Real-Time Analytics:** It provides real-time read and write capabilities, making it useful for applications that require up-to-date data and low-latency access, such as recommendation engines and monitoring systems.
3. **Integration with Data Processing Frameworks:** HBase can be used alongside other big data tools and frameworks, such as Apache Hive for querying, Apache Pig for data processing, and Apache Flink for stream processing.
4. **Supporting High-Volume Data Ingestion:** HBase is well-suited for high-volume data ingestion scenarios, where data is continuously generated and needs to be processed and stored in near real-time.
5. **Column-Oriented Data Access:** The column-oriented nature of HBase makes it particularly efficient for analytical queries that need to access only a subset of columns, reducing the amount of data read from disk.

Overall, HBase plays a crucial role in big data ecosystems by providing a scalable, flexible, and efficient storage solution for large datasets, supporting real-time data access, and integrating well with other big data processing tools.

HBase and HDFS

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

HBase and RDBMS

HBase	RDBMS
HBase is schema-less, it doesn't have the concept of fixed columns schema; defines only column families.	An RDBMS is governed by its schema, which describes the whole structure of tables.
It is built for wide tables. HBase is horizontally scalable.	It is thin and built for small tables. Hard to scale.
No transactions are there in HBase.	RDBMS is transactional.
It has de-normalized data.	It will have normalized data.
It is good for semi-structured as well as structured data.	It is good for structured data.