


Chapter 6

👤 Created by	 chess pog
🕒 Created time	@April 2, 2024 7:03 PM
🏷️ Tags	

Hadoop

Hadoop is an open-source software framework designed for distributed storage and processing of large datasets across clusters of computers using simple programming models. It's primarily used to handle big data processing tasks. The core components of Hadoop include:

1. **Hadoop Distributed File System (HDFS):** A distributed file system that provides high-throughput access to application data. It's designed to be highly fault-tolerant and is suitable for applications with large data sets.
2. **MapReduce:** A programming model for processing and generating large data sets with a parallel, distributed algorithm on a cluster. It consists of two main functions, namely Map() and Reduce(), and is designed to efficiently process large volumes of data in parallel.

Hadoop is used for several reasons:

1. **Scalability:** Hadoop can scale horizontally, meaning you can add more nodes to your Hadoop cluster to handle increasing amounts of data without needing to invest in expensive high-end hardware.
2. **Fault tolerance:** Hadoop's distributed nature inherently provides fault tolerance. If one node in the cluster fails, the data processing framework redistributes the workload to other nodes, ensuring that the job can still be completed.
3. **Cost-effectiveness:** Hadoop can run on commodity hardware, which is relatively inexpensive compared to specialized hardware. This makes it cost-effective for storing and processing large datasets.
4. **Flexibility:** Hadoop is capable of processing various types of data, including structured, semi-structured, and unstructured data. This makes it suitable for a wide range of applications across different industries.

5. **Parallel processing:** Hadoop's MapReduce framework allows for parallel processing of data across multiple nodes in the cluster, enabling faster processing of large datasets.
6. **Data locality:** Hadoop processes data where it resides, which reduces the need to move large datasets across the network. This improves performance by minimizing data transfer time.

Data Flow in Hadoop

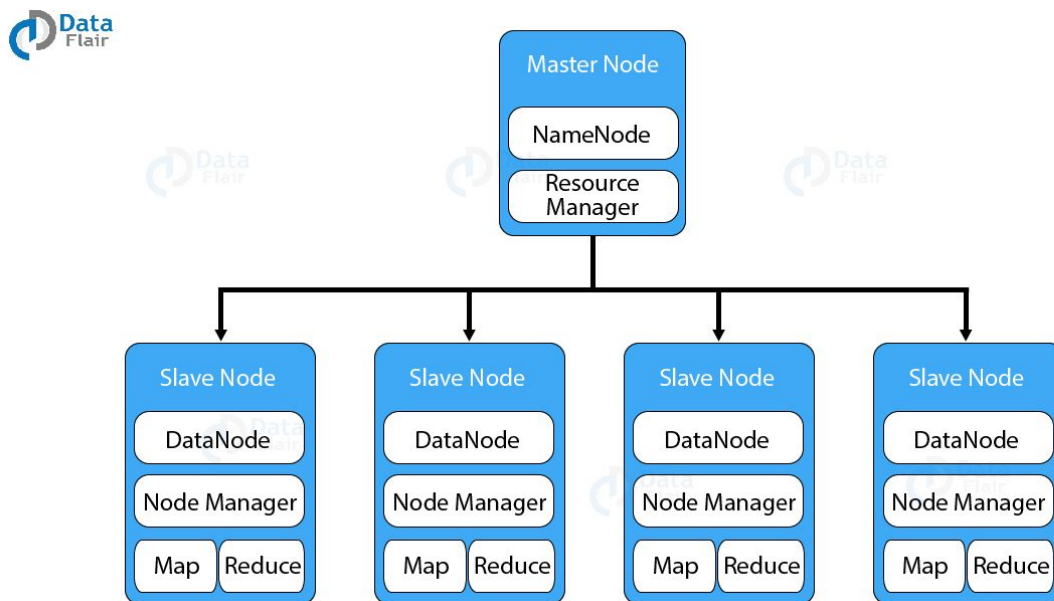


fig : Hadoop Master Slave Architecture

- A basic master and slave architecture of Hadoop is shown in given figure.
- There will be single master node also known as NameNode and multiple slave nodes also known as DataNode.
- Data are stored in DataNode and Metadata are stored in NameNode.

The architecture of Hadoop depicts five daemons:

- NameNode
- Secondary NameNode (Hidden in diagram)
- DataNode
- Job Tracker (Resource Manager)
- Task Tracker (Node Manager)

Typical workflow for handling big data within the Hadoop ecosystem. Let's break down each step:

1. Capture Big Data:

- In this step, the focus is on gathering data from various sources. These sources can include structured data (e.g., databases), semi-structured data (e.g., CSV files), and unstructured data (e.g., text documents). The sources could also include real-time data streams, sensor data, and other devices.
- Hadoop offers various tools for capturing and ingesting data from these diverse sources. Examples include Flume, Sqoop, Storm, and others. These tools help in efficiently transferring data into the Hadoop ecosystem for further processing and analysis.

2. Process and Structure:

- Once the data is captured, it needs to be processed and structured for analysis. This involves tasks like cleansing (removing errors or inconsistencies), filtering (selecting relevant data), and transforming (changing the format or structure of data).
- Hadoop provides several frameworks for processing big data. The traditional one is MapReduce, which divides data processing tasks into smaller parts and distributes them across a cluster of computers. Other frameworks like Hive, Pig, and Spark offer higher-level abstractions and more efficient processing methods compared to MapReduce.

3. Distribute Results:

- After processing, the data is ready for analysis or visualization. This step involves making the processed data available to business intelligence (BI) tools or big data analytics systems.
- These systems can perform various types of analysis, such as statistical analysis, machine learning, or data visualization, to derive insights from the data.

4. Feedback and Retain:

- The insights gained from analyzing the data can be used to improve future data processing and analysis.
- Feedback mechanisms can be established to incorporate insights back into the Hadoop ecosystem. This ensures that the system continuously learns and adapts based on the analysis results.

5. Hadoop Daemons and HDFS:

- Throughout this process, various components of the Hadoop ecosystem, known as daemons, are responsible for managing and processing the data.
- The Hadoop Distributed File System (HDFS) is used to store and distribute large datasets across the cluster of machines in the Hadoop ecosystem. It ensures high availability and reliability of data.

daemons refer to the background processes or services that run on different nodes of a Hadoop cluster to perform various tasks related to distributed storage and processing of data. These daemons collectively manage the storage, processing, and coordination within the Hadoop ecosystem. Each daemon serves a specific purpose and contributes to the overall functioning of the Hadoop cluster. The daemons typically run continuously in the background to ensure the smooth operation of the Hadoop cluster.

1. **NameNode:** The **NameNode** serves as the master node in the Hadoop Distributed File System (HDFS). Its primary responsibility lies in managing metadata, which includes crucial information about the data stored across the Hadoop cluster. This metadata encompasses details such as the location of data blocks within DataNodes, file block division, and system performance metrics. Essentially, the NameNode directs DataNode daemons, which reside on slave nodes, to handle low-level I/O tasks. However, it's important to note that the NameNode represents a single point of failure within the system due to its critical role in coordinating data storage and retrieval operations.
2. **Secondary NameNode:** Complementing the NameNode, the **Secondary NameNode** acts as a backup, safeguarding the metadata stored by the NameNode. It continuously reads metadata from the NameNode's RAM and archives it onto the disk, ensuring that in the event of a system failure or crash, this backed-up metadata can be utilized to restore the system's state and facilitate the creation of a new master node.
3. **DataNode:** On the other hand, the **DataNode** is responsible for storing the actual data within the Hadoop cluster. It operates as a daemon on slave

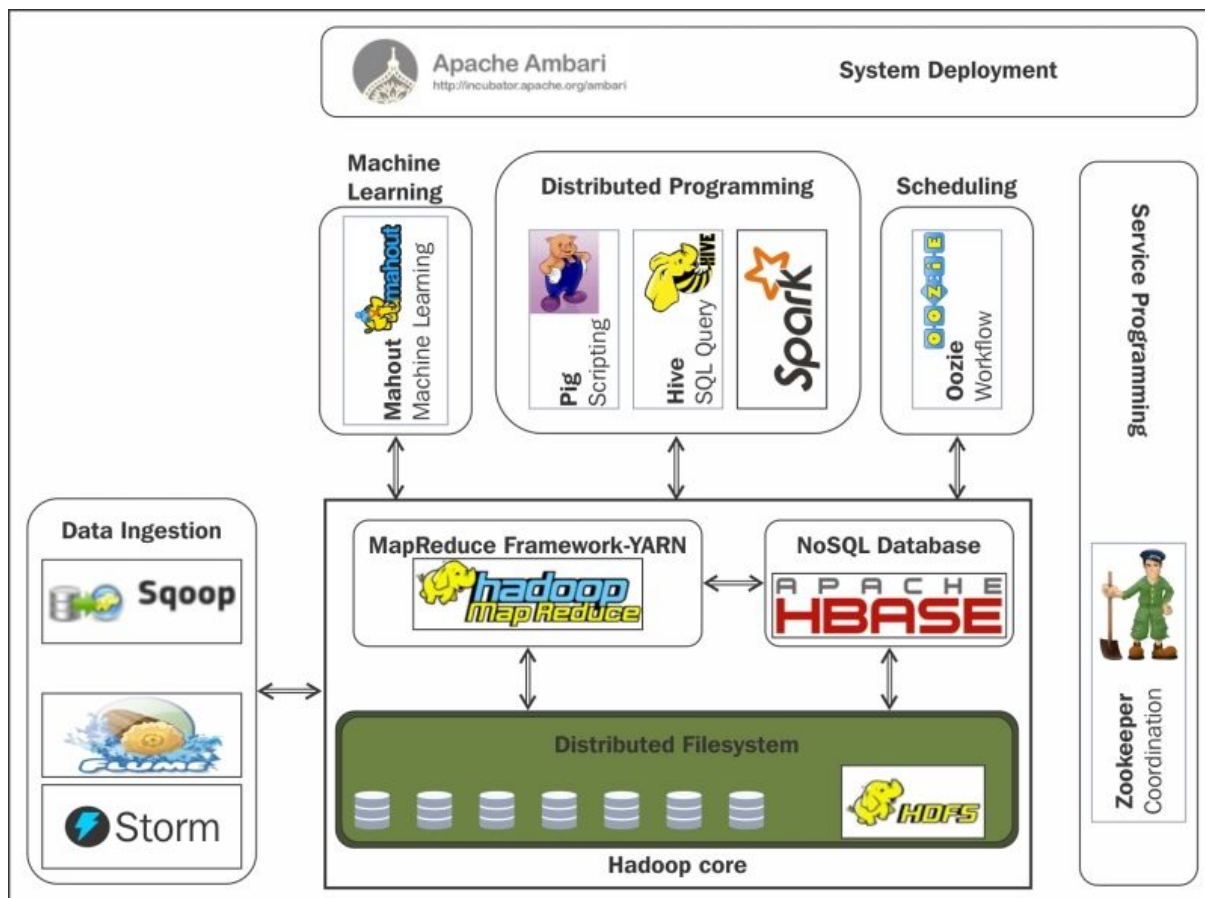
nodes, managing the storage and retrieval of data blocks as instructed by the NameNode. Each DataNode stores data blocks locally on disk and communicates directly with clients, facilitating read and write operations. Additionally, DataNodes play a crucial role in data replication and report back to the NameNode regarding any local changes or updates.

4. **JobTracker (Resource Manager)**: Moving beyond storage, the **Job Tracker** (also known as the Resource Manager) acts as the central coordinator for MapReduce jobs within the Hadoop cluster. It oversees the allocation of resources to applications running on the cluster, creating and managing jobs, and monitoring their execution. The Job Tracker, residing on the NameNode, assigns tasks to TaskTrackers, and in case of task failures, it redistributes tasks across the cluster for efficient execution.
5. **TaskTracker (Node Manager)**: Finally, the **Task Tracker** (Node Manager) operates as a slave node component within the MapReduce layer. Task Trackers manage the execution of individual MapReduce tasks on slave nodes, reporting task statuses back to the Job Tracker. In the event of a task failure, the Job Tracker reschedules the task for execution. Each Task Tracker spawns a separate JVM process to ensure fault tolerance and manages memory resources within the node.



Explain various components of Hadoop in brief (10)

Hadoop Ecosystem



The Hadoop ecosystem is a collection of open-source software utilities and frameworks that enable the distributed processing of large datasets across clusters of computers. These components work together to provide a comprehensive platform for big data storage, processing, and analysis. Here's an explanation of some of the core components of the Hadoop ecosystem:

1. Hadoop Common:

Hadoop Common is a set of common utilities and libraries that provide support for other Hadoop modules. It includes necessary Java libraries and utilities that are required by other Hadoop modules to function. Some of the common functionalities provided by Hadoop Common include file system support, networking, authentication, and utilities for data serialization and deserialization.

2. Hadoop Distributed File System (HDFS):

HDFS is the primary storage system used by Hadoop. It is a distributed file system designed to store large volumes of data reliably and efficiently across a cluster of commodity hardware. HDFS follows a master-slave architecture where the NameNode acts as the master node, responsible for managing metadata and namespace operations, while DataNodes serve as

slave nodes, storing the actual data blocks. HDFS provides high throughput access to application data and is fault-tolerant, enabling data replication across multiple nodes to ensure reliability.

3. **Yet Another Resource Negotiator (YARN):**

YARN is a resource management and job scheduling framework in Hadoop. It is responsible for managing and allocating resources in a Hadoop cluster to various applications running on it. YARN decouples the resource management and job scheduling functionalities from the MapReduce engine, allowing different processing models to run on Hadoop, such as Apache Spark, Apache Flink, and others. YARN consists of two main components: ResourceManager, which manages cluster resources, and NodeManager, which runs on each node and manages resources available on that node.

4. **MapReduce:**

MapReduce is a programming model and processing engine for distributed computing on large datasets. It enables parallel processing of data across multiple nodes in a Hadoop cluster by dividing tasks into two phases: Map and Reduce. In the Map phase, input data is divided into smaller chunks and processed in parallel across nodes to produce intermediate key-value pairs. In the Reduce phase, intermediate results with the same key are aggregated together to produce the final output. MapReduce is widely used for batch processing tasks in Hadoop, such as data transformation, filtering, and aggregation.

5. **Apache Ambari:** Ambari is an open-source management platform for provisioning, managing, and monitoring Apache Hadoop clusters. It provides an intuitive web interface to manage various Hadoop components and their configurations.

6. **Apache Mahout:** Mahout is a machine learning library built on top of Apache Hadoop. It provides scalable implementations of various machine learning algorithms for clustering, classification, and recommendation tasks.

7. **Apache Pig:** Pig is a platform for analyzing large datasets. It consists of a high-level language called Pig Latin, which is used to express data analysis tasks. Pig translates these tasks into MapReduce jobs, which can then be executed on a Hadoop cluster.

8. **Apache Hive:** Hive is a data warehousing framework built on top of Hadoop. It provides a SQL-like query language called HiveQL, which allows users to query and analyze data stored in Hadoop Distributed File System (HDFS) or other compatible file systems.
9. **Apache Spark:** Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in languages like Scala, Java, and Python, as well as an optimized engine that supports in-memory computing and fault tolerance. Spark is commonly used for data processing, machine learning, and real-time analytics.
10. **Apache Oozie:** Oozie is a workflow scheduler system for managing Hadoop jobs. It allows users to define workflows that consist of a series of interconnected Hadoop jobs, and then schedule and monitor the execution of these workflows.
11. **Sqoop:** Sqoop is a tool for transferring data between Hadoop and relational databases. It allows users to import data from a database into Hadoop for analysis, or export data from Hadoop back into a database for storage or further processing.
12. **Apache Storm:** Storm is a distributed real-time computation system. It allows users to process streaming data in real-time, with support for fault tolerance and scalability. Storm is commonly used for tasks such as real-time analytics, event processing, and stream processing.
13. **Hadoop core:** Hadoop core refers to the foundational components of the Apache Hadoop ecosystem, including the Hadoop Distributed File System (HDFS) for distributed storage, and the MapReduce framework for distributed processing of large datasets.
14. **Apache Zookeeper:** Zookeeper is a centralized service for maintaining configuration information, providing distributed synchronization, and implementing group services. It is commonly used in distributed systems such as Hadoop to manage cluster metadata and coordinate operations among nodes.

GFS VS HDFS

File Structure

- In GFS:
- Divided into 64 MB chunks

- Chunks identified by 64-bit handle
- Chunks replicated (default 3 replicas)
- Chunks divided into 64 KB blocks
- Each block has 32 bit checksum
- In HDFS:
 - Divided into 128 MB blocks
 - Name node holds block replica as 2 files (one for data and other for checksum)

Architecture Comparison

- In GFS, leases at primary default to 60 seconds. In HDFS, no leases (client decides where to write.)
- The read and write process by client is similar in both GFS and HDFS:
- **Read Operation:**
 1. Client sends request to master
 2. Caches lists of replicas
 3. Locations provided for limited time
- **Write Operation:**
 1. Client obtains replica locations and identify primary replica
 2. Client pushes data to replica
 3. Client issues update request to primary replica
 4. Primary replica forwards write request.
 5. Primary receives replies from replicas
 6. Primary replica replies to the client

For a Hadoop cluster with 128MB block size, how many mappers will Hadoop infrastructure form while performing mapper functions on 1 GB of data?

Determining the number of mappers for Hadoop

- -----
1. If the file is split table:
 - Calculate total size of input file
 - No of mapper = Total input file size / Block size (Split size)
 2. If the file is not split table:
 - No of mapper = No of input files

- If the file size is too huge, it becomes bottle neck to the performance of MapReduce.

Solution to numerical problem:

- -----

1. Assuming the file to be split table:

No of mappers = $1024 / 128 = 8$

2. Assuming the file to be not split table and data is contained in single file:

No of mappers = 1

s



Describe the daemons of Hadoop. Explain the role of HDFS in Hadoop and write down the syntax for file upload, download, list and view content of file commands for HDFS. (10)

Here's an overview of the role of HDFS in Hadoop:

1. **Scalability:** HDFS is designed to scale horizontally to handle large amounts of data by distributing it across multiple nodes in a Hadoop cluster.
2. **Fault Tolerance:** HDFS achieves fault tolerance by replicating data across multiple nodes in the cluster. If one node fails, the data can still be retrieved from another replica.
3. **Data Locality:** HDFS is optimized for data locality, meaning that computations are performed on the same nodes where the data is stored whenever possible, reducing network traffic and improving performance.
4. **Streaming Access:** HDFS is optimized for large, sequential reads and writes, making it suitable for applications that require high throughput data access, such as batch processing and analytics.
5. **Ease of Use:** HDFS provides a simple and straightforward interface for storing and accessing data, making it easy to integrate with Hadoop applications.

Now, here are the commands for performing basic file operations in HDFS:

1. File Upload:

```
bashCopy code
hdfs dfs -put <local_src> <hdfs_dest>
```

Example:

```
bashCopy code
hdfs dfs -put /path/to/local/file /user/username/
```

1. File Download:

```
bashCopy code
hdfs dfs -get <hdfs_src> <local_dest>
```

Example:

```
bashCopy code
hdfs dfs -get /user/username/file.txt /path/to/local/destination/
```

1. List Files/Directories:

```
bashCopy code
hdfs dfs -ls <hdfs_path>
```

Example:

```
bashCopy code
hdfs dfs -ls /user/username/
```

1. View File Content:

```
bashCopy code
hdfs dfs -cat <hdfs_file>
```

Example:

```
bashCopy code
hdfs dfs -cat /user/username/file.txt
```



Explain about the configuration modes of Hadoop. give an overview of Hadoop ecosystem. (6+6)

The main configuration modes in Hadoop are:

1. **Standalone Mode:** In this mode, Hadoop runs on a single node without any Hadoop daemons running. It's primarily used for debugging and development purposes where you don't need the distributed capabilities of Hadoop. This mode is not suitable for production use.
2. **Pseudo-Distributed Mode:** Also known as single-node cluster mode, this mode is used for running Hadoop on a single machine but with separate processes for each Hadoop daemon. It simulates a distributed environment on a single node, enabling you to test Hadoop's functionality as if it were in a real distributed setting. While still not suitable for production use, it's useful for learning and testing.
3. **Fully Distributed Mode:** This is the mode where Hadoop is deployed across a cluster of machines, each running one or more Hadoop daemons. It's the mode used in production environments where data processing tasks are distributed across multiple nodes for scalability and fault tolerance. In this mode, you typically have separate nodes for the NameNode, ResourceManager, and DataNodes.



Explain the respective components of HDFS and YARN. How client writes data into HDFS? Explain with a suitable block diagram. (10)

HDFS (Hadoop Distributed File System) and YARN (Yet Another Resource Negotiator) are core components of the Hadoop ecosystem, designed to handle distributed storage and resource management, respectively.

1. HDFS (Hadoop Distributed File System):

- **NameNode:** It is the master node responsible for managing the metadata of the file system, such as the directory tree structure and the mapping of file blocks to DataNodes.
- **DataNode:** These are worker nodes responsible for storing the actual data. They manage storage attached to the nodes that they run on. They also communicate with the NameNode to perform tasks such as block creation, deletion, and replication as instructed.
- **Secondary NameNode:** Despite its name, it doesn't act as a failover NameNode. Instead, it periodically merges the namespace and edits from the NameNode's memory and writes them to the disk, helping in preventing corruption and speeding up startup times.

2. YARN (Yet Another Resource Negotiator):

- **ResourceManager:** It is the master daemon in YARN, responsible for allocating resources to applications and scheduling tasks across the cluster. There's typically one ResourceManager per cluster.
- **NodeManager:** These are worker daemons that run on each node in the cluster and are responsible for managing resources such as CPU, memory, and disks locally. They report resource utilization to the ResourceManager and execute tasks allocated by the ResourceManager.
- **ApplicationMaster:** Each application running on the cluster has its own ApplicationMaster, which is responsible for negotiating resources with the ResourceManager and working with NodeManagers to execute and monitor tasks.

Now, regarding how a client writes data into HDFS:

1. **Client:** The client interacts with HDFS using various command-line utilities or client libraries.
2. **Write Process:**
 - The client breaks the data into blocks and calculates the appropriate DataNodes for each block.
 - It then communicates with the NameNode to get information about the DataNodes and the location to write each block.
 - The client then connects directly to the chosen DataNodes and writes the blocks of data to them.
 - Each DataNode acknowledges the receipt of the data, and once all blocks are successfully written, the client informs the NameNode about the completion of the file write operation.
 - The NameNode updates its metadata to reflect the changes.
 - Data replication may occur based on the configured replication factor to ensure fault tolerance and data durability.



List down the different components installed in the hadoop cluster. Explain it's workflow. How fault tolerance and scalability is handled by the hadoop cluster? (2+4+6)

Components Installed in a Hadoop Cluster:

1. **Hadoop Distributed File System (HDFS)**
2. **Yet Another Resource Negotiator (YARN)**
3. **MapReduce**
4. **Hadoop Common**
5. **Hadoop Query Language (Hive)**

Workflow of Hadoop Cluster:

1. **Data Ingestion:** Data is ingested into the Hadoop cluster, typically through tools like Apache Flume, Apache Kafka, or by direct file upload.
2. **Storage:** The data is stored in the Hadoop Distributed File System (HDFS). HDFS splits large files into blocks and distributes them across multiple nodes in the cluster for fault tolerance and scalability.
3. **Processing:** Data processing is done using MapReduce or other processing engines like Apache Spark or Apache Flink. MapReduce jobs are submitted to the YARN ResourceManager, which allocates resources and schedules tasks across the cluster.
4. **Analysis:** Processed data can be analyzed using tools like Apache Hive, Apache Pig, or through custom applications developed using Hadoop APIs.
5. **Visualization:** Analyzed data can be visualized using tools like Apache Zeppelin, Tableau, or other visualization libraries.

Fault Tolerance and Scalability in Hadoop Cluster:

1. **Fault Tolerance:** Hadoop achieves fault tolerance primarily through data replication in HDFS. It divides files into blocks and replicates each block across multiple DataNodes in the cluster. If a DataNode fails or becomes inaccessible, Hadoop automatically re-replicates the data blocks to ensure redundancy and fault tolerance. Additionally, components like YARN also have mechanisms to handle failures gracefully by reallocating resources and restarting failed tasks.
2. **Scalability:** Hadoop clusters are designed to scale horizontally by adding more nodes to the cluster as data volume and processing demands increase. Hadoop's distributed nature allows it to distribute and parallelize data processing tasks across multiple nodes in the cluster, enabling linear scalability. YARN effectively manages resources and ensures efficient utilization across the cluster, allowing it to scale to thousands of nodes without significant performance degradation. Additionally, Hadoop's modular architecture allows for the integration of new components and technologies to meet evolving scalability requirements.



How Client reads data from HDFS? (10)

1. **Locating the Data:** Before reading data from HDFS, the client needs to know where the data is stored. This information is provided by the NameNode, which maintains the metadata about file locations and block assignments.
2. **Communicating with the NameNode:** The client communicates with the NameNode to obtain the metadata about the file it wants to read. This metadata includes the locations of the data blocks comprising the file.
3. **Data Retrieval:** Once the client has the metadata, it knows which DataNodes store the blocks of the file. It then directly communicates with these DataNodes to retrieve the data blocks. HDFS employs a pipeline mechanism where data is streamed from one DataNode to another and finally to the client.
4. **Combining Data Blocks:** If the file spans multiple data blocks, the client will receive these blocks from different DataNodes. It is the responsibility of the client to merge these blocks to reconstruct the original file.
5. **Reading Data:** After receiving the data blocks, the client application can read the data from them according to its requirements.
6. **Error Handling:** Throughout this process, error handling mechanisms are in place to ensure data integrity and fault tolerance. If a DataNode fails to respond or returns corrupted data, the client can request the same block from another replica stored on a different DataNode.



clock synchronization in DFS may be the big challenge. How this clock synchronization problem can be solved? (10)

Clock synchronization is indeed a significant challenge in distributed file systems (DFS), as it's essential for maintaining consistency and coherence

among the distributed nodes. Here are a few approaches commonly used to address this problem:

1. Network Time Protocol (NTP):

NTP is a protocol used to synchronize the clocks of computer systems over packet-switched, variable-latency data networks. It operates by exchanging timestamped messages between servers and clients. DFS nodes can synchronize their clocks using NTP to ensure consistency in timestamps across the distributed system.

2. Vector Clocks:

Vector clocks are used to track causality between events in a distributed system. Each node maintains a vector clock, which is a list of integer counters corresponding to other nodes. When an event occurs at a node, its vector clock is updated. By exchanging and comparing vector clocks between nodes, it's possible to determine the ordering of events and ensure consistency.

3. Logical Clocks:

Logical clocks, such as Lamport clocks or Scalar clocks, provide a logical ordering of events in a distributed system. These clocks don't rely on physical time but instead use a logical notion of time based on the ordering of events. Nodes can use logical clocks to track the ordering of operations and ensure consistency without requiring precise clock synchronization.

4. Clock Synchronization Protocols:

There are specific protocols designed for clock synchronization in distributed systems, such as the Berkeley algorithm or the Cristian's algorithm. These protocols aim to adjust the local clocks of nodes based on network latency and other factors to achieve synchronization.

5. Consensus Algorithms:

Consensus algorithms like Paxos or Raft can be utilized to agree on a common notion of time among distributed nodes. By reaching consensus on the ordering of events and timestamps, these algorithms indirectly address clock synchronization challenges.

6. Hybrid Approaches:

Often, a combination of the above methods may be used to achieve robust clock synchronization in DFS. For instance, combining NTP for coarse-grained synchronization with logical clocks for fine-grained ordering can provide an effective solution.



How do you find max and min occurrence of the words in a given text document. Explain (10)

Here's how you can implement finding the maximum and minimum occurrences of words using MapReduce:

1. **Map phase:** In this phase, each worker node processes a portion of the input data and emits a set of intermediate key-value pairs. For this task, each mapper will tokenize the text, count the occurrences of each word, and emit key-value pairs where the key is the word and the value is the count of occurrences.
2. **Shuffle and Sort:** The MapReduce framework groups together all intermediate values associated with the same intermediate key and sorts them. This prepares the data for the next phase.
3. **Reduce phase:** In this phase, each reducer receives a key and a list of values associated with that key. The reducer then performs the necessary aggregation to produce the final result. For this task, each reducer will find the maximum and minimum occurrences of words from the intermediate key-value pairs it receives.



How HADOOP and GFS are similar in terms of design architecture.

Hadoop and Google File System (GFS) are both designed to handle large-scale distributed data processing, and they share several similarities in their design architecture:

1. **Distributed Storage:** Both Hadoop and GFS are built on the concept of distributed storage, where large files are broken down into smaller blocks and distributed across multiple storage nodes in a cluster. This allows for

scalability and fault tolerance, as data is replicated across nodes to ensure reliability.

2. **Master-Slave Architecture:** Both systems employ a master-slave architecture. In Hadoop, you have the NameNode (master) and multiple DataNodes (slaves). Similarly, GFS has a master node and multiple chunk servers. The master node manages metadata and coordinates operations, while the chunk servers handle data storage and retrieval.
3. **Replication:** Both systems utilize replication for fault tolerance. Hadoop replicates data blocks across multiple DataNodes, typically three replicas by default. GFS also replicates data across chunk servers for redundancy and reliability.
4. **Data Locality:** Both systems prioritize data locality, aiming to perform computations as close to the data as possible. In Hadoop, MapReduce tasks are scheduled on nodes where the data resides to minimize network traffic. Similarly, GFS optimizes data access by storing multiple copies of data across the cluster, allowing computations to be performed on nearby data.
5. **Scalability:** Both Hadoop and GFS are designed to scale horizontally, allowing them to handle petabytes of data across a large number of nodes. As the size of the dataset grows, additional nodes can be added to the cluster to accommodate increased storage and processing requirements.
6. **Fault Tolerance:** Both systems are resilient to failures. They employ mechanisms to detect faults and recover from them seamlessly. In Hadoop, if a DataNode fails, the NameNode can redirect tasks to other replicas of the data. GFS similarly handles failures by replicating data and redistributing workload to healthy chunk servers.
7. **Open Source and Commercial Implementations:** While GFS is proprietary to Google, Hadoop is an open-source project maintained by the Apache Software Foundation. However, there are commercial distributions and implementations of both systems available, such as Google Cloud Storage for GFS and Cloudera, Hortonworks, and MapR for Hadoop.