

Architecture

28 November 2024 11:46 AM

Architecture of Hadoop

The **Hadoop architecture** is designed to process and store vast amounts of structured, semi-structured, and unstructured data. It is built on a **distributed computing model** that allows data to be stored across a cluster of machines and processed in parallel. Hadoop's architecture comprises several key components, each responsible for a specific role.

Core Components of Hadoop Architecture

Hadoop consists of two main layers:

- Storage Layer (HDFS - Hadoop Distributed File System):** Handles storage of data in a distributed manner.
- Processing Layer (MapReduce Framework):** Handles the processing of data.

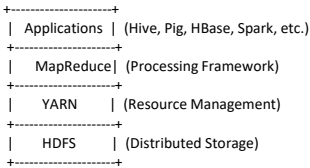
Additionally, there are resource management components like **YARN** and higher-level tools (Hive, Pig, etc.).

Hadoop Architecture Diagram

At a high level, Hadoop architecture includes:

- HDFS (Storage Layer)**
- MapReduce (Processing Layer)**
- YARN (Resource Management)**

Copy code



Detailed Components

1. HDFS (Hadoop Distributed File System)

HDFS is the storage layer of Hadoop that allows data to be stored across a cluster of machines while maintaining fault tolerance and scalability.

- NameNode:**
 - Acts as the **master node**.
 - Manages the **metadata** (e.g., file structure, block locations) of the file system.
 - Does not store actual data but knows where the data blocks are stored.
- DataNode:**
 - Acts as the **worker node**.
 - Stores the actual **data blocks**.
 - Responsible for serving read/write requests from clients and reporting back to the NameNode.
- Secondary NameNode:**
 - Assists the NameNode by taking snapshots of the file system metadata at intervals.
 - Used for checkpointing, not as a backup.

2. YARN (Yet Another Resource Negotiator)

YARN is the **resource management layer** of Hadoop, responsible for managing cluster resources and scheduling tasks.

- ResourceManager:**
 - Manages all cluster resources.
 - Assigns resources to applications based on requirements.
- NodeManager:**
 - Runs on each DataNode and monitors resource usage (CPU, memory, etc.).
 - Reports to the ResourceManager.
- ApplicationMaster:**
 - Manages the lifecycle of a specific application running on YARN.
 - Coordinates with ResourceManager and NodeManager for resource allocation.

3. MapReduce (Processing Layer)

MapReduce is a distributed processing framework that processes data in two stages: **Map** and **Reduce**.

- Map Phase:**
 - Splits the input data into smaller chunks and processes them in parallel.
 - Generates key-value pairs as output.
- Reduce Phase:**
 - Aggregates the output from the Map phase to generate the final result.
- JobTracker and TaskTracker** (deprecated in Hadoop 2.0+):
 - In earlier Hadoop versions, JobTracker managed the MapReduce jobs, and TaskTrackers executed tasks on individual nodes. In Hadoop 2.0+, YARN replaced this functionality.

4. Higher-Level Tools (Built on Core Hadoop)

Hadoop provides tools and libraries to work with HDFS and MapReduce more efficiently:

- Hive:** A data warehouse tool for querying and managing large datasets (SQL-like interface).
- Pig:** A scripting platform for processing large datasets.
- HBase:** A NoSQL database built on HDFS.
- Spark:** A fast, in-memory data processing framework.
- Sqoop:** Tool for transferring data between Hadoop and relational databases.
- Flume:** Used for collecting and transferring large amounts of log data.

Workflow of Hadoop Architecture

- Data Input:**
 - Data is ingested into HDFS using tools like Flume or Sqoop, or directly from files.
- Data Storage:**
 - HDFS splits the data into blocks (default: 128 MB) and distributes them across DataNodes.
 - NameNode maintains the metadata for these blocks.
- Processing:**
 - Users submit a job (e.g., MapReduce, Hive query) to YARN.
 - YARN allocates resources and schedules tasks on the cluster.
 - The MapReduce framework processes the data in parallel.
- Data Output:**
 - The results are stored back in HDFS or exported to external systems.

Key Features of Hadoop Architecture

- Scalability:** Hadoop can scale horizontally by adding more nodes to the cluster.
- Fault Tolerance:** HDFS replicates data blocks (default 3 copies) across multiple nodes, ensuring data availability.

Architecture of HDFS

The **Hadoop Distributed File System (HDFS)** architecture is designed to store and manage large datasets across distributed systems while providing fault tolerance and high throughput for data access. It operates on a **master-slave architecture** model, where the master node manages the file system's metadata, and the slave nodes store the actual data blocks.

Components of HDFS Architecture

HDFS consists of the following key components:

1. NameNode (Master Node)

- Acts as the **master** in the HDFS architecture.
- Manages the **metadata** of the file system, such as:
 - File-to-block mapping.
 - Block-to-DataNode mapping.
 - Namespace hierarchy.
- Does not store the actual data; instead, it keeps metadata in memory for fast access.
- Responsibilities:**
 - Handles client requests for file operations (e.g., open, close, rename).
 - Tracks the health of DataNodes through **heartbeat signals**.
 - Orchestrates replication of data blocks to maintain fault tolerance.

2. DataNode (Worker Node)

- Acts as the **worker** node in the architecture.
- Stores the actual **data blocks** of files.
- Sends regular **heartbeats** and **block reports** to the NameNode to indicate its status.
- Responsibilities:**
 - Handles read/write requests from clients for data blocks.
 - Performs block creation, deletion, and replication as instructed by the NameNode.

3. Secondary NameNode

- Acts as a helper to the NameNode.
- Responsibilities:**
 - Periodically takes snapshots of the NameNode's metadata.
 - Merges the NameNode's **edit logs** with the **FsImage** (a checkpoint of the file system).
 - Reduces the size of the edit logs, improving NameNode restart times.
- It is **not a backup** of the NameNode. If the NameNode fails, manual recovery is needed using the snapshots.

4. HDFS Clients

- Interfaces that allow users and applications to interact with the HDFS.
- Responsibilities:**
 - Split large files into smaller blocks (default: 128 MB or 256 MB).
 - Communicate with the NameNode for metadata and with DataNodes for block storage/retrieval.

HDFS File Storage Mechanism

- File Splitting:**
 - When a file is written to HDFS, it is divided into fixed-size blocks (e.g., 128 MB).
 - Blocks are distributed across DataNodes in the cluster.
- Block Replication:**
 - HDFS maintains multiple copies of each block (default replication factor: 3).
 - Blocks are replicated across different DataNodes for fault tolerance.
- Metadata Management:**
 - NameNode stores metadata, such as:
 - File name.
 - Block ID and locations.
 - Permissions.
- Data Locality:**
 - Computation is moved to the DataNodes storing the required blocks, reducing network overhead.

HDFS Workflow

File Write Operation

- The client sends a file creation request to the NameNode.
- The NameNode verifies if the file already exists and checks permissions.
- The file is divided into blocks and written to DataNodes.
- The NameNode assigns DataNodes for block replication.
- Acknowledgment is sent to the client once all blocks are written successfully.

File Read Operation

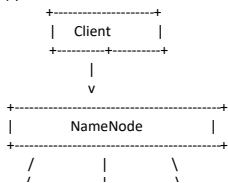
- The client requests file access from the NameNode.
- The NameNode provides the block locations (DataNodes storing the blocks).
- The client retrieves the data directly from the DataNodes.

HDFS Features

- Scalability:** Can store and process petabytes of data by adding more DataNodes.
- Fault Tolerance:** Data is replicated across multiple DataNodes to ensure availability during failures.
- High Throughput:** Optimized for large file reads and writes.
- Data Locality:** Brings computation to the nodes storing the data.
- Write Once, Read Many:** Designed for workloads that involve a single write and multiple reads.
- POSIX-like Permissions:** Ensures access control with user, group, and permission settings.

HDFS Architecture Diagram

Copy code



- The results are stored back in HDFS or exported to external systems.

Key Features of Hadoop Architecture

- **Scalability:** Hadoop can scale horizontally by adding more nodes to the cluster.
- **Fault Tolerance:** HDFS replicates data blocks (default: 3 copies) across multiple nodes, ensuring data availability even if nodes fail.
- **Cost-Effectiveness:** Uses commodity hardware for storage and computation.
- **Parallel Processing:** Processes data across multiple nodes simultaneously for faster computation.
- **Data Locality:** Moves computation to where the data resides, reducing network overhead.

Hadoop Ecosystem Components (Beyond Core)

Hadoop’s ecosystem includes several tools and technologies that extend its capabilities:

- **Zookeeper:** Coordinates distributed applications.
- **Oozie:** Workflow scheduler for Hadoop jobs.
- **Mahout:** Library for machine learning and data mining.
- **Cassandra:** Distributed NoSQL database.
- **Kafka:** Real-time data ingestion and streaming.

Explain Kafka Architecture

Kafka Architecture is designed to handle high-throughput, real-time streaming and processing. It provides a distributed messaging system that allows producers to send messages to a broker, and consumers to read them asynchronously. Kafka is known for its fault tolerance, scalability, and high - performance data pipelines.

Main Parts of Kafka

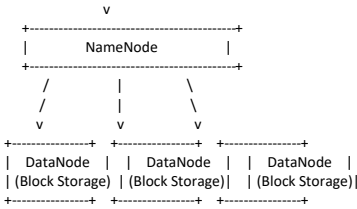
- Topics**
 - A **topic** is like a mailbox. Messages are sent to a specific topic.
 - Topics are split into **partitions** for better organization and faster access.
- Partitions**
 - A partition is like a drawer inside the mailbox, where messages are stored in order.
 - Each message in a partition gets a unique number called an **offset** (like a receipt).
- Producers**
 - Producers are apps or systems that **send messages** to Kafka topics.
 - Example: A weather app sending updates to a "Weather Data" topic.
- Consumers**
 - Consumers are apps or systems that **read messages** from Kafka topics.
 - Example: A dashboard reading weather updates from the "Weather Data" topic.
- Brokers**
 - Brokers are the **servers** that store the topics and partitions.
 - If you have a cluster of brokers (multiple servers), they share the data to balance the load.
- Zookeeper (Old Version)**
 - Keeps track of which broker manages which topic/partition.
 - Newer versions of Kafka use **KRaft** instead of Zookeeper.
- Kafka Connect**
 - Connects Kafka to other systems, like databases or cloud storage.
 - Example: Importing data from MySQL or sending Kafka data to S3.
- Kafka Streams**
 - Processes the messages in real time, like transforming or analyzing data as it flows through Kafka.

How Kafka Works (In Simple Steps)

- Sending Messages:**
 - A producer sends messages to a topic.
 - Kafka stores these messages in partitions.
- Storing Messages:**
 - Messages in a partition are saved in order.
 - Kafka keeps these messages for a set amount of time, even if they’ve been read.
- Reading Messages:**
 - A consumer reads messages from a partition.
 - If there are multiple consumers, they divide the partitions to share the work.
- Replication:**
 - Kafka makes copies of partitions (replicas) to prevent data loss if a broker fails.

Why Use Kafka?

- **Fast:** Handles millions of messages per second.
- **Reliable:** Keeps data safe with backups (replicas).
- **Scalable:** Can grow by adding more servers.
- **Flexible:** Works for many use cases like logs, analytics, and streaming.



HDFS Advantages

- **Cost-Effective:** Uses commodity hardware.
- **Fault Tolerant:** Ensures data availability and integrity.
- **Scalable:** Can grow to accommodate increasing data volumes.

HDFS Limitations

- **Latency:** Not suitable for low-latency access to small files.
- **Write Constraints:** Supports only one writer per file at a time.
- **Metadata Overhead:** The NameNode can become a bottleneck as metadata grows.

Explain Yarn Architecture

YARN is a key component of the Hadoop ecosystem, responsible for **resource management** and **job scheduling**. It allows Hadoop to run multiple data processing frameworks (like MapReduce, Spark) on the same cluster, making it more flexible and efficient.

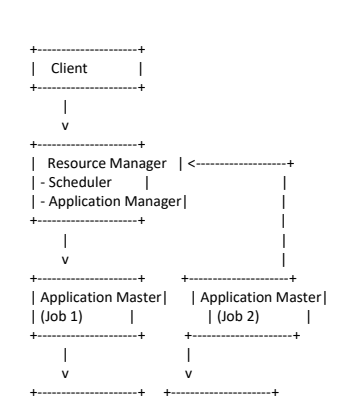
Key Components of YARN Architecture

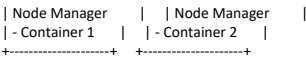
- ResourceManager (Master Node):**
 - The **central authority** for managing cluster resources.
 - Responsible for **allocating resources** to applications.
 - Divided into two parts:
 - **Scheduler:** Allocates resources to running applications based on policies (e.g., FIFO, Fair Scheduler).
 - **Application Manager:** Manages job submissions, monitors application progress, and restarts failed applications.
- NodeManager (Slave Node):**
 - Runs on each node of the cluster.
 - Responsible for **monitoring the resource usage** (CPU, memory) of containers on the node.
 - Communicates with the ResourceManager to update the status of resources.
- ApplicationMaster (Per Job):**
 - Created for each job/application submitted to the cluster.
 - Manages the execution of the specific application (e.g., MapReduce, Spark).
 - Negotiates resources with the ResourceManager.
 - Coordinates with NodeManager to run tasks.
- Containers:**
 - The basic unit of resource allocation in YARN.
 - Encapsulates a fixed amount of resources (CPU, memory).
 - Containers are where the actual tasks or jobs run.

How YARN Works

- Job Submission:**
 - The client submits a job (e.g., MapReduce) to the ResourceManager.
- ApplicationMaster Creation:**
 - ResourceManager starts an ApplicationMaster for the job inside a container.
 - The ApplicationMaster negotiates resources with the ResourceManager.
- Resource Allocation:**
 - ResourceManager allocates resources to the ApplicationMaster based on availability and priority.
 - ApplicationMaster requests containers for tasks.
- Task Execution:**
 - The NodeManager on each node launches containers as instructed by the ApplicationMaster.
 - Containers execute tasks (e.g., map or reduce tasks in MapReduce).
- Monitoring and Completion:**
 - NodeManagers monitor the health of tasks running in their containers and report back to the ResourceManager.
 - Once the job is complete, the ApplicationMaster notifies the ResourceManager and exits.

YARN Architecture Diagram





Key Features of YARN

- **Scalability:** Can handle a large number of nodes and tasks.
- **Resource Utilization:** Allocates resources dynamically, maximizing efficiency.
- **Fault Tolerance:** Reschedules tasks in case of node or container failure.
- **Multi-Tenancy:** Supports multiple frameworks like MapReduce, Spark, Flink, etc.
- **Flexible Scheduling:** Allows custom scheduling policies (e.g., FIFO, Fair Scheduling).

Advantages of YARN

- Supports multiple applications and frameworks on the same cluster.
- Improves resource utilization by separating resource management from task execution.
- Allows dynamic allocation of resources based on demand.