Distributed Computing Using Hadoop

Chapter 06
Instructor: Houssein Dhayne
houssein.dhayne@net.usj.edu.lb

Table of content

Introduction

BIG DATA

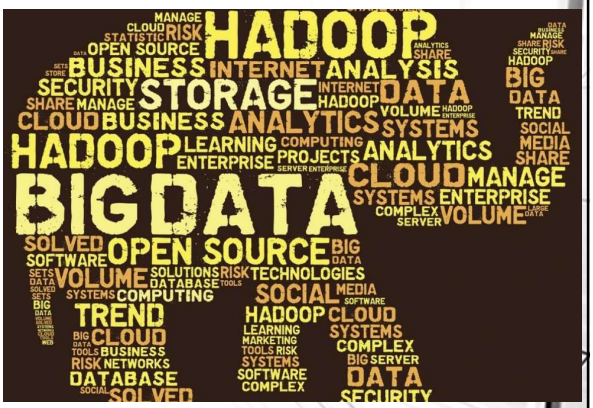
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore

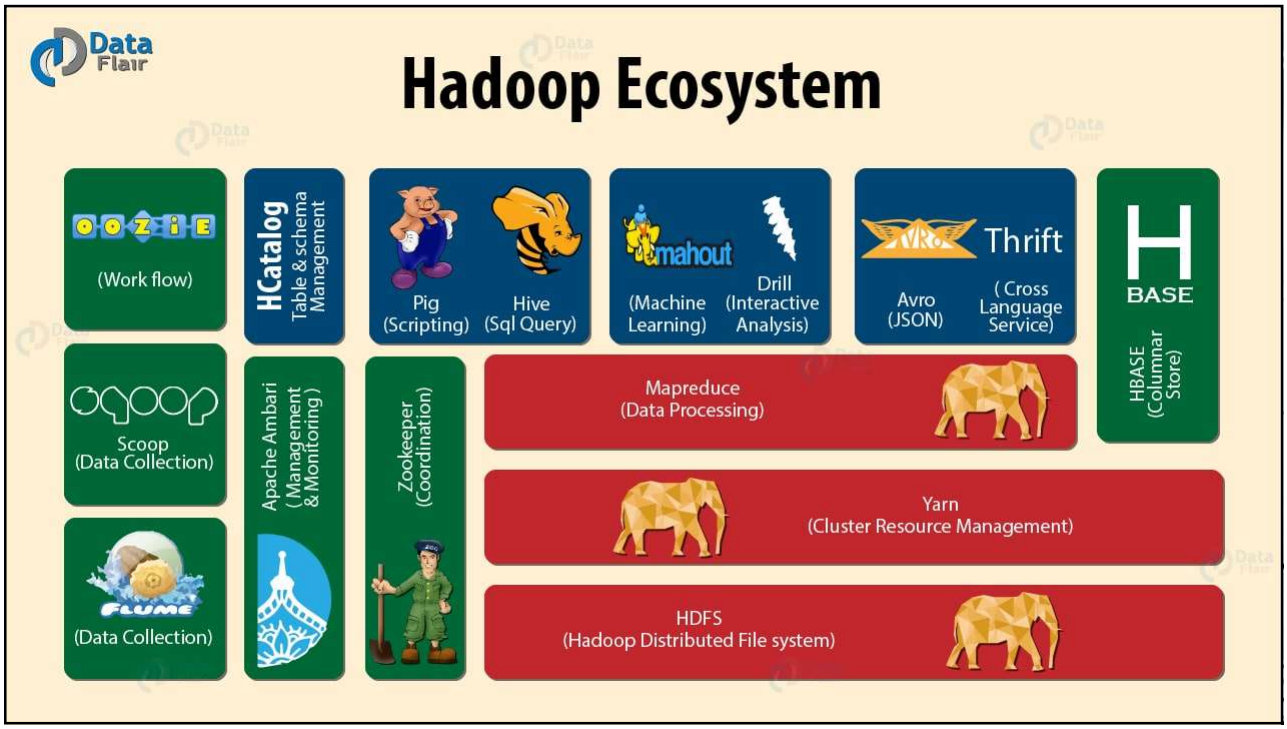
- A distributed file storage system is a clever way of storing huge quantities of data, securely and cost-effectively, for speed and ease, for retrieval and processing, using a networked collection of commodity machines.
- The ideal distributed file system would store infinite amounts of data while making the complexity completely hidden from the user, and enabling instant and easy access to the right data.
- This would be achieved by storing sections of data at different locations, and internally managing the lower- level tasks of integrating and replicating data across the network.

Hadoop Framework

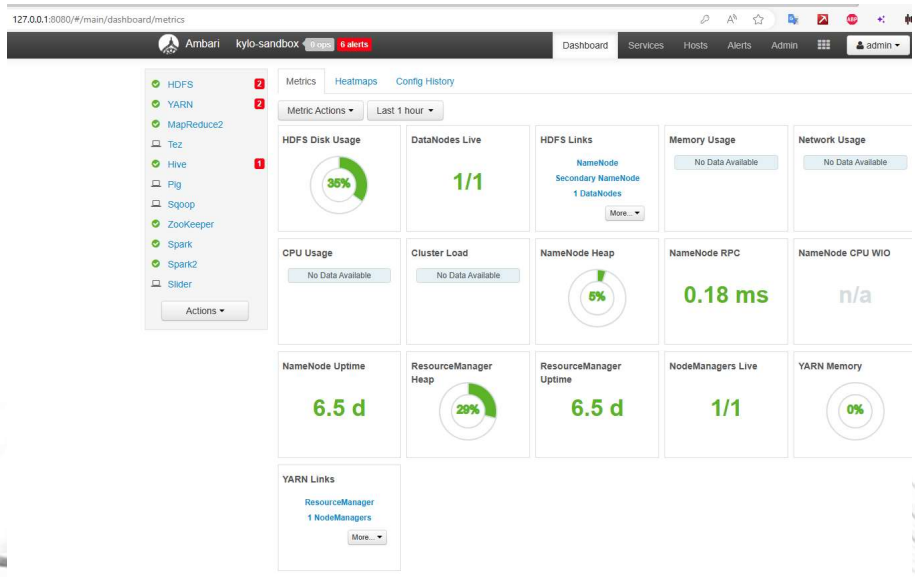
The Apache Hadoop framework comprises key modules:

- **Hadoop Common:** Libraries and utilities for other modules.
- **HDFS:** Distributed file-system with high bandwidth.
- **YARN:** Resource management platform.
- **MapReduce:** Implementation of the programming model for large-scale data processing. This chapter focuses on Hadoop Common, HDFS, and YARN.





Ambari dashboard provides various portlets that display key metrics and information about the Hadoop cluster. Here's an explanation for each portlet:

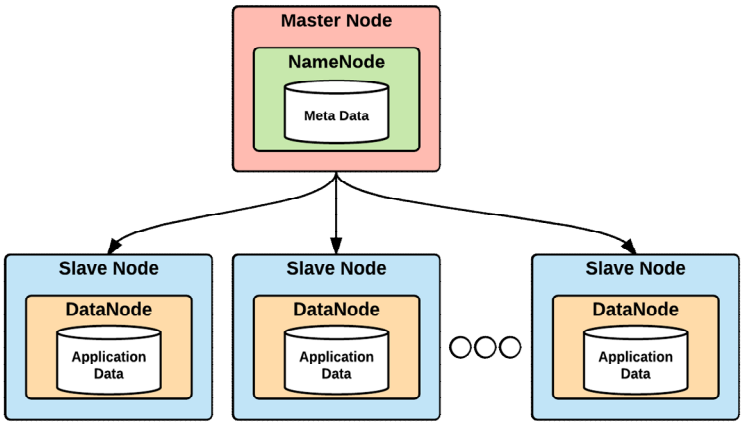


HDFS Design Goals

Hadoop Distributed File System (HDFS) is designed for scalability and large data sizes. Key design goals include:

- **Hardware Failure Management:** Planning for inevitable failures.
- **Huge Volume:** Capacity for large files with fast read/write throughput.
- **High Speed:** Low latency access to streaming applications.
- **High Variety:** Maintaining data coherence for both reading and writing.
- **Plug-and-Play:** Easy data accessibility on any hardware, software, or database platform.
- **Network Efficiency:** Minimizing network bandwidth requirements.

Master-Slave Architecture



Master-Slave Architecture Overview

- Hadoop employs a master-slave architecture for organizing computers, achieving scalability in processing Big Data.
- **Key Components:**
  - Two primary components: NameNode (master) and DataNodes (slaves).
  - Master supervise the file system, namespace, and access control, while slaves store and serve data blocks.
- **Cluster Hierarchy:**
  - A small Hadoop cluster includes a single master and multiple worker nodes (DataNodes).
  - A large cluster features a master and thousands of worker nodes.
- **Transaction Log:**
  - Master node uses a transaction log to persistently record changes in the file system.
  - Ensures data integrity and recovery in case of failures.

Master Node Functions

- **File System Management:**
  - Master node (NameNode) manages the overall file system and its namespace.
  - Controls access to files by clients and is aware of data-nodes storing file blocks.
- **Processing Plan:**
  - Controls the processing plan for all applications running on the cluster.
  - Plans execution of tasks on data stored across the cluster.
- **Single Point of Failure Reduction:**
  - Hot backup is always ready to take over if the master node fails unexpectedly.
  - Transaction log records every change persistently to facilitate recovery.
- **Health Monitoring:**
  - Receives heartbeat messages from DataNodes periodically.
  - Automatically replaces failed DataNodes and ensures high availability.

Worker Nodes (DataNodes)

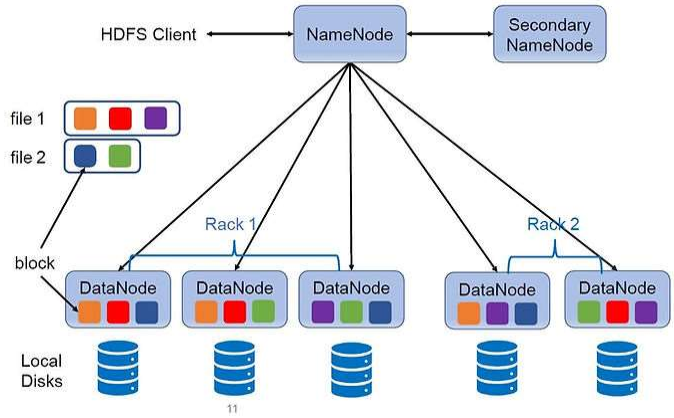
- **Storage and Retrieval:**
  - DataNodes store data blocks in their storage space.
  - typically contains multiple disks to maximize storage capacity and access speed.
- **File System Unawareness:**
  - Worker nodes have no awareness of the distributed file structure.
  - Store each block of data as directed by the NameNode.
- **Replication and Data Movement:**
  - DataNodes store and serve up blocks of data over the network using a block protocol.
  - Replicate data to ensure fault tolerance and replace failed DataNodes.
- **Heartbeat Mechanism:**
  - DataNodes send heartbeat messages to the NameNode periodically.
  - Lack of heartbeat indicates a dead DataNode, triggering replication efforts.

DataNode Replication Process

- **Replication for Fault Tolerance:**
  - If a DataNode fails, data on the failed node is accessed from replicas on other DataNodes.
  - Failed DataNode can be automatically recreated on another machine.
- **Information Stored by NameNode:**
  - NameNode stores information about every DataNode, including name, rack, capacity, and health.
  - For every file, it maintains details such as name, replicas, type, size, location, and health.
- **Balancing Storage and Computing Load:**
  - NameNode ensures files are evenly spread across DataNodes to balance storage and computing load.
  - Optimizes networking load for efficient data retrieval and processing.
- **Fragment Distribution:**
  - Fragments of files are distributed across multiple nodes to balance processing load and speed up overall processing.

Replication Management

- Each block is replicated 3 times and stored on different DataNodes



Hadoop File System Features

- **Splintering and Scattering:**
  - Hadoop File System hides the splintering and scattering of data from users.
  - Enables users to deal with data at a high, logical level.
- **Balanced Storage:**
  - NameNode ensures even distribution of files across DataNodes for balanced storage and computing load.
  - Minimizes loss from the failure of a single node or rack.
- **Optimized Networking:**
  - Optimizes networking load by selecting fragments from multiple nodes during data retrieval or processing.
  - Stores fragments on the same node for speed of read and write operations.
- **DataNode Collaboration:**
  - DataNodes can communicate to rebalance data, move copies, and maintain high data replication.
  - Three replicas are typically stored: two on the same rack, one on a different rack.

Block System

- **Fundamental Storage Unit:**
  - A block of data is the basic storage unit in HDFS.
  - HDFS stores large files by dividing them into segments called blocks.
- **Block Characteristics:**
  - Ranges from 16–128MB (default size: 64MB).
  - Each block resides on a different DataNode if possible.
- **Organization:**
  - Files are organized as consecutively numbered blocks.
  - Blocks stored physically close for efficient access.
- **Configurability:**
  - Block size and replication factor are configurable.
  - Typically, each piece of data is stored on three nodes for fault tolerance.

Ensuring Data Integrity

- **Immutable Files:**
  - Files are written once, never updated in place.
  - Readable many times, ensuring data integrity.
- **Automatic Recovery:**
  - If data on a DataNode is lost or corrupted, healthy replicas are automatically recreated.
  - Replication ensures fault tolerance.
- **Checksum Algorithm:**
  - Applied to all data written to HDFS.
  - Ensures data integrity during storage and processing.
- **Concurrency Control:**
  - Only one client can write or append to a file at a time.
  - No concurrent updates allowed for data consistency.

Installing HDFS

- **Deployment Options:**
  - Hadoop can run on in-house clusters or on the cloud.
- **System Requirements:**
  - Hadoop is written in Java; a working Java installation is required.
  - Hadoop needs enough memory; 1GB per million file fragments is a rule of thumb.
- **Installation Methods:**
  - GUI options like Cloudera Resource Manager for easy installation.
  - Command-line installation involves downloading Hadoop from Apache mirror sites.

Reading and Writing Data

- List Files in a Directory:
  - `hdfs dfs -ls /user/`
- Create a Directory:
  - `hdfs dfs -mkdir /user/test`
- Change Ownership of a Directory:
  - `hdfs dfs -chown hdfs:hdfs /user/test/`
- Copy a Local File to HDFS:
  - `hdfs dfs -put "/usr/mybigdata/test.txt" /user/test/`
- View Contents of a File in HDFS:
  - `hdfs dfs -cat /user/test/test.txt`

Sequence Files

- **Handling Small Files:**
  - Sequence Files are designed for smaller files with smaller record sizes.
  - Uses a key-value pair format for efficient storage of smaller objects.
- **Efficiency in Storage and Processing:**
  - Small files are packed into a Sequence File container for efficient storage and processing.
  - HDFS and MapReduce are optimized to work with large files.
- **Commands and Formats:**
  - Easy commands for creating, reading, and writing Sequence Files.
  - Sorting and merging Sequence Files is native to the MapReduce system.
- **Benefits:**
  - Enhances efficiency for handling smaller files in a Big Data ecosystem.

YARN (Yet Another Resource Negotiator)

- YARN serves as the architectural center of Hadoop, functioning as a large-scale, distributed operating system for big data applications.
- **Multi-Tool Compatibility:**
  - Characterized as a secure multi-tenant environment, YARN allows running various tools and applications (e.g., Hive, Spark, MapReduce) on a single HDFS storage platform.
- **Resource Management:**
  - YARN manages resources and monitors workloads across multiple Hadoop clusters.
  - Ensures high availability and flexibility for dynamic allocation of cluster resources.
- **Scalability and Utilization:**
  - Enhances scalability, expanding clusters beyond 1000 nodes.
  - Dynamically allocates resources to applications, improving overall cluster utilization.
- **Components:**
  - Resource Manager: Includes Scheduler and Applications Manager components.
  - Scheduler allocates resources, while Applications Manager handles job submissions and monitors execution.

Conclusion

- Hadoop stands as the dominant technology for managing Big Data.
- Its distributed nature, particularly the Hadoop Distributed File System (HDFS), securely stores data on large clusters of commodity machines.
- **Key Components and Functionality:**
  - Master-slave architecture, comprising NameNode and DataNodes, ensures fault tolerance and data integrity.
  - YARN, as the resource manager, adds flexibility and scalability to the Hadoop ecosystem.
- **Deployment and Installation:**
  - Hadoop can be deployed on in-house clusters or the cloud.
  - Installation, though resource-intensive, offers multiple options, including GUI tools and command-line methods.
- **Efficient Handling of Data:**
  - From the block system and data integrity to reading/writing data and handling smaller files using Sequence Files, Hadoop provides a comprehensive solution for Big Data challenges.