Hadoop and Its Architecture: In-Depth Understanding

---

1. Introduction to Hadoop

1.1 What is Hadoop?

Hadoop is an open-source framework designed to store and process vast amounts of data efficiently. It enables distributed storage and computation across clusters of commodity hardware, making it an essential tool for Big Data processing.

1.2 Key Features of Hadoop:

- **Scalability:** Easily scale from a single server to thousands of machines.
- **Fault Tolerance:** Automatic data replication ensures data availability.
- **Cost Efficiency:** Leverages inexpensive hardware to handle large-scale data.
- **Flexibility:** Processes structured, semi-structured, and unstructured data.

2. Hadoop Architecture Overview

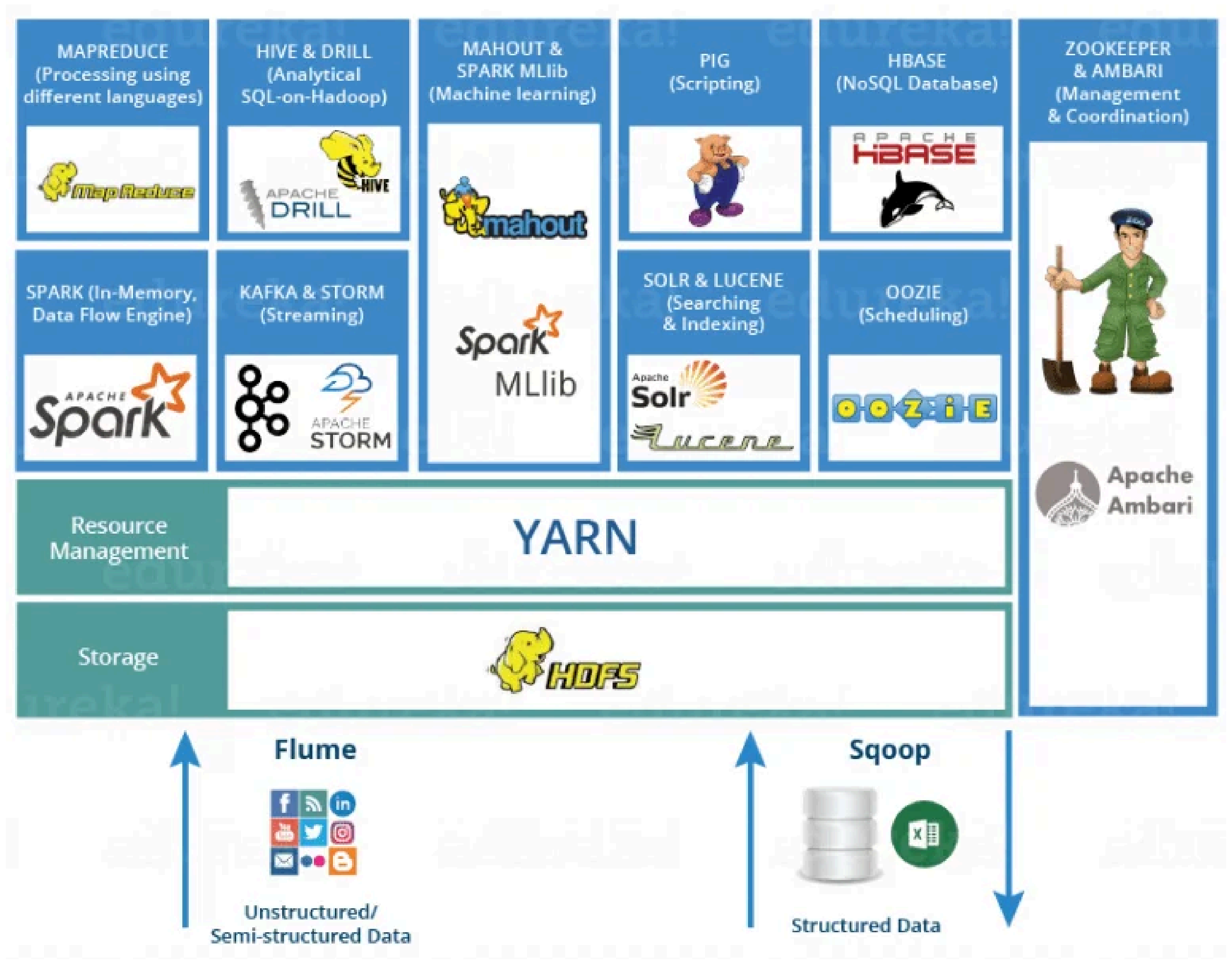
2.1 High-Level Architecture:

Hadoop consists of three core components:

1. **HDFS (Hadoop Distributed File System):** Distributed storage system.
2. **YARN (Yet Another Resource Negotiator):** Resource management layer.
3. **MapReduce:** Programming model for data processing.

2.2 Hadoop Core Modules:

- **HDFS:** Splits large files into smaller blocks and stores them redundantly across multiple nodes.
- **YARN:** Allocates resources for processing tasks.
- **MapReduce:** Processes data in parallel across the cluster.


MAPREDUCE (Processing using different languages)
HIVE & DRILL (Analytical SQL-on-Hadoop)
MAHOUT & SPARK MLlib (Machine learning)
PIG (Scripting)
HBASE (NoSQL Database)
ZOOKEEPER & AMBARI (Management & Coordination)
SPARK (In-Memory, Data Flow Engine)
KAFKA & STORM (Streaming)
SOLR & LUCENE (Searching & Indexing)
OOZIE (Scheduling)
Resource Management
YARN
Storage
HDFS
Apache Ambari
Flume
Unstructured/ Semi-structured Data
Sqoop
Structured Data

source

---

3. HDFS (Hadoop Distributed File System)

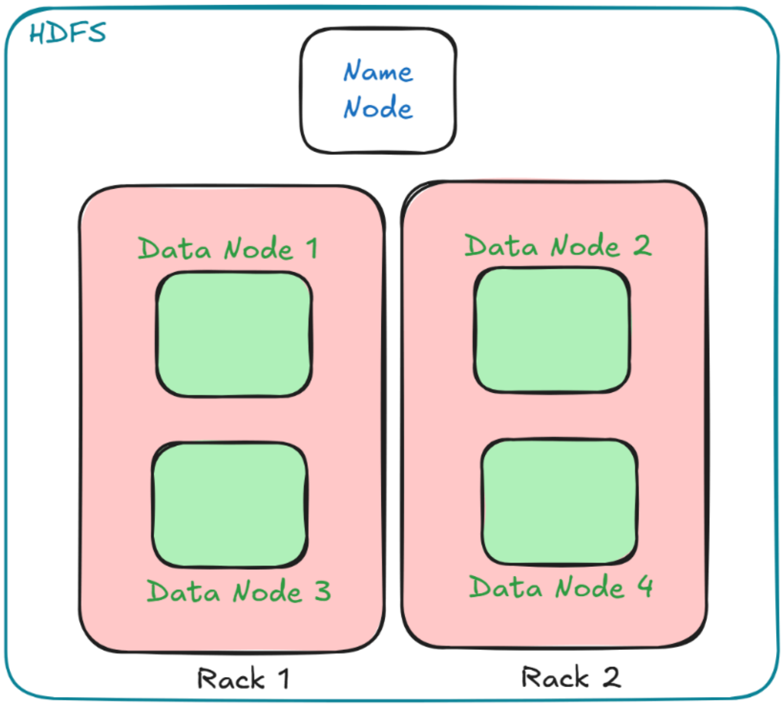
3.1 What is HDFS?

HDFS is a distributed file system designed to store large datasets reliably and stream them at high bandwidth to applications.

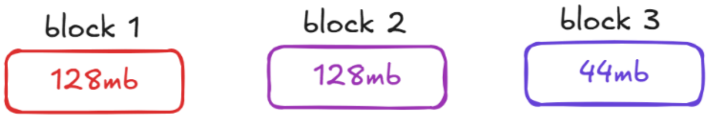
3.2 Components of HDFS:

- **NameNode:** Manages the metadata of files and directories.
- **DataNode:** Stores actual data blocks.
- **Secondary NameNode:** Periodically merges the NameNode's logs and metadata to prevent data loss.

Name node will have the meta data about all the file storage in block and node

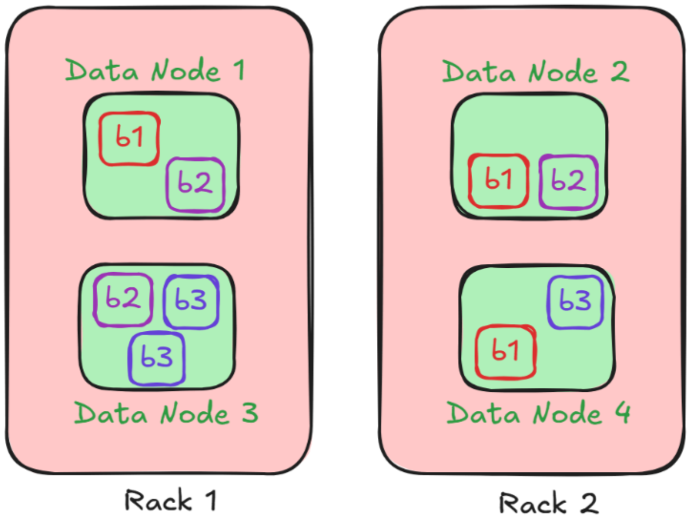


Let us say we have a file for 300mb,
by default our block size in data node is 128mb,
now the file will get split in to three blocks.



Note - available space in block 3 is wasted.

Now let us see how these blocks are placed in data node
with default replication factor 3, HDFS stores 3 copies of each block for fault tolerance.



3.3 Working of HDFS: HDFS is designed to store and retrieve large datasets in a distributed manner. Here is how it works:

1. **File Splitting:**
   - Large files are split into fixed-size blocks (default size: 128MB).
   - This allows parallel storage and retrieval across multiple nodes.
2. **Data Replication:**
   - Each block is replicated across multiple DataNodes (default replication factor: 3).
   - This ensures fault tolerance, as the loss of one node doesn't result in data loss.
3. **Metadata Management:**
   - The NameNode maintains a record of the metadata, including file-to-block mapping and block-to-DataNode mapping.
   - This helps track where each block of a file is stored in the cluster.
4. **Data Write Process:**
   - When a client uploads a file, it first communicates with the NameNode.
   - The NameNode determines the block placement and replication strategy.
   - The client writes the blocks directly to the allocated DataNodes.
5. **Data Read Process:**
   - The client requests the NameNode for file metadata.
   - The NameNode provides the block locations and their corresponding DataNodes.
   - The client reads the blocks directly from the DataNodes in parallel, ensuring high throughput.
6. **Fault Tolerance:**
   - If a DataNode fails, the NameNode detects the failure through heartbeat signals.

    - Missing blocks are replicated to other DataNodes to maintain the desired replication factor.
7. **Rack Awareness:**
    - HDFS uses a rack-awareness algorithm to optimize block placement.
    - Blocks are stored on different racks to ensure data availability even if an entire rack fails.

3.4 Block Size in HDFS:

Why Default is 128MB?

- **Optimized for Large Files:** Minimizes the number of blocks for large datasets, reducing metadata overhead.
- **Efficient Data Transfers:** Larger blocks result in fewer disk I/O operations, improving throughput.
- **Reduced Metadata Management:** Fewer blocks mean less metadata stored on the NameNode, saving memory.

Impact of Increasing/Decreasing Block Size

1. **Increasing Block Size:**
    - Benefits large files by reducing the number of blocks.
    - Improves sequential read/write performance.
    - Reduces parallelism, as fewer tasks are created.
2. **Decreasing Block Size:**
    - Suitable for datasets with smaller files.
    - Increases parallelism by creating more tasks for processing.
    - May lead to higher metadata overhead and increased NameNode memory usage.

Choosing the right block size depends on the workload:

- **Large files:** Opt for larger block sizes to minimize overhead.
- **Many small files:** Use smaller block sizes for better resource utilization and parallelism.

3.5 Advantages and Limitations of HDFS:

- **Advantages:** Scalability, fault tolerance, and cost efficiency.
- **Limitations:** High latency for small files, write-once-read-many architecture.

---

4. YARN (Yet Another Resource Negotiator)

4.1 What is YARN?

YARN is a resource management framework that enables multiple applications to run on Hadoop simultaneously.

4.2 Components of YARN:

- **ResourceManager:** Allocates cluster resources to various applications.
- **NodeManager:** Monitors resource usage on individual nodes.
- **ApplicationMaster:** Manages the execution of a single application.

4.3 Working of YARN:

- ResourceManager assigns resources to ApplicationMaster.
- ApplicationMaster coordinates tasks across NodeManagers.
- NodeManagers execute tasks and report progress to ResourceManager.

---

5. MapReduce

5.1 What is MapReduce?

MapReduce is a programming model for parallel data processing. It divides tasks into smaller sub-tasks to process data efficiently.

5.2 Phases of MapReduce:

1. **Map Phase:** Processes input data and produces intermediate key-value pairs.
2. **Shuffle and Sort Phase:** Groups and sorts data by keys.
3. **Reduce Phase:** Aggregates and processes grouped data to produce the final output.

5.3 Workflow Example: A word count program:

1. Mapper counts words in each line of a text file.
2. Shuffle phase groups words across mappers.
3. Reducer sums word counts and produces the final tally.

5.4 Advantages and Challenges of MapReduce:

- **Advantages:** Scalability, fault tolerance, and simplicity.
- **Challenges:** High disk I/O and complex debugging.

---

6. Hadoop Ecosystem Components

6.1 Hive: SQL-like querying for structured data.
6.2 Pig: Simplified scripting for data transformations.
6.3 HBase: NoSQL database for real-time data access.
6.4 Spark: Fast in-memory processing engine.
6.5 Zookeeper: Coordination and synchronization service.
6.6 Other Tools: Sqoop (data import/export), Flume (log collection), Oozie (workflow scheduling).
