<u>In-Depth Tutorial on Apache Hadoop</u>

Submitted By:

Sachin Kumar Singh

Reg. No: 12415237

Course: MCA (H) Data Science

Table of Contents

1. Introduction

The Apache® Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

2. Core Concepts

Big Data

Refers to datasets that are too large, complex, or fast-changing (characterized by the *3 Vs*: Volume, Variety, and Velocity) to be effectively managed and analyzed using traditional data processing tools and systems.

Distributed Computing

An approach where multiple interconnected computers (nodes) work together to perform tasks. It enables parallel processing, scalability, and efficient handling of big data by breaking the workload into smaller chunks distributed across the system.

Fault Tolerance

The capability of a system to continue operating correctly even when some of its components (e.g., servers, network links) fail. In distributed computing, this is critical and is often achieved through techniques like replication, redundancy, and automated failover mechanisms.

3. Apache Hadoop Ecosystem

1. Hadoop Common

- The shared utilities, libraries, and APIs used by other Hadoop modules.
- Provides the basic infrastructure.

2. HDFS (Hadoop Distributed File System)

- A distributed file system that stores large data sets across multiple machines.
- Provides high throughput access and fault tolerance.

3. YARN (Yet Another Resource Negotiator)

- Manages resources and schedules tasks across the cluster.

- Acts as the operating system of Hadoop for job scheduling and cluster management.

4. MapReduce

- A programming model for processing large data sets in parallel.
- Breaks jobs into tasks and distributes them across nodes.

Ecosystem Tools and Projects

Category	**Tool/Project**	**Description**
Data Ingestion	Flume	Collects, aggregates, and moves large amounts of log data.
	Sqoop	Transfers data between Hadoop and relational databases.
Data Storage	HBase	A distributed NoSQL database on top of HDFS (good for real-time read/write).
Data Processing	Hive	SQL-like query engine for Hadoop. Translates SQL into MapReduce jobs.
	Pig	High-level scripting language for writing data transformation programs.
	Spark	In-memory data processing framework (faster than MapReduce).
Workflow Management	Oozie	Schedules and manages Hadoop jobs.
Data Search	Solr / Lucene	Full-text search engines that integrate with Hadoop.
Data Governance	Atlas	Data governance and metadata management.
Security	Ranger	Centralized security framework for managing access.
Coordination	Zookeeper	Centralized service for configuration, synchronization, and naming.

How It All Fits Together

1. Ingest data using Flume or Sqoop.
2. Store it in HDFS or HBase.
3. Process it using MapReduce, Spark, Hive, or Pig.
4. Schedule jobs with Oozie.
5. Search or access data with Solr or Hive.
6. Secure and monitor the entire system using Ranger, Atlas, and Zookeeper.

4. Architecture Overview

1. HDFS (Hadoop Distributed File System) – *Storage*

- Designed for storing large files across a cluster of machines.
- Stores data in blocks (default 128MB/256MB).
- Replicates data (default 3 copies) for fault tolerance.

2. YARN (Yet Another Resource Negotiator) – *Cluster Resource Management*

- Coordinates resource allocation (CPU, memory) across all nodes.
- Supports multiple data processing engines (not just MapReduce).

3. MapReduce – *Data Processing*

- A batch processing model: splits data into chunks, processes them in parallel, and aggregates the results.
- Consists of Mapper and Reducer tasks.

4. Common – *Shared Utilities*

- Includes libraries and tools used by all other Hadoop modules.
- Provides essential Java classes, file system APIs, and configuration options.

Cluster Nodes and Their Roles

Node Type	**Component**	**Role**
Master Node	NameNode	Manages file system metadata (file names, directories, block locations). No actual data.
	ResourceManager	Manages cluster resources and job scheduling via YARN.
Worker Node	DataNode	Stores the actual data blocks on local disks. Reports to NameNode.
	NodeManager	Runs on each worker node; manages execution of tasks and reports to ResourceManager.

How It Works Together

1. Data Ingestion: Files are split into blocks and stored in HDFS across DataNodes.
2. Metadata Tracking: NameNode keeps track of which blocks belong to which file.
3. Job Submission: User submits a MapReduce job.
4. Scheduling: ResourceManager coordinates resources and assigns tasks to NodeManagers.
5. Task Execution: Map and Reduce tasks are executed where data resides (data locality).
6. Results: Processed data is written back to HDFS.

5. Installation Guide

Requirements

- OS: Ubuntu/Linux
- Java: OpenJDK 8 or higher (Hadoop 3.3.6 works with Java 8+)
- SSH: Required for communication between Hadoop daemons
- Hadoop 3.3.6

Step-by-Step Hadoop Installation (Single Node / Pseudo-Distributed Mode)

Install Java

```
sudo apt update

sudo apt install openjdk-8-jdk -y
```

Verify:

```
java -version
```

Create a Hadoop User

```
sudo adduser hadoop
sudo usermod -aG sudo hadoop
su – Hadoop
```

SSH Key Setup (No Password Login for Localhost)

```
ssh-keygen -t rsa -P ""
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys
ssh localhost
```

Download and Extract Hadoop

```
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
tar -xzvf hadoop-3.3.6.tar.gz
mv hadoop-3.3.6 hadoop
```

Configure Environment Variables

Edit .bashrc:

```
nano ~/.bashrc
```

Add at the end:

```
export HADOOP_HOME=/home/hadoop/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Apply changes:

```
source ~/.bashrc
```

Configure Hadoop XML Files

Edit the following in $HADOOP_HOME/etc/hadoop/:

core-site.xml

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///home/hadoop/hadoopdata/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///home/hadoop/hadoopdata/hdfs/datanode</value>
  </property>
</configuration>
```

mapred-site.xml

First copy:

```
cp mapred-site.xml.template mapred-site.xml
```

Then edit:

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

yarn-site.xml

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

Format the NameNode

```
hdfs namenode -format
```

Start Hadoop Services

```
start-dfs.sh
start-yarn.sh
```

Check if everything's running:

```
Jps
```

Access Hadoop Web UIs

- NameNode: http://localhost:9870
- ResourceManager: http://localhost:8088

6. HDFS – Hadoop Distributed File System

1. Block-Based Storage

- Files are split into fixed-size blocks (default: 128 MB, configurable).
- Blocks are distributed across multiple DataNodes.
- Enables parallel processing and efficient management of large files.

2. Data Replication

- Each block is replicated across multiple nodes (default: 3 copies).
- Ensures data availability and reliability.
- Replication factor can be set per file or globally.

3. Fault Tolerance

- If a DataNode fails, HDFS retrieves the data from other replicas.
- The NameNode monitors health of nodes and re-replicates blocks when needed.
- Ensures high availability of data despite hardware failures.

4. High Throughput Access

- Designed for **batch processing**, not low-latency access.
- Optimized for **large-scale data processing** workloads (e.g., MapReduce, Spark).

5. Write-Once, Read-Many Model

- Files are written once and then read multiple times.
- Simplifies data consistency and replication management.

Data Locality Optimization

- Computation (MapReduce tasks) is moved to where the data resides, reducing network bottlenecks.

HDFS Shell Commands:

```
# Upload file
hdfs dfs -put file.txt /input

# Read file
hdfs dfs -cat /input/file.txt

# List files
hdfs dfs -ls /

# Delete file
hdfs dfs -rm /input/file.txt
```

7. MapReduce – Processing Engine

Two-Phase Processing Model

1. Map Phase

- Purpose: Processes input data line by line.
- Action: Transforms data into key-value pairs.
- Common Use: Filtering, sorting, parsing, or tokenizing raw input.

2. Reduce Phase

- Purpose: Aggregates or summarizes the intermediate key-value pairs from the Map phase.
- Action: Combines values with the same key (e.g., sum, average, count).
- Common Use: Summing totals, counting occurrences, joining data.

Example: Word Count in MapReduce

file.txt:

```
Hello world
Hello Hadoop
```

Map Phase Output (Key-Value Pairs):

Copy code

```
<Hello, 1>

<world, 1>

<Hello, 1>

<Hadoop, 1>
```

Shuffle & Sort (Internally by Hadoop):

Groups by key:

```
<Hello, [1, 1]>

<world, [1]>

<Hadoop, [1]>
```

Reduce Phase Output:

```
<Hello, 2>

<world, 1>

<Hadoop, 1>
```

Result:

A frequency count of each word across all input files.

8. Hands-on Example: Word Count Using MapReduce

Mapper.java

```
public class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);

  private Text word = new Text();

  public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

    StringTokenizer itr = new StringTokenizer(value.toString());

    while (itr.hasMoreTokens()) {

      word.set(itr.nextToken());

      context.write(word, one);
    }
  }
}
```

Reducer.java

```
public class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    context.write(key, new IntWritable(sum));
  }
}
```

Driver.java

```
public class WordCount {
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

9. Use Cases of Apache Hadoop

1. Data Warehousing & ETL

- **Use**: Ingesting, transforming, and storing large-scale data from multiple sources.
- **Example**: Replacing traditional ETL tools to load data into Hadoop for processing and reporting.
- **Tools**: Hive, Pig, Sqoop.

2. Log and Event Processing

- **Use**: Analyzing logs from servers, applications, or IoT devices in real time or batch mode.
- **Example**: Web companies process user logs to understand behavior, detect errors, or monitor performance.
- **Tools**: Flume (for ingestion), Spark/MapReduce (for processing), HDFS (for storage).

3. Data Archiving

- **Use**: Long-term storage of data at low cost.
- **Example**: Banks storing historical transactions, telecom companies storing call records.
- **Advantage**: HDFS is cost-effective compared to traditional enterprise storage.

4. Machine Learning and Predictive Analytics

- **Use**: Training ML models on large datasets.
- **Example**: Retail companies predicting customer churn or product recommendations.
- **Tools**: Apache Mahout, Hadoop + Spark MLlib.

5. Fraud Detection

- **Use**: Detecting suspicious patterns and anomalies in large datasets.
- **Example**: Credit card companies analyzing billions of transactions in near-real time.
- **Tools**: Real-time Hadoop stack (e.g., Kafka + Spark + HDFS).

6. Customer Analytics / Personalization

- **Use**: Understanding customer behavior and preferences.
- **Example**: E-commerce companies recommending products based on user activity.
- **Tools**: Hive, HBase, Spark.

7. Sentiment Analysis and Social Media Mining

- **Use**: Extracting opinions and trends from large volumes of social data.
- **Example**: Brands tracking sentiment across Twitter, Facebook, etc.
- **Tools**: Hadoop + Natural Language Processing (NLP) libraries.

8. Healthcare Analytics

- **Use**: Storing and analyzing medical records, sensor data, or genomics data.
- **Example**: Hospitals identifying disease patterns or optimizing treatment plans.

Bonus Use Case: Infrastructure Monitoring

- Companies use Hadoop to analyze logs and metrics from servers to predict failures, scale resources, or audit usage.

10. Advantages

Scalability

- Hadoop can scale horizontally by simply adding more nodes (machines) to the cluster.
- No major changes to applications or configurations are needed.
- Can handle petabytes of data by distributing storage and processing.

Cost-Effective

- Open-source software—no licensing fees.
- Runs on commodity hardware, not high-end servers.
- Significantly lowers the cost of storing and processing big data compared to traditional systems.

High Throughput

- Optimized for batch processing of large volumes of data.
- Delivers high throughput using HDFS and MapReduce to process data in parallel across nodes.
- Ideal for workloads like log analysis, ETL, and data mining.

Fault Tolerance

- HDFS automatically replicates data blocks (default: 3 copies).
- If a node fails, Hadoop retrieves data from another replica.
- MapReduce reschedules failed tasks on healthy nodes—ensuring reliability and continuous operation.

Parallel Processing

- MapReduce breaks jobs into independent tasks that run in parallel across the cluster.
- Significantly reduces processing time, especially for large datasets.
- Ensures efficient use of all available resources.

11. Challenges and Limitations

Steep Learning Curve

- Hadoop's ecosystem involves many components (HDFS, YARN, MapReduce, Hive, Pig, etc.), each with its own complexity.
- Requires knowledge of Java, Linux/Unix commands, distributed computing concepts, and cluster administration.
- Not beginner-friendly without prior big data or system admin experience.

High Latency for Small Jobs

- MapReduce jobs have significant startup overhead and are optimized for large, batch jobs.
- For small or interactive queries, it performs poorly compared to tools like Apache Spark or Presto.

Complex Configuration and Tuning

- Hadoop requires manual configuration of dozens of settings (memory, block size, replication, JVM options, etc.).
- Performance tuning is non-trivial and often requires deep understanding of the cluster and workloads.

Not Real-Time (Batch Only)

- MapReduce is inherently batch-oriented, meaning it's not suitable for real-time processing.
- Systems like Apache Spark Streaming, Apache Flink, or Apache Kafka are better suited for real-time or near-real-time use cases.

Inefficient for Small Files

- HDFS stores metadata for each file in memory on the NameNode.
- A large number of small files (e.g., thousands of 1KB logs) can overload the NameNode, leading to performance issues.
- Hadoop works best with fewer, larger files.

Common Workarounds or Alternatives

- Use Apache Spark for faster, in-memory processing.
- Use Apache HBase for real-time, NoSQL-like access.
- Use data ingestion frameworks (like Flume or Kafka) to batch small files before sending to HDFS.
- Run Hadoop on managed services like Amazon EMR to simplify setup and scaling.

12. Conclusion

Apache Hadoop remains a foundational technology in Big Data engineering. While newer real-time processing tools like Apache Spark have emerged, Hadoop's reliability, scalability, and robust ecosystem continue to make it a vital tool in the data engineering toolbox. Mastery of Hadoop opens doors to large-scale data analytics and ETL pipelines in various industries.

13. References

- https://hadoop.apache.org
- "Hadoop: The Definitive Guide" by Tom White
- Cloudera Documentation