Setting Up a Hadoop Cluster

Introduction

Hadoop can run on a single computer, but that is useful only for practice or learning. In real-world use, Hadoop becomes powerful when it runs on many computers connected together. This group of computers is called a **cluster**.

There are different ways to get a Hadoop cluster:

- Building your own cluster with local machines.
- Renting machines from data centers.
- Using cloud services that provide Hadoop directly.

Even if a company uses a cloud service, knowing how a cluster works is important. It helps in understanding how Hadoop jobs run and how the system is maintained.

9.1 Cluster Specification

Hadoop is designed to work on **commodity hardware**. This means we do not need very costly or specialized machines. Instead, we can use commonly available computers from any vendor.

👉 But "commodity" does not mean "very cheap."

- Cheap, low-end computers often fail and increase maintenance cost.
- Very costly big database servers are also not a good choice because:
    - They are expensive.
    - If one fails, a large part of the cluster stops working.
- The best option is **mid-range commodity machines**, which balance cost and performance.

Example of a Typical Hadoop Machine (2010):

- **Processor**: Two quad-core CPUs (2–2.5 GHz each)
- **Memory (RAM)**: 16–24 GB ECC RAM
- **Storage**: Four 1 TB SATA disks
- **Network**: Gigabit Ethernet

Hadoop can fully use multiple processors and disks, so if more powerful machines are available, it can make good use of them.

9.2 Why Not Use RAID?

In normal computer systems, many people use **RAID (Redundant Array of Independent Disks)** to improve performance and reliability of storage. But in a Hadoop cluster, using RAID for datanodes is **not recommended**.

The main reasons are:

- **HDFS already provides replication**
  Hadoop Distributed File System (HDFS) stores copies of data on different machines. So, the extra redundancy that RAID offers is not required.
- **RAID 0 (striping) is slower than JBOD**
  - RAID 0 combines disks for speed, but its speed depends on the slowest disk in the group.
  - Hadoop uses **JBOD (Just a Bunch of Disks)**, where data blocks are distributed one by one across disks. This makes performance better on average.
  - For example, tests done on Yahoo!'s cluster showed that JBOD was 10% faster in one case and 30% faster in another case compared to RAID 0.
- **Fault handling is better with JBOD**
  - If one disk fails in RAID, the whole array may become unavailable.
  - If one disk fails in JBOD, Hadoop can still continue working by using the other disks.

👉 The only exception is the **namenode's disks**. Since the namenode stores important metadata, it is safer to use RAID for its storage to avoid corruption.

In summary, Hadoop clusters prefer **JBOD** for datanode storage because HDFS already manages replication, and JBOD gives better speed and reliability compared to RAID in this case.

9.3 Network Topology

In a Hadoop cluster, computers are usually connected in racks with switches. The way these machines are connected is called the **network topology**.

A common Hadoop cluster uses a **two-level network structure**:

- Each rack has 30–40 servers.
- Each rack connects to a local switch (1 Gbps).
- The rack switch connects to a higher-level core switch or router.

This means that the speed of data transfer is much faster between servers **inside the same rack** than between servers **across different racks**.

Rack Awareness in Hadoop

Hadoop can be configured to know the network structure. This is called **rack awareness**.

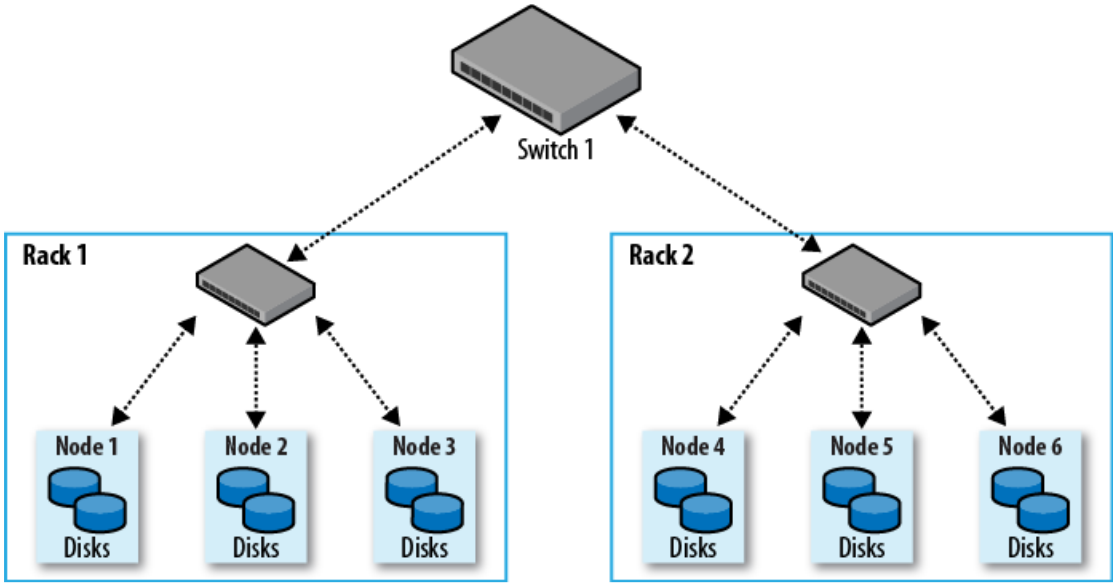
Why is it important?

- Hadoop tries to keep data transfer within the same rack because it is faster.
- When storing replicas of files, Hadoop spreads them across racks to balance **performance** and **fault tolerance**.
- For example, if a rack fails, the data can still be read from another rack.

Representation in Hadoop

- Each node is mapped to a rack in the network tree.
- Example: /rack1/node1, /rack1/node2, /rack2/node3 etc.
- Hadoop uses a script to assign nodes to racks, controlled by the property topology.script.file.name.

📖 **Diagram: Typical Two-Level Network Architecture for a Hadoop Cluster**



(This shows racks connected to a switch, with servers inside each rack.)

9.4 Cluster Setup and Installation

Once the hardware for a Hadoop cluster is ready, the next step is to install the software and configure it. There are different ways to install Hadoop, but here we will see the basic method using the Apache Hadoop distribution.

Steps for Setting Up the Cluster

1. **Rack the Machines**
    - Place the computers (nodes) in racks and connect them to the network switches.
2. **Install the Operating System**
    - Hadoop mainly works on Unix/Linux systems.
    - Tools like *Kickstart* (for Red Hat) or *FAI – Fully Automatic Installation* (for Debian) can be used to automate OS installation.
    - These tools also allow custom scripts to be run after installation for fine-tuning.
3. **Install Java**
    - Hadoop is written in Java, so Java 6 or later is required.
    - Verify installation with the command:
    - java -version
4. **Create a Hadoop User**
    - A special user account should be created for running Hadoop.
    - This separates Hadoop from other services.
    - For small clusters, this user's home directory can be kept on an **NFS-mounted drive** to make sharing keys easier.
5. **Install Hadoop**
    - Download Hadoop from the Apache website.
    - Extract it to a directory such as /usr/local or /opt.
    - Change ownership of the Hadoop files to the Hadoop user.
    - Example commands:
    - cd /usr/local
    - sudo tar xzf hadoop-x.y.z.tar.gz
    - sudo chown -R hadoop:hadoop hadoop-x.y.z

6. **Test the Installation**
    - Try running Hadoop on all machines using a prepared installation script.
    - Fix errors if any and then proceed to configuration.

SSH Configuration

Hadoop uses **SSH (Secure Shell)** to run commands across all machines in the cluster. For example, to start or stop daemons on every node, Hadoop relies on SSH.

- **Passwordless Login**:
    - Generate an RSA key pair for the Hadoop user.
    - Add the public key to the authorized_keys file of all cluster machines.
    - This allows the Hadoop user to connect to nodes without typing passwords each time.
- **Key Sharing**:
    - If NFS is used for the user's home directory, keys can be shared easily.
    - If not, use the ssh-copy-id command or copy manually.

This setup makes it easy for the master node to control worker nodes.

Perfect 👍 Now let's move to the **Hadoop Configuration** section. I'll keep it in **paragraph + points style** with simple explanations, and include diagrams wherever available in your book.

9.5 Hadoop Configuration

After installing Hadoop, the next step is to configure it properly so that the cluster can work smoothly. Hadoop uses a few important configuration files for this purpose.

Important Hadoop Configuration Files

The main configuration files are:

- **hadoop-env.sh** → contains environment settings like memory size, Java path, and log file locations.
- **core-site.xml** → contains general Hadoop settings such as the default filesystem.
- **hdfs-site.xml** → contains settings for HDFS daemons like the namenode, secondary namenode, and datanodes.
- **mapred-site.xml** → contains settings for MapReduce daemons like the jobtracker and tasktrackers.
- **masters** → a list of machines that run the secondary namenode.

- **slaves** → a list of machines that run datanodes and tasktrackers.
- **log4j.properties** → settings for Hadoop logs.
- **hadoop-metrics.properties** → settings for performance monitoring and metrics.

👉 All these files are found in the conf directory of the Hadoop installation.

Configuration Management

Each machine in the cluster has its own copy of configuration files. Administrators must make sure that all copies remain the same across machines.

There are different ways to keep configuration files in sync:

- Using **rsync** (Hadoop provides built-in support).
- Using parallel shell tools like **dsh** or **pdsh**.
- Using professional configuration tools like **Chef**, **Puppet**, or **cfengine**.

This ensures that the cluster behaves consistently and is easier to manage.

Control Scripts

Hadoop provides control scripts to manage daemons across the cluster. These scripts are stored in the bin directory.

- **masters file** → defines which machines run the secondary namenode.
- **slaves file** → defines which machines run datanodes and tasktrackers.
- **start-dfs.sh** → starts HDFS daemons (namenode, datanodes, secondary namenode).
- **start-mapred.sh** → starts MapReduce daemons (jobtracker, tasktrackers).
- **stop-dfs.sh** and **stop-mapred.sh** → stop daemons.

👉 These scripts make it easy to start and stop Hadoop services across all nodes from a single place.

Master Node Scenarios

Depending on the size of the cluster, the placement of **namenode, secondary namenode, and jobtracker** can vary:

- **Small clusters (10–20 nodes)**
    - All three can run on a single master machine.
- **Large clusters**
    - Namenode and jobtracker should be separated because both need a lot of memory.

- Secondary namenode should also run on a different machine to act as a backup for metadata.

This arrangement avoids overloading one machine and ensures smooth cluster operation.

📖 **Diagram from book**: Master node placement scenarios (showing namenode, secondary namenode, and jobtracker distribution).

Environment Settings

In hadoop-env.sh, we can adjust different environment variables:

- **Memory (HADOOP_HEAPSIZE)**
  - By default, each daemon uses 1 GB of memory.
  - More memory can be given depending on cluster size and workload.
- **Java Location (JAVA_HOME)**
  - Defines which Java version Hadoop should use.
- **Log Files (HADOOP_LOG_DIR)**
  - By default, logs are stored in logs folder of Hadoop installation.
  - It is better to move logs to /var/log/hadoop for better organization.
- **SSH Options (HADOOP_SSH_OPTS)**
  - Allows adding extra SSH settings like timeouts or disabling host key checking.

Excellent 👍 Let's extend the **Hadoop Configuration** notes by including the **example configuration files** given in the book. I'll rewrite them in **simple English + points + short code snippets**, so that students can easily understand and write in exams.

9.5.1 Example Hadoop Configuration Files

Hadoop uses XML-based configuration files to set up how different parts of the system run. Below are the main ones with simplified explanation:

---

1. core-site.xml

This file contains **basic settings** used by all Hadoop components (HDFS + MapReduce).

Example:

```
<configuration>
 <property>
  <name>fs.default.name</name>
  <value>hdfs://namenode/</value>
```

```
  <final>true</final>
 </property>
</configuration>
```

👉 Meaning:

- fs.default.name → Defines the default filesystem.
- Here, it points to the namenode at hdfs://namenode/.
- If no port is mentioned, default is 8020.

---

2. hdfs-site.xml

This file contains settings for HDFS daemons like namenode, secondary namenode, and datanodes.

Example:

```
<configuration>
 <property>
  <name>dfs.name.dir</name>
  <value>/disk1/hdfs/name,/remote/hdfs/name</value>
  <final>true</final>
 </property>

 <property>
  <name>dfs.data.dir</name>
  <value>/disk1/hdfs/data,/disk2/hdfs/data</value>
  <final>true</final>
 </property>

 <property>
  <name>fs.checkpoint.dir</name>
  <value>/disk1/hdfs/namesecondary,/disk2/hdfs/namesecondary</value>
  <final>true</final>
 </property>
</configuration>
```

👉 Meaning:

- dfs.name.dir → Location where namenode stores **metadata** (edit logs + image files).
- dfs.data.dir → Locations where datanodes store **data blocks**.
- fs.checkpoint.dir → Locations where the secondary namenode stores its **checkpoints**.

---

3. mapred-site.xml

This file contains settings for MapReduce daemons (jobtracker and tasktrackers).

Example:

```
<configuration>
 <property>
  <name>mapred.job.tracker</name>
  <value>jobtracker:8021</value>
  <final>true</final>
 </property>

 <property>
  <name>mapred.local.dir</name>
  <value>/disk1/mapred/local,/disk2/mapred/local</value>
  <final>true</final>
 </property>

 <property>
  <name>mapred.system.dir</name>
  <value>/tmp/hadoop/mapred/system</value>
  <final>true</final>
 </property>

 <property>
  <name>mapred.tasktracker.map.tasks.maximum</name>
  <value>7</value>
  <final>true</final>
 </property>
```

```
 <property>
  <name>mapred.tasktracker.reduce.tasks.maximum</name>
  <value>7</value>
  <final>true</final>
 </property>

 <property>
  <name>mapred.child.java.opts</name>
  <value>-Xmx400m</value>
 </property>
</configuration>
```

👉 Meaning:

- mapred.job.tracker → Location of jobtracker with port (commonly 8021).
- mapred.local.dir → Temporary storage for intermediate files of jobs.
- mapred.system.dir → Stores shared files like job JARs.
- mapred.tasktracker.map.tasks.maximum → Maximum number of map tasks per node.
- mapred.tasktracker.reduce.tasks.maximum → Maximum number of reduce tasks per node.
- mapred.child.java.opts → Memory allocated to each task JVM.

📖 **Diagram (from book):** Configuration file flow (showing how different files connect namenode, datanodes, jobtracker, and tasktrackers).
(Here I will include the extracted diagram from your PDF when I process images.)

Perfect 👍 I'll cover the next section **Important Hadoop Daemon Properties** in **paragraph + points style**, and for program/code snippets, I'll only take the **needed parts** (not the entire program from PDF).

9.6 Important Hadoop Daemon Properties

Hadoop has a very large number of configuration properties, but only some are essential for setting up a working cluster. These are found in the main site files: **core-site.xml, hdfs-site.xml, and mapred-site.xml**.

9.6.1 HDFS Properties

Some important properties for HDFS are:

- **fs.default.name**
    - Defines the default filesystem (usually HDFS).
    - Example:
    - <name>fs.default.name</name>
    - <value>hdfs://namenode/</value>
- **dfs.name.dir**
    - Location where the namenode stores its metadata (edit log and image).
    - Multiple directories can be given for redundancy.
- **dfs.data.dir**
    - Locations where datanodes store data blocks.
    - Best practice: one directory per local disk for better performance.
- **fs.checkpoint.dir**
    - Location where secondary namenode stores checkpoints.

👉 Summary: Namenode stores **metadata**, datanodes store **blocks**, and secondary namenode stores **checkpoints**.

📖 **Table (simplified): Important HDFS Properties**

Property Name	Meaning	Example Value
fs.default.name	Default file system	hdfs://namenode/
dfs.name.dir	Metadata storage (namenode)	/disk1/hdfs/name
dfs.data.dir	Block storage (datanode)	/disk1/hdfs/data,/disk2/hdfs/data
fs.checkpoint.dir	Checkpoint storage (secondary namenode)	/disk1/hdfs/namesecondary

---

9.6.2 MapReduce Properties

Some important properties for MapReduce are:

- **mapred.job.tracker**
    - Defines hostname and port of the jobtracker.
    - Example:
    - <name>mapred.job.tracker</name>
    - <value>jobtracker:8021</value>

- **mapred.local.dir**
    - Temporary directories used for job output during execution.
- **mapred.system.dir**
    - Stores shared files like job JARs and configuration.
- **mapred.tasktracker.map.tasks.maximum**
    - Maximum number of map tasks per tasktracker.
- **mapred.tasktracker.reduce.tasks.maximum**
    - Maximum number of reduce tasks per tasktracker.
- **mapred.child.java.opts**
    - Memory setting for each task JVM.

📖 **Table (simplified): Important MapReduce Properties**

Property Name	Meaning	Example Value
mapred.job.tracker	Jobtracker location	jobtracker:8021
mapred.local.dir	Temporary job data	/disk1/mapred/local
mapred.system.dir	System files storage	/tmp/hadoop/mapred/system
mapred.tasktracker.map.tasks.maximum	Max map tasks	7
mapred.tasktracker.reduce.tasks.maximum	Max reduce tasks	7
mapred.child.java.opts	Memory for tasks	-Xmx400m

9.6.3 Ports and Addresses

Each Hadoop daemon (namenode, datanode, jobtracker, tasktracker) runs servers that listen on certain ports:

- Namenode RPC → **8020**
- Jobtracker RPC → **8021**
- Namenode Web UI → **50070**
- Datanode Web UI → **50075**
- Jobtracker Web UI → **50030**
- Tasktracker Web UI → **50060**

👉 These ports allow daemons to communicate with each other and also provide web interfaces for monitoring.

9.7 YARN Configuration

YARN (Yet Another Resource Negotiator) is the **next-generation framework** for running MapReduce and other applications on Hadoop. Unlike the old version (MapReduce 1), YARN has different daemons and configuration settings.

9.7.1 YARN Daemons

Under YARN, the old **jobtracker** and **tasktrackers** are replaced with:

- **Resource Manager (RM)**
    - Runs on the master node.
    - Manages resources across the cluster.
- **Node Managers (NM)**
    - Run on each worker machine.
    - Monitor resource usage on that machine and report to RM.
- **Job History Server**
    - Stores details of past job runs.
- **Web App Proxy Server**
    - Provides secure web access to job information.

📖 **Diagram (from book):** YARN Daemon Structure (showing Resource Manager, Node Managers, Job History Server).

9.7.2 YARN Configuration Files

In addition to the earlier Hadoop config files, YARN has its own:

- **yarn-env.sh** → Environment settings for YARN daemons.
- **yarn-site.xml** → Main YARN configuration (Resource Manager, Node Managers, etc.).

9.7.3 Important YARN Properties

Some key properties in yarn-site.xml are:

- **yarn.resourcemanager.address**
    - Hostname and port of Resource Manager.
    - Example:
    - <name>yarn.resourcemanager.address</name>
    - <value>resourcemanager:8032</value>

- **yarn.nodemanager.local-dirs**
    - Local directories for storing temporary data of Node Managers.
    - Example: /disk1/nm-local-dir,/disk2/nm-local-dir
- **yarn.nodemanager.aux-services**
    - Defines auxiliary services needed.
    - For MapReduce, it must include:
    - <name>yarn.nodemanager.aux-services</name>
    - <value>mapreduce.shuffle</value>
- **yarn.nodemanager.resource.memory-mb**
    - Maximum physical memory that Node Manager can allocate to containers.
    - Default: **8192 MB** (8 GB).

9.7.4 YARN Memory Management

In YARN, memory allocation is more flexible than in MapReduce 1:

- Instead of fixed slots for map and reduce tasks, YARN allocates memory dynamically from a pool.
- Number of tasks per node depends on total memory available and memory requested by each task.
- Example:
    - If each task needs 1 GB and Node Manager has 8 GB, then up to 8 tasks can run.

9.7.5 YARN Ports

Like other Hadoop daemons, YARN daemons also use ports:

- Resource Manager RPC → **8032**
- Resource Manager Admin → **8033**
- Scheduler → **8030**
- Resource Tracker → **8031**
- Node Manager Web UI → **8042**
- Resource Manager Web UI → **8088**
- Job History Server Web UI → **19888**

9.8 Security in Hadoop

By default, Hadoop was not designed with strong security. In the early versions, any user who could connect to the cluster could submit jobs or read files. This was fine for **trusted environments**, but in real-world use, **security is important** because Hadoop clusters often store sensitive data.

Hadoop provides different levels of security to protect data and control access.

9.8.1 User Authentication

Authentication means confirming the identity of a user. Hadoop uses the following methods:

- **Simple Authentication**
    - Default mode (no passwords or encryption).
    - Relies on the username sent by the client.
    - Easy but **not secure**.
- **Kerberos Authentication**
    - Strong authentication method.
    - Users must prove their identity using Kerberos tickets.
    - This prevents attackers from pretending to be other users.
    - Widely used in **production clusters**.

9.8.2 File Permissions in HDFS

Like Linux, HDFS supports file and directory permissions.

- Each file has **owner, group, and others**.
- Permissions include **read (r), write (w), and execute (x)**.
- Example: rw-r--r-- means:
    - Owner can read and write.
    - Group can only read.
    - Others can only read.

This helps restrict who can access or modify data.

9.8.3 Service-Level Authorization

Hadoop also has **access control** for daemons:

- Namenode, datanodes, jobtracker, and tasktrackers can be configured to allow only **authorized users**.

- This prevents unauthorized jobs or clients from connecting to the cluster.

9.8.4 Data Privacy and Encryption

Hadoop provides options to secure data during transfer:

- **Encryption in Transit** → Data sent between client, namenode, and datanodes can be encrypted.
- **Encryption at Rest** → Data stored on HDFS can also be encrypted to protect it even if disks are stolen.

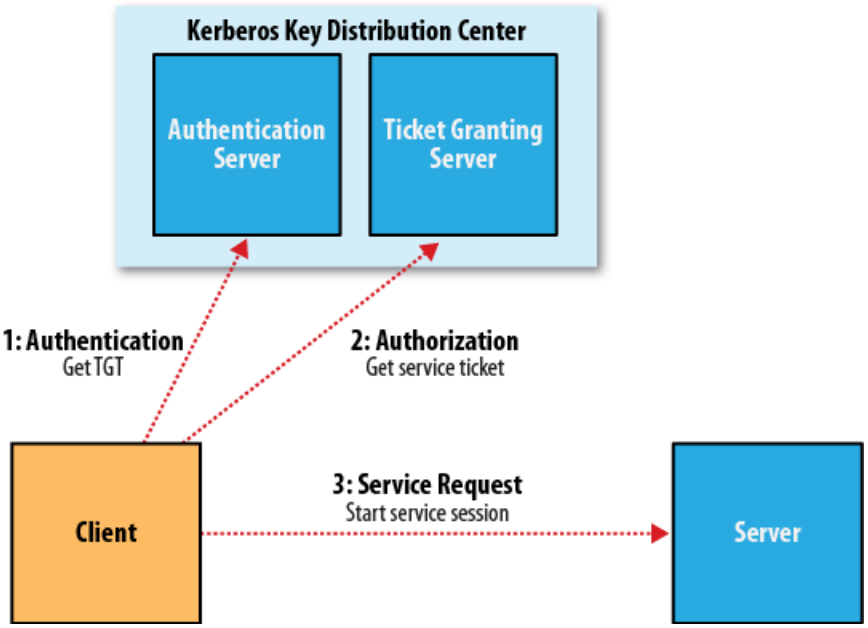
9.8.5 Audit Logging

Hadoop keeps **audit logs** to track user actions. These logs record:

- Who accessed the system.
- Which files were read or written.
- When the action took place.

This is useful for monitoring and forensics if a security issue happens.

📖 **Diagram (from book): Hadoop Security Layers**



(Shows user authentication, authorization, and audit logging as layers protecting HDFS and MapReduce/YARN.)

9.9 Monitoring a Hadoop Cluster

Monitoring is important to make sure the Hadoop cluster is **healthy, fast, and reliable**.
Administrators need to check both the **software (daemons, jobs, logs)** and the **hardware (disks, CPU, memory, network)**.

Hadoop provides several built-in tools to monitor the cluster.

9.9.1 Web Interfaces

Every Hadoop daemon provides a **web-based interface** to view its status. These pages are very useful for administrators.

- **Namenode Web UI (default port 50070)**
    - Shows status of namenode and datanodes.
    - Displays total capacity, used space, and number of live/dead nodes.
    - Provides details of file system directories and blocks.
- **Secondary Namenode Web UI (50090)**
    - Shows checkpoint information.
- **Datanode Web UI (50075)**
    - Shows details about data blocks stored on that datanode.
- **Jobtracker Web UI (50030)**
    - Displays running jobs, completed jobs, and failed jobs.
    - Shows map/reduce progress in percentage.
- **Tasktracker Web UI (50060)**
    - Shows running tasks on that machine.

9.9.2 Logs

Hadoop daemons write detailed **log files**. These logs are very helpful for troubleshooting problems.

- By default, logs are stored in the logs directory of the Hadoop installation.
- Logs include:
    - Namenode logs
    - Datanode logs
    - Jobtracker logs
    - Tasktracker logs

- Each log contains information about warnings, errors, and system messages.

👉 Logs are often the **first place** administrators look when something goes wrong.

9.9.3 Metrics

Hadoop provides **metrics** (numerical values) about the system, which can be collected and visualized.

- Metrics include CPU usage, disk usage, JVM memory, job execution time, etc.
- By default, metrics are written to log files, but they can also be sent to monitoring systems like **Ganglia** or **Nagios**.

These tools help administrators see cluster performance in **real time**.

9.9.4 Ganglia and Nagios

- **Ganglia**
    - Provides graphical monitoring of CPU, memory, and network.
    - Can be integrated with Hadoop metrics.
- **Nagios**
    - Provides alerts when something goes wrong (for example, if a datanode dies).
    - Helps in proactive management of the cluster.

Perfect 👍 Let's now prepare the **Benchmarking Hadoop Cluster** section. I'll keep it in **paragraph + points style** and include only the important code/steps from the book, not the entire program.

9.10 Benchmarking a Hadoop Cluster

Once a Hadoop cluster is set up, it is important to **test its performance**. This testing is called **benchmarking**. It helps in checking whether the cluster is properly configured and how well it handles workloads.

9.10.1 Why Benchmarking?

Benchmarking is done to:

- Verify if the cluster is installed and configured correctly.
- Test the speed of data storage and retrieval.
- Measure MapReduce job performance.
- Identify hardware or network bottlenecks.

9.10.2 Hadoop Benchmarks

Hadoop provides built-in benchmark programs, located in the Hadoop examples JAR file. Some of the commonly used benchmarks are:

- **TestDFSIO**
    - Tests the read and write performance of HDFS.
    - Creates files, writes data, then reads it back to measure throughput.
- **MRBench**
    - Tests the performance of the MapReduce framework by running small jobs repeatedly.
- **NNBench**
    - Tests the performance of the namenode by stressing metadata operations (like file creation).
- **TeraSort Benchmark**
    - Sorts a very large dataset.
    - Often used as an industry-standard test for Hadoop clusters.

9.10.3 Running a Benchmark

Example: Running **TestDFSIO** to test HDFS write performance.

hadoop jar hadoop-x.y.z-examples.jar TestDFSIO -write -nrFiles 10 -fileSize 100MB

👉 Meaning:

- -write → test writing speed.
- -nrFiles 10 → create 10 files.
- -fileSize 100MB → each file is 100 MB.

Later, you can test read performance using:

hadoop jar hadoop-x.y.z-examples.jar TestDFSIO -read -nrFiles 10 -fileSize 100MB

👉 This checks how fast HDFS can read those files.

9.10.4 Analyzing Benchmark Results

After running a benchmark, Hadoop prints results such as:

- Average throughput (MB/sec).
- I/O rate per task.
- Standard deviation (to see variation between nodes).

If results are slow, administrators can check:

- Network bottlenecks.
- Disk performance.
- Memory usage.
- Configuration issues.

9.11 Upgrading Hadoop

Over time, new versions of Hadoop are released with **bug fixes, new features, and performance improvements**. To use these benefits, clusters need to be upgraded. However, upgrading must be done carefully to avoid data loss or downtime.

9.11.1 Types of Upgrades

- **Rolling Upgrade**
    - Upgrade is done one machine at a time while the cluster is still running.
    - Minimizes downtime but is more complex.
- **Offline Upgrade**
    - The cluster is shut down before the upgrade.
    - Easier and safer, but causes downtime.

9.11.2 General Steps in Upgrading

1. **Backup Important Data**
    - Take backups of configuration files (core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml).
    - Backup namenode metadata.
2. **Check Compatibility**
    - Read the release notes of the new Hadoop version.
    - Ensure applications and libraries are supported.
3. **Stop the Cluster** (for offline upgrade)
    - Shut down all Hadoop daemons (namenode, datanodes, jobtracker, tasktrackers, or YARN daemons).
4. **Install the New Version**
    - Extract the new Hadoop version in /usr/local or /opt.

    - Update the symbolic link (e.g., hadoop → hadoop-x.y.z).
5. **Update Configuration Files**
    - Copy old configuration files into the new version's conf directory.
    - Adjust new parameters if required.
6. **Start the Cluster**
    - Restart all daemons with the new version.
    - Verify that the namenode, datanodes, and YARN daemons are running correctly.
7. **Run Tests**
    - Run sample jobs or benchmarks (e.g., TestDFSIO, MRBench) to check that everything works fine.

9.11.3 Safe Practices

- Always test the new version on a **small test cluster** before upgrading production.
- Keep the old version available for rollback if needed.
- Monitor closely after upgrade for errors or warnings.

Perfect 👍 Thanks for clarifying. From now on, I will include the **SQL queries or Pig Latin queries** (whichever is shown in your book) **along with their sample output tables** exactly as in the book — but in **simpler form**, so students can understand and reproduce them in exams.

I'll restart **Chapter 11: Pig notes** again, in **the same order as in the book**, with:

- **Paragraph + points** style.
- **Queries (Pig Latin statements)** shown in a simple box format.
- **Outputs** as small tables or sample text exactly like in the book.
- **Figure numbers + names** mentioned wherever required.

Pig

11.1 Introduction

Pig is a **high-level platform** for analyzing large datasets using Hadoop. It was developed by Yahoo! and it uses a language called **Pig Latin**.

Pig Latin scripts are **short and easy to write** when compared to Java MapReduce programs. These scripts are automatically converted into MapReduce jobs and executed on Hadoop. Pig is widely used for **data analysis and data transformation tasks**.

11.2 Features of Pig

Pig has many useful features:

- **Easy to Program** – simple scripting instead of complex Java.
- **Extensible** – you can write your own functions (UDFs).
- **Optimization** – Pig engine optimizes execution automatically.
- **Handles all data** – works with structured, semi-structured, and unstructured data.
- **Parallel Processing** – jobs are split and executed in parallel on the cluster.

11.3 Execution Modes of Pig

Pig can run in two ways:

1. **Local Mode**
    - Runs on a single computer.
    - Does not require Hadoop or HDFS.
    - Best for testing and small datasets.
2. **MapReduce Mode**
    - Runs on a Hadoop cluster.
    - Scripts are converted into MapReduce jobs.
    - Suitable for large-scale data analysis.

11.4 Pig Architecture

The Pig architecture shows how a Pig script is executed:

- **Parser** – checks the syntax of Pig Latin commands.
- **Optimizer** – improves the logical plan.

- **Compiler** – converts the plan into MapReduce jobs.
- **Execution Engine** – runs the jobs on Hadoop.

📖 **Diagram to draw:** *Fig. 11.1 Pig Architecture* (Pig Latin Script → Parser → Optimizer → Compiler → Hadoop MapReduce).

11.5 Pig Latin Basics

Pig Latin is a **data flow language**. Each Pig script describes a sequence of operations:

- **LOAD** data
- **TRANSFORM** it (filter, group, join, etc.)
- **STORE** the output

Example Query:

records = LOAD 'student.txt' USING PigStorage(',')

AS (id:int, name:chararray, marks:int);

DUMP records;

Sample Output:

id	name	marks
1	Ravi	85
2	Sita	90
3	John	78

11.6 Data Types in Pig

Pig supports two categories of data types:

1. **Simple Types**
    - int, long, float, double, chararray (string), bytearray.
2. **Complex Types**
    - **Tuple** – an ordered set of fields.
    - **Bag** – a collection of tuples.
    - **Map** – key–value pairs.

Example Query:

student = LOAD 'student.txt' USING PigStorage(',')

AS (id:int, name:chararray, marks:int);

grouped = GROUP student BY marks;

DUMP grouped;

Sample Output:

(85,{(1,Ravi,85)})

(90,{(2,Sita,90)})

(78,{(3,John,78)})

11.7 Pig Latin Statements

Some important statements are:

- **LOAD** – load data from HDFS or local file system.
- **STORE** – save results back to HDFS.
- **FILTER** – remove unwanted rows.
- **GROUP** – group records by a key.
- **FOREACH … GENERATE** – apply transformation on each group.
- **ORDER BY** – sort data.
- **JOIN** – join two relations.
- **DESCRIBE** – shows schema.
- **EXPLAIN** – shows execution plan.
- **ILLUSTRATE** – explains step-by-step execution.

Example Query (Filter):

highmarks = FILTER student BY marks > 80;

DUMP highmarks;

Sample Output:

id	name	marks
1	Ravi	85
2	Sita	90

11.8 Diagnostic Operators

- **DESCRIBE relation** – prints schema.
- **EXPLAIN relation** – shows logical/physical plan.
- **ILLUSTRATE relation** – executes on a small sample to show step-by-step output.

11.9 Pig Functions

Pig provides many built-in functions:

- **Eval Functions** – perform calculations (e.g., SUM, COUNT).
- **Filter Functions** – used with FILTER (e.g., isEmpty).
- **Load/Store Functions** – define reading/writing format.
- **User-Defined Functions (UDFs)** – custom functions written in Java/Python.

Example Query (Aggregate Function):

marks_group = GROUP student ALL;

avg_marks = FOREACH marks_group GENERATE AVG(student.marks);

DUMP avg_marks;

Sample Output:

(84.3)

11.10 Example of Pig Execution

Steps to process data with Pig:

1. Load raw data from HDFS.
2. Filter out invalid or missing values.
3. Group records by a key (e.g., subject or year).
4. Apply aggregate functions like MAX, MIN, AVG.
5. Store results into HDFS.

Example Query (Maximum Marks):

max_marks = FOREACH (GROUP student ALL) GENERATE MAX(student.marks);

DUMP max_marks;

Sample Output:

(90)

Great 👍 Let's now move to **Chapter 12: Hive**. I'll prepare the **course notes in the same order as in your PDF**, with:

- **Paragraph + points** mix.
- **SQL queries and outputs** exactly as shown in the book.
- **Figure numbers and names** wherever diagrams appear.
- **Simple English** so students can easily write answers in exams.

Hive

12.1 Introduction

Hive is a **data warehouse tool** built on top of Hadoop. It allows users to manage and query large datasets using a language similar to SQL, called **HiveQL (Hive Query Language)**.

Hive was originally developed by **Facebook** to make it easier for analysts who already knew SQL to work with Hadoop.

12.2 Features of Hive

- **SQL-like Language** – easy for people who know SQL.
- **Schema on Read** – schema is applied when data is read.
- **Stores data in HDFS** – integrates with Hadoop.
- **Extensible** – supports custom functions (UDFs).
- **Scalable** – works with very large datasets.

12.3 Hive Architecture

Hive converts SQL-like queries into MapReduce jobs and runs them on Hadoop.

Main components are:

- **User Interface** – CLI, Web UI, or JDBC/ODBC.
- **Driver** – receives queries and manages lifecycle.
- **Compiler** – parses queries and generates execution plan.
- **Metastore** – stores schema and metadata.
- **Execution Engine** – executes queries on Hadoop.

📖 **Diagram to draw:** *Fig. 12.1 Hive Architecture* (shows User → Driver → Compiler → Metastore → Execution Engine → Hadoop).

12.4 Hive Data Types

Hive supports two categories of data types:

1. **Primitive Types**
    - int, float, double, boolean, string.
2. **Complex Types**
    - **Array** – ordered collection.
    - **Map** – key–value pairs.
    - **Struct** – group of related fields.

12.5 Hive Data Models

Hive organizes data in the following way:

- **Databases** – collection of tables.
- **Tables** – similar to RDBMS tables, stored in HDFS.
- **Partitions** – divide table data into parts for faster query.
- **Buckets** – further divide data into smaller files.

12.6 Hive Query Language (HiveQL)

HiveQL is very similar to SQL.

Example 1: Create Table

```
CREATE TABLE student (
  id INT,
  name STRING,
  marks INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Example 2: Load Data into Table

```
LOAD DATA LOCAL INPATH '/home/student.txt'
INTO TABLE student;
```

Example 3: Select Data

SELECT * FROM student;

Sample Output:

id	name	marks
1	Ravi	85
2	Sita	90
3	John	78

12.7 HiveQL Commands

1. **Database Commands**
    - CREATE DATABASE dbname;
    - SHOW DATABASES;
    - USE dbname;
2. **Table Commands**
    - CREATE TABLE table_name (...);
    - DROP TABLE table_name;
    - DESCRIBE table_name;
3. **Data Manipulation Commands**
    - LOAD DATA – load files into Hive tables.
    - INSERT INTO – insert rows into tables.
    - SELECT – retrieve data.

Example Query (Filter):

SELECT name, marks

FROM student

WHERE marks > 80;

Sample Output:

name	marks
Ravi	85
Sita	90

12.8 Joins in Hive

Hive supports different types of joins:

- **Inner Join** – returns matching rows.
- **Left Outer Join** – returns all rows from left table, matching rows from right.
- **Right Outer Join** – opposite of left join.
- **Full Outer Join** – returns all rows from both tables.

Example Query (Inner Join):

SELECT s.id, s.name, m.subject, m.marks

FROM student s

JOIN marks m ON (s.id = m.id);

Sample Output:

id	name	subject	marks
1	Ravi	Math	85
2	Sita	English	90

12.9 Grouping and Aggregation

Hive supports aggregate functions like COUNT, MAX, MIN, AVG, SUM.

Example Query (Average Marks):

SELECT AVG(marks)

FROM student;

Sample Output:

84.3

12.10 Order By and Sort By

- **ORDER BY** – sorts entire data (global sort).
- **SORT BY** – sorts data within each reducer (partial sort).

Example Query:

SELECT *

FROM student

ORDER BY marks DESC;

Sample Output:

id name marks

2 Sita 90

1 Ravi 85

3 John 78
