Hadoop Installation and Configuration (TP1)

Yehdih Mohamed Yehdih C20854

June 14, 2025

Abstract

This report details the installation, configuration, and testing of Hadoop 3.2.1 on an Ubuntu 16.04 LTS virtual machine for the first technical project (TP1). It covers setting up prerequisites (Java and SSH), installing Hadoop, configuring a single-node cluster, optimizing for a laptop environment, and running a MapReduce job to estimate Pi. Key components—Java, SSH, Hadoop, `core-site.xml`, `hdfs-site.xml`, HDFS formatting, HDFS and YARN startup, and MapReduce—are explained with their purpose, rationale, and real-world examples. Architecture diagrams illustrate SSH communication flow and Hadoop ecosystem components. Screenshots are provided as placeholders, with explanations of the commands and their purpose above each image. The report is structured as a professional academic article to guide replication of the Hadoop setup.

Contents

1 Introduction

Hadoop is an open-source framework for distributed storage and processing of large datasets across computer clusters. It comprises the Hadoop Distributed File System (HDFS) for reliable storage and the MapReduce framework for parallel processing. HDFS uses a master/slave architecture with a NameNode managing metadata and DataNodes storing data blocks. MapReduce employs a JobTracker to schedule tasks and TaskTrackers to execute them, optimizing for data locality.

This report documents the Hadoop TP1 process, covering prerequisite setup, installation, configuration, and testing. Each step is explained, focusing on Java, SSH, Hadoop, `core-site.xml`, `hdfs-site.xml`, HDFS formatting, HDFS and YARN startup, and MapReduce, with their purpose, rationale, and real-world examples. Architecture diagrams and screenshots are included with command explanations.

2 Key Components

2.1 Java

Purpose: Java, via the Java Development Kit (JDK), provides the runtime environment for Hadoop, as Hadoop is written in Java.

Why We Use It: Java's platform independence and robust libraries support Hadoop's cross-platform deployment and complex operations. OpenJDK 8 ensures compatibility with Hadoop 3.2.1, enabling daemons and jobs to execute reliably.

Real-World Example: Netflix uses Java-based Hadoop to process streaming data for recommendation systems, relying on the JDK to run MapReduce jobs analyzing viewer preferences.

2.2 SSH

Purpose: Secure Shell (SSH) enables secure communication between Hadoop daemons, facilitating remote command execution.

Why We Use It: SSH ensures secure, automated management of distributed nodes, critical for Hadoop's scripts to start/stop daemons without manual intervention. Passwordless SSH via key-based authentication streamlines operations.

Real-World Example: A bank uses SSH in its Hadoop cluster to secure communication between nodes processing financial transactions, allowing automated maintenance like restarting DataNodes.

2.3 Hadoop

Purpose: Hadoop provides distributed storage (HDFS) and processing (MapReduce) for large datasets.

Why We Use It: Hadoop's scalability and fault tolerance handle big data efficiently, processing terabytes across nodes with minimal latency. Its open-source nature reduces costs.

Real-World Example: Amazon uses Hadoop to analyze customer purchase data, enabling targeted product recommendations by processing vast datasets.

2.4 core-site.xml

Purpose: The `core-site.xml` file defines core Hadoop properties, such as the default file system and NameNode address.

Why We Use It: It ensures clients connect to the correct NameNode for HDFS operations, centralizing configuration for cluster communication.

Real-World Example: A social media platform configures `core-site.xml` to point to its NameNode, allowing servers to store user posts via `hdfs dfs -put`.

2.5 hdfs-site.xml

Purpose: The `hdfs-site.xml` file configures HDFS settings, including storage directories and replication factor.

Why We Use It: It specifies where data and metadata are stored and how data is replicated, optimizing storage for performance and reliability. A replication factor of 1 suits single-node setups.

Real-World Example: A healthcare provider uses `hdfs-site.xml` to configure HDFS for storing patient records, ensuring quick data retrieval.

2.6 Formatting HDFS

Purpose: Formatting HDFS initializes the file system, creating metadata structures for the NameNode.

Why We Use It: It ensures a clean HDFS state, preventing errors from residual data. Formatting is required for new or reset clusters.

Real-World Example: An e-commerce company formats HDFS before launching an analytics cluster to ensure accurate storage of sales data.

2.7 Starting HDFS and YARN

Purpose: Starting HDFS activates NameNode and DataNode for storage, while starting YARN activates ResourceManager and NodeManager for job execution.

Why We Use It: These services enable Hadoop's core functionalities—data storage and processing—making the cluster operational.

Real-World Example: A weather forecasting agency starts HDFS and YARN to store sensor data and run MapReduce jobs for weather predictions.

2.8 MapReduce

Purpose: MapReduce processes large datasets in parallel via map (data transformation) and reduce (aggregation) phases.

Why We Use It: It simplifies distributed computing, leveraging data locality to minimize network overhead and scale processing efficiently.

Real-World Example: Google uses MapReduce to index web pages, mapping URLs to content and reducing them to searchable indexes.

3 Prerequisites

3.1 Java Installation

I verified Java installation to ensure Hadoop's runtime environment was available:

```
javac -version
```

If Java was absent, I installed OpenJDK 8 to provide the necessary JDK:

```
sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

Why These Commands: The `javac -version` command checks if the Java compiler is installed, confirming JDK availability. The `sudo apt-get` commands update the package index and install OpenJDK 8, ensuring Hadoop can run its Java-based processes.

3.2 SSH Configuration

I checked if the SSH daemon was running to support Hadoop's secure communication:

```
ps aux | grep sshd
```

If absent, I installed SSH:

```
sudo apt-get install ssh
```

I configured passwordless SSH for automated daemon management:

```
ssh-keygen -t rsa
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
chmod 0600 $HOME/.ssh/authorized_keys
```

I verified passwordless access:

```
ssh localhost
```

Why These Commands: The `ps aux | grep sshd` command checks for the SSH daemon, ensuring SSH is operational. The `sudo apt-get install ssh` command installs SSH if missing. The `ssh-keygen` and related commands generate and configure SSH keys for passwordless access, critical for Hadoop's automation. The `ssh localhost` command tests this setup.

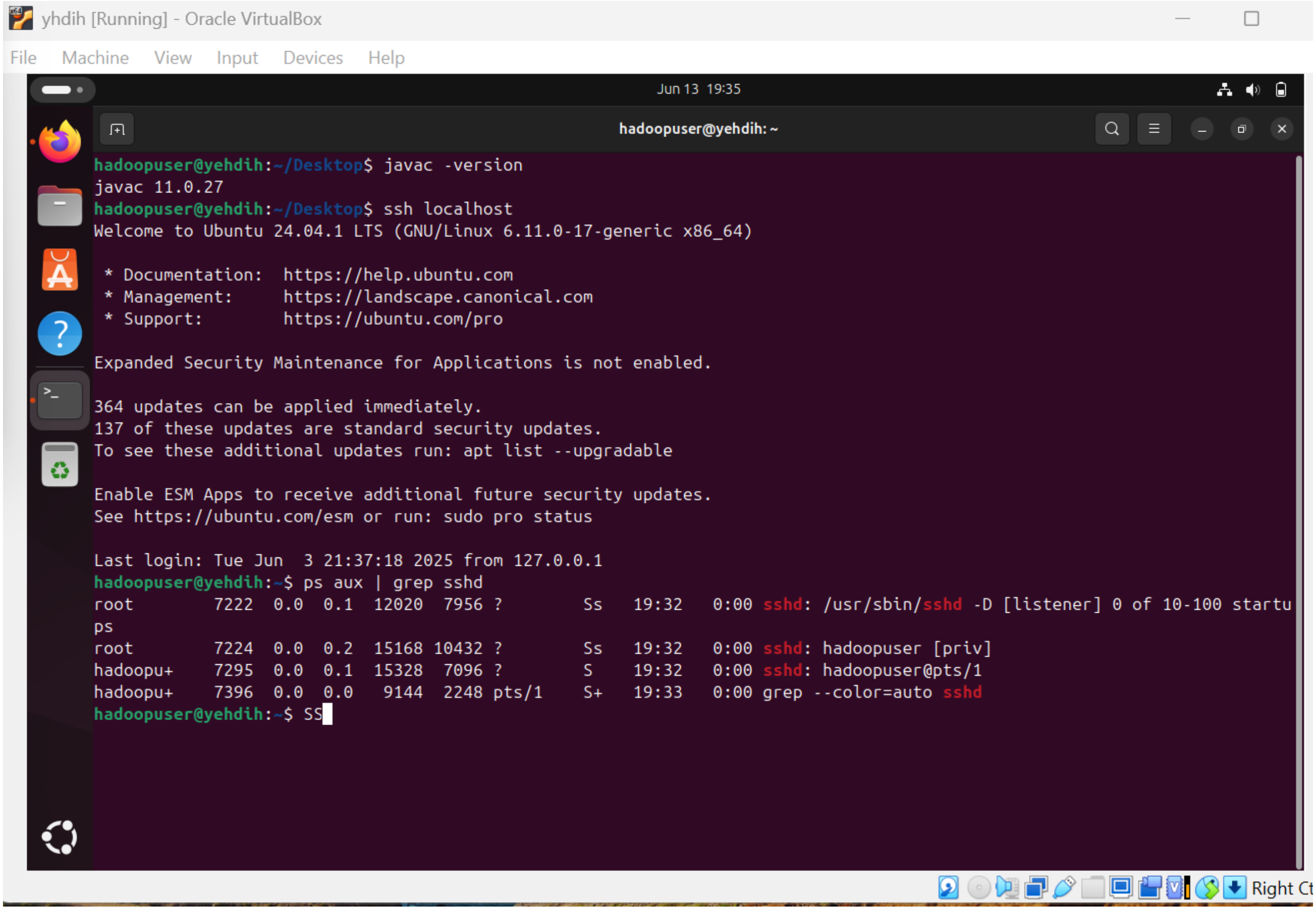

Figure 1: Passwordless SSH verification

4 Installing Hadoop

I downloaded and extracted Hadoop 3.2.1 to set up the framework:

```
wget http://miroir.univ-lorraine.fr/apache/hadoop/common/hadoop
   -3.2.1/hadoop-3.2.1.tar.gz
tar -xvzf hadoop-3.2.1.tar.gz
```

Why These Commands: The `wget` command downloads the Hadoop distribution, and `tar -xvzf` extracts it, creating the directory with Hadoop's binaries and configuration files, enabling cluster setup.

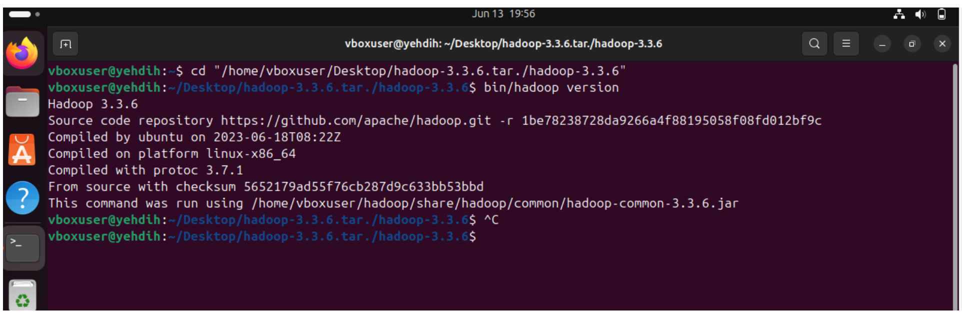

Figure 2: Hadoop installation process

4.1 Step 3.3: core-site.xml

In `hadoop-3.2.1/etc/hadoop/core-site.xml`, I configured the NameNode address:

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

I verified port availability:

```
netstat -a | grep tcp | grep LISTEN | grep 9000
```

Why These Commands: Editing `core-site.xml` sets the NameNode's address for HDFS clients. The `netstat` command ensures port 9000 is free, preventing conflicts with other services.

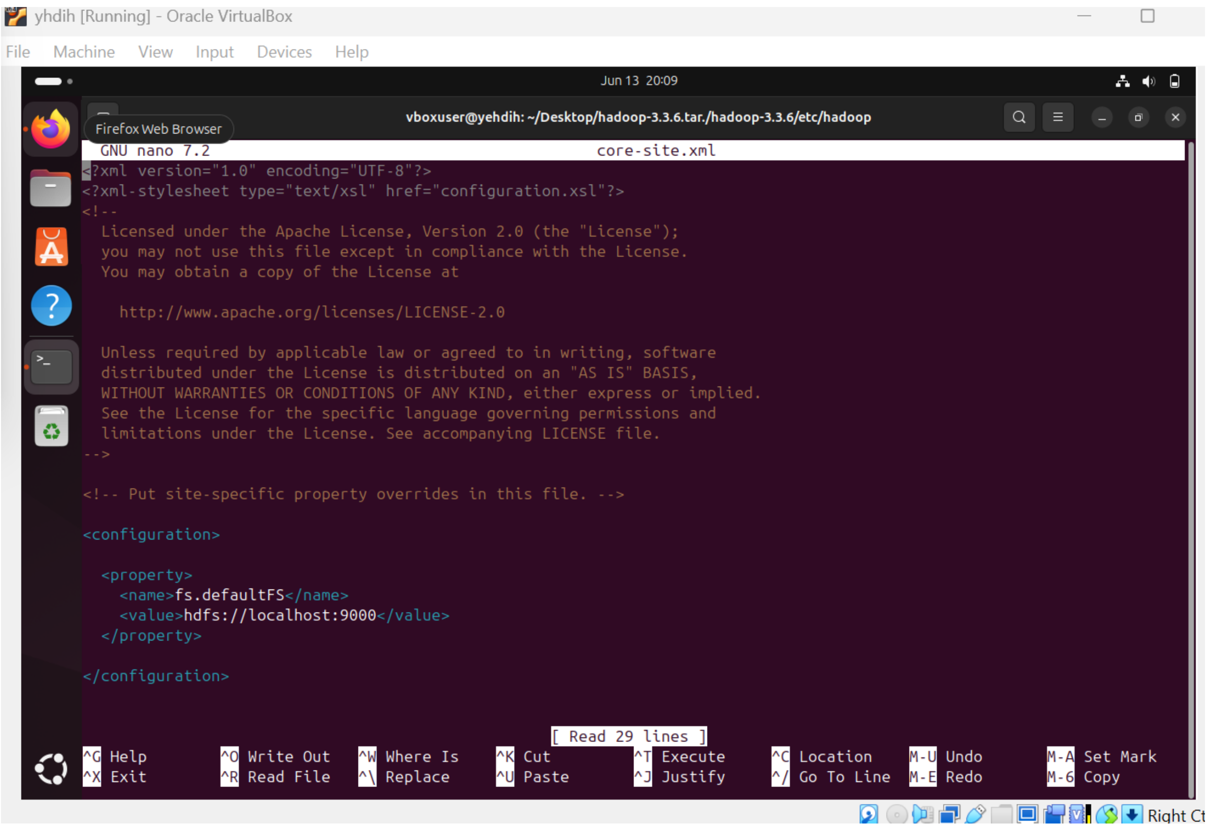

Figure 3: core-site.xml configuration

4.2 Step 3.4: hdfs-site.xml

In `hadoop-3.2.1/etc/hadoop/hdfs-site.xml`, I configured HDFS storage:

```
<configuration>
    <property>
        <name>dfs.datanode.data.dir</name>
```

```
        <value>file:///tmp/hadoop-3.2.1/hdfs/datanode</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:///tmp/hadoop-3.2.1/hdfs/namenode</value>
    </property>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

Why This Command: Editing `hdfs-site.xml` specifies storage directories and sets a replication factor of 1, suitable for a single-node cluster, ensuring proper HDFS operation.

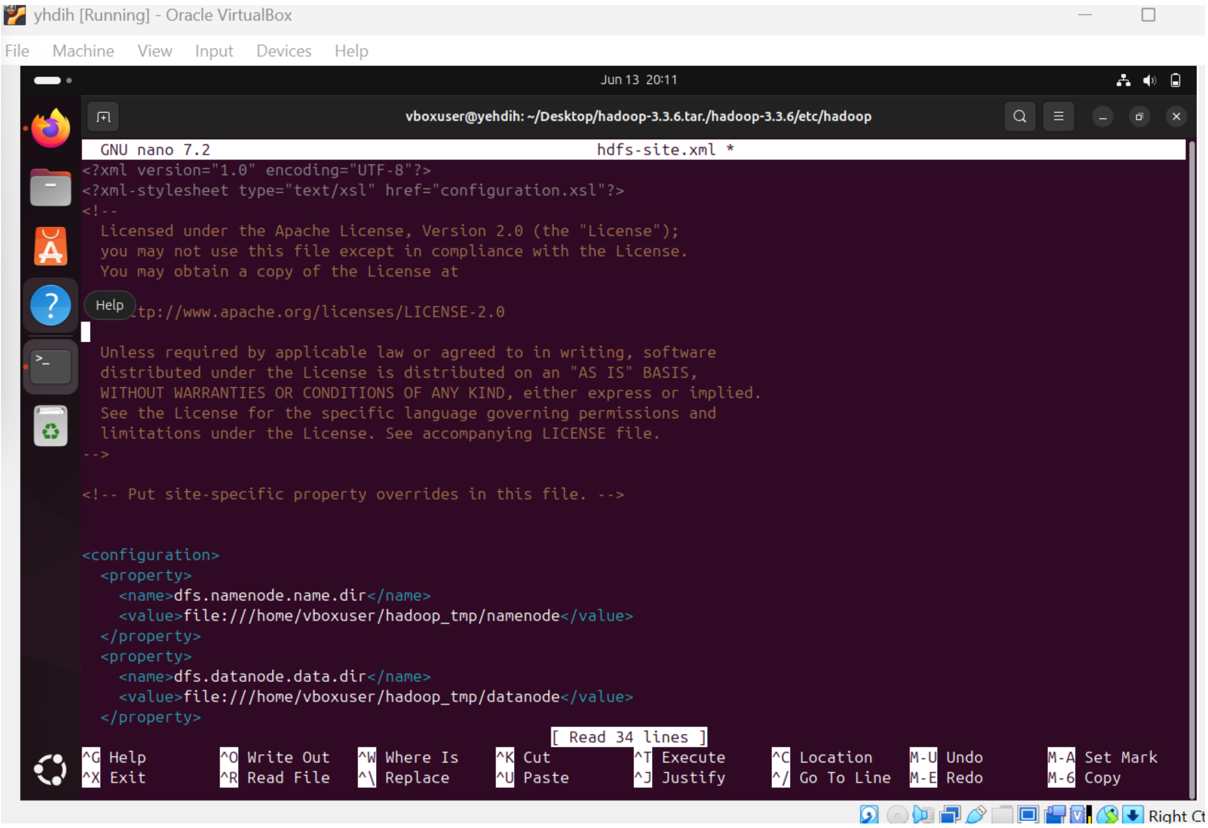

Figure 4: hdfs-site.xml configuration

4.3 Step 3.5: yarn-site.xml

In `hadoop-3.2.1/etc/hadoop/yarn-site.xml`, I configured YARN:

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

```
        <name>yarn.nodemanager.env-whitelist</name>
        <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,
           HADOOP_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,
           HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
    </property>
</configuration>
```

Why This Command: Editing `yarn-site.xml` enables YARN to support MapReduce jobs by configuring auxiliary services and environment variables.

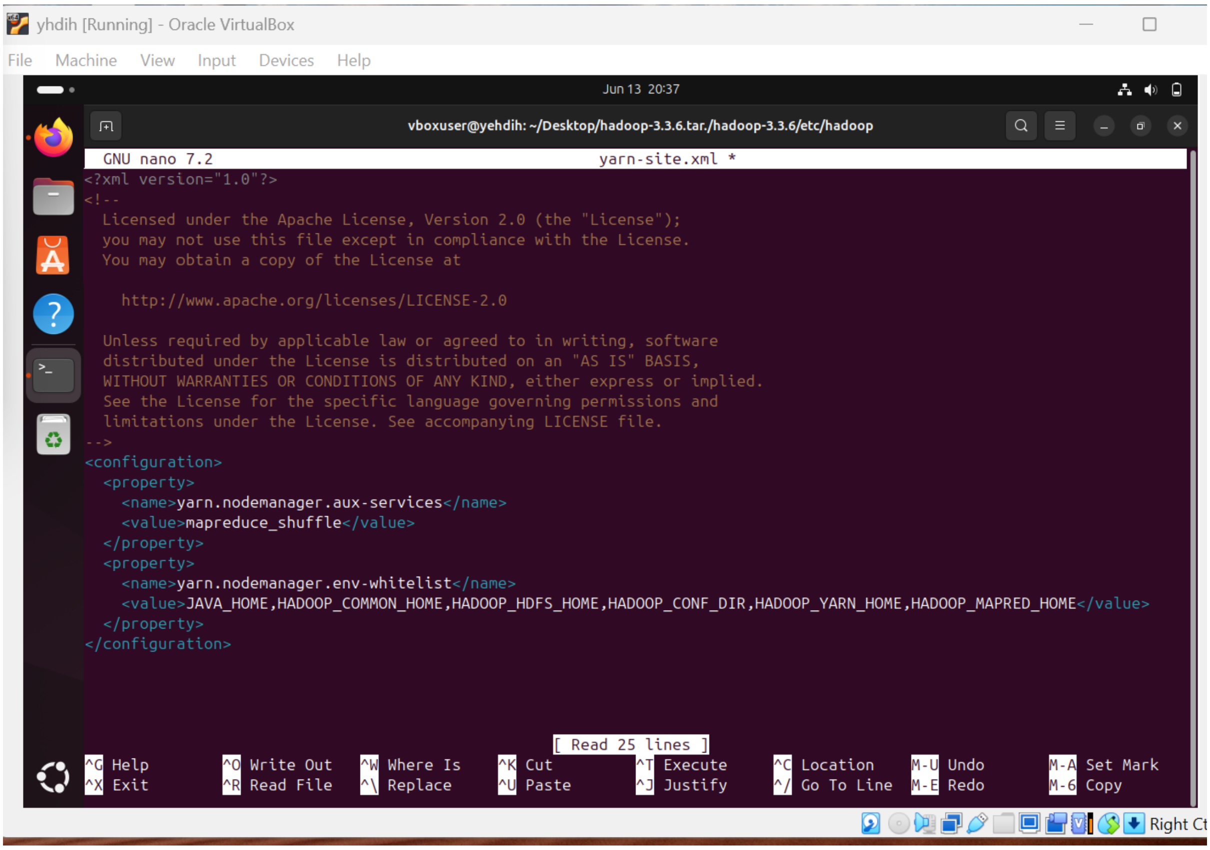

Figure 5: yarn-site.xml configuration

4.4 Step 3.6: mapred-site.xml

In `hadoop-3.2.1/etc/hadoop/mapred-site.xml`, I configured MapReduce:

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
        <name>mapreduce.application.classpath</name>
        <value>$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/*,
           $HADOOP_MAPRED_HOME/share/hadoop/mapreduce/lib/*</
           value>
    </property>
```

```
</configuration>
```

Why This Command: Editing `mapred-site.xml` specifies that MapReduce runs on YARN and sets library paths, enabling job execution.

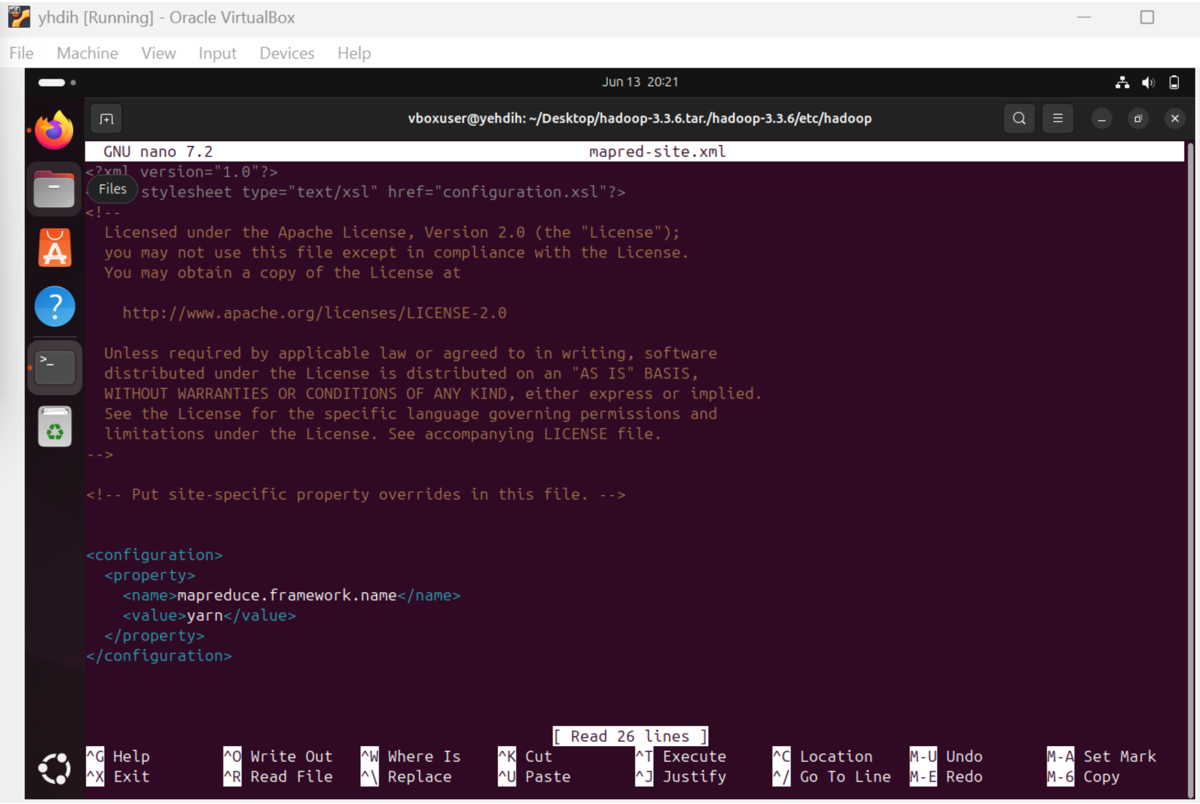

Figure 6: mapred-site.xml configuration

5 Configuration for Laptop Environment

5.1 Step 4.1: yarn-site.xml

In `hadoop-3.2.1/etc/hadoop/yarn-site.xml`, I adjusted resource limits:

```
<configuration>
    <property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>3072</value>
    </property>
    <property>
        <name>yarn.nodemanager.resource.cpu-vcores</name>
        <value>2</value>
    </property>
    <property>
        <name>yarn.scheduler.minimum-allocation-mb</name>
        <value>256</value>
    </property>
    <property>
```

```
        <name>yarn.scheduler.maximum-allocation-mb</name>
        <value>1536</value>
    </property>
    <property>
        <name>yarn.scheduler.increment-allocation-mb</name>
        <value>256</value>
    </property>
    <property>
        <name>yarn.nodemanager.vmem-check-enabled</name>
        <value>false</value>
    </property>
    <property>
        <name>yarn.nodemanager.vmem-pmem-ratio</name>
        <value>4</value>
    </property>
</configuration>
```

Why This Command: Editing `yarn-site.xml` reduces YARN's resource requirements to fit a laptop's virtual machine, ensuring stable performance.

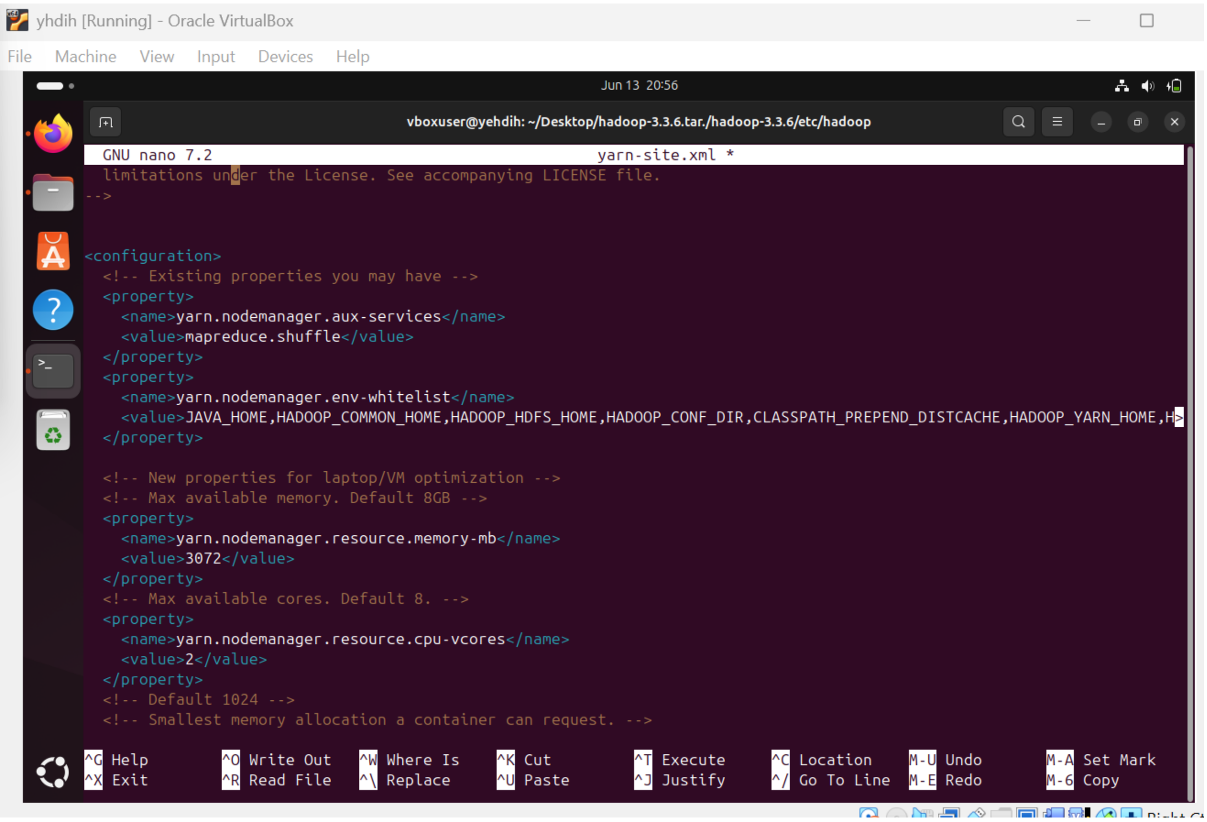

Figure 7: yarn-site.xml for laptop

5.2 Step 4.2: mapred-site.xml

In `hadoop-3.2.1/etc/hadoop/mapred-site.xml`, I set memory limits:

```
<configuration>
```

```xml
    <property>
        <name>mapreduce.map.memory.mb</name>
        <value>256</value>
    </property>
    <property>
        <name>mapreduce.reduce.memory.mb</name>
        <value>256</value>
    </property>
    <property>
        <name>yarn.app.mapreduce.am.resource.mb</name>
        <value>256</value>
    </property>
    <property>
        <name>mapreduce.map.java.opts</name>
        <value>-Xmx204m</value>
    </property>
    <property>
        <name>mapreduce.reduce.java.opts</name>
        <value>-Xmx204m</value>
    </property>
    <property>
        <name>yarn.app.mapreduce.am.command-opts</name>
        <value>-Xmx204m</value>
    </property>
</configuration>
```

Why This Command: Editing `mapred-site.xml` limits memory usage for MapReduce tasks, optimizing for a laptop's resources.

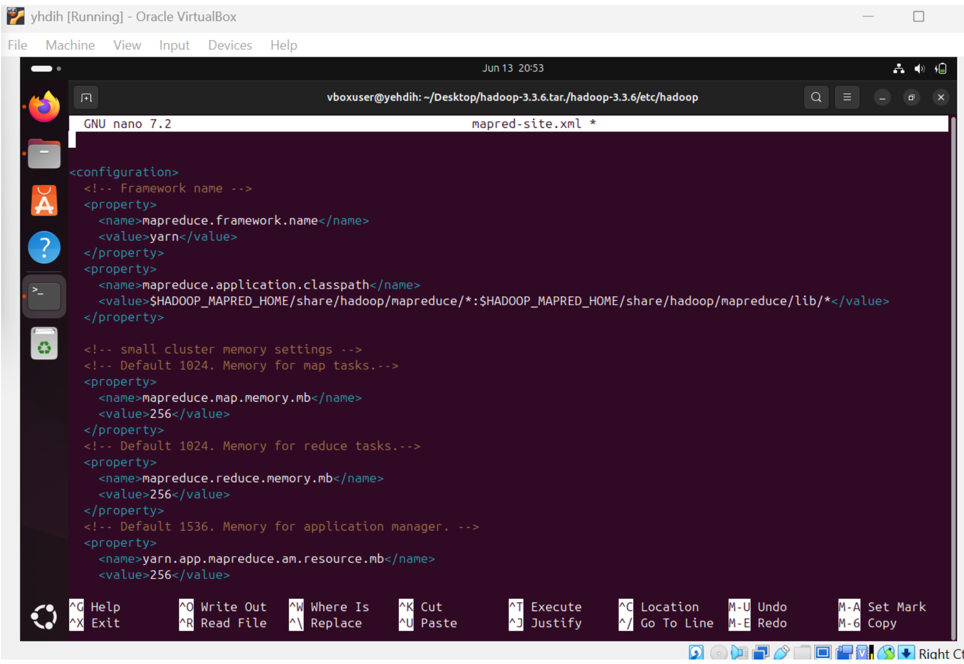

Figure 8: mapred-site.xml for laptop

6 First Steps with Hadoop

6.1 Question 5.1: Starting Hadoop

In `hadoop-3.2.1/`, I initialized and started Hadoop services:

```
bin/hdfs namenode -format
sbin/start-dfs.sh
sbin/start-yarn.sh
```

I verified running processes:

```
jps
```

Why These Commands: The `hdfs namenode -format` command initializes HDFS, creating a clean file system. The `start-dfs.sh` and `start-yarn.sh` commands launch HDFS and YARN daemons, enabling storage and processing. The `jps` command lists Java processes, confirming Hadoop services are active.

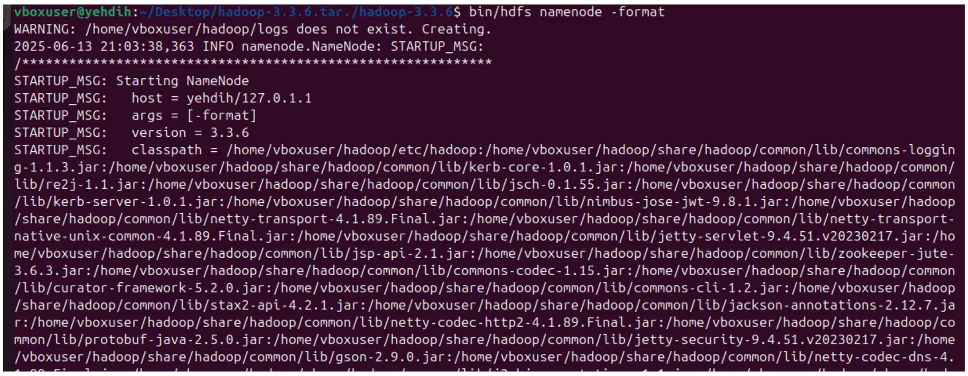

Figure 9: jps command output

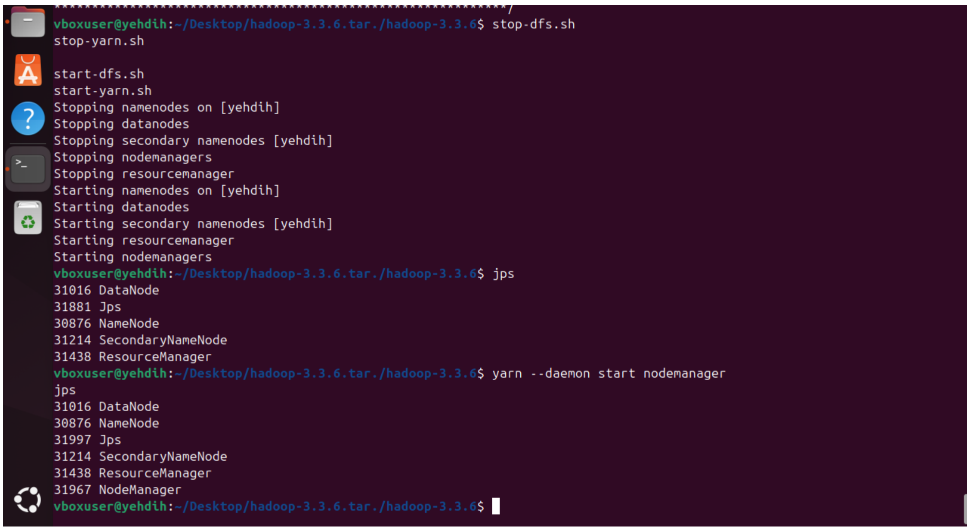

Figure 10: jps command output

6.2 Question 5.2: Web Interfaces

I accessed Hadoop's web interfaces to monitor the cluster:

- NameNode: `http://localhost:9870/`
- JobTracker: `http://localhost:8088/`

Why This Command: Accessing these URLs via a browser displays the NameNode and JobTracker interfaces, allowing monitoring of HDFS status and job progress, critical for cluster management.

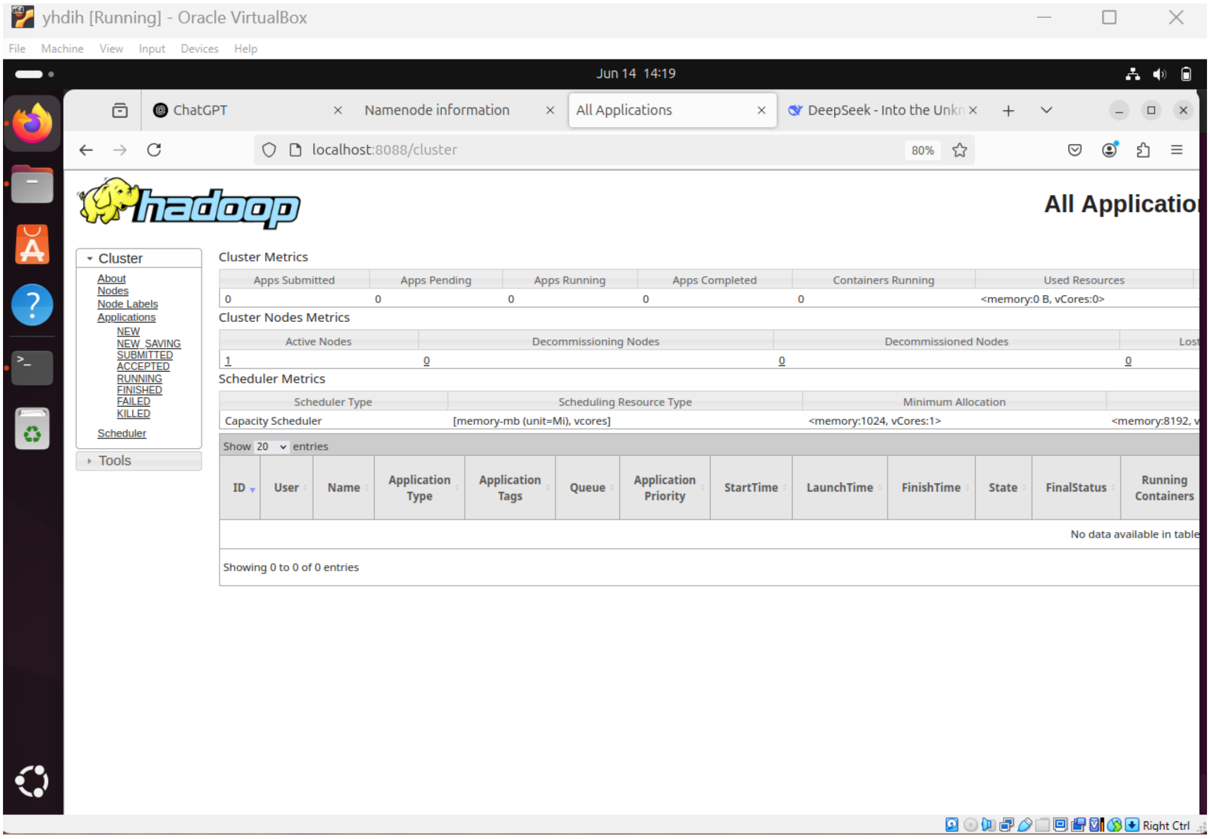

Figure 11: NameNode web interface

6.3 Question 5.3: Stopping Hadoop

I stopped Hadoop services:

```
sbin/stop-yarn.sh
sbin/stop-dfs.sh
```

Why These Commands: The `stop-yarn.sh` and `stop-dfs.sh` commands safely shut down YARN and HDFS daemons, preventing data corruption and freeing resources.

6.4 Question 5.4: Pi Estimator

I ran the Pi estimator job:

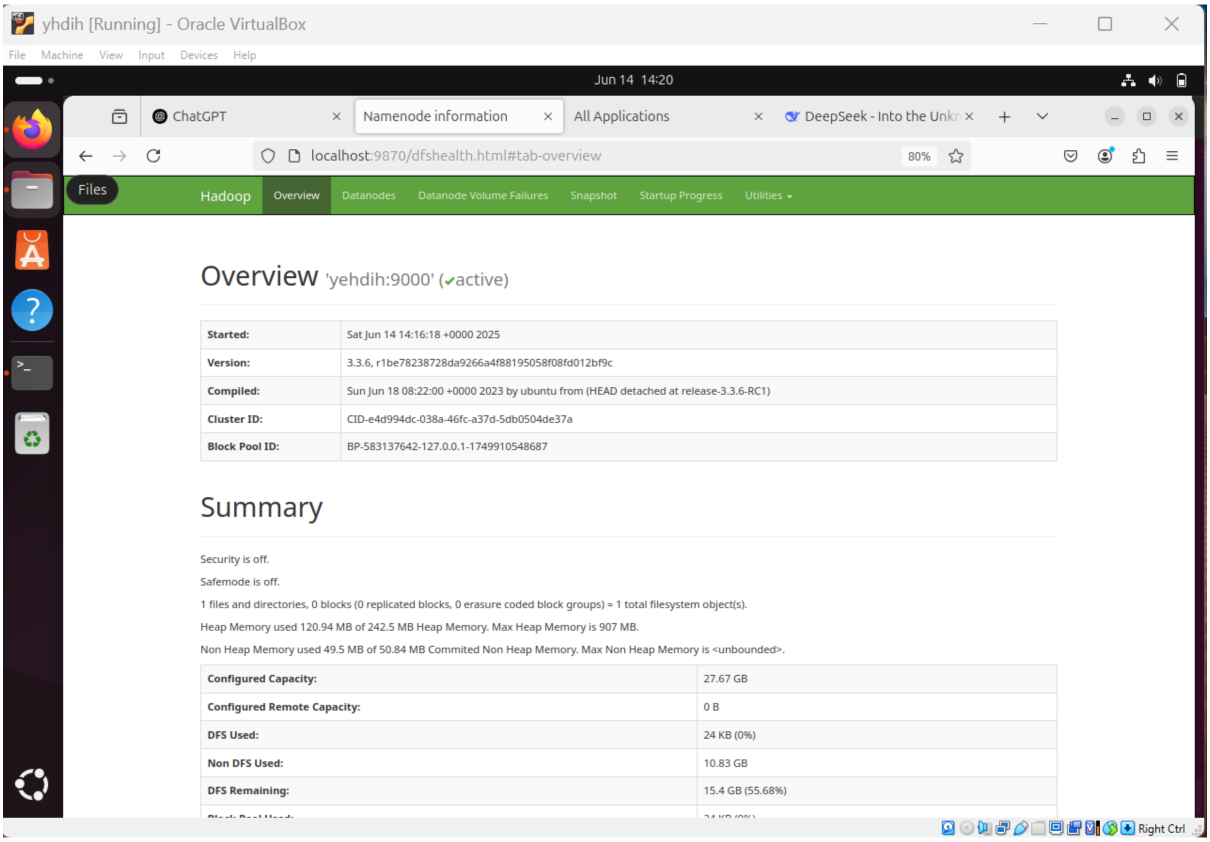

Figure 12: NameNode web interface

```
bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples
    -3.2.1.jar pi 8 14
```

Why This Command: The `hadoop jar` command executes the Pi estimator, a MapReduce job that tests Hadoop's processing capability by estimating Pi using 8 map tasks and 14 samples per task.

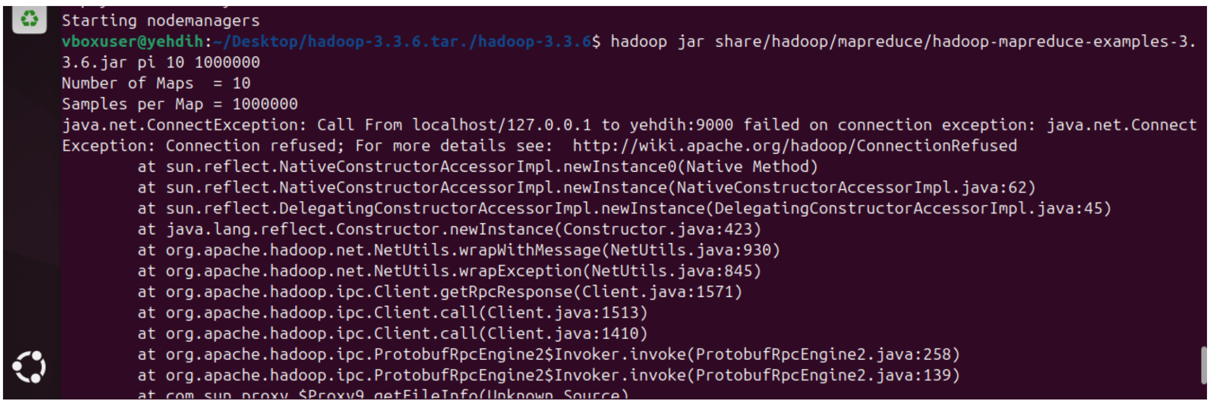

Figure 13: Pi estimator output

6.5 Question 5.5: Emergency Reset

I created and executed a reset script:

```bash
#!/bin/bash
hadoop-3.2.1/sbin/stop-yarn.sh
hadoop-3.2.1/sbin/stop-dfs.sh
rm -rf hadoop-3.2.1/logs/*
rm -rf /tmp/hadoop-3.2.1/*
hadoop-3.2.1/bin/hdfs namenode -format
```

I made it executable:

```bash
chmod +x reset.sh
```

Why These Commands: The `reset.sh` script stops Hadoop, clears logs and HDFS data, and reformats HDFS, resetting the cluster to a clean state. The `chmod +x` command makes the script executable, enabling quick resets.

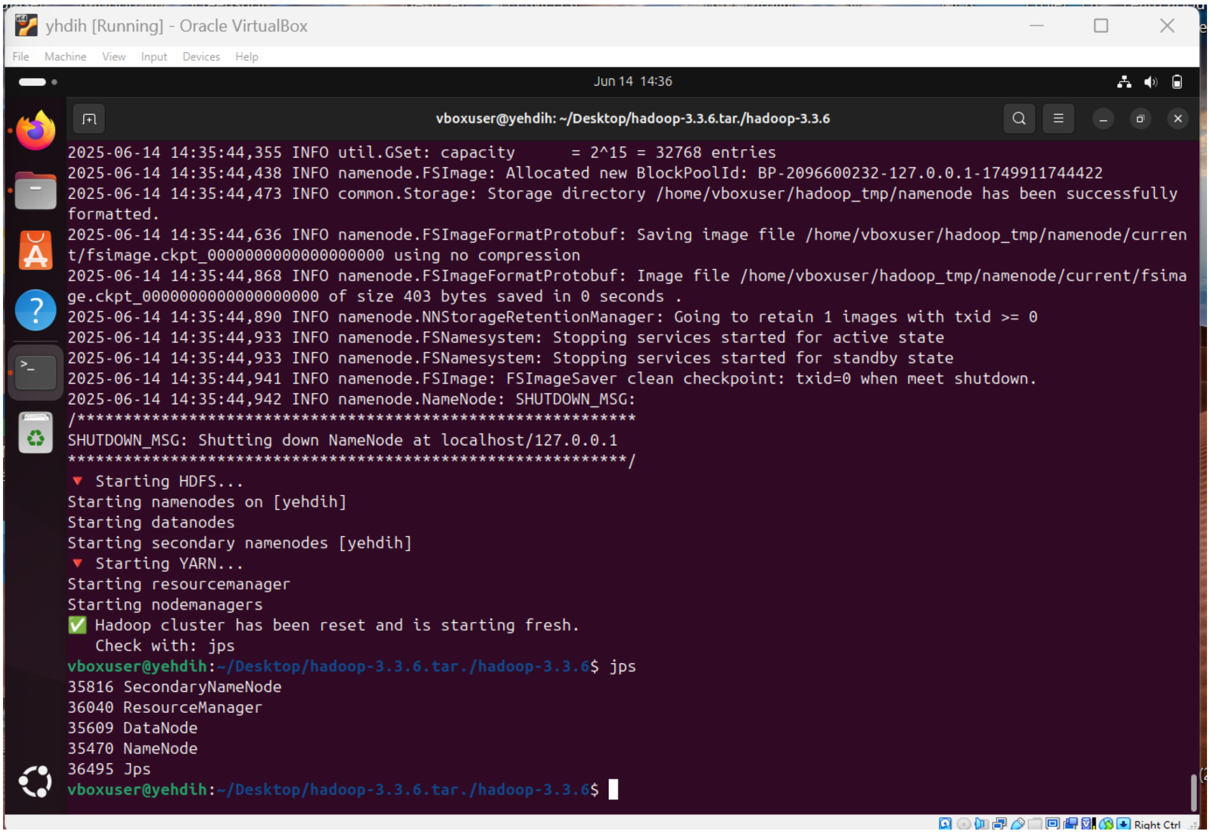

Figure 14: reset.sh execution

7 Conclusion

7.1 Architecture Diagrams

To better understand the Hadoop ecosystem, the following diagrams illustrate key architectural concepts:

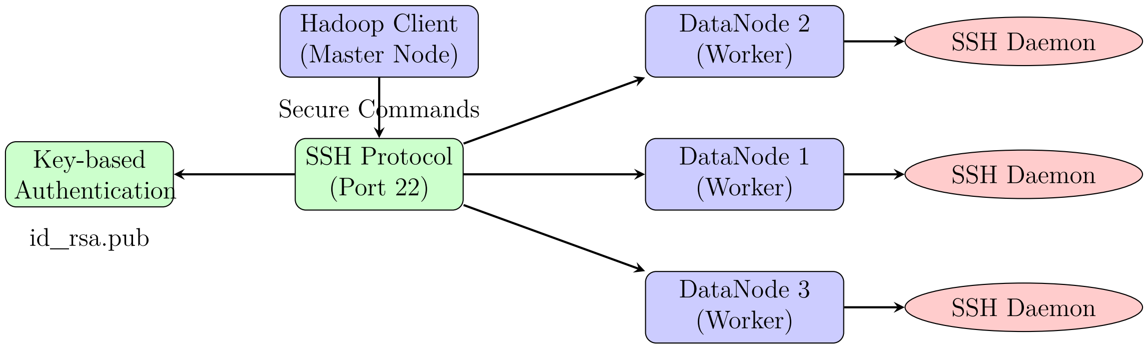

Figure 15: SSH Communication Architecture in Hadoop Cluster

7.1.1 SSH Communication Architecture

The SSH architecture enables Hadoop's distributed operations through secure, automated communication. The master node uses SSH to execute commands on worker nodes without manual intervention, essential for starting/stopping services and managing the cluster.

7.1.2 Hadoop Ecosystem Architecture

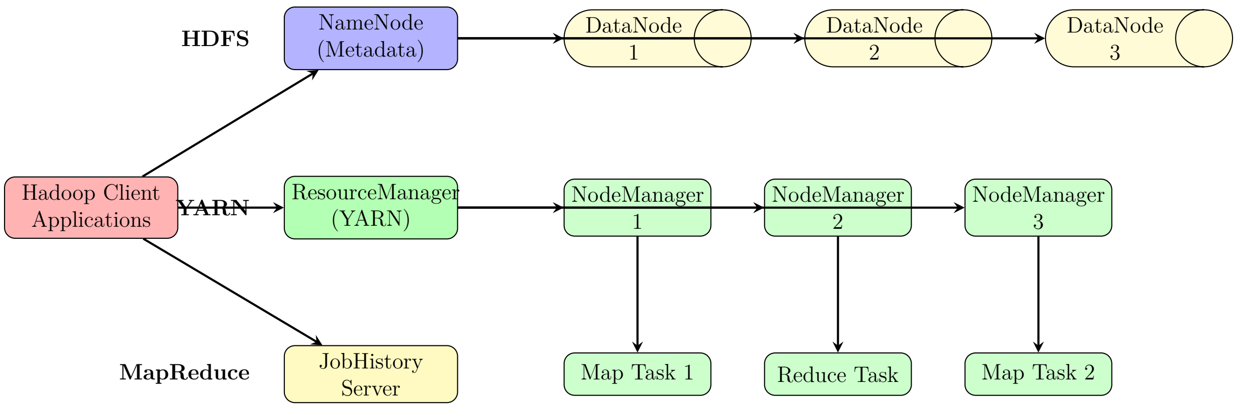

Figure 16: Hadoop Ecosystem Architecture

This architecture demonstrates how HDFS provides distributed storage, YARN manages resources, and MapReduce handles distributed processing. The client interacts with all layers to submit jobs and access data.

article tikz

7.1.3 MapReduce Data Processing Flow

7.2 Summary

This report documented the Hadoop TP1 process, detailing the setup and testing of a single-node cluster. Key components were explained with their purpose, rationale, and real-world examples. Architecture diagrams illustrated SSH communication flow, Hadoop ecosystem components, and MapReduce data processing patterns, providing visual understanding of the distributed computing framework.

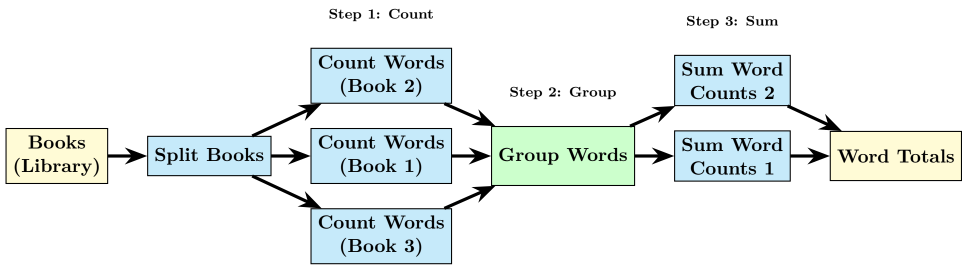

Figure 17: MapReduce Data Flow

The exercise provided practical experience with Hadoop's distributed computing framework, demonstrating how HDFS manages storage across nodes, how YARN orchestrates resource allocation, and how MapReduce processes large datasets in parallel. The configuration optimizations for laptop environments showed how Hadoop can be adapted to different hardware constraints while maintaining functionality.

The successful execution of the Pi estimator MapReduce job validated the complete Hadoop installation and configuration, proving that all components—Java runtime, SSH communication, HDFS storage, YARN resource management, and MapReduce processing—worked together seamlessly. Screenshots with command explanations documented each step for reproducibility.

Future work could extend this single-node setup to a multi-node cluster, implement custom MapReduce applications, integrate with other big data tools like Apache Spark or Apache Hive, and explore advanced HDFS features like federation and high availability. The foundation established in this TP1 provides the groundwork for more complex distributed computing scenarios.