

Lab01 Assignment's Report

Le Anh Thu¹, Nguyen Nhat Truyen¹, Nguyen Minh Dat¹, and Huynh Duc Thien¹

¹Faculty of Information Technology, University of Science

1 Team's Result

After dedicating several weeks to this lab assignment, our team's work reveal the following percentages in this Table 1.

Table 1. The contribution of our team in each task

	Thu Le	Truyen Nguyen	Dat Nguyen	Thien Huynh	Total
Setting up SNC	25%	25%	25%	25%	100%
Introduction to MapReduce	25%	-	75%	-	100%
Running WordCount	-	50%	-	50%	100%
Word Length Count	-	75%	-	25%	100%
Fully Distributed Mode	-	-	50%	50%	100%
Latex Report	100%	-	-	-	100%
Contribution	150%	150%	150%	150%	600%

2 Answer in Each Section

2.1 Setting up SNC - Single Node Cluster

Member 1. Le Anh Thu (Linux Native)

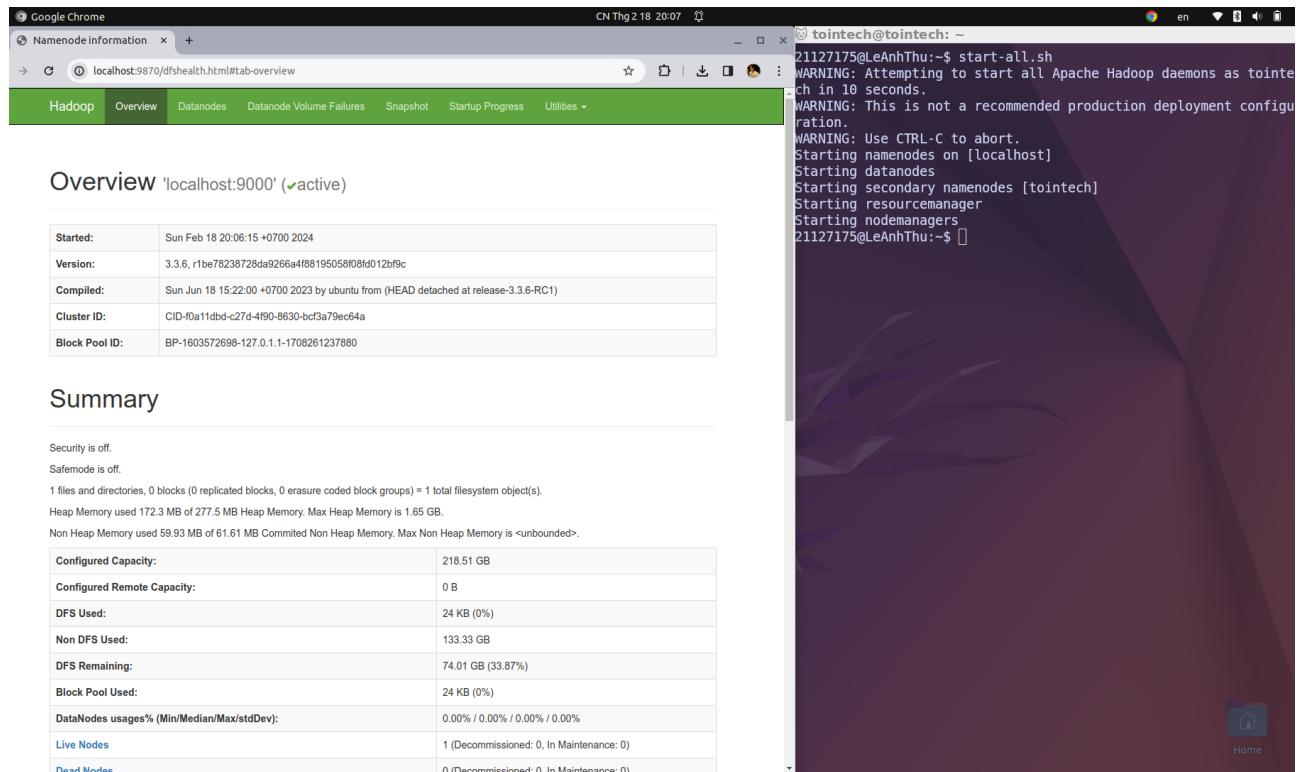


Fig. 1. Installing Hadoop natively on a Linux system

Member 2. Nguyen Nhat Truyen (VMWare in Windows)

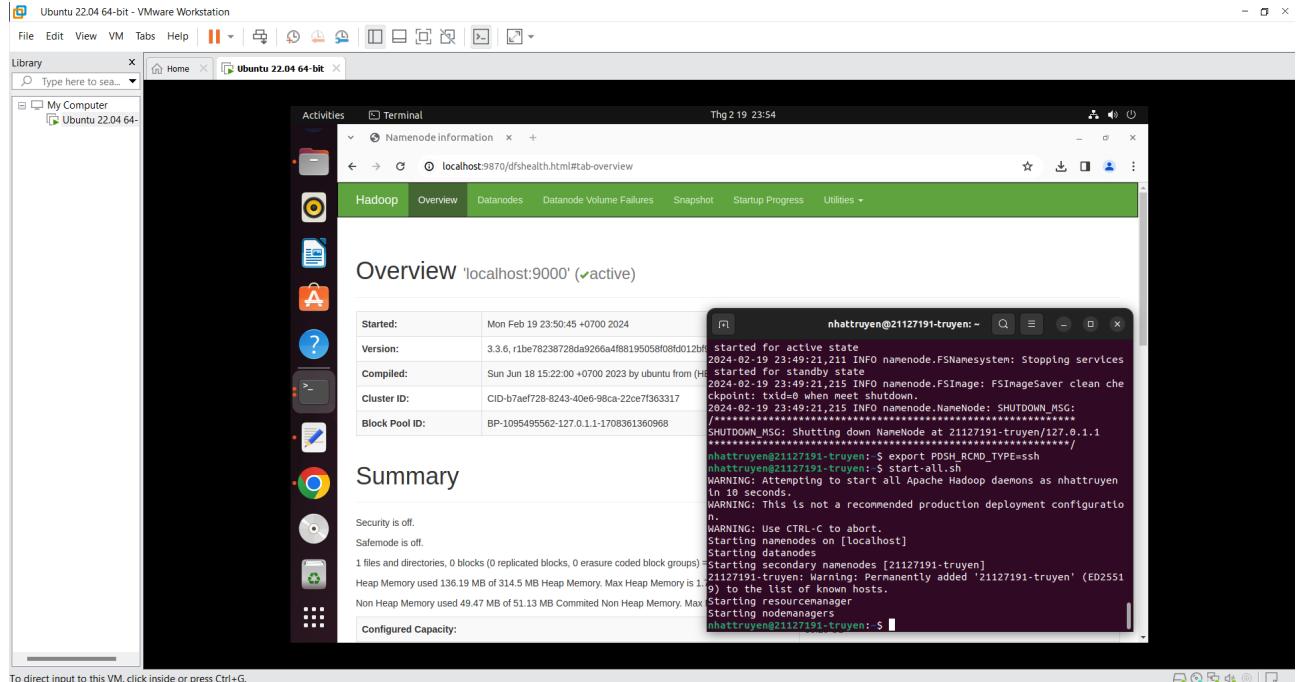


Fig. 2. Installing Hadoop within a VMWare virtual environment

Member 3. Nguyen Minh Dat (MacOS Native)

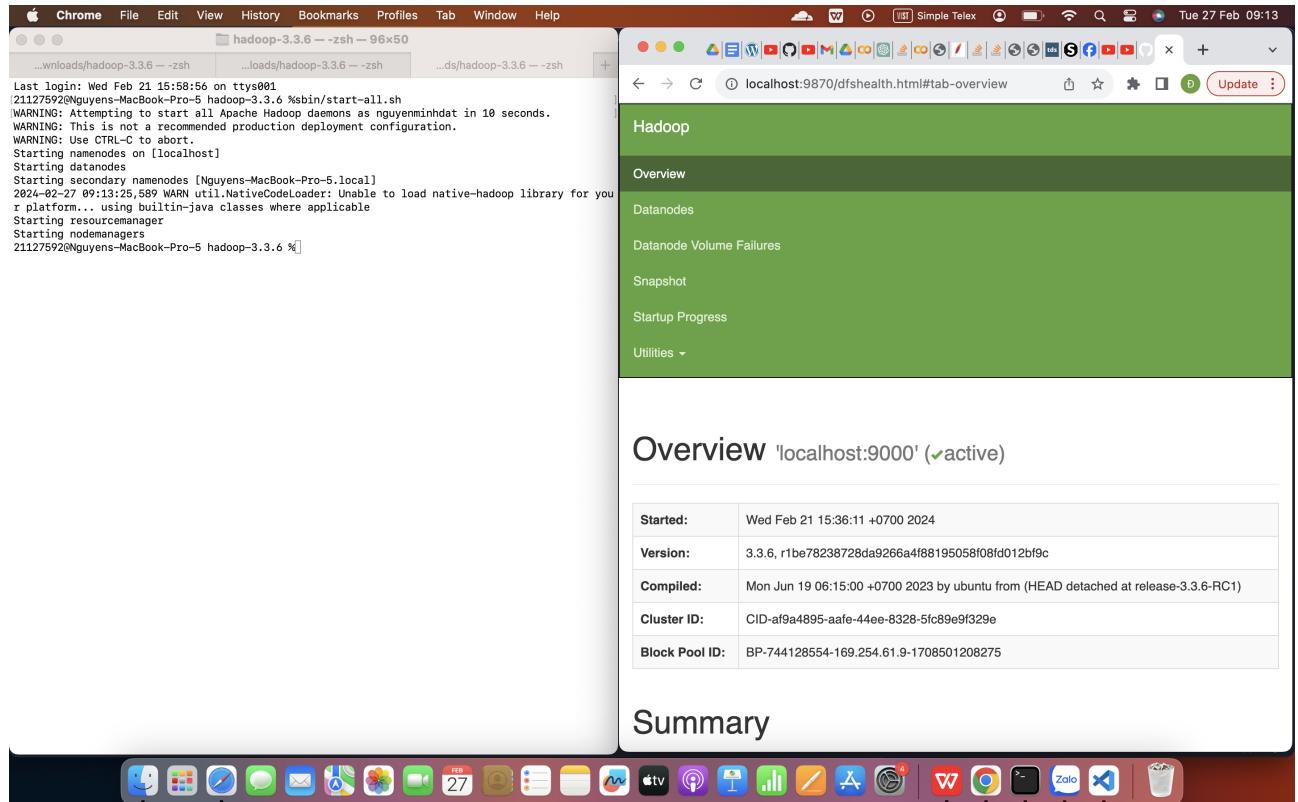


Fig. 3. Installing Hadoop directly on a macOS system

Member 4. Huynh Duc Thien (Linux in MacOS)

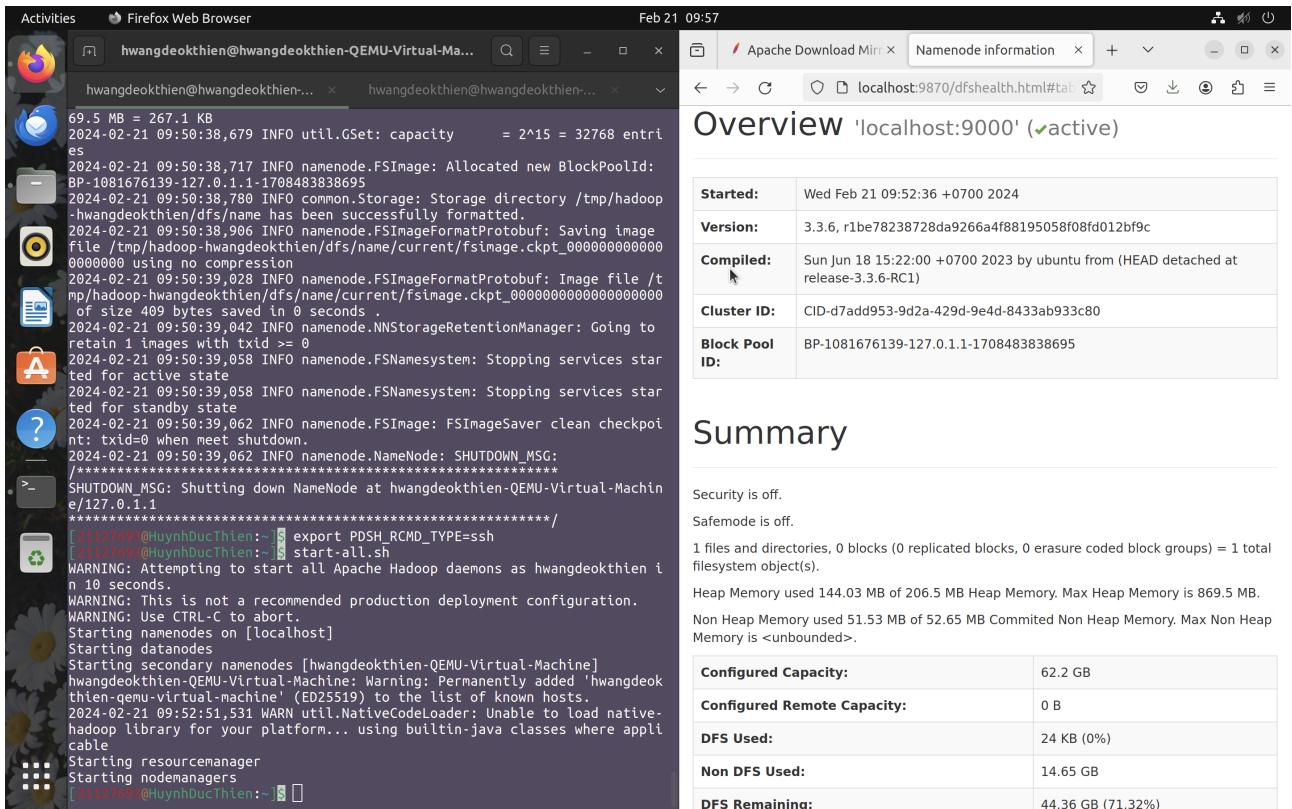


Fig. 4. Installing Hadoop on a Linux environment within a MacOS system

During this assignment, our team members collaborated seamlessly, achieving a shared understanding and successfully installing a Hadoop cluster on their respective devices.

2.2 Introduction to MapReduce

After thoroughly studying the original MapReduce paper authored by Dean and Ghemawat [3], we have addressed the associated questions.

- How do the input keys-values, the intermediate keys-values, and the output keys-values relate?
 - An input key-value pair is passed to the Map function to produce a set of intermediate key-value pairs. The MapReduce library groups all intermediate values associated with the same intermediate key and passes them to the Reduce function.
 - The Reduce function accepts an intermediate key and a set of values for that key. It merges those values to form a possibly smaller set of values. Typically, just zero or one output value is produced per Reduction call.
 - How does MapReduce deal with node failures?
- Work Failure
- The master pings every worker periodically. If no response is received from a worker in a certain amount of time, the master marks the worker as failed.
 - For in-progress tasks: any map task or reduce task in progress on a failed worker are reset to idle state and become eligible for rescheduling.
 - For completed tasks: Completed map tasks are re-executed because their output is stored on the local disk of the failed machine and is therefore inaccessible. Completed reduce tasks do not need to be re-executed because their output is stored in a global file system.
- MapReduce is **resilient** to large-scale worker failures. During one operation, if multiple machines become unreachable, the master simply re-executes the work done by the unreachable worker machines and continues to make progress, eventually completing the operation.

Master Failure

- There is only a single master, so failure is unlikely. → Abort the computation if the master fails.
3. What is the meaning and implication of locality? What does it use?
- Why locality? Because network bandwidth is scarce.
 - Solution: GFS divides each file into 64MB blocks and stores several copies of each block on different machines. The master manages the location information of the input files and schedules a map task on a machine that contains a replica of the corresponding input data. → Most input data is read locally and consumes no network bandwidth.
4. Which problem is addressed by introducing a combiner function to the MapReduce model?
- Problem: Repetition in the intermediate keys produced by each map task.
 - Example: word counting. Each map task produces thousands of records of the form <the, 1>. All of these counts will be sent over the network to a single reduce task and then added together by the Reduce function to produce one number. The Combiner function does the partial merging of this data before it is sent over the network.

2.3 Running a warm-up problem: Word Count

This section provide a comprehensive, step-by-step for running the Word Count problem [1].

Step 1. We need to make sure Hadoop and Java have already been installed, by running these commands.

```
hadoop version
javac -version
```

```
[21127693@HuynhDucThien:~]$ hadoop version
Hadoop 3.3.6
Source code repository https://github.com/apache/hadoop.git -r 1be78238728da9266
a4f88195058f08fd012bf9c
Compiled by ubuntu on 2023-06-18T08:22Z
Compiled on platform linux-x86_64
Compiled with protoc 3.7.1
From source with checksum 5652179ad55f76cb287d9c633bb53bbd
This command was run using /home/hwangdeokthien/hadoop-3.3.6/share/hadoop/common
/hadoop-common-3.3.6.jar

[21127693@HuynhDucThien:~]$ javac -version
javac 1.8.0_392
```

Fig. 5. Step 1 in Word Count problem

Step 2. Create a directory to store input, compile, and related files. For convenience, we will create it at /Desktop.

```
mkdir Lab1
mkdir Lab1/Compiled
mkdir Lab1/Input
```

By using command `mkdir <path>` we will create the directory at a given path and name. We will check the result using `ls <path>` command, which will show the folders included.

```
[21127693@HuynhDucThien:~/Desktop]$ mkdir Lab1
[21127693@HuynhDucThien:~/Desktop]$ mkdir Lab1/Compiled
[21127693@HuynhDucThien:~/Desktop]$ mkdir Lab1/Input
[21127693@HuynhDucThien:~/Desktop]$ ls Lab1
Compiled  Input
```

Fig. 6. Step 2 in Word Count problem

Step 3. Create a Java file to run the WordCount problem, using the source code given in the Apache Hadoop documentation [2].

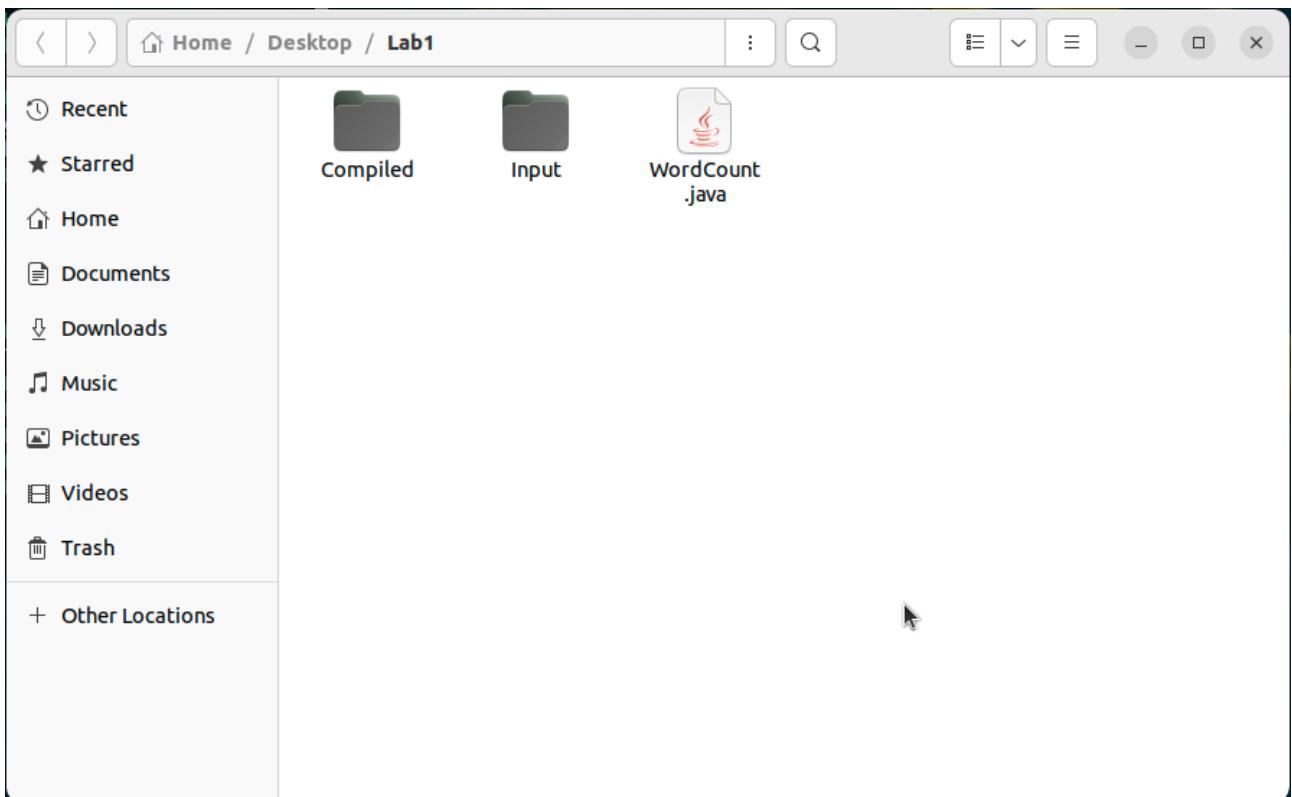


Fig. 7. Step 3 in Word Count problem

The file will be stored in the `Lab1` directory.

Step 4. We continue to create an input file for this problem, the input file is stored in the `Input` directory, and each line of the file will include one word.

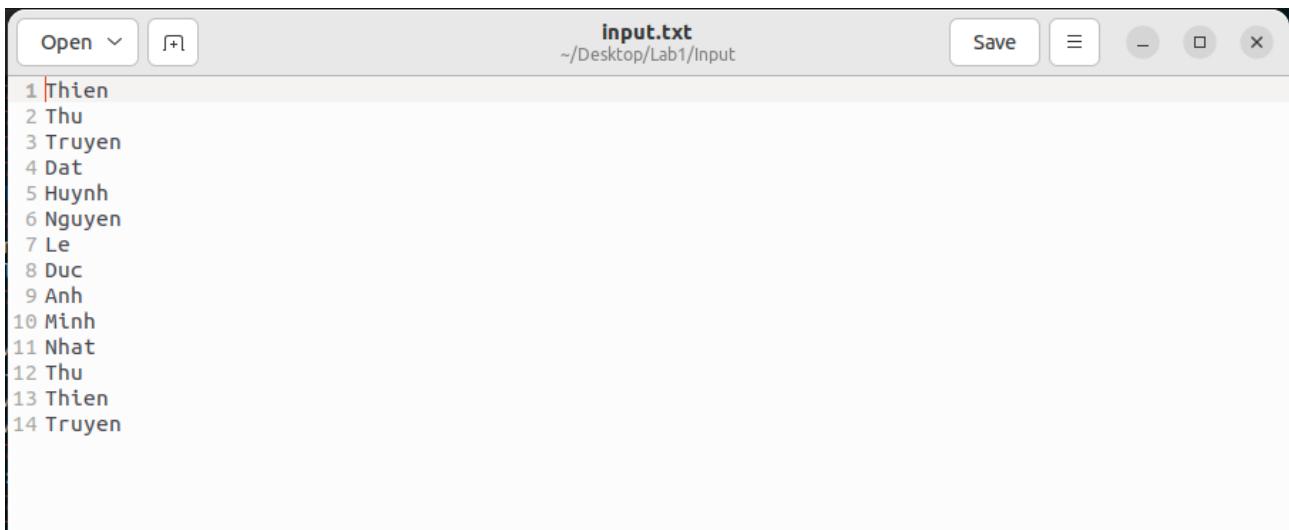


Fig. 8. Step 4 in Word Count problem

Step 5. Start Hadoop and create corresponding directories on HDFS.

```
hadoop fs -mkdir /WordCountProblem
hadoop fs -mkdir /WordCountProblem/Input
hadoop fs -put ~/Desktop/Lab1/Input/input.txt /WordCountProblem/Input
```

Next, we create a directory named `WordCountProblem/Input` in HDFS and copy the `input.txt` file in the local file system to HDFS.

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-r--r--r--	hwangdeokthien	supergroup	72 B	Feb 27 00:23	1	128 MB	input.txt

Fig. 9. Step 5 in Word Count problem

Step 6. Exporting hadoop classpath into bash, ensures that the Java compiler and packaging tool can access Hadoop's classes and libraries, which are essential for building and running Hadoop jobs.

```
export HADOOP_CLASSPATH=$(hadoop classpath)
```

Step 7. Compile the local `WordCount.java` file and store the output files in the `Compiled` directory, then create a jar file for future runs in Hadoop.

```
javac -classpath $HADOOP_CLASSPATH -d Compiled WordCount.java
```

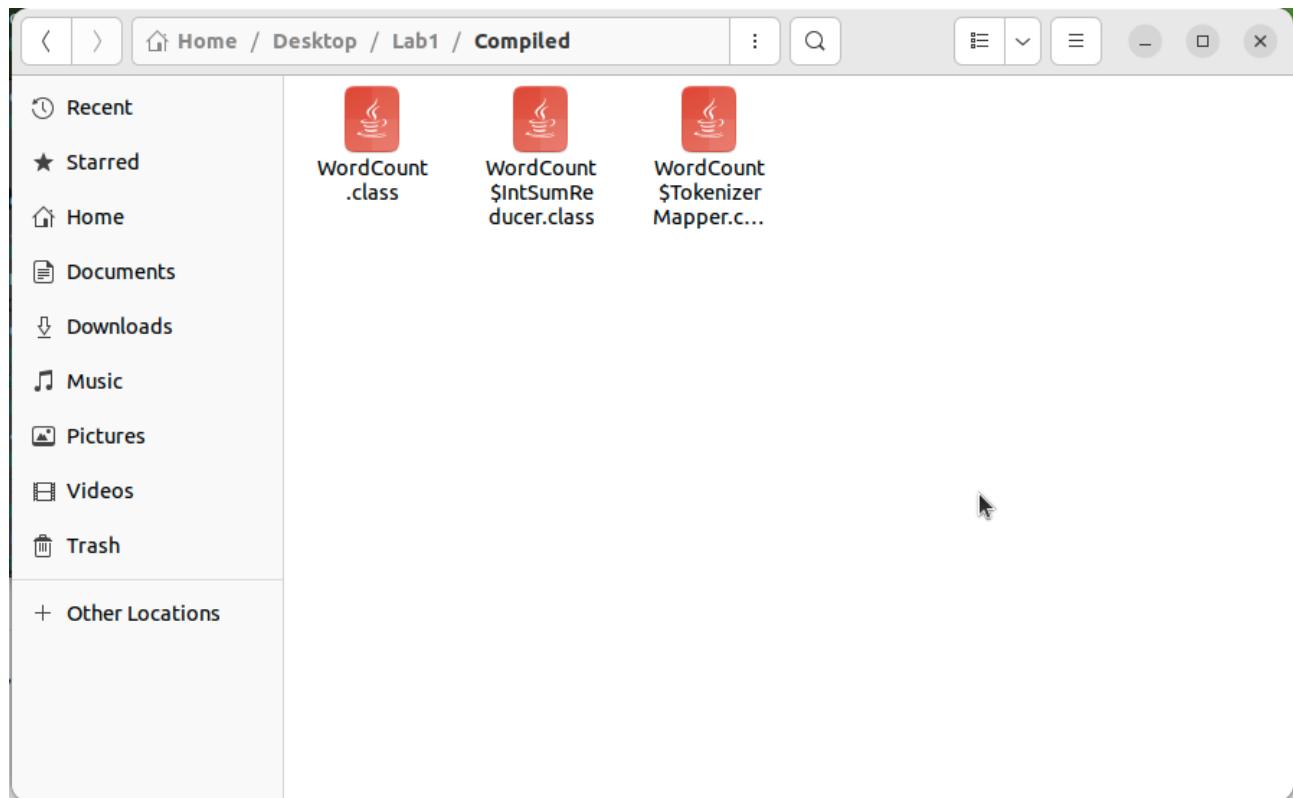


Fig. 10. Step 7.1 in Word Count problem

```
jar -cvf WordCount.jar -C Compiled .
```

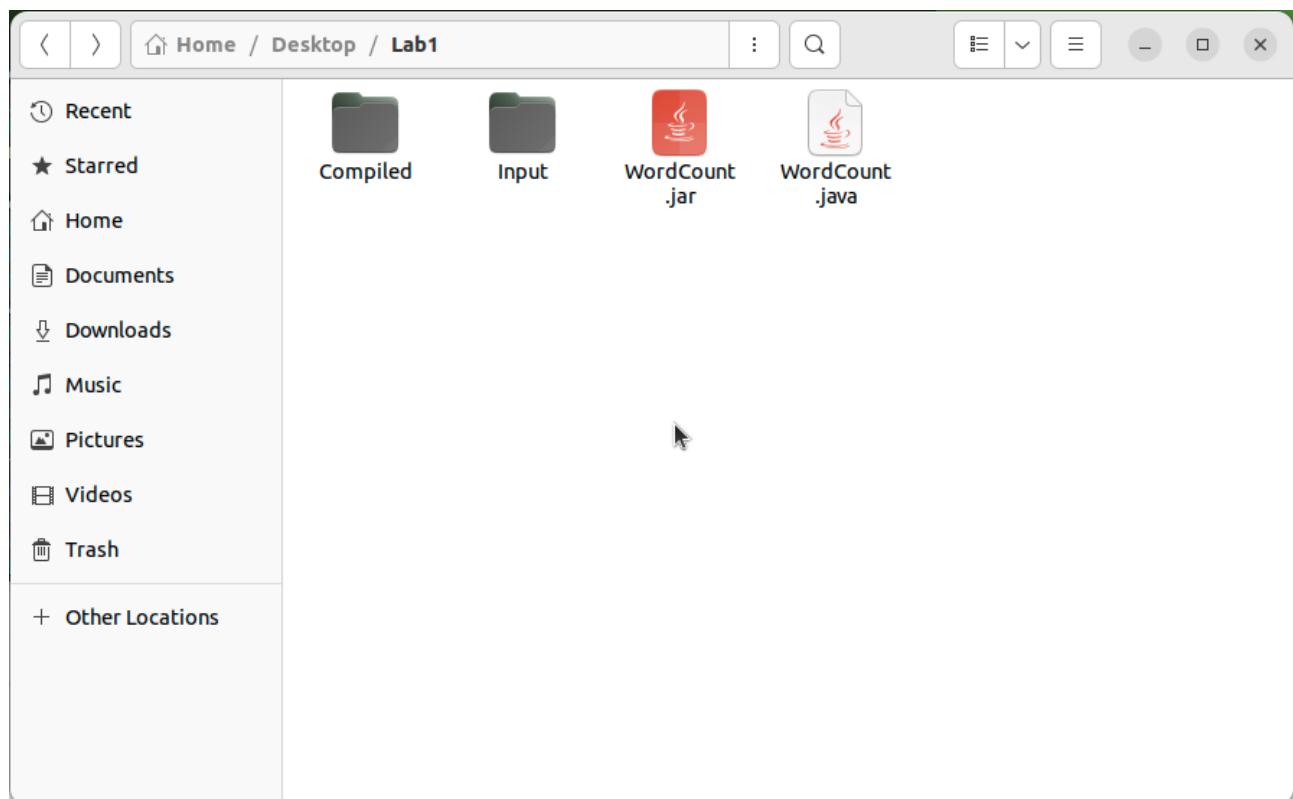


Fig. 11. Step 7.2 in Word Count problem

Step 8. Run the jar file on Hadoop and output the result stored in /WordCountProblem/Output.

```
hadoop jar WordCount.jar WordCount /WordCountProblem/Input /WordCountProblem/Output
```

The screenshot shows the Hadoop Web UI interface. At the top, there's a navigation bar with links for Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities. Below the navigation bar, the main title is "Browse Directory" with the path "/WordCountProblem/Output". There are several action buttons (Copy, Paste, Delete, etc.) and a search bar labeled "Search: []". The table below lists the directory contents:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
<input type="checkbox"/>	-rw-r--r--	hwangdeokthien	supergroup	0 B	Feb 27 01:15	1	128 MB	_SUCCESS
<input type="checkbox"/>	-rW-r--r--	hwangdeokthien	supergroup	77 B	Feb 27 01:15	1	128 MB	part-r-00000

At the bottom left, it says "Showing 1 to 2 of 2 entries". On the right, there are "Previous" and "Next" buttons.

Fig. 12. Step 8.1 in Word Count problem

```
hadoop dfs -cat /WordCountProblem/Output/*
```

```
[21127693@HuynhDucThien:~/Desktop/Lab1]$ hadoop dfs -cat /WordCountProblem/Output/*
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

2024-02-27 01:15:35,386 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Anh      1
Dat      1
Duc      1
Huynh   1
Le       1
Minh    1
Nguyen   1
Nhat    1
Thien    2
Thu     2
Truyen   2
```

Fig. 13. Result of the Word Count problem

Explanations for the algorithm [6].

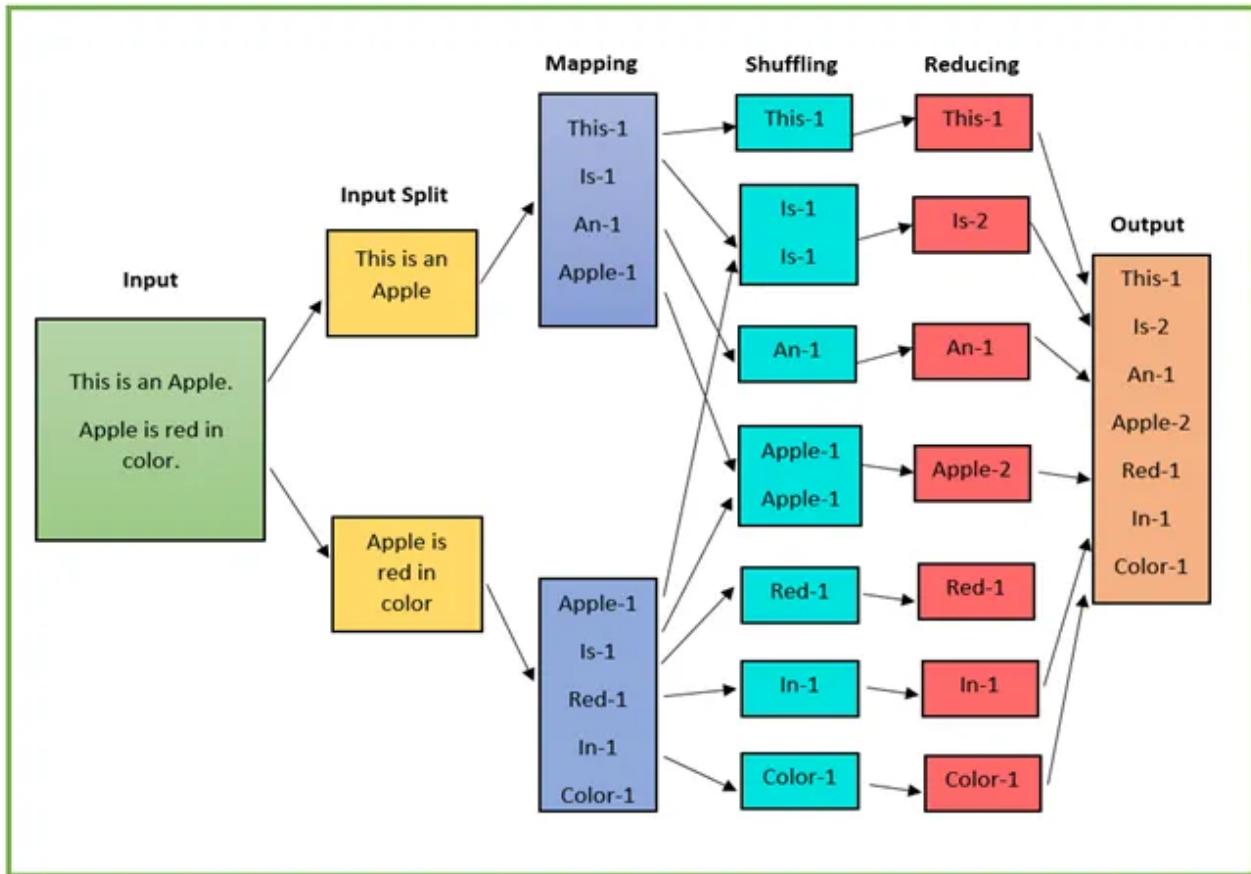


Fig. 14. Overview of MapReduce word count process

Splitting. The input text is split by a comma, space, a new line or a semicolon.

Mapping. Mapping each word with a number represents the frequency.

Shuffle/Intermediate splitting. The process is usually parallel on cluster keys. The output of the map gets into the Reducer phase and all the similar keys of data are aligned in a cluster.

Reducing. The word and its frequency have now been combined.

Output. Show the words and their corresponding frequency.

3 Word Length Count

Now, we will reanalyze the idea of the WordCount problem.

```
public static 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);
        }
    }
}
```

TokenizerMapper: is a transformation function that takes an input pair < Key, Value > and needs to return one or more new pairs of < Key, Value >.

```

public static class IntSumReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

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

```

IntSumReducer: is an aggregation function that takes an input pair < Key, Value[] > where the input values list contains all the values with the same key, and needs to return one result pair < Key, Value >.

→ In this program, the **TokenizerMapper** function will split the input text into words, and for each successfully split word, it will return a pair consisting of the word and a count of 1. The **IntSumReducer** function takes as input a word along with a list of all counts of that word, and it will sum up the counts to obtain the total number of occurrences of that word.

Now, let's return to the WordLengthCount problem. Based on the idea of the WordCount problem, but this time we will map the words accordingly:

- Tiny: Words with 1 letter.
- Small: Words with 2 to 4 letters.
- Medium: Words with 5 to 9 letters.
- Big: Words with more than 10 letters.

```

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

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

    public void map(Object key, Text value, Context context
    ) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            String word = itr.nextToken();
            int wordLength = word.length();
            if (wordLength == 1)
                wordLengthCategory.set("Tiny");
            else if (wordLength >= 2 && wordLength <= 4)
                wordLengthCategory.set("Small");
            else if (wordLength >= 5 && wordLength <= 9)
                wordLengthCategory.set("Medium");
            else
                wordLengthCategory.set("Big");
            context.write(wordLengthCategory, one);
        }
    }
}

```

After mapping the words to their corresponding categories, we will reuse the `IntSumReducer` function from the WordCount problem to obtain the output in Figure 15.

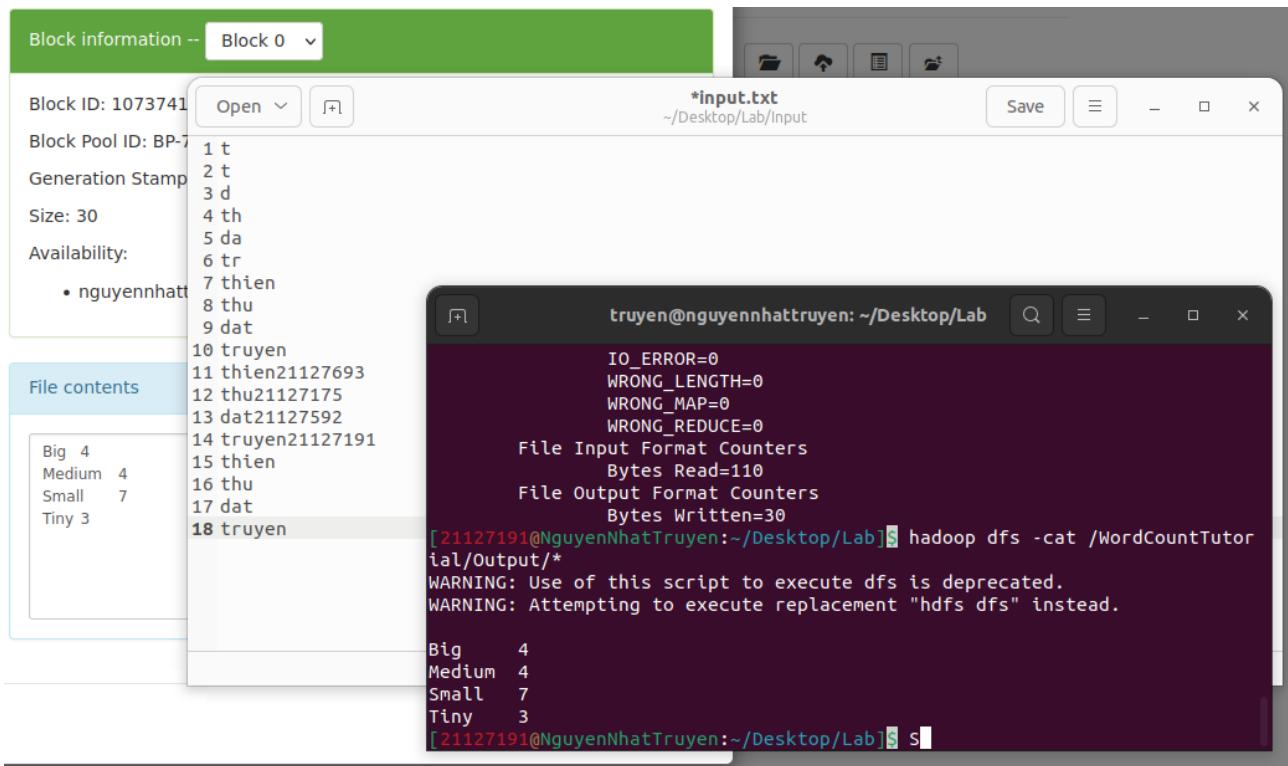


Fig. 15. Result of the Word Length Count problem

4 Setting up Fully Distributed Mode

4.1 Hadoop Cluster Setup in Non-Secure Mode

For the implementation of fully distributed mode for Hadoop, the configuration proposed consists of 1 master node and 2 slave nodes used for backup and data storage purposes.

We need to perform several necessary steps:

Step 1. Set up the network and establish ssh connections between the slave nodes and the master node. The purpose is to establish communication links between the slave nodes and the master node within the same internal network, and to be able to access the slave machines through the terminal of the master node.[4]

```
GNU nano 6.2          /etc/hosts
127.0.0.1      localhost
192.168.64.8    master
192.168.64.9    node1
192.168.64.10   node2
```

Fig. 16. Network setup between nodes

```

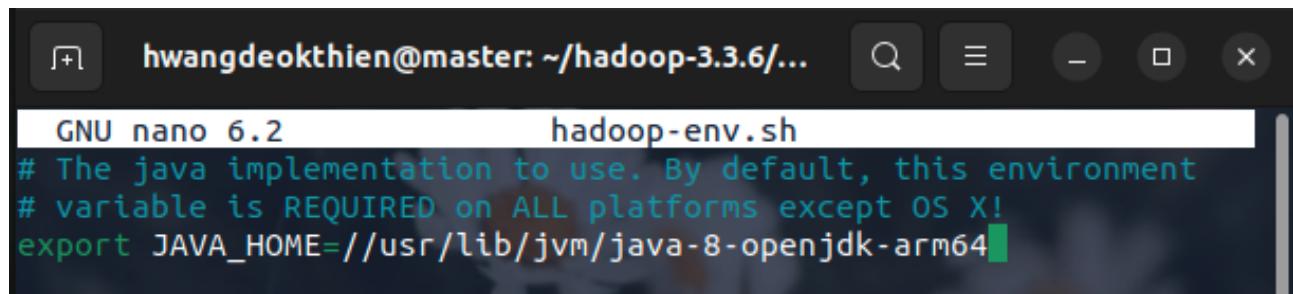
hwangdeokthien@master: ~
[21127693@HuynhDucThien:~/Desktop]$ cd ..
[21127693@HuynhDucThien:~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hwangdeokthien/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hwangdeokthien/.ssh/id_rsa
Your public key has been saved in /home/hwangdeokthien/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:OsPdU61he42Q0pNHkjOP5zkf0S6J8p0P3NfCvIuo3AA hwangdeokthien
@master
The key's randomart image is:
+---[RSA 3072]---+
| . . . |
| = . . |
| . @ . |
| E S . @ =.. |
| . + . + @ *.. |
| = o + ++0.+ |
| + o *o=+++. |
| o.o =o==. |
+---[SHA256]---+
[21127693@HuynhDucThien:~]$ cd ~/.ssh
[21127693@HuynhDucThien:~/.ssh]$ ls
id_rsa id_rsa.pub
[21127693@HuynhDucThien:~/.ssh]$ cp id_rsa.pub authorized_keys
[21127693@HuynhDucThien:~/.ssh]$ ls
authorized_keys id_rsa id_rsa.pub
[21127693@HuynhDucThien:~/.ssh]$ cd
[21127693@HuynhDucThien:~]$ chmod 700 .ssh
[21127693@HuynhDucThien:~]$ scp ~/.ssh/id_rsa.pub node1:~/.ssh/au
thorized_keys
The authenticity of host 'node1 (192.168.64.9)' can't be establis
hed.
ED25519 key fingerprint is SHA256:QC02aW/Xxs2B3diHQ8x110J+2RtNMt7
5Xw3KE1nSKPY.

```

Fig. 17. Authorized ssh connection between nodes

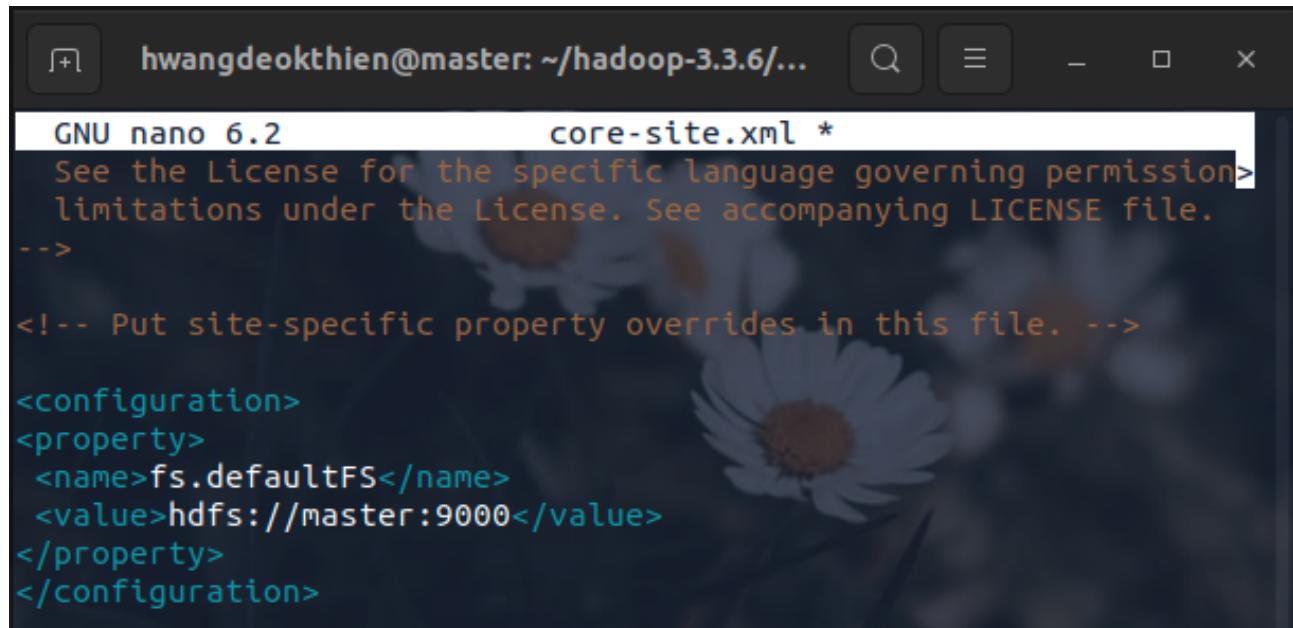
Step 2. Install Hadoop and Java for all nodes. This step will be similar to the installation process of a single-node cluster mentioned above.

Step 3. Proceed to configure the necessary `hadoop-env.sh` and `core-site.xml` files for all nodes as done in section 1. [7]



```
GNU nano 6.2          hadoop-env.sh
# The java implementation to use. By default, this environment
# variable is REQUIRED on ALL platforms except OS X!
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-arm64
```

Fig. 18. Adding Java path into hadoop-env.sh



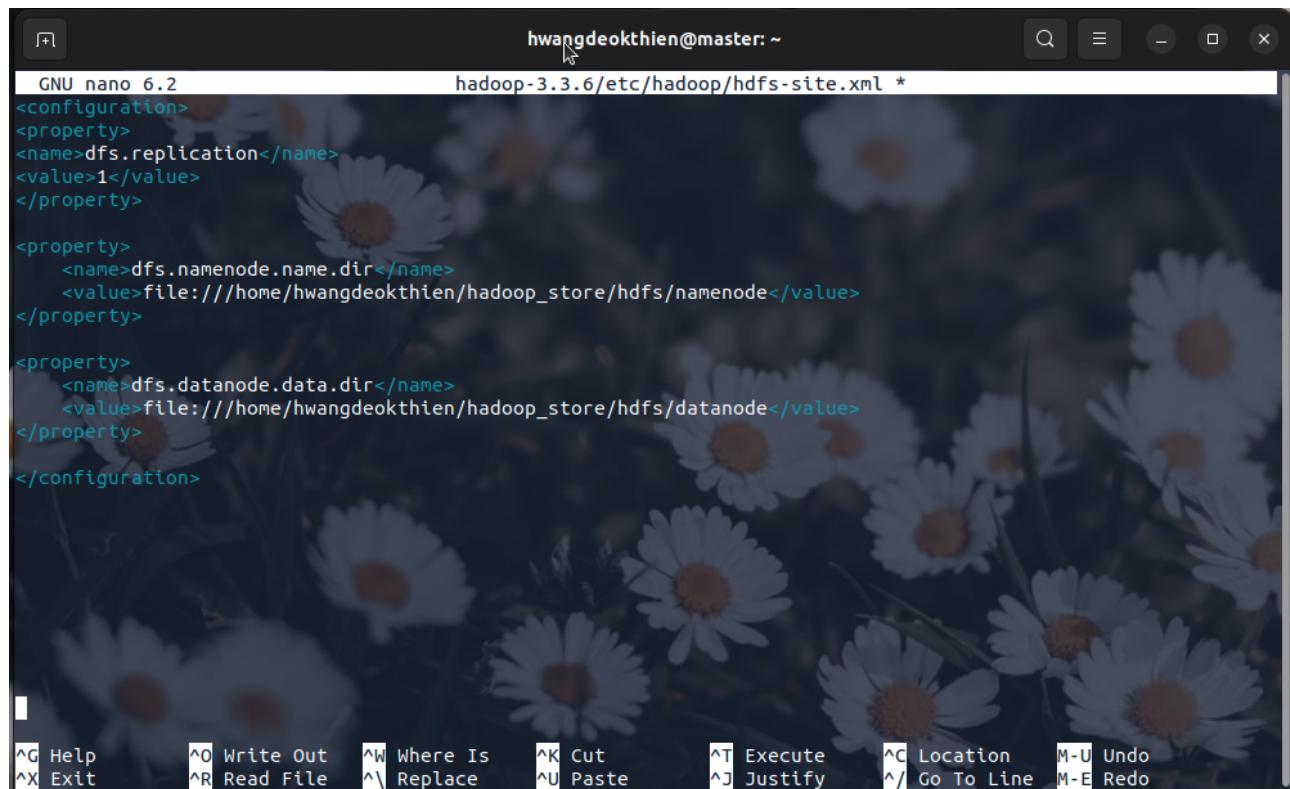
```
GNU nano 6.2          core-site.xml *
See the License for the specific language governing permission>
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

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

Fig. 19. Adjusting core-site.xml files on all nodes

Step 4. Configure specific settings in the corresponding `hdfs-site.xml` files for each node, and additionally install `mapred-site.xml` and `yarn-site.xml` files for the master node.



```

GNU nano 6.2          hadoop-3.3.6/etc/hadoop/hdfs-site.xml *
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>

<property>
<name>dfs.namenode.name.dir</name>
<value>file:///home/hwangdeokthien/hadoop_store/hdfs/namenode</value>
</property>

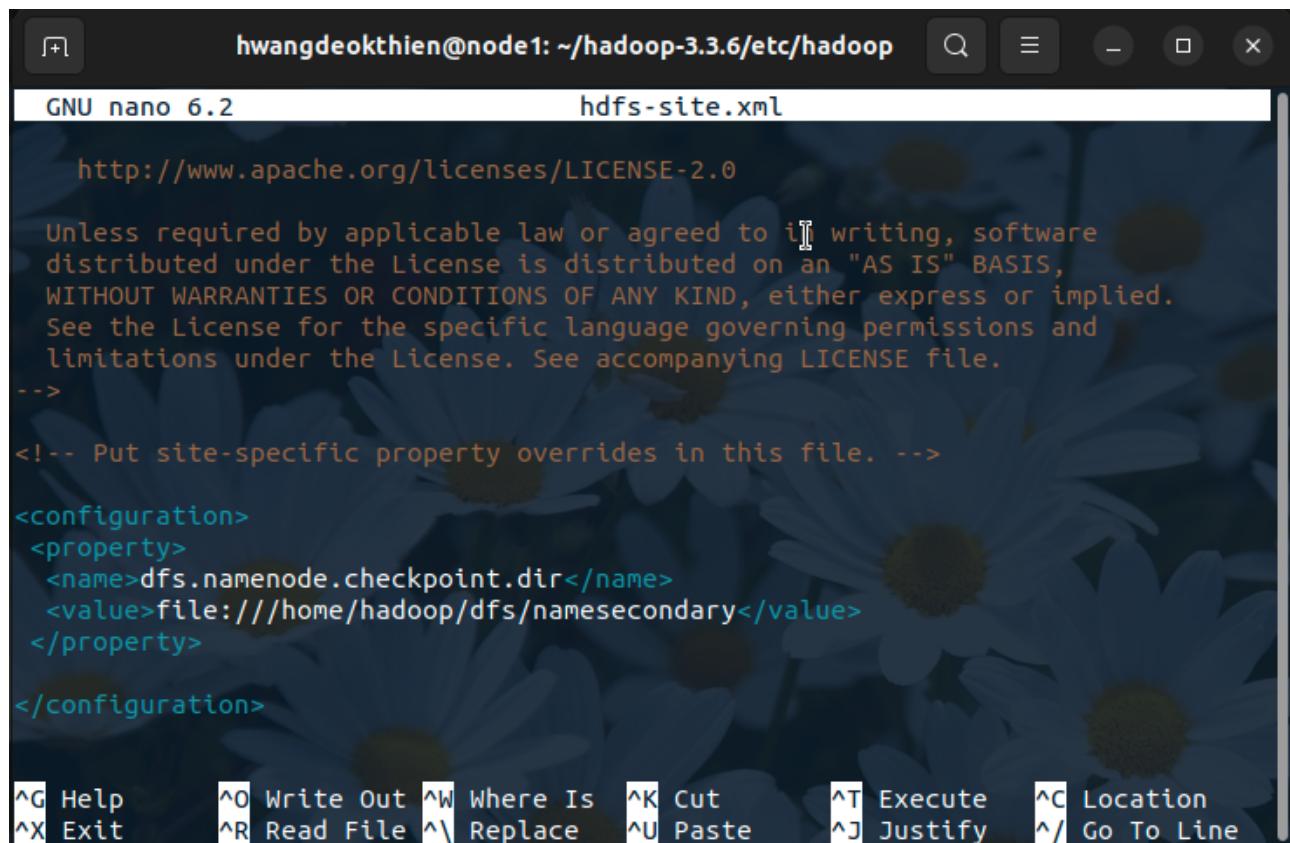
<property>
<name>dfs.datanode.data.dir</name>
<value>file:///home/hwangdeokthien/hadoop_store/hdfs/datanode</value>
</property>

</configuration>

```

File menu: New, Open, Save, Exit
 Edit menu: Undo, Redo, Cut, Copy, Paste, Select All, Find, Replace, Go To Line
 View menu: Where Is, Location
 Help menu: About, Exit

Fig. 20. hdfs-site.xml configuration for master node



```

GNU nano 6.2          hdfs-site.xml
http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

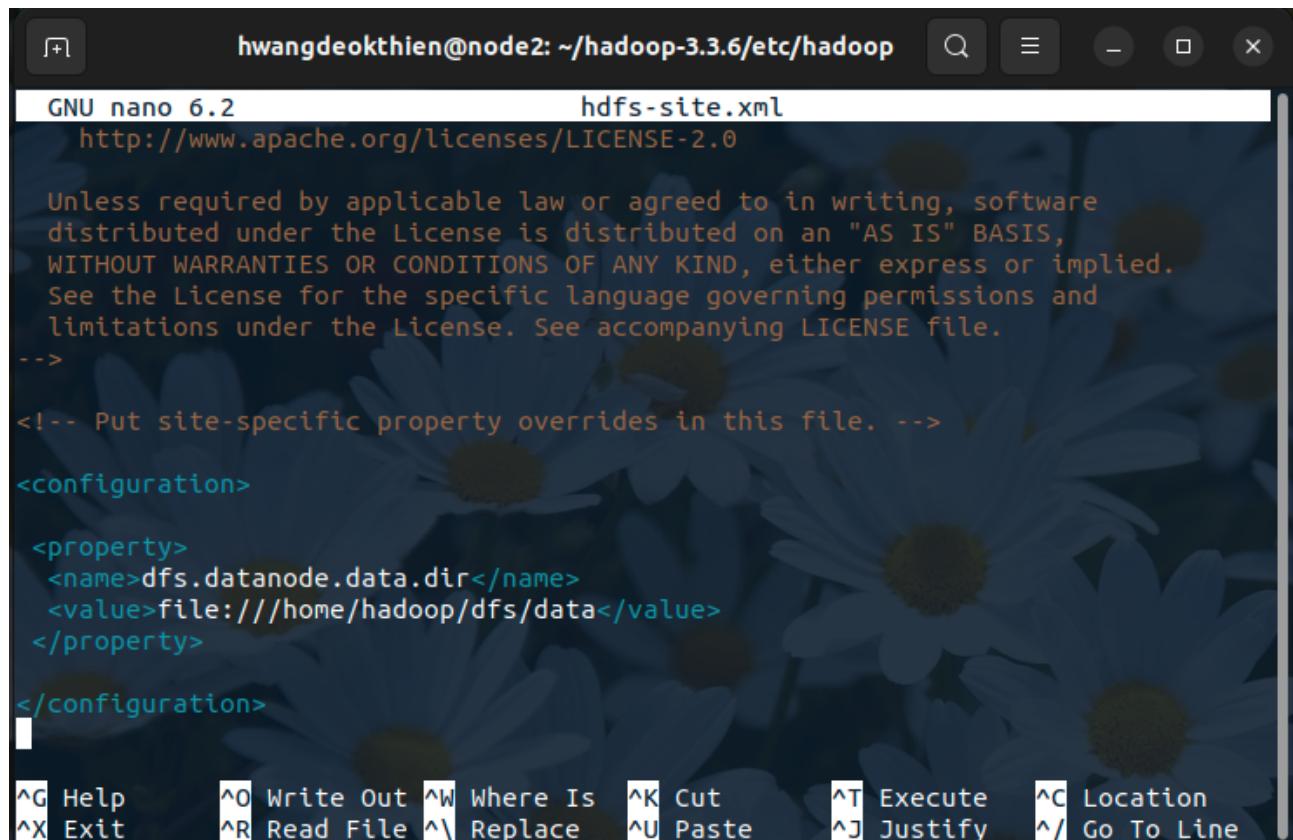
<configuration>
<property>
<name>dfs.namenode.checkpoint.dir</name>
<value>file:///home/hadoop/dfs/namesesecondary</value>
</property>

</configuration>


```

File menu: New, Open, Save, Exit
 Edit menu: Undo, Redo, Cut, Copy, Paste, Select All, Find, Replace, Go To Line
 View menu: Where Is, Location
 Help menu: About, Exit

Fig. 21. hdfs-site.xml configuration for backup node



```

hadoop@node2: ~/hadoop-3.3.6/etc/hadoop
GNU nano 6.2          hdfs-site.xml
http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.

-->

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:///home/hadoop/dfs/data</value>
</property>

</configuration>

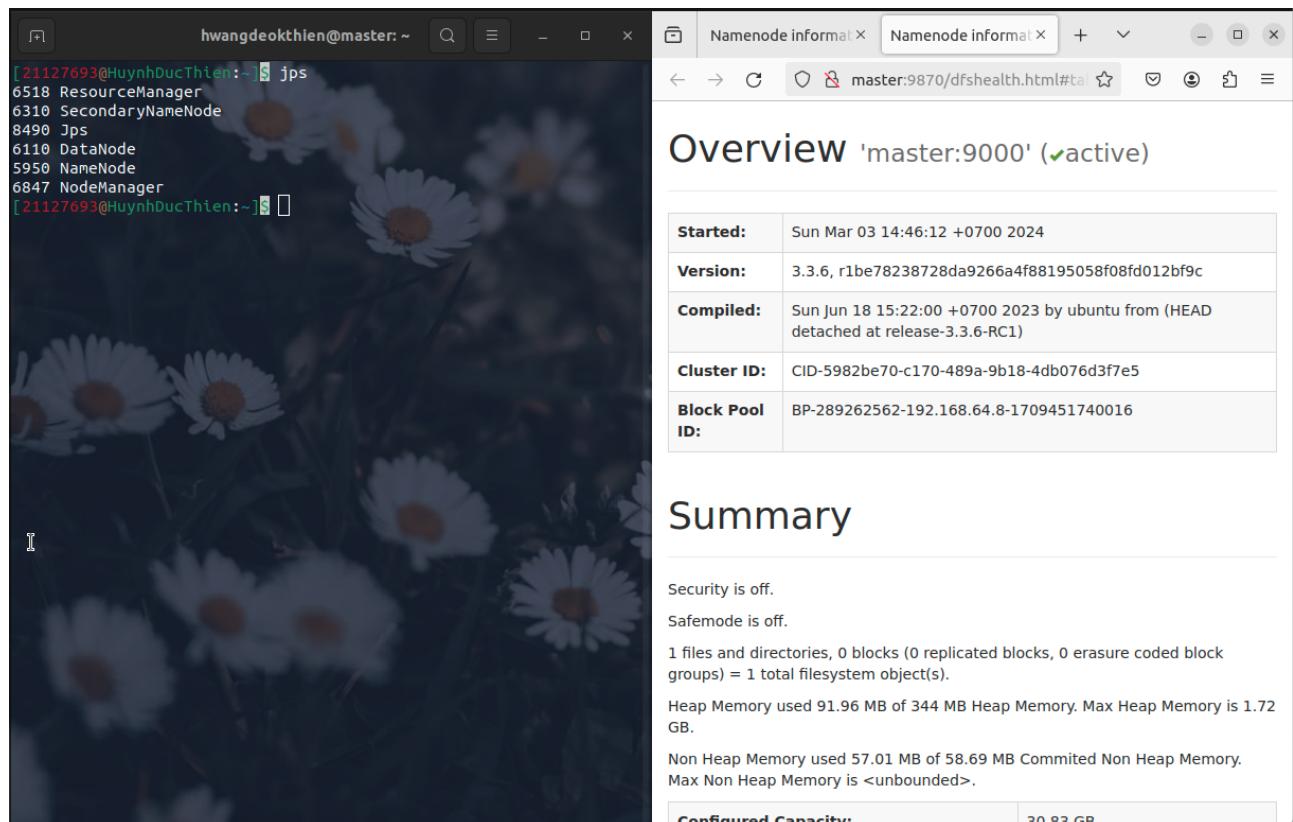
```

Keyboard Shortcuts:

- ^G Help
- ^O Write Out
- ^W Where Is
- ^K Cut
- ^T Execute
- ^C Location
- ^X Exit
- ^R Read File
- ^V Replace
- ^U Paste
- ^J Justify
- ^L Go To Line

Fig. 22. hdfs-site.xml configuration for data node

Step 5. Finally, we format the name node and begin using Hadoop.



Terminal Output (jps):

```

[21127693@HuynhDucThien: ~] $ jps
6518 ResourceManager
6310 SecondaryNameNode
8490 Jps
6110 DataNode
5950 NameNode
6847 NodeManager
[21127693@HuynhDucThien: ~] $ 

```

Browser Overview Page:

Overview 'master:9000' (active)

Started:	Sun Mar 03 14:46:12 +0700 2024
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 15:22:00 +0700 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-5982be70-c170-489a-9b18-4db076d3f7e5
Block Pool ID:	BP-289262562-192.168.64.8-1709451740016

Summary

Security is off.
Safemode is off.
1 files and directories, 0 blocks (0 replicated blocks, 0 erasure coded block groups) = 1 total filesystem object(s).
Heap Memory used 91.96 MB of 344 MB Heap Memory. Max Heap Memory is 1.72 GB.
Non Heap Memory used 57.01 MB of 58.69 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	30.83 GB
-----------------------------	----------

Fig. 23. Overview site

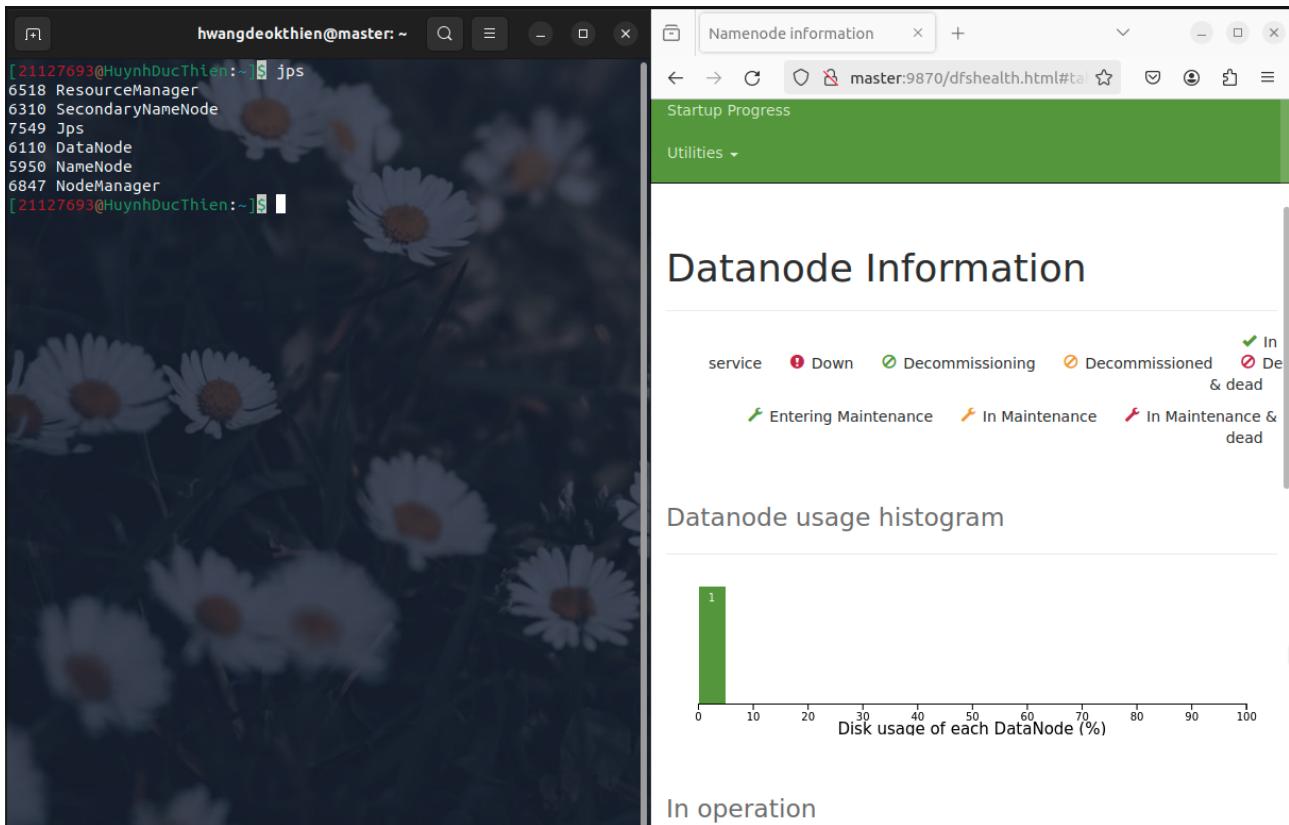


Fig. 24. Datanode site

4.2 Research about Security in Hadoop Set-up

After reviewing the documentation on configuring Hadoop in Secure Mode [5], we have addressed the associated questions.

1. Is your Hadoop secure? Give a short explanation if your answer is yes. Otherwise, give some examples of risks to your system.
 - The security of a Hadoop cluster depends on how well it has been configured and managed. Security features in Hadoop include authentication, authorization, and encryption. If these features are properly implemented and configured, the Hadoop cluster can be considered secure.
 - Examples of risks if not properly secured include unauthorized access to sensitive data, data breaches, and potential disruption of the cluster by malicious activities.
2. From your perspective, which method is better when securing your HDFS: authentication, authorization, or encryption? Give an explanation about your choices. Let's examine those security methods.
 - Authentication: This involves verifying the identity of users and ensuring they are who they claim to be. It is crucial for preventing unauthorized access. Implementing strong authentication mechanisms, such as Kerberos, is essential.
 - Authorization: This defines what actions authenticated users are allowed to perform. Proper authorization ensures that users can only access the data and resources they are permitted to. It is crucial for maintaining the principle of least privilege.
 - Encryption: Encrypting data at rest and in transit adds an extra layer of security. It ensures that even if unauthorized access occurs, the data remains unreadable without the proper decryption keys. In general, a combination of all three (authentication, authorization, and encryption) is recommended for a comprehensive security strategy. Authentication and authorization control access, while encryption protects the confidentiality of the data.

If we have to choose only one method for securing Hadoop Distributed File System (HDFS), **authentication** would be the primary priority. Reasoning,

- Authentication ensures that only authorized users or systems can access HDFS by verifying their identity.
- Without proper authentication, authorization and encryption measures may become irrelevant because unauthorized entities could gain access to the system.

5 Reflection

Throughout the lab, with the goal of learning and exploring the Hadoop tool, the entire group experienced installing big data processing tools, becoming familiar with the command-line environment and environment variables, and also delving into MapReduce and its application to solve the WordCount Problem.

Tasks within the group were evenly distributed among all four members with the intention that each member would contribute and gain understanding of the topics covered in the lab. Each member of the group also contributed their utmost to complete their assigned tasks and ensure that the group's progress was maintained steadily.

During the execution of this lab, both individuals and the group as a whole encountered certain difficulties, which also served as opportunities for members to develop further. Some of the difficulties encountered include:

- The original Apache documentation was difficult to access, with relatively complex content for members due to limited time to familiarize themselves with the tool. Additionally, the progress of the class was slow, resulting in students lacking some necessary knowledge to understand the lab content. To overcome this difficulty, group members combined references from external necessary documentation alongside the original material to gain a more comprehensive and accessible perspective. Though documentation on the topic was limited, it helped the group improve its situation.
- Difficulties in installing Hadoop were also notable, as members often struggled with installation due to some members' lack of understanding about the installation environment, including virtual machines, Linux environments, and environment variables. Alongside consulting online instructional materials, group members frequently communicated to clarify doubts and gain a deeper understanding of the peculiarities and conditions to consider during installation.
- Reading and studying various topics posed a significant challenge, as individual documents often did not encompass the entire context and there were ongoing changes in technology over the years, requiring students to update their knowledge with newer information. This limitation arose as newer architectures had fewer additional reading sources. Consequently, group members proactively shared the documents they read and organized meetings to clarify knowledge for each other, creating comprehensive perspectives, and enhancing the group's understanding of the topics.
- Group members lacked extensive knowledge of the Java programming language, making the process of familiarization more time-consuming and requiring more effort.

This lab serves as a good foundation for future endeavors and upcoming labs, as group members have equipped themselves with the necessary knowledge. The goal of the group is to continue exploring and further developing their understanding of big data and the essential tools surrounding this topic.

References

- [1] Installation of hadoop and mapreduce wordcount example.
- [2] Source code wordcount v1.0. Accessed: February 27, 2024.

- [3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, jan 2008.
- [4] Hyejung Lim. Setting up a fully distributed hadoop cluster, 2020. Accessed: March 1, 2024.
- [5] Owen O’Malley. Hadoop security architecture, 2014. Accessed: February 27, 2024.
- [6] Tirth Shah. Word count in apache hadoop (mapreduce), 2023. Accessed: February 27, 2024.
- [7] Muhammad Zaman. Setting-up fully distributed hadoop cluster, 2018. Accessed: March 1, 2024.