Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – IV Laboratory AY:
2024-25

Experiment 1

**Name:Jhanvi Parekh**

**Sap: 60009210033**

Aim: Implement a word count algorithm using MapReduce to count the number of times each word appears in a document.

Theory:

Hadoop can be developed in programming languages like Python and C++. MapReduce Hadoop is a software framework for ease in writing applications of software processing huge amounts of data. MapReduce Word Count is a framework which splits the chunk of data, sorts the map outputs and input to reduce tasks. A File-system stores the output and input of jobs. Re-execution of failed tasks, scheduling them and monitoring them is the task of the framework.

Introduction:

MapReduce is a programming model and processing technique designed for processing and generating large datasets that can be parallelized across a distributed cluster of computers. It is commonly used for big data processing tasks and is well-suited for tasks like word counting.

MapReduce Workflow:

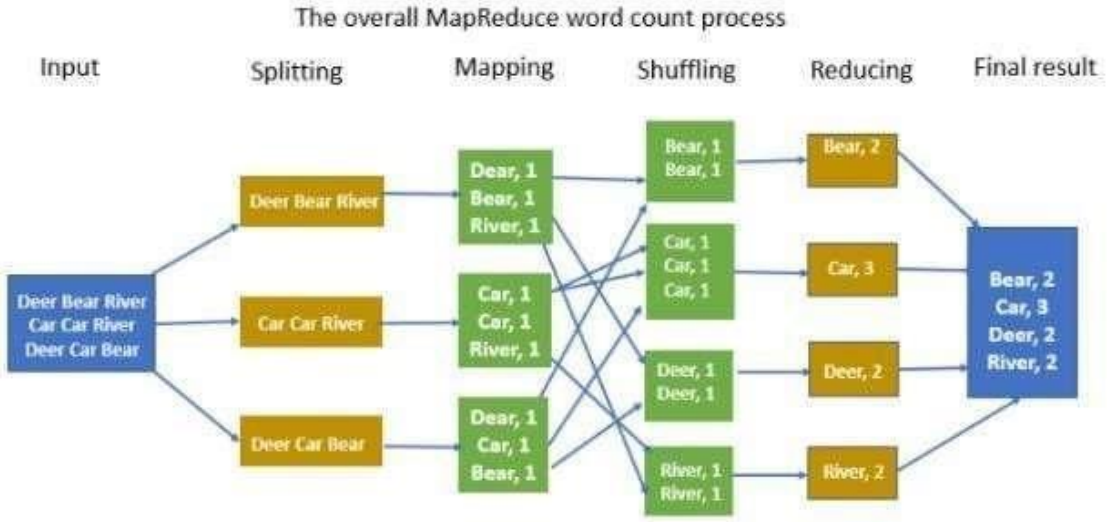
The MapReduce workflow consists of two main phases: the Map phase and the Reduce phase.

1. Map Phase:
    - Input Splitting: The input data, such as a large text document, is divided into smaller equal-sized splits.
    - Mapping: In this phase, the Mapper function processes each split independently. The Mapper takes the input data, extracts the required information, and emits key-value pairs. In the case of word counting, the key is typically a word, and the value is a count of 1.
2. Reduce Phase:

- Shuffling and Sorting: The MapReduce framework groups and sorts the output keyvalue pairs from the Mappers based on their keys. This ensures that all instances of the same word are grouped together.



- Reducing: The Reducer function processes the key-value pairs generated by the Map phase. In the word counting task, the Reducer takes the word as a key and a list of counts associated with that word. It then sums up the counts to produce the final count for that word.
- Final Output: The Reducers' output consists of key-value pairs where the key is the word and the value is the total count of that word in the input document.




Word Count using MapReduce:

The input given is converted into the string. Then it tokenizes them into words as if it need to break them. The mapper will append a single number or digit to each word and mapper outputs are shown above. Once we get the outputs as key-value pairs, once we pass the offset address as input to the mapper, the output of the value would be key-value pairs.

The output is getting into the sorting and shuffling phase. When we sort based on keys, all the keys will come to once a particular place. Sorting on the keys and shuffling the keys is done. A single word will go to a single reducer. Input to the reducer is key-value pairs. Once we pass outputs to reducer as input, the reducer will sum up all the values to keys.

That is, it groups up all the similar keys and output would be the concatenated key-value pair. The reducer will pick the result from the temp path and it will arrive at the final result. When we execute map-reduce, the input and output should be created in HDFS.

Sample Problem:

Input Data: Consider the following input text:

Line 1: Hello world, how are you? Hello, world!

Line 2: I'm doing well, thank you. How about you?



Map Phase: In this phase, we process each line and tokenize it into words. For each word found, we emit key-value pairs (word, 1):

"Hello" 1 "world" 1 "how" 1 "are" 1 "you" 1 "Hello" 1 "world" 1 "I'm" 1 "doing" 1 "well" 1 "thank you" 1 "How" 1 "about" 1 "you" 1

Shuffling and Sorting (Intermediate Step): The MapReduce framework groups and sorts the emitted key-value pairs by the key (word):

"Hello" 1, 1 "world" 1, 1 "how" 1 "are" 1 "you" 1, 1 "I'm" 1 "doing" 1 "well" 1 "thank you" 1 "How" 1 "about" 1

Reduce Phase: In this phase, we process each group of key-value pairs, summing up the values for each word to count its occurrences. We emit the word and its total count:

"Hello" 2 "world" 2 "how" 1 "are" 1 "you" 2 "I'm" 1 "doing" 1 "well" 1 "thank you" 1 "How" 1 "about" 1

This illustrates the Word Count process using key-value pairs, where the words serve as keys and the counts as values. The process efficiently counts the occurrences of each word in the input text by distributing the workload across multiple mappers and reducers.

Lab Exercise:

Step 1: Obtain a text document that you want to perform word count on. This can be a plain text file or a dataset in a suitable format.

Step 2: Implement a MapReduce program to count the number of times each word appears in the document. This program should consist of:

- A Mapper function that tokenizes the text, emits key-value pairs (word, 1) for each word.

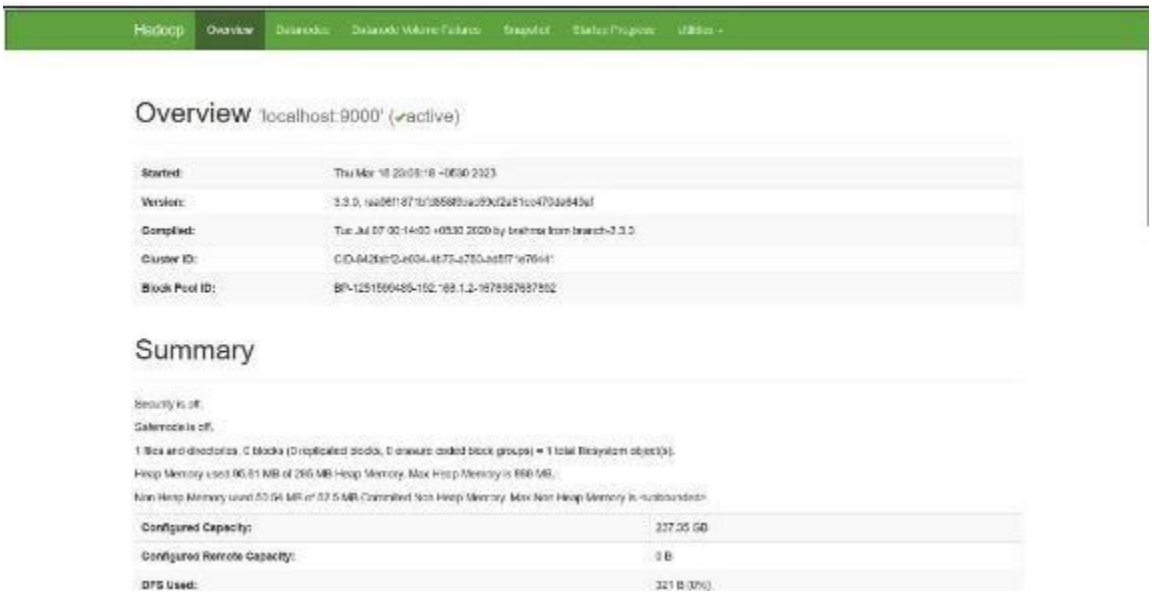
- A Reducer function that receives key-value pairs, groups them by key (word), and sums up the values to get the word count.

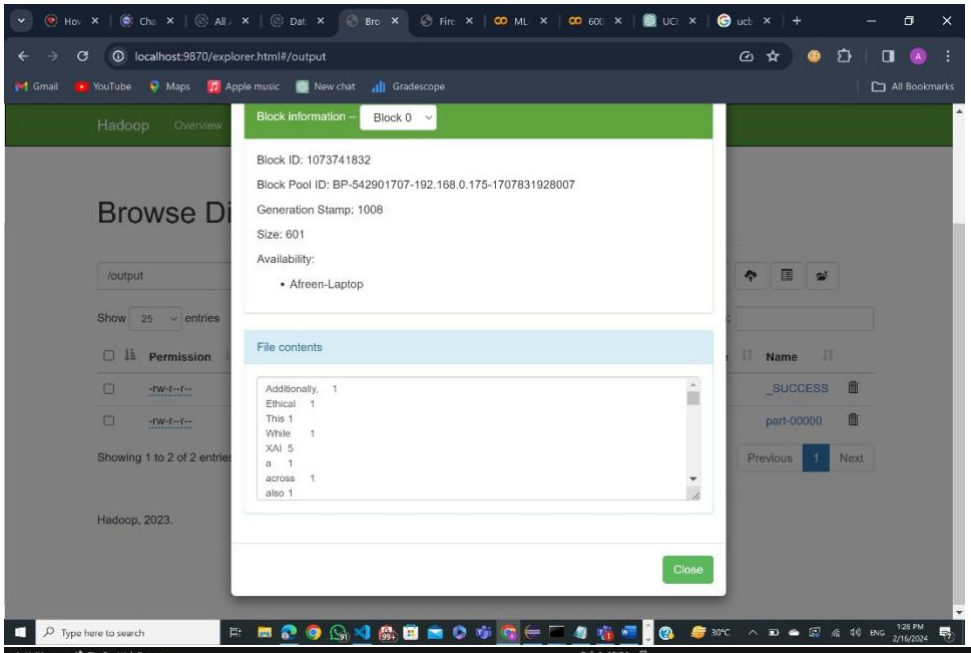
Code with Output:

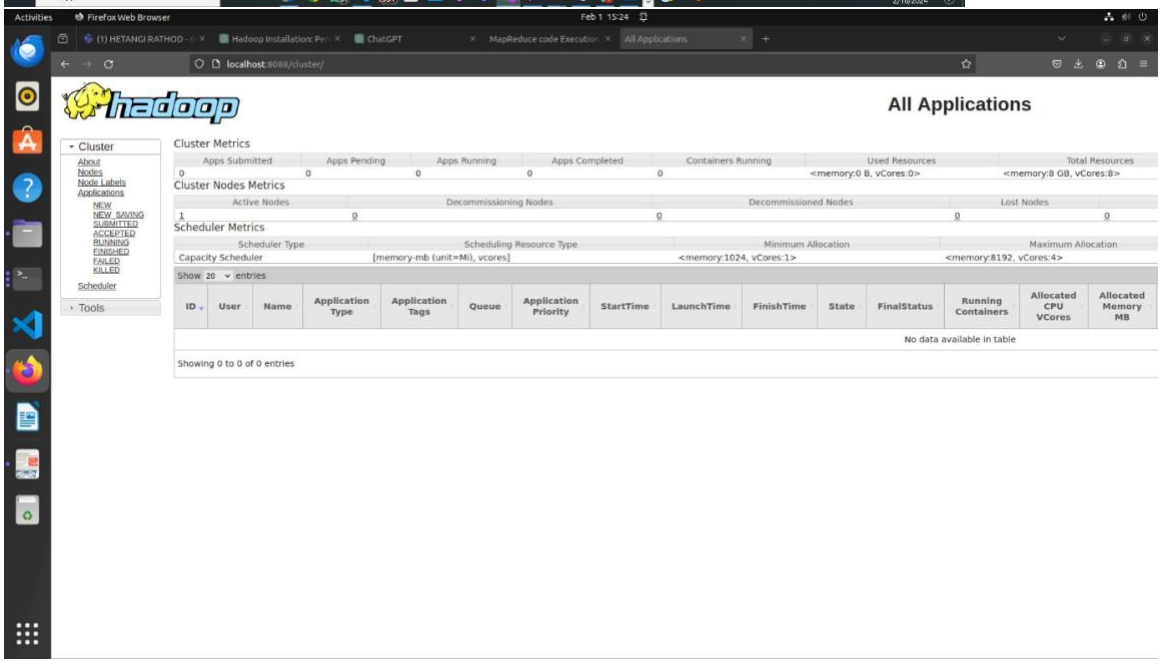
```
1347  sudo apt update
1348  sudo apt upgrade
1349  sudo apt openjdk-8-jdk
1350  sudo apt install openjdk-8-jdk
1351  wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
1352  tar -xzvf hadoop-3.3.6.tar.gz
1353  sudo mv hadoop-3.3.6/usr/local/hadoop
1354  sudo mv hadoop-3.3.6 /usr/local/hadoop
1355  cd /usr/local/hadoop/etc/hadoop
1356  nano hadoop-env.sh
1357  nano hdfs-site.xml
1358  nano core-site.xml
1359  nano hdfs-site.xml
1360  cp mapred-site.xml.template mapred-site.xml
1361  nano mapred-site.xml
1362  nano yarn-site.xml
1363  nano hadoop-env.sh
1364  nano hdfs-site.xml
1365  nano mapred-site.xml
1366  sudo apt update
1367  sudo apt upgrade -y
1368  sudo apt install openjdk-8-jdk -y
1369  wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
1370  tar xzf hadoop-3.3.6.tar.gz
1371  tar xzf hadoop-3.3.6.tar.gz.1
1372  sudo mv hadoop-3.3.6 /usr/local/hadoop
1373  sudo chown -R hdoop:hdoop /usr/local/hadoop
1374  echo 'export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")' | sudo tee -a /usr/local/hadoop/etc/hadoop/hadoop-env.sh
1375  sudo apt update
1376  sudo apt upgrade -y
1377  sudo apt install openssh-server openssh-client -y
1378  sudo apt install openjdk-11-jdk -y
1379  sudo adduser hdoop
1380  su - hdoop
1381  sudo nano/etc/profile
1382  source ~/.bashrc
1383  sudo apt update
1384  sudo apt upgrade -y
1385  sudo apt install openssh-server openssh-client -y
1386  sudo apt install openjdk-11-jdk -y
1387  sudo adduser hdoop
1388  sudo adduser devansh
1389  su - devansh
1390  sudo apt update && sudo apt install openjdk-8-jdk
1391  java -version
1392  sudo apt install ssh
1393  sudo adduser dhruv209
1394  su - dhruv209
1395  sudo apt update
1396  sudo apt upgrade
1397  tar -xzvf hadoop-3.3.0.tar.gz
```

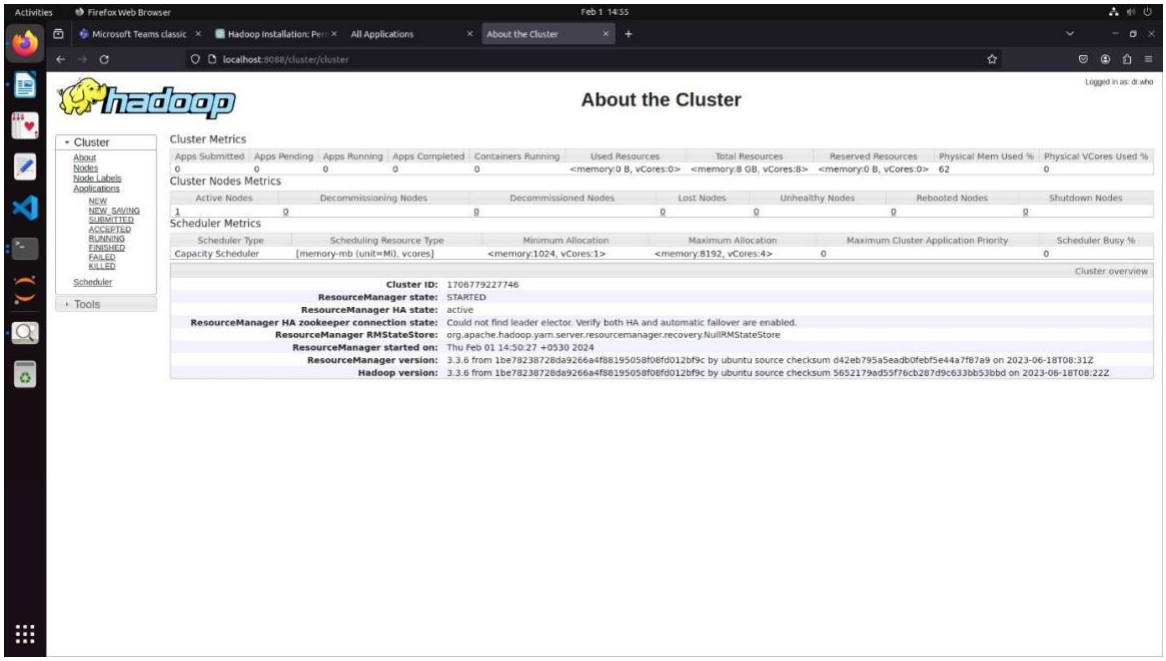
Administrator: Command Prompt

```
6648 Jps
6224 NodeManager
2624 NameNode
6788 ResourceManager
3576 Eclipse
0764 DataNode
2700 org.eclipse.equinox.launcher_1.6.700.v20231214-2017.jar

:\windows\system32>hadoop fs -mkdir /input

:\windows\system32>hadoop fs -put C:\Users\Admin\OneDrive\Desktop\input.txt \input

:\windows\system32>hadoop fs -cat /output/part-00000
dditionally,   1
thical 1
his    1
hile   1
AI     5
       1
cross  1
lso    1
nd     3
pplication     1
reas   1
een    1
onsiderations  1
ontinued       1
eployment      1
evelopment     2
evelopment.    1
ifferent       1
omains.        1
fficacy        1
merging        1
xisting        1
xploration     1
xplore 1
or     1
rameworks      1
urther 1
ave    1
mprove 1
n      1
ncludes        1
ntegration     1
nvestigation.  1
anguage        1
earning        1
ike    1
ade    1
erit   1
atural 1
eed    1
f      2
rocessing.     1
efining        1
einforcement   1
esearch,       1
```




localhost:9870/explorer.html#/output
Hadoop
Overview
Block information –
Block 0
Block ID: 1073741832
Block Pool ID: BP-542901707-192.168.0.175-1707831928007
Generation Stamp: 1008
Size: 601
Availability:
• Afreen-Laptop
File contents
Additionally, 1
Ethical 1
This 1
While 1
XAI 5
a 1
across 1
also 1
Close
Browse Di
/output
Show 25 entries
Permission
Name
-rw-r--r--
_SUCCESS
part-00000
Showing 1 to 2 of 2 entries
Previous
1
Next
Hadoop, 2023.


localhost:8088/cluster/
All Applications
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Tools
Cluster Metrics
Apps Submitted
Apps Pending
Apps Running
Apps Completed
Containers Running
Used Resources
Total Resources
<memory:0 B, vCores:0>
<memory:8 GB, vCores:8>
Cluster Nodes Metrics
Active Nodes
Decommissioning Nodes
Decommissioned Nodes
Lost Nodes
Scheduler Metrics
Scheduler Type
Scheduling Resource Type
Minimum Allocation
Maximum Allocation
Capacity Scheduler
[memory-mb (unit=Mi), vcores]
<memory:1024, vCores:1>
<memory:8192, vCores:4>
Show 20 entries
ID
User
Name
Application Type
Application Tags
Queue
Application Priority
StartTime
LaunchTime
FinishTime
State
FinalStatus
Running Containers
Allocated CPU VCores
Allocated Memory MB
No data available in table
Showing 0 to 0 of 0 entries




Conclusion:

Thus, we successfully implemented a Word Count using MapReduce. This exercise allowed us to harness the power of distributed computing to efficiently count word occurrences in a large dataset. We learned about the MapReduce paradigm and the importance of data preprocessing. This experiment provided valuable insights into handling big data and parallel processing.