

## Data Engineering-1 Assignment – 2



### Topic: Hadoop and MapReduce

Submitted by – Rajib Datta

#### **Content:**

Sl. No.	Task details	:	Page No.
Part-1	<a href="#">Introduction to the Hadoop Framework</a>	:	2
1.0	<a href="#">Setup VM with Hadoop (1-6)</a>	:	2
1.0.7	<a href="#">Setting up my VM</a>	:	2
1.1	<a href="#">Word count example in local (standalone) mode</a>	:	3
1.1.4	<a href="#">Answer the following questions: (local standalone) mode</a>	:	4
1.2	<a href="#">Setup pseudo-distributed mode</a>	:	5-6
1.2.3	<a href="#">HDFS web GUI: (screenshot)</a>	:	7
1.2.4	<a href="#">Question and Answers</a>	:	7-9
1.3	<a href="#">Word count in pseudo-distributed mode</a>	:	10-12
1.3.6	<a href="#">Question and Answers</a>	:	12-13
1.4.1	<a href="#">Modified word count example</a>	:	14
1.4.1-2	<a href="#">Word Count Coding Results with Plotting and (Modify Codes)</a>	:	14-20
1.5	<a href="#">NoSQL and MongoDB</a>	:	21-22
Part-2	Analyzing twitter data using Hadoop streaming and Python	:	23
2.1	<a href="#">Analyzing twitter data using Hadoop streaming and Python (Mapper &amp; Reducer python code)</a>	:	23-27
2.2	<a href="#">Analyzing twitter data using Hadoop streaming and Mango DB(Mapper &amp; Reducer python code)</a>	:	28-31
2.2	<a href="#">Question and Answer</a>	:	32-33

## 1. Introduction to the Hadoop Framework.

**1.0.(1-6): Setup VM with Hadoop:** Creation is done as per instruction and below is the screenshot of Intanse associate with VM

Image:1

<input checked="" type="checkbox"/>	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State
<input checked="" type="checkbox"/>	<a href="#">RajibDatta_A2</a>	Ubuntu 20.04 - 2023.12.07	192.168.2.133, 130.238.29.191	<a href="#">ssc.medium</a>	RD_SSH	Active	nova	None	Running

Displaying 1 item

Image:2

UPPMAX 2024/1-1 Inter...		192.168.2.133, 130.238.29.191
Security Groups		
	default	ALLOW IPv6 from default ALLOW IPv4 to 0.0.0.0/0 ALLOW IPv6 to ::/0 ALLOW IPv4 from default ALLOW IPv4 8888/tcp from 0.0.0.0/0 ALLOW IPv4 22/tcp from 0.0.0.0/0 ALLOW IPv4 icmp from 0.0.0.0/0
Metadata		
	Key Name	RD_SSH
	Image Name	<a href="#">Ubuntu 20.04 - 2023.12.07</a>
	Image ID	31dbaa8-2200-44bc-b4f2-44ab4be099ca
Volumes Attached		
	Attached To	<a href="#">00521144-5763-42b4-b805-5a5bb0b2cf28</a> on /dev/vda
	Attached To	<a href="#">Rajib_DattaV_A2</a> on /dev/vdb

**1.0.7.1: Setting up my VM:** Screenshot of my hostname in below.

```
ubuntu@rajibdatta-a2: /etc
GNU nano 4.8 hosts
127.0.0.1 localhost rajibdatta-a2

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

### 1.0.7.3: Jave update and path setup screenshot.

```
ubuntu@rajibdatta-a2: /etc
GNU nano 4.8 environment
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin"

JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64/

ubuntu@rajibdatta-a2:/etc$ $JAVA_HOME
-bash: /usr/lib/jvm/java-17-openjdk-amd64/: Is a directory
```

### 1.0.7.5: Screenshot of Hadoop version in below.

```
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hadoop version
Hadoop 3.3.6
Source code repository https://github.com/apache/hadoop.git -r 1be78238728da9266a4f88195058f08fd012bf9c
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/ubuntu/hadoop-3.3.6/share/hadoop/common/hadoop-common-3.3.6.jar
ubuntu@rajibdatta-a2:~$
```

## 1.1: Word count example in local (standalone) mode.

1.1.1: Create a folder “input” in my home directory (/home/ubuntu) on my instance using command mkdir. (Screenshot is below).

```
ubuntu@rajibdatta-a2: ~
ubuntu@rajibdatta-a2:~$ ls -l
total 713012
drwxr-xr-x 11 ubuntu ubuntu      4096 Feb  5 13:06 hadoop-3.3.6
-rw-rw-r--  1 ubuntu ubuntu 730107476 Jun 25  2023 hadoop-3.3.6.tar.gz
drwxrwxr-x  2 ubuntu ubuntu      4096 Jan 31 10:07 input
```

1.1.2: Download the following data file and place it in that directory.(screenshot is below)

```
ubuntu@rajibdatta-a2:~/input$ ls -l
total 660
-rw-rw-r-- 1 ubuntu ubuntu 674684 Jan  9 10:13 20417.txt.utf-8
ubuntu@rajibdatta-a2:~/input$
```

1.1.3: Then go back to my home directory to execute the following command to run the canonical MapReduce example: (Screenshot is below)

```
ubuntu@rajibdatta-a2: ~
ubuntu@rajibdatta-a2:~$ ls -l
total 713012
drwxr-xr-x 11 ubuntu ubuntu 4096 Feb  5 13:06 hadoop-3.3.6
-rw-rw-r-- 1 ubuntu ubuntu 730107476 Jun 25 2023 hadoop-3.3.6.tar.gz
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 31 10:07 input
drwxr-xr-x 2 ubuntu ubuntu 4096 Jan 31 10:17 output
ubuntu@rajibdatta-a2:~$
```

#### 1.1.4. Answer the following questions:

a. Look at the contents of the folder “output”, what files are placed in there? What do they mean?

Ans: After viewing the output folder I found a screenshot of the mentioned folder after running the instruction mentioned code.

```
ubuntu@rajibdatta-a2:~$ ls -l
total 713012
drwxr-xr-x 11 ubuntu ubuntu 4096 Jan 31 10:21 hadoop-3.3.6
-rw-rw-r-- 1 ubuntu ubuntu 730107476 Jun 25 2023 hadoop-3.3.6.tar.gz
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 31 10:07 input
drwxr-xr-x 2 ubuntu ubuntu 4096 Jan 31 10:17 output
ubuntu@rajibdatta-a2:~$ cd output/
ubuntu@rajibdatta-a2:~/output$ ls -la
total 208
drwxr-xr-x 2 ubuntu ubuntu 4096 Jan 31 10:17 .
drwxr-xr-x 7 ubuntu ubuntu 4096 Jan 31 10:17 ..
-rw-r--r-- 1 ubuntu ubuntu 8 Jan 31 10:17 ._SUCCESS.crc
-rw-r--r-- 1 ubuntu ubuntu 1540 Jan 31 10:17 .part-r-00000.crc
-rw-r--r-- 1 ubuntu ubuntu 0 Jan 31 10:17 _SUCCESS
-rw-r--r-- 1 ubuntu ubuntu 195945 Jan 31 10:17 part-r-00000
ubuntu@rajibdatta-a2:~/output$
```

The output folder consists of the results of the WordCount job will be stored.

The WordCount example counts the occurrences of each word in the input files and produces output in the specified output directory. The output typically consists of multiple files in the specified output directory, and the actual content of these files will be the word and its corresponding count.

b) Looking at the output files, how many times did the word ‘Discovery’ (casesensitive) appear?

Ans: By putting below command cases I got below outputs.

**Case-1. Command:** `cat /home/ubuntu/output/part-* | grep -c 'Discovery'`

**Output:** 1

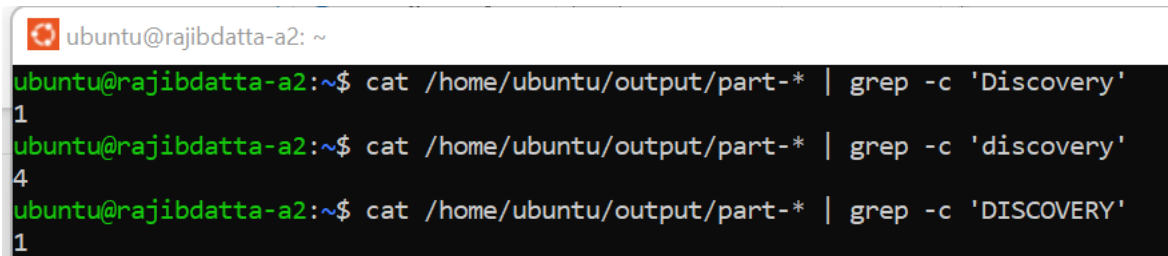
**Case-2. Command:** `cat /home/ubuntu/output/part-* | grep -c 'discovery'`

**Output:** 4

**Case-3. Command:** `cat /home/ubuntu/output/part-* | grep -c 'DISCOVERY'`

**Output: 1**

**Screenshot of outputs:**



```
ubuntu@rajibdatta-a2: ~  
ubuntu@rajibdatta-a2:~$ cat /home/ubuntu/output/part-* | grep -c 'Discovery'  
1  
ubuntu@rajibdatta-a2:~$ cat /home/ubuntu/output/part-* | grep -c 'discovery'  
4  
ubuntu@rajibdatta-a2:~$ cat /home/ubuntu/output/part-* | grep -c 'DISCOVERY'  
1
```

c) In this example we used Hadoop in "Local (Standalone) Mode". What is the difference between this mode and the Pseudo-distributed mode?

Ans: Local (Standalone) Mode and Pseudo-distributed Mode are two configurations where we can run Hadoop for development and testing purposes.

Local (Standalone) Mode: In Local Mode, Hadoop runs on a single machine as a single Java process. It is mainly used for development and debugging purposes. It simulates the distributed nature of Hadoop but doesn't take advantage of parallel processing or distributed storage. It's suitable for testing our MapReduce programs on a small scale before deploying them to a full Hadoop cluster.

To run a job in Local Mode, we might use a command like:

**hadoop jar our-mapreduce-job.jar input-directory output-directory**

**Our code here is:**

```
hadoop-3.3.6/bin/hadoop jar ./Hadoop-  
3.3.6/share/hadoop/mapreduce/hadoopmapreduce-  
examples-3.3.6.jar wordcount ./input ./output
```

## **1.2: Setup pseudo-distributed mode:**

**1.2.1:** Follow the instructions in the link below to set up Hadoop in pseudo-distributed operation. Execute ONLY the instructions for Configuration and step 1 ("format the namenode") of "Execution". (Screenshots are below):

```
ubuntu@rajibdatta-a2: ~/hadoop-3.3.6/etc/hadoop
GNU nano 4.8                                core-site.xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

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

```
ubuntu@rajibdatta-a2: ~/hadoop-3.3.6/etc/hadoop
GNU nano 4.8                                hdfs-site.xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    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.replication</name>
    <value>1</value>
  </property>
</configuration>
```

And this command: `hadoop-3.3.6/bin/hdfs --daemon start namenode`

```
ubuntu@rajibdatta-a2: ~/hadoop-3.3.6
ubuntu@rajibdatta-a2:~/hadoop-3.3.6$ /home/ubuntu/hadoop-3.3.6/bin/hdfs --daemon start namenode
namenode is running as process 26316. Stop it first and ensure /tmp/hadoop-ubuntu-namenode.pid file is empty before rety.
ubuntu@rajibdatta-a2:~/hadoop-3.3.6$
```

- Pseudo-distributed Mode is also a single-node setup, but it emulates a distributed environment more closely. Each Hadoop daemon (such as NameNode, DataNode, ResourceManager, and NodeManager) runs as separate Java processes.
- While it runs on a single machine, it gives us a more realistic environment to test Hadoop applications because different components are running as separate processes.
- It allows us to evaluate the behavior of a Hadoop cluster on a small scale, which can be useful for debugging and understanding how the different components interact.
- To run a job in Pseudo-distributed Mode, we would typically follow the same commands and procedures as we would in a fully distributed cluster, but everything runs on a single machine.

In summary, Local Mode is simpler and primarily used for development and initial testing on a single machine, while Pseudo-distributed Mode provides a more realistic simulation of a distributed environment on a single machine, allowing us to test Hadoop applications in a setting that resembles a larger cluster.

### 1.2: After running daemons, I got the below output with 'jps' command.

```
ubuntu@rajibdatta-a2: ~
ubuntu@rajibdatta-a2:~$ jps
36001 NameNode
44371 Jps
36107 DataNode
ubuntu@rajibdatta-a2:~$
```

### 1.2.3: HDFS web GUI: (screenshot)

Overview 'localhost:9000' (✓active)

Started:	Mon Feb 05 12:56:52 +0100 2024
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 10:22:00 +0200 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-e277c481-4d15-46e9-b281-428f13c0c987
Block Pool ID:	BP-1343876030-127.0.0.1-1707134089831

Summary

Security is off.  
Safemode is off.

### 1.2.4: Question and Answer:

a) What are the roles of files core-site.xml and hdfs-site.xml ?

**Ans:** In Hadoop, core-site.xml and hdfs-site.xml are two important configuration files that define settings for the Hadoop core components and Hadoop Distributed File System (HDFS), respectively. In below, I discuss the roles of each file:

## 1. core-site.xml:

The core-site.xml file contains configuration properties that are common to both Hadoop MapReduce and HDFS. It includes settings related to the underlying Hadoop Distributed File System (HDFS) and general configurations for the Hadoop framework. Some key properties include:

- i. fs.defaultFS: Specifies the default file system URI. It defines the default filesystem for Hadoop, and it is used by both HDFS and MapReduce. For example XML code is below:

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

- ii. hadoop.tmp.dir: Specifies the base directory for Hadoop's temporary data. It is used for storing intermediate data during Hadoop job execution.
- iii. io.file.buffer.size: Defines the buffer size for reading and writing files.

## 2. hdfs-site.xml:

The hdfs-site.xml file contains configuration properties specific to the Hadoop Distributed File System (HDFS). It includes settings that are used by the Namenode, Datanode, and other components of HDFS. Some key properties include:

- i. dfs.replication: Defines the default replication factor for HDFS blocks. This property specifies the number of replicas for each block.

```
<property>
  <name>dfs.replication</name>
  <value>3</value>
</property>
```

- ii. dfs.namenode.name.dir: Specifies the directory where the Namenode stores its metadata, such as the namespace and the block metadata.
- iii. dfs.datanode.data.dir: Specifies the directory where the Datanode stores its data blocks.
- iv. dfs.permissions.enabled: Indicates whether HDFS permissions are enabled (true or false).

These files are typically located in the **conf** directory of our Hadoop installation, and their configurations are essential for the proper functioning of Hadoop services. We may need to customize these configurations based on our specific Hadoop cluster setup and requirements.

b) Describe the different services listed when executing 'jps'. What are their functions in Hadoop and HDFS?

**Ans:** The jps command is used to list Java Virtual Machine (JVM) processes running on a machine. In the context of Hadoop and HDFS (Hadoop Distributed File System), jps can be used to view the various services and daemons that are part of a Hadoop cluster. Here are some common services we might see when executing jps in a Hadoop environment:



## 1. Hadoop Services:

- ✓ NameNode:

Process Name: NameNode

Description: The NameNode is a critical component of HDFS. It manages the metadata, including the file system namespace and the mapping of blocks to DataNodes.

- ✓ SecondaryNameNode:

Process Name: SecondaryNameNode

Description: The SecondaryNameNode performs periodic checkpoints of the HDFS metadata from the NameNode. It does not act as a standby or backup; its primary role is to merge the edits log with the fsimage.

- ✓ DataNode:

Process Name: DataNode

Description: DataNodes are responsible for storing actual data blocks. They periodically send heartbeats and block reports to the NameNode and respond to requests for read and write operations.

- ✓ ResourceManager:

Process Name: ResourceManager

Description: The ResourceManager is the master daemon for resource management in YARN (Yet Another Resource Negotiator). It manages and schedules resources across applications.

- ✓ NodeManager:

Process Name: NodeManager

Description: NodeManagers run on each worker node and are responsible for managing resources and containers on the individual nodes. They communicate with the ResourceManager and launch containers.

- ✓ JobHistoryServer:

Process Name: JobHistoryServer

Description: The JobHistoryServer stores and serves information about completed MapReduce jobs. It helps in tracking job history and logs.

## 2. Other Services:

- ✓ Jps:

Process Name: Jps

Description: The Jps process itself is included in the output. It lists all the Java processes running on the machine.

- ✓ Hadoop MapReduce Services:

In addition to the core Hadoop services, if we are running MapReduce jobs, we might also see:

- ✓ JobTracker:

Process Name: JobTracker

Description: The JobTracker is the master daemon for MapReduce. It manages the execution of all MapReduce jobs, tracks the progress, and schedules tasks on TaskTrackers.

✓ TaskTracker:

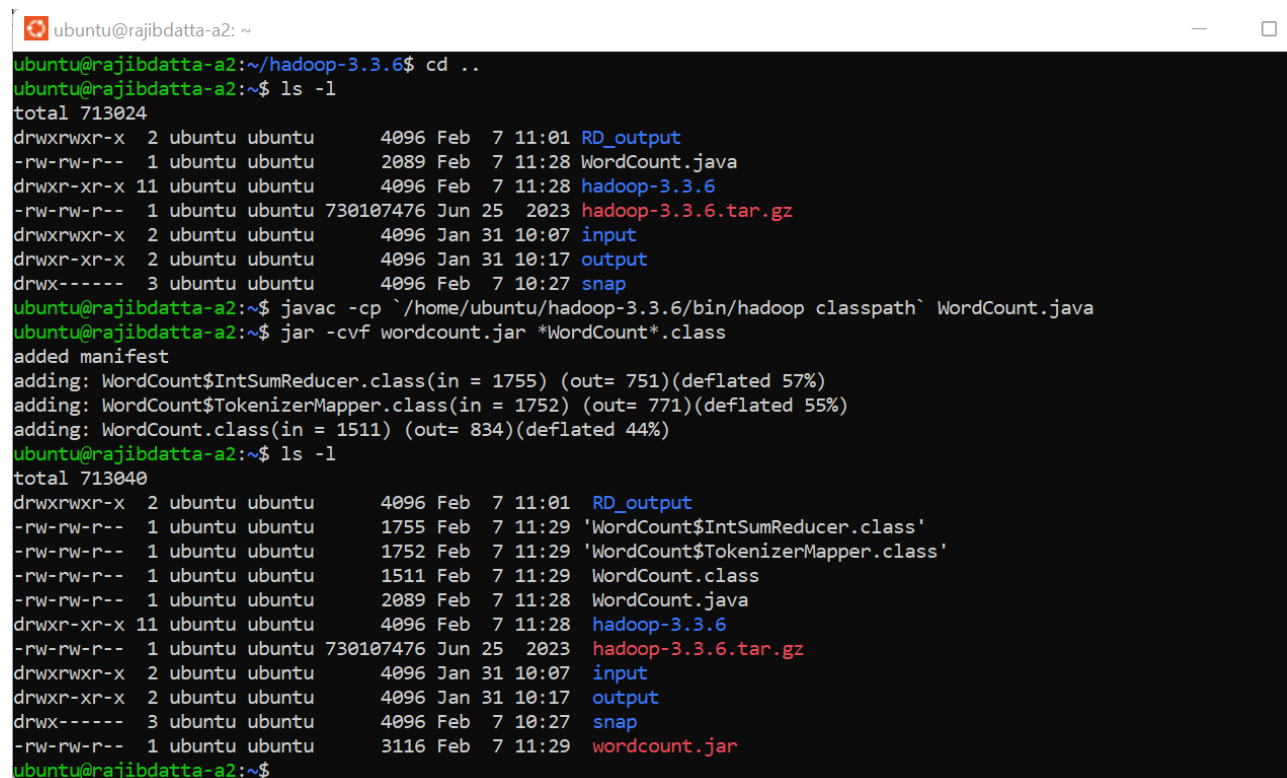
Process Name: TaskTracker

Description: TaskTrackers run on each worker node and are responsible for executing individual tasks (map or reduce) as directed by the JobTracker.

These processes collectively make up a Hadoop cluster, with components responsible for data storage, processing, and resource management. The actual output of jps may vary based on the Hadoop version and the specific configuration of our cluster.

### Task 1.3: Word count in pseudo-distributed mode.

1.3.0-1: Create a WordCount.java after compiling and make jar file by instruction code screenshot is below.



```
ubuntu@rajibdatta-a2: ~  
ubuntu@rajibdatta-a2:~/hadoop-3.3.6$ cd ..  
ubuntu@rajibdatta-a2:~$ ls -l  
total 713024  
drwxrwxr-x 2 ubuntu ubuntu 4096 Feb 7 11:01 RD_output  
-rw-rw-r-- 1 ubuntu ubuntu 2089 Feb 7 11:28 WordCount.java  
drwxr-xr-x 11 ubuntu ubuntu 4096 Feb 7 11:28 hadoop-3.3.6  
-rw-rw-r-- 1 ubuntu ubuntu 730107476 Jun 25 2023 hadoop-3.3.6.tar.gz  
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 31 10:07 input  
drwxr-xr-x 2 ubuntu ubuntu 4096 Jan 31 10:17 output  
drwx----- 3 ubuntu ubuntu 4096 Feb 7 10:27 snap  
ubuntu@rajibdatta-a2:~$ javac -cp `~/home/ubuntu/hadoop-3.3.6/bin/hadoop classpath` WordCount.java  
ubuntu@rajibdatta-a2:~$ jar -cvf wordcount.jar *WordCount*.class  
added manifest  
adding: WordCount$IntSumReducer.class(in = 1755) (out= 751)(deflated 57%)  
adding: WordCount$TokenizerMapper.class(in = 1752) (out= 771)(deflated 55%)  
adding: WordCount.class(in = 1511) (out= 834)(deflated 44%)  
ubuntu@rajibdatta-a2:~$ ls -l  
total 713040  
drwxrwxr-x 2 ubuntu ubuntu 4096 Feb 7 11:01 RD_output  
-rw-rw-r-- 1 ubuntu ubuntu 1755 Feb 7 11:29 'WordCount$IntSumReducer.class'  
-rw-rw-r-- 1 ubuntu ubuntu 1752 Feb 7 11:29 'WordCount$TokenizerMapper.class'  
-rw-rw-r-- 1 ubuntu ubuntu 1511 Feb 7 11:29 WordCount.class  
-rw-rw-r-- 1 ubuntu ubuntu 2089 Feb 7 11:28 WordCount.java  
drwxr-xr-x 11 ubuntu ubuntu 4096 Feb 7 11:28 hadoop-3.3.6  
-rw-rw-r-- 1 ubuntu ubuntu 730107476 Jun 25 2023 hadoop-3.3.6.tar.gz  
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 31 10:07 input  
drwxr-xr-x 2 ubuntu ubuntu 4096 Jan 31 10:17 output  
drwx----- 3 ubuntu ubuntu 4096 Feb 7 10:27 snap  
-rw-rw-r-- 1 ubuntu ubuntu 3116 Feb 7 11:29 wordcount.jar  
ubuntu@rajibdatta-a2:~$
```

### 1.3.2-3: below screenshot as per instructions code:

Now we should have added the text file from the file system of my VM to HDFS. Use the command below to verify that the file is indeed in HDFS:

hadoop-3.3.6/bin/hdfs dfs -ls input

```
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hdfs dfs -ls input
Found 1 items
-rw-r--r-- 1 ubuntu supergroup 674684 2024-02-07 13:35 input/20417.txt.utf-8
ubuntu@rajibdatta-a2:~$
```

1.3.4: After running Hadoop instructed command in my output directory I got below screenshot output.

```
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hdfs dfs -ls RD_output
Found 2 items
-rw-r--r-- 1 ubuntu supergroup 0 2024-02-07 11:52 RD_output/_SUCCESS
-rw-r--r-- 1 ubuntu supergroup 0 2024-02-07 11:52 RD_output/part-r-00000
ubuntu@rajibdatta-a2:~$
```

1.3.5: After running cat command I got below screenshot output.

```

|-----| 1
|-----| 3
|-----| 2
$ 77
Aeschylus 1
Aesop 3
Aeons 1
Aesthetic 1
'AS-IS', 1
"Defects," 1
"Information 1
"Plain 2
"Project 5
"Right 1
• 4
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hdfs dfs -cat RD_output3/part-r-00000 | grep -w Discovery
Discovery 5
ubuntu@rajibdatta-a2:~$
```

Output is : 5

```
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hdfs dfs -cat RD_output3/part-r-00000 | grep -w discovery
discovery 30
discovery, 1
discovery. 2
discovery: 1
ubuntu@rajibdatta-a2:~$

ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hdfs dfs -cat RD_output3/part-r-00000 | grep -w DISCOVERY
DISCOVERY 4
ubuntu@rajibdatta-a2:~$
```

### **1.3.6: Answer the following questions:**

a) Explain the roles of the different classes in the file WordCount.java.

Ans: In this small code snippet, there are three main classes, and each serves a specific purpose:

- a) WordCount: This is the main class containing the main method, which is the entry point for the MapReduce job. It configures and runs the MapReduce job.
- b) TokenizerMapper: This class extends Mapper, which is a base class for mapping input key/value pairs to a set of intermediate key/value pairs. It tokenizes each input line into words and emits key-value pairs, where the key is a word and the value is the count of occurrences of that word (initialized to 1).
- c) IntSumReducer: This class extends Reducer, which is a base class for reducing a set of intermediate key/value pairs to a smaller set of key/value pairs. It sums up the counts of each word emitted by the mapper and emits a final key-value pair, where the key is the word and the value is the total count of occurrences.

With these there are few classes import from Java library and org.apache.hadoop library such as

- java.io.IOException: This class is imported to handle input and out Exception.
- java.util.StringTokenizer: This class is used in the TokenizerMapper class within the map method to tokenize input text into words.
- org.apache.hadoop.conf.Configuration: This class in Apache Hadoop is responsible for managing configuration settings for Hadoop-based applications.
- org.apache.hadoop.fs.Path: This class in Apache Hadoop is responsible for representing file system paths in a platform-independent manner within the Hadoop ecosystem.
- org.apache.hadoop.io.IntWritable: This is responsible for representing integer values in a format suitable for storage, transmission, and processing within the Hadoop ecosystem, playing a vital role in the efficient handling of integer data in Hadoop-based applications.
- org.apache.hadoop.io.Text: This is responsible for representing textual data in a format suitable for storage, transmission, and processing within the Hadoop ecosystem, playing a vital role in the efficient handling of text-based data in Hadoop-based applications.
- org.apache.hadoop.mapreduce.Job: This serves as the primary interface for configuring, submitting, monitoring, and controlling MapReduce jobs in Apache Hadoop, providing developers with a comprehensive API for managing the execution of distributed data processing tasks.
- org.apache.hadoop.mapreduce.Mapper: This class in Apache Hadoop is responsible for the mapping phase of the MapReduce computation.
- org.apache.hadoop.mapreduce.Reducer: This class in Apache Hadoop is responsible for the reducing phase of the MapReduce computation
- org.apache.hadoop.mapreduce.lib.input.FileInputFormat: This class in Apache Hadoop is responsible for defining the input format for reading data from files in a Hadoop MapReduce job.
- org.apache.hadoop.mapreduce.lib.output.FileOutputFormat: This class in Apache Hadoop is responsible for defining the output format for writing data to files in a Hadoop MapReduce job.

So, to summarize, there are three classes:

- ✓ WordCount: Configures and runs the MapReduce job.
- ✓ TokenizerMapper: Tokenizes input text and emits word count key-value pairs.
- ✓ IntSumReducer: Reduces word counts by summing them up and emitting final counts.

b) Describe the different services listed when executing 'jps'. What are their functions in Hadoop and HDFS?

Ans: When we execute the jps command in a Hadoop environment, we typically see several Java processes (JVMs) running, each representing a different component or service of the Hadoop ecosystem. Here are some common services we might see and their functions in Hadoop and HDFS:

1. **NameNode:** The NameNode is a critical component of the Hadoop Distributed File System (HDFS). It manages the file system namespace and metadata, including the directory tree, file permissions, and block locations.
  - a. **Role in Hadoop/HDFS:** NameNode stores metadata about the data blocks and their locations. It serves clients' requests for file system operations and coordinates data access and replication across DataNodes.
2. **DataNode:** DataNodes store actual data blocks of files in HDFS. They perform read and write operations on the data blocks and replicate blocks for fault tolerance.
  - a. **Role in Hadoop/HDFS:** DataNodes are responsible for storing and serving data blocks to clients. They communicate with the NameNode to report block information and perform block replication as needed.
3. **ResourceManager:** The ResourceManager is the central authority for resource management and job scheduling in a Hadoop cluster. It allocates resources (memory, CPU) among competing applications and schedules tasks on individual nodes.
  - a. **Role in Hadoop/HDFS:** ResourceManager manages the allocation of cluster resources to different applications (MapReduce, YARN-based applications). It maintains information about available cluster resources and queues, and it tracks application progress.
4. **NodeManager:** NodeManagers run on individual nodes in the cluster and manage resources on that node. They monitor resource usage (CPU, memory) and report back to the ResourceManager.
  - a. **Role in Hadoop/HDFS:** NodeManagers launch and monitor containers, which are execution environments for running tasks (MapReduce, Spark, etc.). They communicate with the ResourceManager to receive instructions and report node status.
5. **SecondaryNameNode:** The SecondaryNameNode performs periodic checkpoints of the HDFS metadata from the NameNode. It merges the edits log with the current state of the file system namespace to create a new, updated snapshot of the metadata.
  - a. **Role in Hadoop/HDFS:** While the NameNode stores the entire file system namespace and metadata in memory, the SecondaryNameNode helps in reducing the recovery time in case of NameNode failure by maintaining a backup of metadata.
6. **JobHistoryServer:** The JobHistoryServer stores historical information about completed MapReduce jobs, including job configuration, status, and logs.
  - a. **Role in Hadoop/HDFS:** JobHistoryServer allows users and administrators to view and analyze the history of MapReduce job executions. It provides insights into job performance, resource utilization, and debugging information.

These services work together to provide fault-tolerant distributed storage (HDFS) and scalable distributed processing (MapReduce, YARN) capabilities in the Hadoop ecosystem. They ensure efficient data storage, processing, and resource management in large-scale distributed computing environments.

### 1.4.1: Change the WordCount.java as per instructions in below.

ubuntu@rajibdatta-a2: ~

```
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()) {
            String token = itr.nextToken();
            // Extract the first letter of the token and convert to lowercase
            if (token.length() > 0 && Character.isLetter(token.charAt(0))) {
                char firstLetter = Character.toLowerCase(token.charAt(0));
                word.set(String.valueOf(firstLetter));
                context.write(word, one);
            }
        }
    }
}

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);
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
```

Got below output on output file as copied.

```
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hdfs dfs -ls RD_output1.4.1
Found 2 items
-rw-r--r--  1 ubuntu supergroup          0 2024-02-07 15:24 RD_output1.4.1/_SUCCESS
-rw-r--r--  1 ubuntu supergroup       180 2024-02-07 15:24 RD_output1.4.1/part-r-00000
ubuntu@rajibdatta-a2:~$
```

Output that insides

```
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hdfs dfs -cat RD_output1.4.1/part-r-00000
a      12893
b      4862
c      4308
d      2734
e      3491
f      4126
g      1769
h      3099
i      8690
j      342
k      465
l      2699
m      4820
n      2067
o      9720
p      3878
q      173
r      2503
s      7404
t      18613
u      1122
v      935
w      5897
x      32
y      484
z      55
æ      6
ubuntu@rajibdatta-a2:~$
```

Letter Counts:

```
a: 12893
b: 4862
c: 4308
d: 2734
e: 3491
f: 4126
g: 1769
h: 3099
i: 8690
j: 342
k: 465
l: 2699
m: 4820
n: 2067
o: 9720
p: 3878
q: 173
r: 2503
s: 7404
t: 18613
u: 1122
v: 935
w: 5897
x: 32
y: 484
z: 55
æ: 6
```

1.4.2:

```

get: /hadoop-3.3.6/bin/hdfs/RD_output1.4.1/part-r-00000* : No such file or directory
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hadoop fs -get /user/ubuntu/RD_output1.4.1/part-r-00000* /home/ubuntu/input
ubuntu@rajibdatta-a2:~$

rdatta3822@DattaEdu:~$ scp -i .ssh/RD_SSH.pem ubuntu@130.238.29.191:/home/ubuntu/input/part-r-00000* /home/rdatta3822/Da
ta_Engineering-1/
part-r-00000                                100% 180   18.4KB/s   00:00
rdatta3822@DattaEdu:~$

```

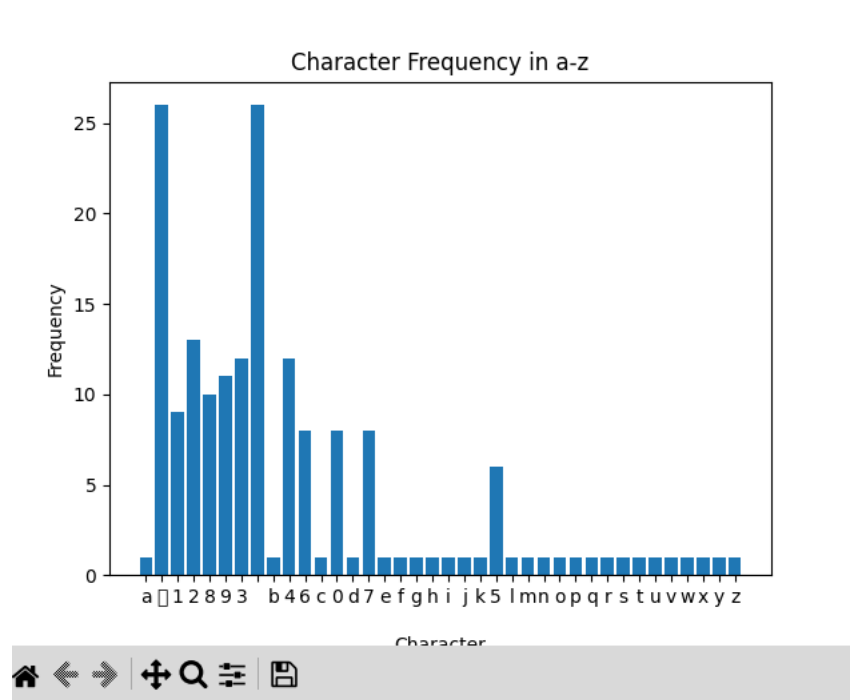
Local copy:

```

rdatta3822@DattaEdu:~/Data_Engineering-1$ ls -la
total 24
drwxr-xr-x  4 rdatta3822 rdatta3822 4096 Feb 10 10:31 .
drwxr-xr-x 15 rdatta3822 rdatta3822 4096 Feb 10 10:01 ..
drwxr-xr-x  8 rdatta3822 rdatta3822 4096 Jan 24 00:06 .git
drwxr-xr-x  3 rdatta3822 rdatta3822 4096 Jan 24 00:03 A1
-rw-r--r--  1 rdatta3822 rdatta3822   6 Jan 23 23:47 Abc.text
-rw-r--r--  1 rdatta3822 rdatta3822  180 Feb 10 10:31 part-r-00000
rdatta3822@DattaEdu:~/Data_Engineering-1$

```

Plotting:



**Modified WordCount.java(First Letter) is below**

```

import java.io.IOException;
import java.util.StringTokenizer;
import java.util.TreeMap;
import java.util.Map;
import java.util.regex.Pattern;
import java.util.ArrayList;
import java.io.BufferedReader;

```



```

import java.io.FileReader;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    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()) {
                String token = itr.nextToken();
                // Extract the first letter of the token and convert to lowercase
                if (token.length() > 0 && Character.isLetter(token.charAt(0))) {
                    char firstLetter = Character.toLowerCase(token.charAt(0));
                    if (firstLetter >= 'a' && firstLetter <= 'z') { // Check if the first letter is within 'a' to 'z'
                        word.set(String.valueOf(firstLetter));
                        context.write(word, one);
                    }
                }
            }
        }
    }
}

```

```

    }
}
}
}
}

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);
    }
}

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

```

```

// System.exit(job.waitForCompletion(true) ? 0 : 1);
if(job.waitForCompletion(true)) {
    // Plotting the results
    TreeMap<String, Integer> counts = getLetterCounts(job);
    plotCounts(counts);
} else {
    System.exit(1);
}
}

private static TreeMap<String, Integer> getLetterCounts(Job job) throws IOException {
    TreeMap<String, Integer> counts = new TreeMap<>();
    String outputPath = job.getConfiguration().get("mapreduce.output.fileoutputformat.outputdir");
    Path resultFile = new Path(outputPath + "/part-r-00000");
    try (BufferedReader br = new BufferedReader(new FileReader(resultFile.toString()))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] parts = line.split("\\s+");
            counts.put(parts[0], Integer.parseInt(parts[1]));
        }
    }
    return counts;
}

private static void plotCounts(TreeMap<String, Integer> counts) {
    // code for plotting the counts using JFreeChart or any other plotting library
    // Here is just a simple example using print statements
    System.out.println("Letter Counts:");
    for (Map.Entry<String, Integer> entry : counts.entrySet()) {
        System.out.println(entry.getKey() + ": " + entry.getValue());
    }
}

```

```
}
```

Plotting Python code is below.

```
import glob
```

```
import matplotlib.pyplot as plt
```

```
#define filename with file path
```

```
file_pattern='/home/rdata3822/Data_Engineering-1/part-r-00000*'
```

```
# Count occurrences of each character
```

```
char_count = {}
```

```
#loop for matching the pattern
```

```
for filename in glob.glob(file_pattern):
```

```
    #Read the contents of the file
```

```
    with open(filename, 'r') as file:
```

```
        data = file.read()
```

```
for char in data:
```

```
    #if char.isalpha(): # Check if the character is alphabetic
```

```
    # char = char.lower(); # Convert the character to lowercase
```

```
    if char in char_count:
```

```
        char_count[char] +=1;
```

```
    else:
```

```
        char_count[char] = 1;
```

```
#char_count[char] = char_count.get(char, 0) + 1;
```

```
# Plotting
```

```
plt.bar(char_count.keys(), char_count.values())
```

```
plt.xlabel('Character')
plt.ylabel('Frequency')
plt.title('Character(Alphabetic) Frequency')
plt.show()
```

### **1.5.1 NoSQL and MongoDB**

**Ans:**

Tweets are the basic atomic building block of all things Twitter. Tweets are also known as “status updates.” The Tweet object has a long list of ‘root-level’ attributes, including fundamental attributes such as id, created\_at, and text. Tweet objects are also the ‘parent’ object to several child objects. Tweet child objects include user, entities, and extended\_entities. Tweets that are geo-tagged will have a place child object. Based on the Twitter documentation, JSON-formatted tweets would typically be classified as semi-structured data.

Here's why:

**Structured Data:** Structured data is highly organized and follows a strict schema. Each data element is well-defined, and the structure is uniform across all records. Examples of structured data include databases and spreadsheets. JSON-formatted tweets do have a defined structure with specific fields such as "text", "created\_at", "user", etc. However, the content within these fields can vary, and not all fields are guaranteed to be present in every tweet. Therefore, JSON-formatted tweets do not fit the definition of structured data perfectly.

**Semi-Structured Data:** Semi-structured data has some structure but does not conform to a rigid schema like structured data. It contains tags or markers that separate the elements and groups of data, but the structure may vary between records. JSON-formatted tweets fall under this category because while they have a defined structure with key-value pairs, the presence and content of some fields can vary between tweets. Additionally, tweets may include user-defined hashtags, mentions, URLs, and other metadata, adding further variability to the structure.

**Unstructured Data:** Unstructured data lacks a predefined data model or structure. It typically includes text-heavy content such as articles, emails, or social media posts without any explicit organization. While JSON-formatted tweets contain structured information, the actual text content of the tweets (e.g., the message itself) is unstructured. Therefore, while tweets are encoded in a structured format (JSON), the content within the tweets is considered unstructured data.

In summary, JSON-formatted tweets are best classified as semi-structured data due to their defined but flexible structure, which includes varying fields and metadata.

### **1.5.2:**

**Ans:** SQL (Structured Query Language) and NoSQL (Not Only SQL) databases are two different approaches to storing and managing data, each with its own set of advantages and disadvantages. Elaboration on the pros and cons of SQL and NoSQL solutions, respectively in below.

**SQL Pros:**

**Structured Data:** SQL databases are ideal for structured data with a well-defined schema. They enforce data integrity through predefined schemas, ensuring consistency and accuracy.

**ACID Compliance:** SQL databases typically adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties, providing strong consistency and reliability.

**Mature Ecosystem:** SQL databases have been around for decades, resulting in a mature ecosystem with robust tools, libraries, and support.

**Complex Queries:** SQL databases excel at complex queries involving joins, aggregations, and analytics. They provide powerful query languages for data manipulation and analysis.

**Transactions:** SQL databases support transactions, allowing multiple operations to be grouped together and executed atomically.

### **SQL Cons:**

**Scalability:** Traditional SQL databases may struggle with horizontal scalability, especially for large-scale distributed systems. Scaling vertically (adding more resources to a single server) can be costly and has limits.

**Schema Rigidity:** While the predefined schema ensures data consistency, it can also be a limitation, especially in scenarios where the data structure is dynamic or evolves frequently.

**Performance:** SQL databases may experience performance bottlenecks, especially with complex queries or high transaction volumes. Indexing and query optimization are critical for maintaining performance.

**Cost:** Commercial SQL databases can be expensive, both in terms of licensing fees and hardware requirements for scaling.

### **Examples of suitable data sets/scenarios for SQL databases:**

- ✓ **Financial Transactions:** SQL databases are well-suited for managing financial data, ensuring data integrity and consistency.
- ✓ **Enterprise Applications:** Traditional business applications such as Customer Relationship Management (CRM) systems, Enterprise Resource Planning (ERP) systems, and Human Resource Management (HRM) systems benefit from the structured nature of SQL databases.
- ✓ **Inventory Management:** SQL databases are suitable for managing inventory data in retail or manufacturing environments, where data consistency and transactional integrity are crucial.

### **NoSQL Solutions:**

#### **Pros:**

**Flexible Schema:** NoSQL databases offer schema flexibility, allowing for the storage of unstructured or semi-structured data. This flexibility accommodates evolving data models and rapid development cycles.

**Scalability:** NoSQL databases are designed for horizontal scalability, making them suitable for handling large volumes of data and distributed systems. They can scale out by adding more nodes to the cluster.

**High Performance:** NoSQL databases can handle high-throughput workloads and large-scale data processing efficiently. They are optimized for read and write operations, making them suitable for real-time applications.

**Cost-Effective:** Many NoSQL databases are open-source and free to use, reducing the total cost of ownership compared to commercial SQL databases.

**Specialized Use Cases:** NoSQL databases are well-suited for specific use cases such as document storage, key-value pairs, graph databases, and time-series data.

#### **Cons:**

**Consistency Trade-offs:** Some NoSQL databases sacrifice strong consistency for scalability and performance. This eventual consistency model may lead to data inconsistency in certain scenarios.

**Limited Query Support:** NoSQL databases may have limited query capabilities compared to SQL databases, especially for complex joins and aggregations.

**Less Mature Ecosystem:** NoSQL databases are relatively newer compared to SQL databases, resulting in a less mature ecosystem with fewer tools, libraries, and community support.

**Learning Curve:** Adopting NoSQL databases may require a learning curve for developers accustomed to SQL databases. The lack of a standardized query language and varying data models can be challenging.

### **Examples of suitable data sets/scenarios for NoSQL databases:**

- ✓ **Big Data Analytics:** NoSQL databases like Apache Cassandra or MongoDB are suitable for storing and analyzing large volumes of unstructured or semi-structured data in real time.
- ✓ **IoT (Internet of Things) Data:** NoSQL databases are ideal for handling time-series data generated by IoT devices, such as sensor readings, telemetry data, and event logs.
- ✓ **Content Management:** NoSQL databases are commonly used for content management systems, where flexibility in data modeling and schema design is essential for handling diverse content types.
- ✓ **Social Media Feeds:** NoSQL databases can efficiently handle social media feeds, where the data structure may vary between users and includes multimedia content such as images and videos.

In summary, the choice between SQL and NoSQL databases depends on factors such as data structure, scalability requirements, performance needs, and development preferences. Both types of databases have their strengths and weaknesses, and selecting the right solution involves evaluating these factors in the context of specific use cases and business requirements.

## **Part-2: Analyzing twitter data using Hadoop streaming and Python**

### **2.1:**

#### **Keep tweets Data in HDFS**

```
ubuntu@rajibdatta-a2:~$  
ubuntu@rajibdatta-a2:~$  
ubuntu@rajibdatta-a2:~$  
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hdfs dfs -ls tweets  
Found 1 items  
drwxr-xr-x  - ubuntu supergroup          0 2024-02-10 15:49 tweets/files  
ubuntu@rajibdatta-a2:~$  
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hdfs dfs -ls input  
Found 6 items  
-rw-r--r--  1 ubuntu supergroup    674684 2024-02-07 13:35 input/20417.txt.utf-8  
-rw-r--r--  1 ubuntu supergroup      786 2024-02-10 19:51 input/mapper.py  
-rw-r--r--  1 ubuntu supergroup    1087 2024-02-10 19:51 input/reducer.py  
-rw-r--r--  1 ubuntu supergroup 586300456 2024-02-10 19:52 input/tweets_0.json  
-rw-r--r--  1 ubuntu supergroup 573803563 2024-02-10 17:14 input/tweets_1.txt  
-rw-r--r--  1 ubuntu supergroup    1324 2024-02-10 17:12 input/tweets_mapper.py  
ubuntu@rajibdatta-a2:~$
```

## Run the Streamline code with below screenshot

**Code is:** `hadoop-3.3.6/bin/hadoop jar hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \`

`-input /user/ubuntu/input/tweets*.txt \`

`-output /user/ubuntu/RD_output2.1/ \`

`-mapper "python3 mapper.py" \`

`-reducer "python3 reducer.py" \`

`-file /home/ubuntu/input2/mapper.py \`

`-file /home/ubuntu/input2/reducer.py`

```
ubuntu@rajibdatta-a2: ~  
Map input records=367358  
Map output records=151718  
Map output bytes=1520212  
Map output materialized bytes=1823678  
Input split bytes=520  
Combine input records=0  
Combine output records=0  
Reduce input groups=7  
Reduce shuffle bytes=1823678  
Reduce input records=151718  
Reduce output records=7  
Spilled Records=303436  
Shuffled Maps =5  
Failed Shuffles=0  
Merged Map outputs=5  
GC time elapsed (ms)=20  
Total committed heap usage (bytes)=1056964608  
Shuffle Errors  
BAD_ID=0  
CONNECTION=0  
IO_ERROR=0  
WRONG_LENGTH=0  
WRONG_MAP=0  
WRONG_REDUCE=0  
File Input Format Counters  
Bytes Read=573824043  
File Output Format Counters  
Bytes Written=98  
2024-02-11 21:51:42,761 INFO streaming.StreamJob: Output directory: /user/ubuntu/RD_output2.1/
```

## Python Mapper Code is: (mapper.py)

```
#!/usr/bin/env python3
```

```
import sys
```

```
import json
```

```
import re
```



```

# Compile regular expression patterns for the pronouns
pronoun_patterns = [
    re.compile(r'\bhan\b', re.IGNORECASE),
    re.compile(r'\bhon\b', re.IGNORECASE),
    re.compile(r'\bden\b', re.IGNORECASE),
    re.compile(r'\bdet\b', re.IGNORECASE),
    re.compile(r'\bdenna\b', re.IGNORECASE),
    re.compile(r'\bdenne\b', re.IGNORECASE),
    re.compile(r'\bhen\b', re.IGNORECASE)
]

# Read input from STDIN (standard input)
for line in sys.stdin:
    try:
        # Parse the JSON tweet
        tweet = json.loads(line)

        # Check if the tweet is a retweet, if yes, skip it
        if 'retweeted_status' in tweet:
            continue

        # Extract text from the tweet
        text = tweet.get('text', "").lower()

        # Check if any pronoun appears in the tweet and emit key-value pair
        for pattern in pronoun_patterns:
            if pattern.search(text):
                print(f'{pattern.pattern}\t1")

    except Exception as e:
        # Print any errors to STDERR (standard error)
        print("Error:", e, file=sys.stderr)

```

Python Code for Reducer: (reducer.py)

ubuntu@rajibdatta-a2:~/input2\$ cat reducer.py

```
#!/usr/bin/env python3
```

```
import sys
```

```
# Initialize variables to keep track of the current pronoun and count
```

```
current_pronoun = None
```

```
current_count = 0
```

```
# Process input from STDIN (standard input)
```

```
for line in sys.stdin:
```

```
    # Split the input into key and value
```

```
    pronoun, count_str = line.strip().split('\t', 1)
```

```
    # Convert count from string to integer
```

```
    count = int(count_str)
```

```
    # If it's the first iteration or we're still on the same pronoun
```

```
    if current_pronoun == pronoun:
```

```
        current_count += count
```

```
    else:
```

```
        # If it's a new pronoun, output the previous pronoun's count
```

```
        if current_pronoun:
```

```
            print(f"{current_pronoun}\t{current_count}")
```

```
    # Update variables for the new pronoun
```

```
    current_pronoun = pronoun
```

```
    current_count = count
```

```
# Output the last pronoun's count
```

```
print(f"{current_pronoun}\t{current_count}")
```

```
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hadoop fs -ls RD_output2.1
Found 2 items
-rw-r--r--    1 ubuntu supergroup          0 2024-02-11 21:51 RD_output2.1/_SUCCESS
-rw-r--r--    1 ubuntu supergroup      98 2024-02-11 21:51 RD_output2.1/part-00000
ubuntu@rajibdatta-a2:~$
```

```
ubuntu@rajibdatta-a2:~$ sudo su
root@rajibdatta-a2:~# cd /mnt/rd_output2.1
root@rajibdatta-a2:/mnt/rd_output2.1# ls -l
total 16
-rw-r--r-- 1 ubuntu supergroup 58 2024-02-11 21:51 RD_output2.1/part-00000
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hadoop fs -cat RD_output2.1/part-00000
\bden\b 71943
\bdenne\b 1262
\bdenne\b 254
\bdet\b 25889
\bhan\b 34094
\bhen\b 1957
\bhon\b 16319
ubuntu@rajibdatta-a2:~$
```

## ubuntu@rajibdatta-a2: ~

```

FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=2489882710
HDFS: Number of bytes written=96
HDFS: Number of read operations=63
HDFS: Number of large read operations=0
HDFS: Number of write operations=8
HDFS: Number of bytes read erasure-coded=0
Map-Reduce Framework
  Map input records=367358
  Map output records=122652
  Map output bytes=1227088
  Map output materialized bytes=1472422
  Input split bytes=520
  Combine input records=0
  Combine output records=0
  Reduce input groups=7
  Reduce shuffle bytes=1472422
  Reduce input records=122652
  Reduce output records=7
  Spilled Records=245304
  Shuffled Maps =5
  Failed Shuffles=0
  Merged Map outputs=5
  GC time elapsed (ms)=21
  Total committed heap usage (bytes)=1088421888
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=573824043
File Output Format Counters
  Bytes Written=96
2024-02-11 22:38:52,911 INFO streaming.StreamJob: Output directory: /user/ubuntu/RD_output2.2/
ubuntu@rajiibdatta-a2:~$

```

Inside of output folder:

```
Bytes Written: 96
2024-02-11 22:38:52,911 INFO streaming.StreamJob: Output directory: /user/ubuntu/RD_output2.2/
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hadoop fs -ls RD_output2.2
Found 2 items
-rw-r--r--  1 ubuntu supergroup          0 2024-02-11 22:38 RD_output2.2/_SUCCESS
-rw-r--r--  1 ubuntu supergroup    96 2024-02-11 22:38 RD_output2.2/part-00000
ubuntu@rajibdatta-a2:~$
```

RD\_output2.2/part-00000 Data: (MapReduce by MangoDB)

```
ubuntu@rajibdatta-a2:~$ hadoop-3.3.6/bin/hadoop fs -cat RD_output2.2/part-00000
\bden\b 68693
\bdenna\b      107
\bdenne\b      177
\bdet\b 2447
\bhan\b 34094
\bhen\b 1302
\bhon\b 15832
ubuntu@rajibdatta-a2:~$
```

MongoDB running status:

```
ubuntu@rajibdatta-a2:~$ sudo systemctl status mongod
● mongod.service - MongoDB Database Server
   Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2024-02-11 22:14:19 UTC; 28min ago
     Docs: https://docs.mongodb.org/manual
    Main PID: 104641 (mongod)
      Memory: 546.7M
    CGroup: /system.slice/mongod.service
            └─104641 /usr/bin/mongod --config /etc/mongod.conf

Feb 11 22:14:19 rajibdatta-a2 systemd[1]: Started MongoDB Database Server.
Feb 11 22:14:19 rajibdatta-a2 mongod[104641]: {"t":{"$date":"2024-02-11T22:14:19.153Z"},"s":"I",  "c":"CONTROL",  "id":7484500, "ctx":"main","msg":"Environment variable MO
lines 1-11/11 (END)
```

Mango Mapper file code : (mapper.py)

rdatta3822@DattaEdu:~/Data\_Engineering-1\$ cat mapper.py

```
#!/usr/bin/env python3
```

```
import sys
```

```
import json
```

```
import re
```

```
from pymongo import MongoClient
```

```
# Connect to MongoDB
```

```
client = MongoClient('localhost', 27017)
```

```

db = client['twitter'] # Connect to MongoDB database
collection = db['tweets'] # Connect to MongoDB collection

# Compile regular expression patterns for the pronouns
pronoun_patterns = [
    re.compile(r'\bhan\b', re.IGNORECASE),
    re.compile(r'\bhon\b', re.IGNORECASE),
    re.compile(r'\bden\b', re.IGNORECASE),
    re.compile(r'\bdet\b', re.IGNORECASE),
    re.compile(r'\bdenna\b', re.IGNORECASE),
    re.compile(r'\bdenne\b', re.IGNORECASE),
    re.compile(r'\bhen\b', re.IGNORECASE)
]

# Read input from STDIN (standard input)
for line in sys.stdin:
    try:
        # Parse the JSON tweet
        tweet = json.loads(line)

        # Check if the tweet is a retweet, if yes, skip it
        if 'retweeted_status' in tweet:
            continue

        # Extract text from the tweet
        text = tweet.get('text', '').lower()

        # Check if any pronoun appears in the tweet and emit key-value pair
        for pattern in pronoun_patterns:
            if pattern.search(text):
                # Insert the tweet into MongoDB
                collection.insert_one(tweet)

```

```
print(f' {pattern.pattern}\t1')
```

```
except Exception as e:
```

```
    # Print any errors to STDERR (standard error)
```

```
    print("Error:", e, file=sys.stderr)
```

```
Mango Reducer file code: reducer.py
```

```
rdatta3822@DattaEdu:~/Data_Engineering-1$ cat reducer.py
```

```
#!/usr/bin/env python3
```

```
import sys
```

```
from pymongo import MongoClient
```

```
# Connect to MongoDB
```

```
client = MongoClient('localhost', 27017)
```

```
db = client['twitter'] # Connect to MongoDB database
```

```
collection = db['tweets'] # Connect to MongoDB collection
```

```
# Initialize variables to keep track of the current pronoun and count
```

```
current_pronoun = None
```

```
current_count = 0
```

```
# Process input from STDIN (standard input)
```

```
for line in sys.stdin:
```

```
    # Split the input into key and value
```

```
    pronoun, count_str = line.strip().split('\t', 1)
```

```
# Convert count from string to integer
```

```
count = int(count_str)
```

```
# If it's the first iteration or we're still on the same pronoun
```

```
if current_pronoun == pronoun:
```

```
    current_count += count
```

else:

```
# If it's a new pronoun, output the previous pronoun's count
```

```
if current_pronoun:
```

```
    print(f'{current_pronoun}\t{current_count}')
```

```
# Update variables for the new pronoun
```

```
current_pronoun = pronoun
```

```
current_count = count
```

```
# Output the last pronoun's count
```

```
if current_pronoun:
```

```
    print(f'{current_pronoun}\t{current_count}')
```

**Part: 2 : Question and Answer:** Motivate your chosen implementation and how you did it. What are some pros and cons of the MongoDB solution compared to the implementation you did in Task 2.1?

Ans: Implementing Mapper and Reducer functions in Hadoop to analyze tweets for Swedish language pronouns can be an exciting and rewarding project. Here are some motivating factors:

- **Understanding Big Data Processing:** Working with Hadoop allows us to gain hands-on experience with big data processing frameworks. It's a valuable skill in today's data-driven world, opening up career opportunities in data engineering and analytics.
- **Real-world Application:** Analyzing tweets for Swedish language pronouns can provide valuable insights into language usage patterns, sentiment analysis, or even cultural trends within the Swedish-speaking community.
- **Language Processing Skills:** Developing Mapper and Reducer functions requires a good understanding of text processing techniques, which are fundamental in natural language processing (NLP) and computational linguistics.
- **Contribution to Research:** In project could contribute to linguistic research by providing data on the usage of pronouns in Swedish tweets. This data could be valuable for linguists studying language evolution, sociolinguistics, or language teaching methodologies.
- **Personal Development:** Building a Hadoop project from scratch is a challenging task that requires problem-solving skills, attention to detail, and persistence. Successfully completing this project will boost our confidence and enhance our programming abilities.
- **Community Engagement:** Sharing our findings and code with the open-source community can spark discussions, collaborations, and further developments in text analysis and big data processing.
- **Potential Impact:** Understanding language usage patterns can have broader societal implications, such as helping organizations tailor their communication strategies or assisting in disaster response efforts by analyzing social media data.
- **Learning Experience:** Even if we are new to Hadoop and text processing, this project offers an excellent opportunity to learn by doing. We will encounter challenges along the way, but overcoming them will deepen our understanding of the technologies involved.

Overall, implementing Mapper and Reducer functions for Hadoop to analyze tweets for Swedish language pronouns is a worthwhile endeavor that offers numerous learning and growth opportunities, both personally and professionally.

Part-2 answer: The pros and cons of using MongoDB compared to implementing Mapper and Reducer with Python code in the Hadoop framework are as follows:

### **Pros of MongoDB Solution:**

**Schema Flexibility:** MongoDB's schema-less design allows for easy storage and retrieval of unstructured data like tweets without the need for predefined schemas.

**Ease of Use:** MongoDB's document-oriented model is intuitive and easy to work with, particularly for developers familiar with JSON-like data structures.

**Scalability:** MongoDB is designed to scale horizontally, making it suitable for handling large volumes of tweet data.

**Querying Power:** MongoDB offers powerful querying capabilities, including text search and aggregation pipelines.

**Community Support:** MongoDB has a vibrant community of developers, providing resources, tutorials, and support forums. This community support can be valuable for troubleshooting issues, optimizing performance, and staying updated on best practices.

### **Cons of MongoDB Solution:**

**Performance:** While MongoDB performs well for many use cases, complex queries and aggregations on large datasets can be slower compared to traditional relational databases.

**Data Consistency:** MongoDB prioritizes availability and partition tolerance over strict consistency, which can lead to eventual consistency in distributed environments. Ensuring data consistency across nodes may require careful planning and implementation.

**Infrastructure Complexity:** Setting up and managing a MongoDB cluster involves configuring servers, maintaining replication, and handling failover.

### **Pros of Mapper and Reducer with Python Code in Hadoop Framework:**

**Scalability:** Hadoop is designed for distributed processing of large datasets, making it well-suited for handling massive volumes of tweet data. Mapper and Reducer functions can be scaled across multiple nodes for parallel processing, ensuring efficient analysis and computation.

**Performance Optimization:** Python offers extensive libraries and tools for optimizing data processing tasks. With careful design and implementation, Mapper and Reducer functions can be optimized for performance and efficiency, enabling fast and scalable tweet analysis.

**Integration Flexibility:** Python seamlessly integrates with Hadoop ecosystem tools and libraries, allowing for seamless integration with other data processing frameworks and technologies.

### **Cons of Mapper and Reducer with Python Code in Hadoop Framework:**

**Development Complexity:** Implementing Mapper and Reducer functions in Python for the Hadoop framework requires expertise in distributed computing, Hadoop MapReduce programming, and Python development.

**Operational Overhead:** Managing a Hadoop cluster and deploying Python code for MapReduce jobs involves infrastructure setup, configuration, and monitoring.



Resource Utilization: Hadoop clusters require dedicated resources for infrastructure setup, maintenance, and operation.

In summary, choosing between MongoDB and implementing Mapper and Reducer with Python code in the Hadoop framework depends on factors such as performance requirements, development expertise, scalability needs, and operational considerations. Each approach has its strengths and weaknesses, so it's essential to evaluate them based on the specific goals and constraints of our tweet analysis project.