T.C. AYDIN ADNAN MENDERES UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING

CSE401 Graduation Thesis 1, Fall 2022
Supervisor: Hüseyin ABACI

Data Analysis with Hadoop Framework Final Report

(Bachelor of Science Thesis)

01.09.2022 (11.06.2023)

By:
Berfin TEK, Student ID: 181805065

PLAGIARISM STATEMENT

This report was written by the group members and in our own words, except for quotations from published and unpublished sources which are clearly indicated and acknowledged as such. We are conscious that the incorporation of material from other works or a paraphrase of such material without acknowledgement will be treated as plagiarism according to the University Regulations. The source of any picture, graph, map or other illustration is also indicated, as is the source, published or unpublished, of any material not resulting from our own experimentation, observation or specimen collecting.

Project Group Members:

Name, Lastname	Student Number	Signature	Date
Berfin Tek	181805065		

Project Supervisors:

Name, Lastname	Department	Signature	Date

ACKNOWLEDGEMENTS

I would like to thank my advisor Hüseyin Abacı. My preferred person supported me in this matter, shared his knowledge and suggested resources. In this way, I gained more skills than I did in this chosen field. I also want to thank my friends who helped me with my technical problems. My friends solved all my problems with me.

KEYWORDS

Big data: Big data is defined as collections of datasets whose volume, velocity or variety is so large that it is difficult to store, manage, process and analyse the data using traditional databases and data processing tools.

Hadoop framework: Hadoop is a framework that allows distributed processing of large datasets across computer clusters using simple programming models. [1]

Analytics: Analytics is a broad term that encompasses the processes, technologies, frameworks and algorithms to extract meaningful insights from data.

Pig: Apache Pig is a platform for analysing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. [2]

Spark: Apache Spark is an open-source cluster computing framework for data analytics. [3]

Flume: Apache Flume is a tool for collecting and moving large amounts of data from various sources or local to the data store. [4]

Hive: Apache Hive is a platform used to develop SQL type scripts to do MapReduce operations. [5]

Sqoop: Apache Sqoop is used to import and export data to and from between HDFS and RDBMS. [6]

RabbitMQ: RabbitMQ is a messaging tool written in the java programming language. [7]

HBase: Apache HBase is an open source, distributed, column-oriented database built on Hadoop. [8]

ABSTRACT

The project required a substantial amount of research on the Internet, as well as written sources and research from industry professionals with specialized knowledge in their fields. Hadoop is an open source framework developed with the JAVA programming language that enables the processing and analysis of large datasets using some programming models. The main purpose of the project is to process and analyze weather data using hadoop framework and similar tools. The data files used for this project were retrieved from weather data site. The datasets were processed using the Python programming language and then exported to the hadoop file system and analyzed with data analysis tools such as Apache Pig, Apache Flume, Apache Hive, Apache Sqoop, Apache HBase, Apache Spark, RabbitMQ .

ÖZET

Büyük veri daha fazla çeşitlilik içeren ve hacmi sürekli artan verilerdir. Büyük veri analizi ise bu verilerin işlenmesini ve analizini gerçekleştirir. Bu proje büyük hava durumu verileri ile analiz işlemini amaçlamaktadır. Projede büyük veri araçları olarak Hadoop, Pig, Flume, Sqoop, RabbitMQ, Spark, Hbase ve Hive kullanıldı. Proje çıktısı analiz verileri içeren dosyalar, senaryolar ve konsol çıktıları olarak sunuldu. Proje çeşitli araştırmaları ve kaynak takibini gerektirdi.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF OUTPUTS

LIST OF ACRONYMS/ABBREVIATIONS

CMD: Command
HDFS: Hadoop Distributed File System
RAM: Random Access Memory
VM: Virtual Machine
RDD: Resilient Distributed Dataset
API: Application Programming Interface
SQL: Structured Query Language
RDBMS: Relational Database Management System

1 PROJECT DEFINITION

1.1 Introduction

1.1.1 Objectives

This project aims to make a case study of using big data stack for data analysis. The intended case study is based on the roadmap of “Big Data Science Analytics: A Hands-On Approach” [9] by Arshdeep Bahga and Vijay Madisetti. The collected weather datasets will be analyzed in a virtual environment using big data analysis tools. Before the analysis process, datasets will be preprocessed from the same virtual machine. Outputs of the project are csv, txt etc. Data formats such as and console outputs will be presented as screenshots.

1.1.2 Scope / Resources

The project is implemented on the virtual machine to save physical space, time management, hardware and software costs. The virtual machine is created by the Oracle VM VirtualBox virtualization program.

All the programs and systems used in this project are as follows.

- PC computer: Windows 64-bit operating system, x64-based processor
- Oracle VM VirtualBox virtualization software: VirtualBox-6.1.36
- Virtual machine: Linux operating system – Ubuntu (64 bit)
- Python development environment: PyCharm (version 2022.3.1 on Linux)
- Big data analytics tools: Ubuntu (22.04), Hadoop (3.3.5), PyCharm (2022.3.1), Flume (1.11.0), HBase (2.3.7), Sqoop (1.4.7), Hive (3.1.3), Pig (0.17.0), Spark (3.3.1), Scala (2.9)

1.1.3 Challenges

The biggest challenge in this project is installing big data tools and configuring configuration files. Installation and configuration procedures will be explained. During the installation and configuration processes of the tools, attention should be paid to version compatibility.

1.1.4 Mapping Analysis Flow

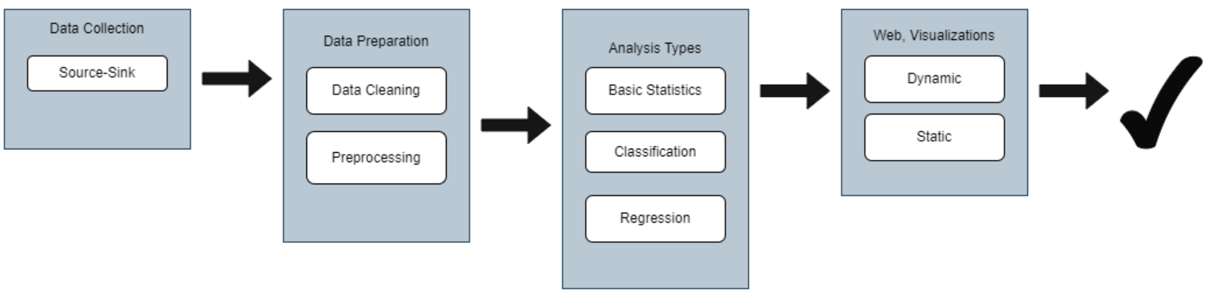


Figure 1.1 Analytics flow for data analysis applications

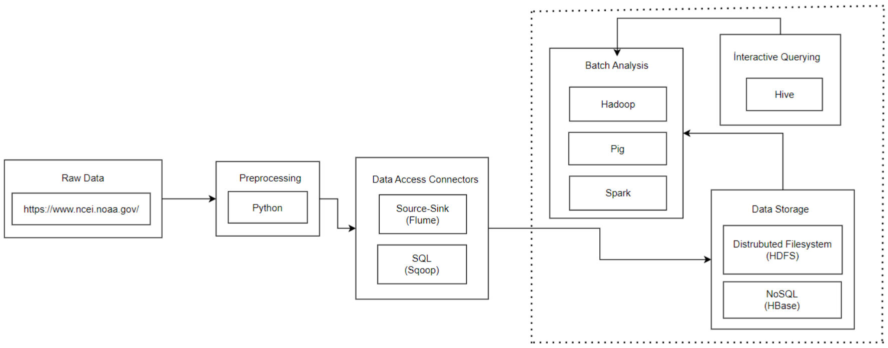


Figure 1.2 Big data stack for weather data analysis

I downloaded the raw data from the site in .txt format for analysis. I used Python to merge and clean up the data files. I used a source-sink connector like Flume and Sqoop to move the data to HDFS. I used Hadoop, Pig and Spark to analyze the data.

2 VIRTUAL MACHINE AND PROGRAMS

2.1 Introduction to VM

Virtual machines run on a single physical computer, providing the ability to use multiple operating systems. So, it acts like a computer within a computer. In general, a virtual machine provides a virtual system with the same underlying architecture as the real machine.

The most common operating system used for data science and big data is Linux. This is the reason why Ubuntu, a Linux-based operating system, is used in this project. The Linux kernel comes with tools and software support that make the data science experience better in many ways.

2.2 VM Setup

Step 1: Download iso file in Ubuntu web site [10].
Step 2: Open “VirtualBox” and click on the “New” button.
Step 3: Name the virtual machine and choose the path to install it. (Type: Linux, Version: Ubuntu (64-bit)). Click on the “Next” button
Step 4: Create RAM size to your Virtual Machine. Click “Next” button.
Step 5: Choose “Create a virtual hard disk now” for the machine to store files. Click “Create” button.
Step 6: Select “VDI”. Click “Next” button
Step 7: Either of the physical storage type can be selected. I choose “Dynamically allocated”. Click “Next” button.
Step 8: Select disk size and provide the destination folder to install. Click “Create” button.

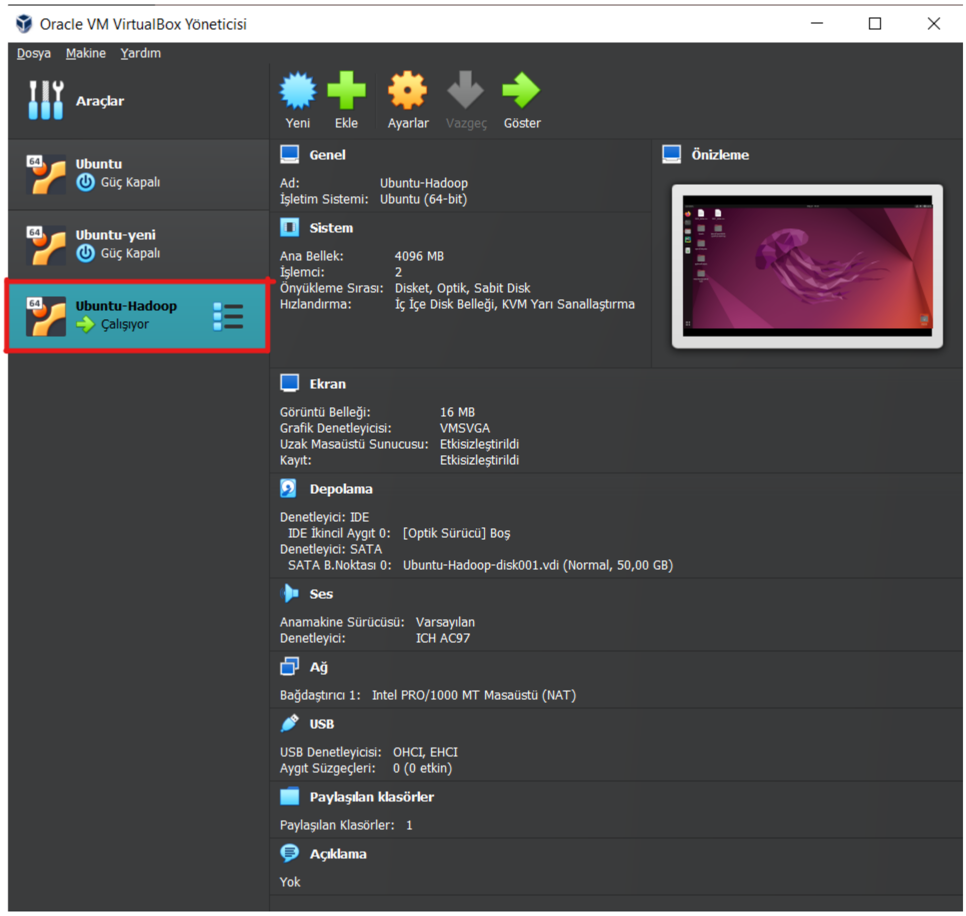


Figure 2.1 Ubuntu VM

Step 9: After that, start the Virtual Machine and begin installing Ubuntu.
Step 10: If the installation disk is not detected, select the iso file for Ubuntu. Click “Start” button.

2.3 Hadoop Setup

Step 1: Open terminal and install OpenJDK on Ubuntu

```
sudo apt update
sudo apt install openjdk-8-jdk -y
java -version
```

Step 2: Setup root user for Hadoop environment

```
sudo apt install openssh-server openssh-client -y
sudo adduser hadoop
su - hadoop
```

Step 3: Enable Passwordless SSH for Hadoop User

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys
```

```
ssh localhost
```

Step 4: Download and Install Hadoop on Ubuntu

```
wget
https://dlcdn.apache.org/hadoop/common/hadoop-3.3.5/hadoop-3.3
.5.tar.gz
tar xzf hadoop-3.3.5.tar.gz
sudo mv -T hadoop-3.3.5 hadoop
```

Step 5: Configure Hadoop Environment Variables.
Edit .bashrc shell configuration file using text editor (nano).

```
sudo nano .bashrc
```

Step 6: Add the following content to the end of the file

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre

# Hadoop
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export
HADOOP_OPTS="$HADOOP_OPTS-Djava.library.path=$HADOOP_HOME/lib/
native"
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

Save Press (Enter) and exit (ctrl+x). It is very important to apply the changes to the current working environment:

```
source .bashrc
```

Step 7: Edit Hadoop-env.sh file:

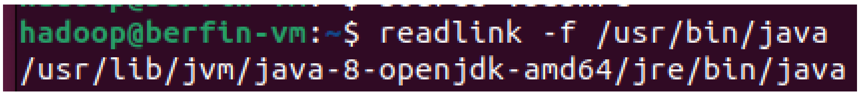
```
sudo nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

Add the following content to the end of the file

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Save Press (Enter) and exit (ctrl+x).

```
readlink -f /usr/bin/java
```



Output 2.1 All path of java file to final file

Step 8: Edit core-site.xml file:

```
sudo nano $HADOOP_HOME/etc/hadoop/core-site.xml
```

Add the following content to the file

```
<configuration>
<property>
     <name>hadoop.tmp.dir</name>
     <value>/usr/local/hadoop/tmpdata</value>
</property>
<property>
     <name>fs.default.name</name>
     <value>hdfs://127.0.0.1:9000</value>
</property>
</configuration>
```

Save Press (Enter) and exit (ctrl+x).

Step 9: Edit hdfs-site.xml file:

```
sudo nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

Add the following content to the file

```
<configuration>
<property>
     <name>dfs.data.dir</name>
     <value>/usr/local/hadoop/dfsdata/namenode</value>
</property>
<property>
     <name>dfs.data.dir</name>
     <value>/usr/local/hadoop/dfsdata/datanode</value>
</property>
<property>
     <name>dfs.replication</name>
     <value>1</value> </property>
</configuration>
```

Save Press (Enter) and exit (ctrl+x).

Step 10: Edit mapred-site.xml file:

```
sudo nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

Add the following content to the file:

```
<configuration>
<property>
     <name>mapreduce.framework.name</name>
     <value>yarn</value>
</property>
</configuration>
```

Save Press (Enter) and exit (ctrl+x).

Step 11: Edit yarn-site.xml file:

```
sudo nano $HADOOP_HOME/etc/hadoop/yarn-site.xml
```

Add the following content to the file:

```
<configuration>
<property>
     <name>yarn.nodemanager.aux-services</name>
     <value>mapreduce_shuffle</value>
</property>
<property>
     <name>yarn.nodemanager.aux-services.mapreduce.shuffle.cla
ss</name>
     <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
     <name>yarn.resourcemanager.hostname</name>
     <value>127.0.0.1</value>
</property>
<property>
     <name>yarn.acl.enable</name>
     <value>0</value>
</property>
<property>
     <name>yarn.nodemanager.env-whitelist</name>
     <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADO
OP_CONF_DIR,CLASSPATH_PERPEND_DISTCACHE,HADOOP_YARN_HOME,HADOO
P_MAPRED_HOME</value>
</property>
</configuration>
```

Save Press (Enter) and exit (ctrl+x).

2.4 PyCharm Setup

I used PyCharm (version 2022.3.1) for data preprocessing [11]. The libraries used in data preprocessing with PyCharm, and their versions are given in the "requiremenets.txt" file in the PyCharm project.

2.5 Apache Flume Setup

Step 1: Download Apache Flume.

```
wget
https://www.apache.org/dyn/closer.lua/flume/1.11.0/apache-flum
e-1.11.0-bin.tar.gz
tar xzf apache-flume-1.11.0-bin.tar.gz
sudo mv -T apache-flume-1.11.0-bin flume
```

Step 2: Configure Flume Environment Variables (bashrc)

```
sudo nano .bashrc
```

Add the following content to the file:

```
#Flume
export FLUME_HOME=/usr/local/flume
export PATH=$PATH:$FLUME_HOME/bin
```

Save Press (Enter) and exit (ctrl+x).

Step 3: Apply the changes to the current working environment:

```
source .bashrc
```

We are looking at the version to check if the installation is correct.

```
cd /usr/local/flume
flume-ng version
```

```
hadoop@berfin-vm:/usr/local/flume$  flume-ng version
Flume 1.11.0
Source code repository: https://git.apache.org/repos/asf/flume.git
Revision: 1a15927e594fd0d05a59d804b90a9c31ec93f5e1
Compiled by rgoers on Sun Oct 16 14:44:15 MST 2022
From source with checksum bbbca682177262aac3a89defde369a37
```

Output 2.2 Flume version

2.6 Apache HBase Setup

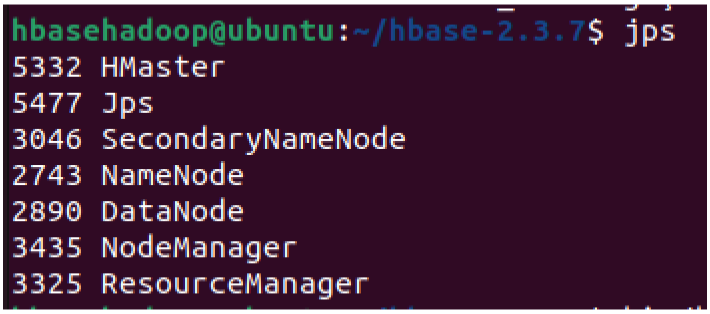
Step 1: Download Apache HBase. Extract the tar file and rename it to hbase.

```
tar -xvf hbase-2.3.7-bin.tar.gz
sudo mv -T hbase-2.3.7-bin hbase
```

Step 2: Enter the hbase file into and start hbase.sh file

```
bin/start-hbase.sh
```

Step 3: We need to make sure that HMaster is doing it by running the jps code.



Output 2.3 HBase jps

2.7 Apache Sqoop Setup

Step 1: Download Apache Sqoop, Extract the tar file and rename it to sqoop.

```
wget
https://archive.apache.org/dist/sqoop/1.4.7/sqoop-1.4.7.bin__h
adoop-2.6.0.tar.gz
tar -xvf sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz
sudo mv -T sqoop-1.4.7.bin__hadoop-2.6.0 sqoop
```

Step 2: Configure Sqoop Environment Variables (bashrc)

```
sudo nano .bashrc
```

Add the following content to the file:

```
#Sqoop
export SQOOP_HOME=/usr/lib/sqoop
export PATH=$PATH:$SQOOP_HOME/bin
```

Save Press (Enter) and exit (ctrl+x).

Step 3: Apply the changes to the current working environment:

```
source .bashrc
```

Step 4: We change the name of the sqoop-env-template.sh file in sqoop/conf to sqoop-env.sh and open the file. To configure Sqoop with Hadoop we write the lines given below.

```
export HADOOP_COMMON_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=/usr/local/hadoop
```

Step 5: We download and configure mysql-connector-java.

```
wget
http://ftp.ntu.edu.tw/MySQL/Downloads/Connector-J/mysql-connec
tor-java-8.0.26.tar.gz
tar -zxf mysql-connector-java-8.0.26.tar.gz
```

```
cd mysql-connector-java-5.1.30
mv mysql-connector-java-5.1.30-bin.jar /usr/lib/sqoop/lib
```

After all the processes are finished, we enter the sqoop/bin folder and verify the sqoop.

```
cd $SQOOP_HOME/bin
sqoop-version
```

NOTE: Some installations may receive the following error. This is because some .jar files in mysql file are missing. Downloading and manually adding the appropriate .jar file will solve the problem.

```
ERROR sqoop.Sqoop: Got exception running Sqoop:
java.lang.RuntimeException: Could not load db driver class:
com.mysql.jdbc.Driver java.lang.RuntimeException: Could not
load db driver class: com.mysql.jdbc.Driver
```

To solve the error I encountered, we download the file from https://downloads.mysql.com/archives/c-j/ and extract it from tar, then copy the jar file inside into sqoop/lib.

2.8 Apache Hive Setup

Step 1: Download Apache Hive, extract the tar file and rename it to hive.

```
wget
https://archive.apache.org/dist/hive/hive-3.1.3/apache-hive-3.
1.3-bin.tar.gz
tar -xvf apache-hive-3.1.3-bin.tar.gz
sudo mv -T apache-hive-3.1.3-bin hive
```

Step 2: Configure Sqoop Environment Variables (bashrc)

```
sudo nano .bashrc
```

Add the following content to the file:

```
#Hive
export HIVE_HOME=/usr/local/hive
Export PATH=$PATH:$HIVE_HOME/bin
Export HIVE_CONF_DIR=$HIVE_HOME/conf
Export CLASSPATH=$CLASSPATH:$HADOOP_HOME/lib/*:.
Export CLASSPATH=$CLASSPATH:$HIVE_HOME/lib/*:.
```

Save Press (Enter) and exit (ctrl+x).

Step 3: Apply the changes to the current working environment:

```
source .bashrc
```

Step 4: We rename the hive-env.sh.template file in hive/conf to hive-env.sh and open the file. To configure Hive with Hadoop we write the lines given below.

```
HADOOP_HOME=/usr/local/hadoop-install
```

Step 5: We change the name of the hive-default.xml.template file in hive/conf to hive-default.xml and open the file. To configure Hive, we write the lines given below.

```
<property>
<name>system:java.io.tmpdir</name>
<value>/tmp/hive/java</value>
</property>
<property>
<name>system:user.name</name>
<value>${user.name}</value>
<property>
```

Step 6: We are creating HDFS file for hive. And we grant read-write permissions.

```
hdfs dfs -mkdir /user/hive/warehouse
hdfs dfs -chmod g+w /user/hive/warehouse
```

Step 7: Finally, we must check its version to see if we have successfully installed Hive.

```
hive -version
```

Output 2.4 Hive version

2.9 Apache Pig Setup

Step 1: Download Apache Pig from site.

```
wget
https://downloads.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz
```

Step 2: Extract this tar file

```
tar -xvf pig-0.17.0.tar.gz
```

Step 3: Move extracted file to Hadoop user

```
sudo mv pig-0.17.0 usr/local/pig
```

Step 4: Change Pig's environment variable.

`sudo gedit ~/.bashrc` or `sudo nano ~/.bashrc`

Add the following content to the file:

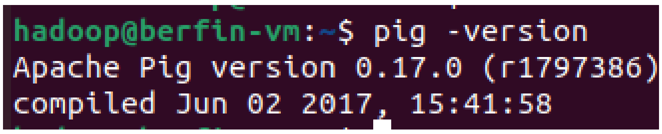
```
# Pig
export PIG_HOME=/usr/local/pig
export PIG_CLASSPATH=$HADOOP_HOME/conf
export PATH=$PATH:$PIG_HOME/bin
```

Save Press (Enter) and exit (ctrl+x).

Step 5: Apply the changes to the current working environment:

```
source ~/.bashrc
```

Step 6: Finally, we do version control to find out if it is installed correctly.



Output 2.5 Pig version

2.10 Apache Spark Setup

Step 1: Download Apache Spark from site.

```
wget
https://dlcdn.apache.org/spark/spark-3.3.1/spark-3.3.1-bin-had
oop3.tgz
```

Step 2: Extract this tar file

```
tar xvf spark-3.3.1-bin-hadoop3.tgz
```

Step 3: Move extracted file to home folder

```
sudo mv spark-3.3.1-bin-hadoop3/ /home/spark
```

Step 4: Change Spark's environment variable.

`sudo gedit ~/.bashrc` or `sudo nano ~/.bashrc`

Add the following content to the file:

```
export SPARK_HOME=/home/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
```

Save Press (Enter) and exit (ctrl+x).

Step 5: Apply the changes to the current working environment:

```
source ~/.bashrc
```

Step 6: Start a standalone master server

```
start-master.sh
```

2.11 RabbitMQ Setup

Step 1: Start the installation process with the prerequisites:

```
sudo apt install curl gnupg apt-transport-https
```

Step 2: Add repository signing keys for RabbiMQ main, ErLang and RabbitMQ PackageCloud repositories

```
curl -1sLf
"https://keys.openpgp.org/vks/v1/by-fingerprint/0A9AF2115F4687
BD29803A206B73A36E6026DFCA" | sudo gpg --dearmor | sudo tee
/usr/share/keyrings/com.rabbitmq.team.gpg > /dev/null

curl -1sLf
"https://keyserver.ubuntu.com/pks/lookup?op=get&search=0xf77f1
eda57ebb1cc" | sudo gpg --dearmor | sudo tee
/usr/share/keyrings/net.launchpad.ppa.rabbitmq.erlang.gpg >
/dev/null

curl -1sLf
"https://packagecloud.io/rabbitmq/rabbitmq-server/gpgkey" |
sudo gpg --dearmor | sudo tee
/usr/share/keyrings/io.packagecloud.rabbitmq.gpg > /dev/null
```

Step 3: Add repository for ErLang and RabbitMQ.

```
sudo nano /etc/apt/sources.list.d/rabbitmq.list
```

We need to add the following lines to this file.

```
deb
[signed-by=/usr/share/keyrings/net.launchpad.ppa.rabbitmq.erla
ng.gpg]
http://ppa.launchpad.net/rabbitmq/rabbitmq-erlang/ubuntu jammy
main deb-src
[signed-by=/usr/share/keyrings/net.launchpad.ppa.rabbitmq.erla
ng.gpg]
http://ppa.launchpad.net/rabbitmq/rabbitmq-erlang/ubuntu jammy
main deb
```

```
[signed-by=/usr/share/keyrings/io.packagecloud.rabbitmq.gpg]
https://packagecloud.io/rabbitmq/rabbitmq-server/ubuntu/ jammy
main deb-src
[signed-by=/usr/share/keyrings/io.packagecloud.rabbitmq.gpg]
https://packagecloud.io/rabbitmq/rabbitmq-server/ubuntu/ jammy
main
```

The “jammy” keyword represents this Ubuntu 22.04 distribution name, so if you're on a different version you should change it. Here are other distributions with their names:

DISTRIBUTION VERSION	DISTRIBUTION NAME
Ubuntu 18.04	bionic
Ubuntu 20.04	focal
Ubuntu 22.04	jammy

Figure 2.2 Table of distribution

Step 4: We need to update the repositories for the changes to take effect:

```
sudo apt update
```

Step 5: We have to install the erlang packages.

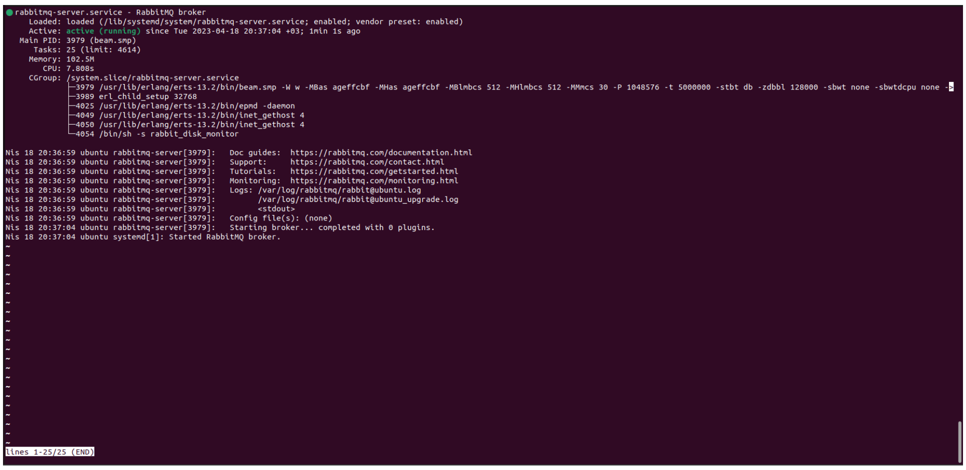
```
sudo apt install -y erlang-base \ erlang-asn1 erlang-crypto
erlang-eldap erlang-ftp erlang-inets \ erlang-mnesia
erlang-os-mon erlang-parsetools erlang-public-key \
erlang-runtime-tools erlang-snmp erlang-ssl \
erlang-syntax-tools erlang-tftp erlang-tools erlang-xmerl
```

Step 6: We can use the given command to install RabbitMQ.

```
sudo apt install rabbitmq-server -y --fix-missing
sudo rabbitmqctl version
```

Step 7: We must configure RabbitMQ in Ubuntu.

```
sudo systemctl status rabbitmq-server
```



Output 2.6 RabbitMQ status

NOTE: If it is not working as intended,

```
sudo systemctl start rabbitmq-server
```

To stop the RabbitMQ server:

```
sudo systemctl stop rabbitmq-server
```

Step 8: RabbitMQ server supports various protocols and one of the most used is AMQP running on port 5672. We must open this port in firewall to allow connection using this protocol.

```
sudo ufw allow 5672/tcp
```

Rabbitmq_management is a GUI interface accessible through the browser and allows you to manage RabbitMQ in the easiest way possible. We can enable it as follows.

```
sudo rabbitmq-plugins enable rabbitmq_management
```

NOTE: Instead of 127.0.1.1 you must enter your own server address. [12]

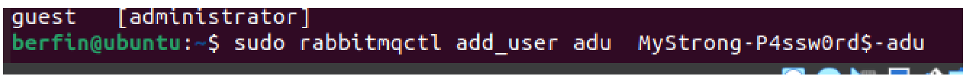
Figure 2.3 RabbitMQ interface

Step 9: A new user can be created to login, but I am a guest login.

Username: `guest`
Password: `guest`

But if the user is wanted to be created:

```
sudo rabbitmqctl add_user adu MyStrong-1p4ssw0rd$-adu
```



```
guest    [administrator]
berfin@ubuntu:~$ sudo rabbitmqctl add_user adu  MyStrong-P4ssw0rd$-adu
```

Output 2.7 RabbitMQ add user name and password

```
rabbitmqctl set_user_tags adu administrator
rabbitmqctl list_users
```

```
berfin@ubuntu:~$ sudo rabbitmqctl list_users
Listing users ...
user    tags
guest   [administrator]
adu     [administrator]
berfin@ubuntu:~$
```

Output 2.8 RabbitMQ users list

If to delete:

```
sudo rabbitmqctl delete_user adu
```

2.12 Install Jupyter Notebook

I used classic Jupyter Notebook, run this command:

```
pip install notebook
```

2.13 Apache Spark Library Setup

NOTE: These operations are for using spark only as a library. This will be enough for us within the scope of the project. If Spark will be used for a different purpose, it should be downloaded as a file and configuration operations should be done in the .bashrc file.

Step 1: First we need to download scala. For this, we open cmd and run the following command.

```
sudo apt-get install scala
```

```
hadoop@berfin-vm:~$ scala -version
Scala code runner version 2.11.12 -- Copyright 2002-2017, LAMP/EPFL
```

Output 2.9 Scala version

Step 2: In the same way, we download spark.

```
sudo apt-get install spark
```

3 INTRODUCTION THE PROJECT

3.1 Big Data

Big data is data that is more diverse, increasing in volume and can grow, and arriving faster. These datasets are larger than normal datasets, so traditional data processing software may not be able to process them.

3.1.1 Big Data Characteristics

Volume: Indicates the amount of data that can be collected. For example, Twitter feeds, clickstreams from web pages and mobile apps, or data from sensor-enabled equipment.

Velocity: It is the rate at which data is received. Some internet-enabled products work in real time, which requires fast data flow.

Variety: Big data includes data types in multiple formats. Traditional datasets are structured or properly saved in the database. Various data includes unstructured data such as text, audio, video and sensor data.

Veracity: It expresses how accurate and high quality the data is. Because data is collected from multiple sources, we need to check the accuracy of the data before using it.

Value: It expresses how useful the data is. We must extract the value of the data using appropriate analysis methods.

3.2 Data Collection and Preprocessing Steps

3.2.1 Data Collection

I did a study on using big data analysis tools for weather data analysis. I will use NCDC weather dataset [13] 2020 and 2021 data for the project. NCDC provides access to daily data from the US Climate Reference Network / US Regional Climate Reference Network (USCRN/USRCRN) via FTP.

3.2.2. Information About Datasets

Data files are kept in .txt format. Each .txt file is kept as log data for one state of the United States.

This information is taken from the readme.txt file. Each station file contains fixed-width formatted fields with a single day's data per line.

All daily data are calculated over the station's 24-hour LST day.

Name	Units
WBANNO	XXXXX
LST_DATE	YYYYMMDD
CRX_VN	XXXXXX
LONGITUDE	Decimal_degrees
LATITUDE	Decimal_degrees
T_DAILY_MAX	Celsius
T_DAILY_MIN	Celsius
T_DAILY_MEAN	Celsius
T_DAILY_AVG	Celsius
P_DAILY_CALC	mm
SOLARAD_DAILY	MJ/m^2
SUR_TEMP_DAILY_TYPE	X
SUR_TEMP_DAILY_MAX	Celsius

SUR_TEMP_DAILY_MIN	Celsius
SUR_TEMP_DAILY_AVG	Celsius
RH_DAILY_MAX	%
RH_DAILY_MIN	%
RH_DAILY_AVG	%
SOIL_MOISTURE_5_DAILY	m^3/m^3
SOIL_MOISTURE_10_DAILY	m^3/m^3
SOIL_MOISTURE_20_DAILY	m^3/m^3
SOIL_MOISTURE_50_DAILY	m^3/m^3
SOIL_MOISTURE_100_DAILY	m^3/m^3
SOIL_TEMP_5_DAILY	Celsius
SOIL_TEMP_10_DAILY	Celsius
SOIL_TEMP_20_DAILY	Celsius
SOIL_TEMP_50_DAILY	Celsius
SOIL_TEMP_100_DAILY	Celsius

WBANNO: The station WBAN number.

LST_DATE: The Local Standard Time (LST) date of the observation.

CRX_VN: The version number of the station datalogger program that was in effect at the time of the observation. Note: This field should be treated as text (i.e., string).

LONGITUDE: Station longitude, using WGS-84.

LATITUDE: Station latitude, using WGS-84.
T_DAILY_MAX: Maximum air temperature, in degrees C.

T_DAILY_MI: Minimum air temperature, in degrees C.

T_DAILY_: Mean air temperature, in degrees C, calculated using the typical historical approach: (T_DAILY_MAX + T_DAILY_MIN) / 2.

T_DAILY_AVG: Average air temperature, in degrees C.

P_DAILY_: Total amount of precipitation, in mm.

SOLARAD_DAILY: Total solar energy, in MJ/meter^2, calculated from the hourly average solar radiation rates and converted to energy by integrating time.

SUR_TEMP_DAILY_TYPE: Type of infrared surface temperature measurement. 'R' denotes raw measurements, 'C' denotes corrected measurements, and 'U' indicates unknown/missing.

SUR_TEMP_DAILY_MAX: Maximum infrared surface temperature, in degrees C.

SUR_TEMP_DAILY_MIN: Minimum infrared surface temperature, in degrees C.

SUR_TEMP_DAILY_AVG: Average infrared surface temperature, in degrees C.

RH_DAILY_MAX: Maximum relative humidity, in %.

RH_DAILY_MIN: Minimum relative humidity, in %.

RH_DAILY_AVG: Average relative humidity, in %.

SOIL_MOISTURE_5_DAILY: Average soil moisture, in fractional volumetric water content (m^3/m^3), at 5 cm below the surface.

SOIL_MOISTURE_10_DAILY: Average soil moisture, in fractional volumetric water content (m^3/m^3), at 10 cm below the surface.

SOIL_MOISTURE_20_DAILY: Average soil moisture, in fractional volumetric water content (m^3/m^3), at 20 cm below the surface.

SOIL_MOISTURE_50_DAILY: Average soil moisture, in fractional volumetric water content (m^3/m^3), at 50 cm below the surface.

SOIL_MOISTURE_100_DAILY: Average soil moisture, in fractional volumetric water content (m^3/m^3), at 100 cm below the surface.

SOIL_TEMP_5_DAILY: Average soil temperature, in degrees C, at 5 cm below the surface.

SOIL_TEMP_10_DAILY: Average soil temperature, in degrees C, at 10 cm below the surface.

SOIL_TEMP_20_DAILY: Average soil temperature, in degrees C, at 20 cm below the surface.

SOIL_TEMP_50_DAILY: Average soil temperature, in degrees C, at 50 cm below the surface.

SOIL_TEMP_100_DAILY: Average soil temperature, in degrees C, at 100 cm below the surface.

NOTE: Missing data are indicated by the lowest possible integer for a given column format, such as -9999.0 for 7-character fields with one decimal place or -99.000 for 7-character fields with three decimal places.

3.2.3. Data Preprocessing

In the data preprocessing process, I folder each .txt file I downloaded according to the year names and combine 155 .txt files with the "data_preparation" function I wrote in the folder and convert them into a single .txt and .csv file.

I then cleared the missing data in the new dataset created. Lost weather data is kept as -9999.0 and -99.000 as mentioned above. In the "data_preprocessing" function I wrote, it uses a for loop to synchronize this data with the data held the previous day. Since there was no big temperature difference between the two days, I preserved the integrity of the dataset with this process. At the same time, I divided the date variable into three different variables as “day, month, year" and deleted the columns in the data set that I did not want to use in my data analysis work and that had missing data in terms of terms. These (SUR_TEMP_DAILY_TYPE, SOIL_MOISTURE_5_DAILY, SOIL_MOISTURE_10_DAILY, SOIL_MOISTURE_20_DAILY, SOIL_MOISTURE_50_DAILY, SOIL_MOISTURE_100_DAILY, SOIL_TEMP_5_DAILY, SOIL_TEMP_10_DAILY, SOIL_TEMP_20_DAILY, SOIL_TEMP_50_DAILY, SOIL_TEMP_100_DAILY). I will use the final version of the data file for data analysis. Please see Appendix A-1 for Python codes.

4 BIG DATA ANALYSIS

Big data analytics is the process of processing large volumes of raw data and uncovering trends in data to help make data-driven decisions. Big data analytics software and tools can be used to make data-driven decisions.

4.1 Steps of Big Data Analytics Process

1. **Data is collected from various data sources:** Sensor data, internet streaming data, mobile app data, social media content data, etc. The data collected is usually unstructured or semi-structured data.
2. **Data is prepared and processed:** Data should be properly organized, structured, and partitioned. A comprehensive data processing requires higher performance than data analysis.
3. **Data Cleaning is done:** Data are cleaned and formatted for analysis using data quality software.
4. **Data is analyzed with analytical software:** Predictive analytics, machine learning algorithms, deep learning, statistical analysis, artificial intelligence and data visualization.

4.2 Hadoop

4.3 Big Data Storage (HDFS)

HDFS is a distributed file system that provides access to data in Hadoop clusters. Hadoop ships with HDFS. Data is distributed and replicated across several machines to ensure high availability.

4.3.1 Characteristics of HDFS

High Scalability: HDFS can scale hundreds of nodes in a single cluster.
Replication: HDFS always keeps a copy of the data on a different machine.
Fault tolerance: HDFS is highly fault tolerant such that if any machine fails, the other machine containing the copy of that data will automatically activate.
Distributed data storage: In HDFS, data is split into multiple blocks and stored in nodes. This is one of the most important features that makes Hadoop powerful.
Portable: HDFS can be easily moved from one platform to another.

4.3.2 HDFS Architecture

Namenode

Namenode manages the filesystem namespace within HDFS. It is software that can be run on commodity hardware. It is responsible for executing operations such as opening and closing files. Regulates the client's access to files. There is almost no data flow in Namenode.

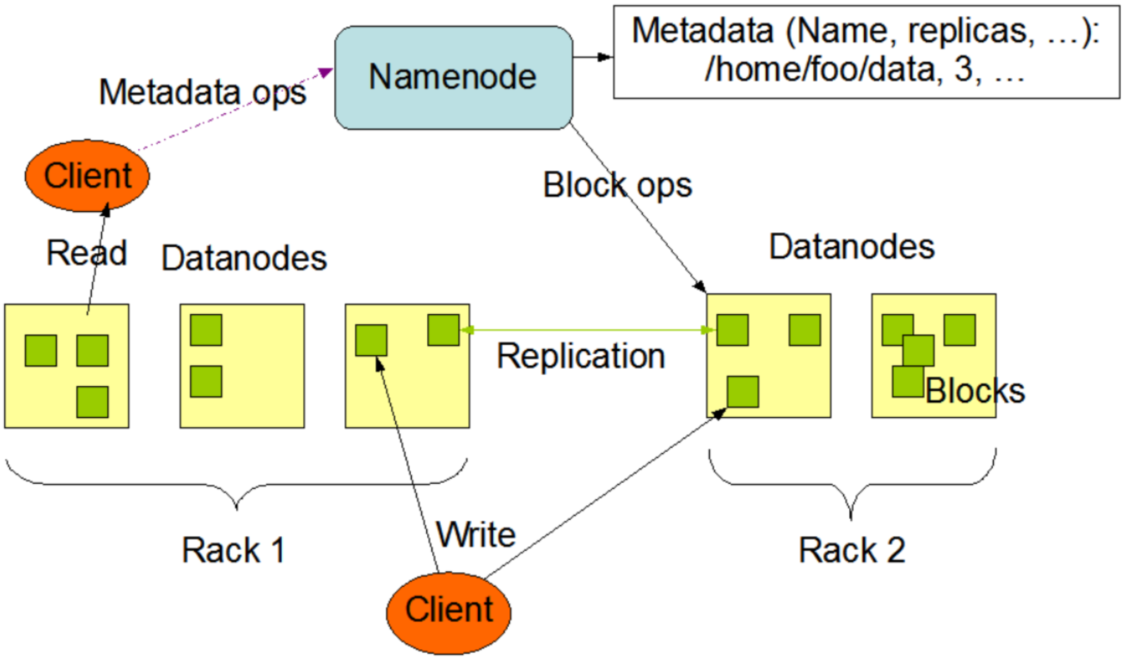


Figure 4.1 HDFS Architecture [14]

HDFS splits files into blocks and blocks are stored in Datanodes. Multiple copies are kept for each block. Namenode stores and manages its metadata about the file system in a file called fsimage and edits the log. All these servers communicate with each other via TCP-based protocols.

Secondary Namenode

This is not a backup node. Its main function is to obtain checkpoints of filesystem metadata located in the namenode. Namenode may not have enough resources to perform other operations. If the namenode fails, the entire HDFS file system is lost. To avoid this situation, the secondary namenode keeps a copy of the fsimage file and edits the edits log file.

Datanode

Datanodes store blocks of data. It performs read-write operations on the file system with requests from clients. It performs block creation, deletion and replication with the request from Namenode.

Data Blocks and Replication

Any number of replicas are created in blocks, Datanodes (by default this number is 3, but the replication factor can be changed from hdfs-site.xml). The default size of these blocks is

64MB (this size can be changed manually in hdfs-site.xml). This system is called fault tolerance.

4.3.3 HDFS Usage

To connect to Hadoop from cmd screen:

```
su – hadoop
```

After connecting to Hadoop start NameNode and DataNode, start yarn source.

```
start-dfs.sh
start-yarn.sh
```
or `start-all.sh`

HDFS’s interface:

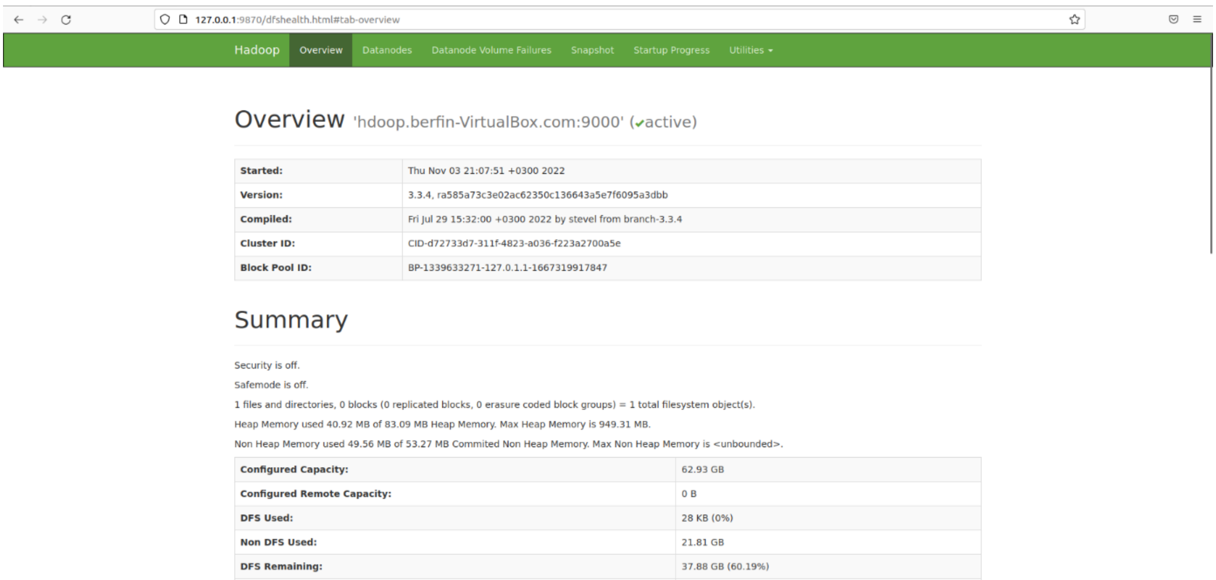


Figure 4.2 Hadoop browser

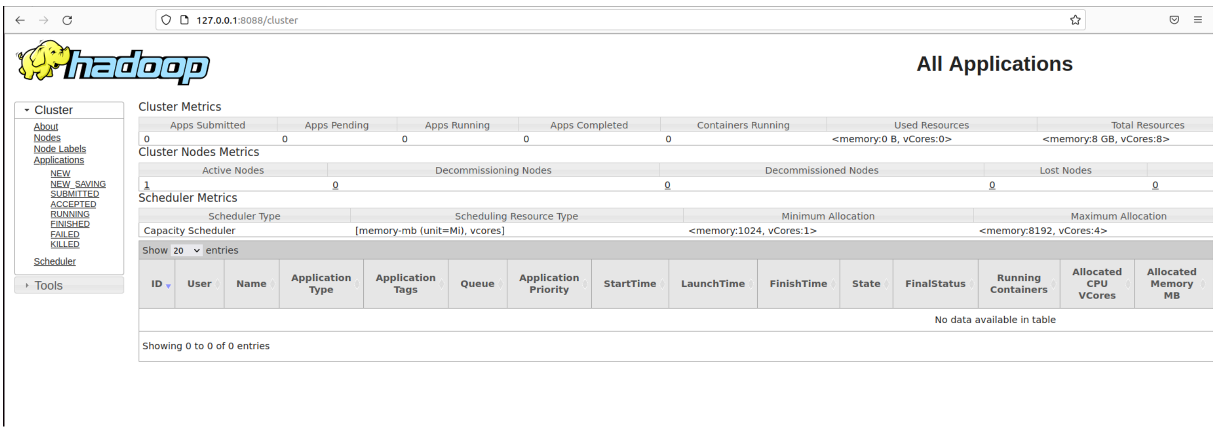


Figure 4.3 Hadoop cluster browser

Create ‘weatherDataInputs’ file on HDFS:

```
hdfs dfs -mkdir /weatherDataInputs
```

Remove ‘example’ file on HDFS:

```
hdfs dfs -rm -R /example
```

Sending the file sent in hadoop to hdfs namenode:

```
hdfs dfs -copyFromLocal
'/home/hadoop/Desktop/weatherDataAnalysis/OutputData/weather_d
ata.csv' /weatherDataInputs
```

NOTE: Care must be taken when writing file paths. To add a more reliable file path, the file to be processed should be dragged and dropped into the cmd window.

Finally, when we enter the “Utilities -> Browse the file system” section in the Hadoop interface, we see the data we have stored in HDFS.

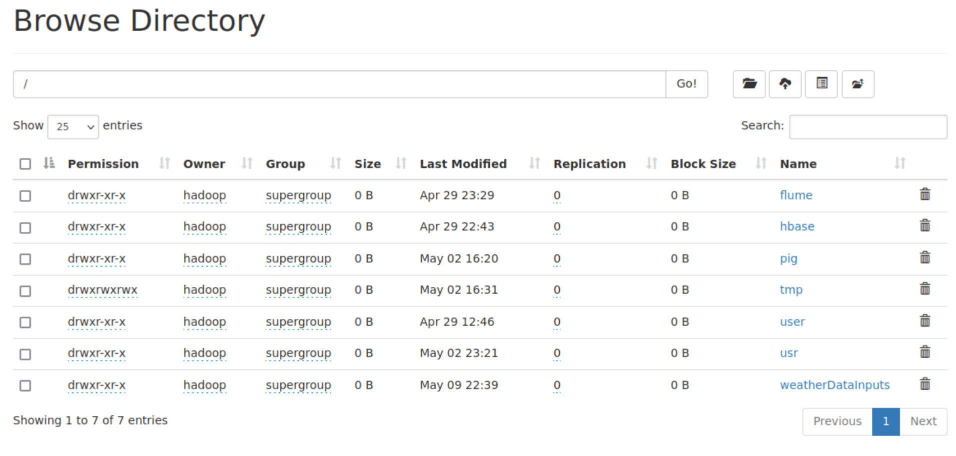


Figure 4.4 Hadoop browser file system

4.4 Data Access Connectors

4.4.1 Apache Flume

Apache Flume is a distributed, reliable and configurable system used to collect large amounts of data from different data sources and aggregate it into a central data repository.

Flume Architecture

• **Source:** It is the component that receives data from external sources. The data flow starts from a source.

• **Channel:** Data from the source is transmitted to the channel, which is a temporary storage. Each channel is connected to a pool. A data stream can consist of more than one channel.

• **Sink:** A component that receives data from a channel and transfers it to a data store or distributed file system. Each pool is connected to a canal.

Sources	Channels	Sinks
Avro Source Thrift Source Exec Source JMS Source Spooling Directory Source Twitter Source Kafka Source NetCat Source Sequence Generator Source Syslog Sources HTTP Source Custom Source	Memory Channel File Channel JDBC Channel Kafka Channel Spillable Memory Channel Custom Channel	HDFS Sink Avro Sink Thrift Sink File Roll Sink Logger Sink IRC Sink HBase Sink Custom Sink Hive Sink Logger Sink Null Sink HBase Sink Kafka Sink ElasticSearch Sink

Figure 4.5 Flume components

• **Agent:** It is a process that hosts resources, channels, and pools. Flume can have more than one agent.

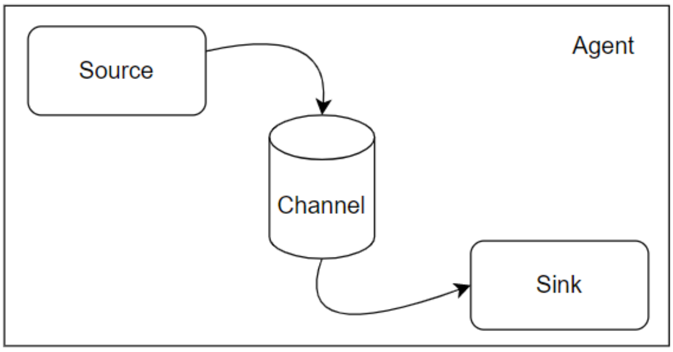


Figure 4.6 Flume agent architecture

Event: It is the main unit of data flow. An event consists of zero or more headers and a body.

Using HDFS Sink, we will upload our data file in Local to the HDFS file system using flume. We update and save the "flume-conf.properties" file according to the configurations given below. Or you can create a new file with a different name. Make sure the file extension is ". properties".

NOTE: You will find the "flume-conf.properties" file as "flume-conf.properties.templates" in flume/conf. You need to delete the template extension by renaming it.

```
agent2.sources = src1
agent2.channels = chan1
agent2.sinks = sink1
```

Exec Source: It is used to get data from standard output. When an agent with this resource is started, the command runs and continues to receive data as long as the process is running. This resource is the cat command, which writes data from a given file to a standard output.

```
agent2.sources.src1.type = exec
agent2.sources.src1.command = cat
/home/hadoop/Desktop/weatherDataAnalysis/OutputData/weather_da
ta.txt
agent2.sources.src1.channels = chan1
```

Memory Channel: It is used to store events in memory and provides high efficiency. The downside of this channel is that events can be lost in the event of an agent failure.

```
agent2.channels.chan1.type = memory
agent2.channels.chan1.capacity = 1000
```

HDFS Sink: HDFS Sink streams events from a channel to HDFS. Data is written in a configurable file type format. This repository supports SequenceFile, DataStream, and CompressedStream file types.

```
agent2.sinks.sink1.channel = chan1
agent2.sinks.sink1.type = hdfs
agent2.sinks.sink1.hdfs.path = hdfs://localhost:9000/flume
agent2.sinks.sink1.hdfs.fileType = DataStream
agent2.sinks.sink1.hdfs.rollInterval = 60
agent2.sinks.sink1.hdfs.rollSize = 0
agent2.sinks.sink1.hdfs.rollCount = 0
```

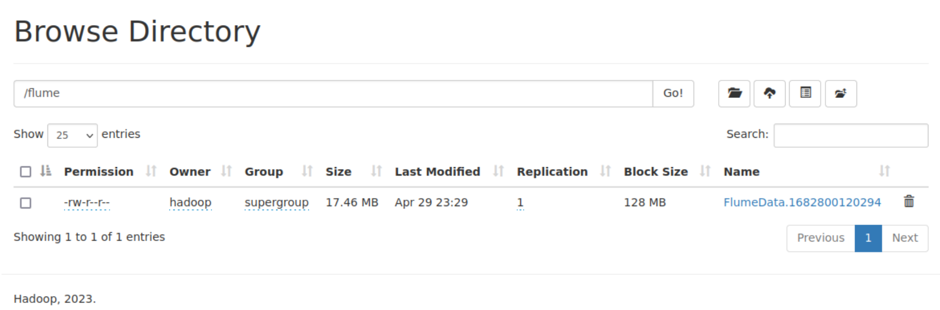


Figure 4.7 FlumeData in HDFS browser

Post-process data file moved to HDFS. The file is stored in a different format in HDFS. When we open the data file, we see that the data has been moved without any corruption (bottom picture).

Figure 4.8 Inside FlumeData file

4.4.2 Apache HBase

HBase is a non-relational, scalable database that provides structured data storage for large tables. It is a column-oriented database built on Hadoop.

HBase Architecture

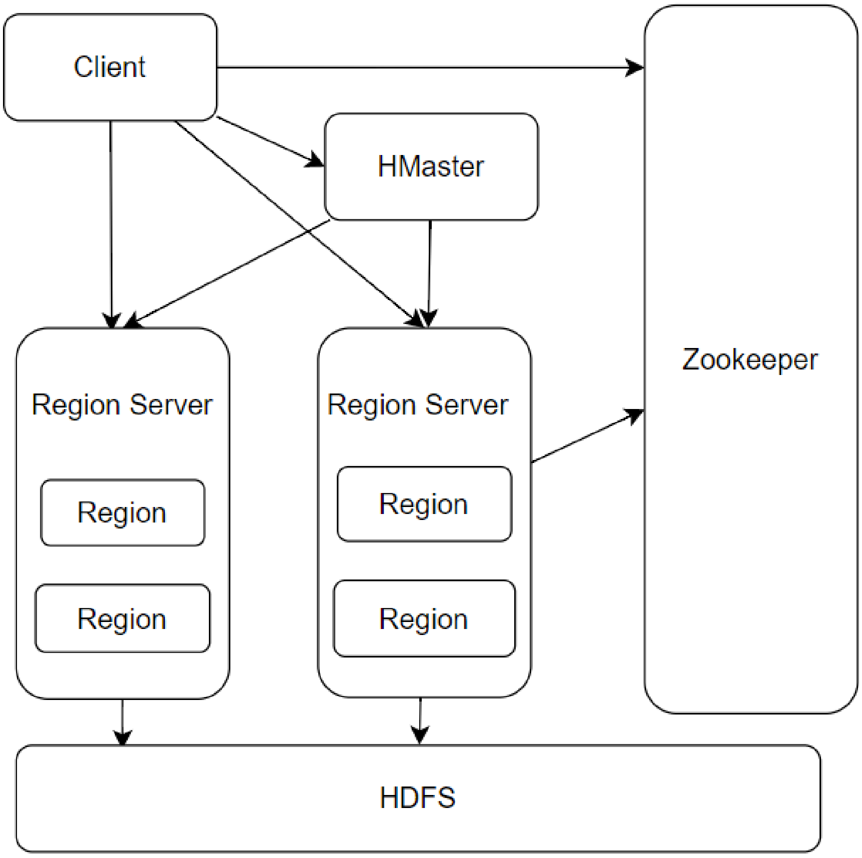


Figure 4.9 HBase architecture

HMaster: It is the use of the main server in HBase. A server where zones are assigned to zone server as well as DDL (table creation, table deletion) operations. HMaster has many structures such as load link, failover.

Region Server: HBase tables are horizontally divided into zones. The zone server runs on the hdfs data node in the hadoop cluster. It is responsible for various things such as processing, managing, executing and reading and writing HBase transactions in these regions. The default size of a zone is 256 MB.

Zookeeper: Clients communicate with zone servers through zookeeper.

Hbase Usage Example:

We enter hbase to write hbase shell commands.

```
cd hbase
bin/hbase shell
```

```
hbasehadoop@ubuntu:~/hbase-2.3.7$ bin/hbase shell
/home/hbasehadoop/hadoop-3.2.4/libexec/hadoop-functions.sh: satır 2366: HADOOP_ORG.APACHE.HADOOP
.HBASE.UTİL.GETJAVAPROPERTY_USER: geçersiz değişken adı
/home/hbasehadoop/hadoop-3.2.4/libexec/hadoop-functions.sh: satır 2461: HADOOP_ORG.APACHE.HADOOP
.HBASE.UTİL.GETJAVAPROPERTY_OPTS: geçersiz değişken adı
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hbasehadoop/hadoop-3.2.4/share/hadoop/common/lib/slf4j-r
eload4j-1.7.35.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hbasehadoop/hbase-2.3.7/lib/client-facing-thirdparty/slf
4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.3.7, r8b2f5141e900c851a2b351fccd54b13bcac5e2ed, Tue Oct 12 16:38:55 UTC 2021
Took 0.0011 seconds
[INFO] Unable to bind key for unsupported operation: beginning-of-line
[INFO] Unable to bind key for unsupported operation: end-of-line
[INFO] Unable to bind key for unsupported operation: quoted-insert
[INFO] Unable to bind key for unsupported operation: beginning-of-line
[INFO] Unable to bind key for unsupported operation: end-of-line
hbase(main):001:0>
```

Output 4.1 HBase shell

status - Provides the status of HBase, for example, the number of servers.
version - Provides the version of HBase being used.
list - Lists all the tables in HBase.

```
hbase(main):001:0> list
TABLE
0 row(s)
Took 0.6794 seconds
=> []
hbase(main):002:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load
Took 0.1995 seconds
hbase(main):003:0> version
2.3.7, r8b2f5141e900c851a2b351fccd54b13bcac5e2ed, Tue Oct 12 16:38:55 UTC 2021
Took 0.0004 seconds
hbase(main):004:0>
```

Output 4.2 HBase list, status and version command

I am creating a table named “weatherdata” with the `create` command.

```
create 'weatherdata', {NAME => 'cf'}
```

```
hbase(main):013:0> create 'weatherdata', {NAME => 'cf'}
Created table weatherdata
Took 0.6388 seconds
=> Hbase::Table - weatherdata
hbase(main):014:0> list
TABLE
weatherdata
1 row(s)
Took 0.0035 seconds
=> ["weatherdata"]
hbase(main):015:0>
```

Output 4.3 HBase create data table

The table named “weatherdata” is created in HDFS.

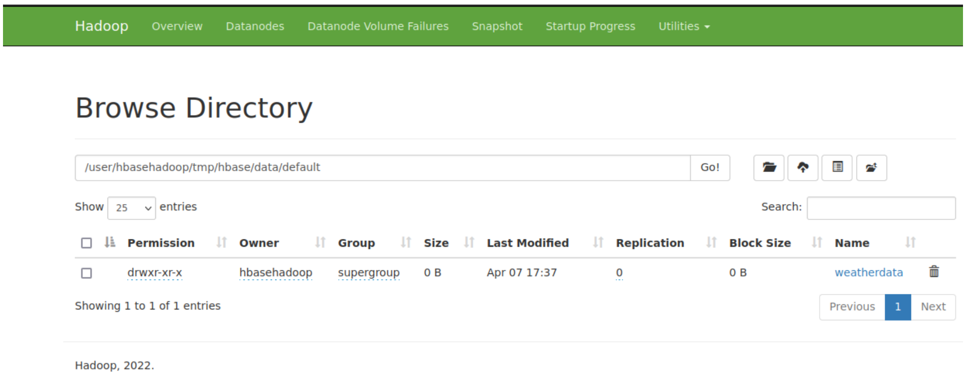


Figure 4.10 HBase table in HDFS browser 1

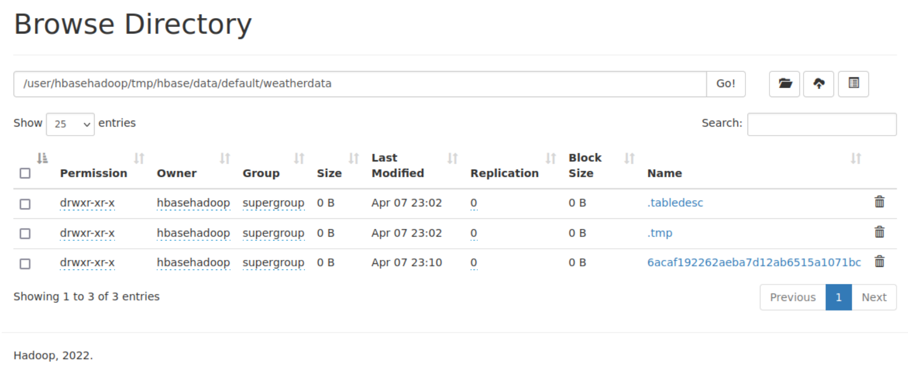


Figure 4.11 HBase table in HDFS browser 1

Then I open a new command line and enter the hbase folder

```
bin/hbase org.apache.hadoop.hbase.mapreduce.ImportTsv
-Dimporttsv.separator=’,’
-Dimporttsv.columns=’HBASE_ROW_KEY,cf:WBANNO,cf:lLST_DATE,cf:C
RX_VN,cf:lLONGITUDE,cf:lLATITUDE,cf:T_DAILY_MAX,cf:T_DAILY_MIN
,cf:T_DAILY_MEAN,cf:T_DAILY_AVG,cf:P_DAILY_CALC,cf:SOLARAD_DAI
LY,cf:SUR_TEMP_DAILY_MAX,cf:SUR_TEMP_DAILY_MIN,cf:SUR_TEMP_DAI
LY_AVG,cf:RH_DAILY_MAX,cf:RH_DAILY_MIN,cf:RH_DAILY_AVG,cf:YEAR
,cf:MONTH,cf:DAY’ weatherdata /home/hadoop/weather_data.csv
```

I import our data file named “weather_data.csv” into my table that I created by importing the column names.

```
2023-04-07 23:08:42,514 INFO  [main] conf.Configuration: resource-types.xml not found
2023-04-07 23:08:42,515 INFO  [main] resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-04-07 23:08:42,525 INFO  [main] resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
2023-04-07 23:08:42,528 INFO  [main] resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
2023-04-07 23:08:44,670 INFO  [main] impl.YarnClientImpl: Submitted application application_1680872948809_0002
2023-04-07 23:08:44,743 INFO  [main] mapreduce.Job: The url to track the job: http://ubuntu:8088/proxy/application_1680872948809_0002/
2023-04-07 23:08:44,744 INFO  [main] mapreduce.Job: Running job: job_1680872948809_0002
2023-04-07 23:09:33,586 INFO  [main] mapreduce.Job: Job job_1680872948809_0002 running in uber mode : false
2023-04-07 23:09:33,625 INFO  [main] mapreduce.Job:  map 0% reduce 0%
2023-04-07 23:10:06,492 INFO  [main] mapreduce.Job:  map 37% reduce 0%
2023-04-07 23:10:12,536 INFO  [main] mapreduce.Job:  map 89% reduce 0%
2023-04-07 23:10:15,692 INFO  [main] mapreduce.Job:  map 100% reduce 0%
2023-04-07 23:10:17,714 INFO  [main] mapreduce.Job: Job job_1680872948809_0002 completed successfully
2023-04-07 23:10:18,357 WARN  [main] counters.FileSystemCounterGroup: HDFS_BYTES_READ_EC is not a recognized counter.
2023-04-07 23:10:18,559 WARN  [main] counters.FrameworkCounterGroup: MAP_PHYSICAL_MEMORY_BYTES_MAX is not a recognized counter.
2023-04-07 23:10:18,561 WARN  [main] counters.FrameworkCounterGroup: MAP_VIRTUAL_MEMORY_BYTES_MAX is not a recognized counter.
2023-04-07 23:10:18,602 INFO  [main] mapreduce.Job: r of bytes read=16271049
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=2
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=0
        Job Counters
                Launched map tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=35904
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=35904
                Total vcore-milliseconds taken by all map tasks=35904
                Total megabyte-milliseconds taken by all map tasks=36765696
        Map-Reduce Framework
                Map input records=170273
                Map output records=170273
                Input split bytes=110
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=293
                CPU time spent (ms)=10330
                Physical memory (bytes) snapshot=232255488
                Virtual memory (bytes) snapshot=1928417280
                Total committed heap usage (bytes)=101187584
        ImportTsv
                Bad Lines=0
        File Input Format Counters
                Bytes Read=16270939
        File Output Format Counters
                Bytes Written=0
hbasehadoop@ubuntu:~/hbase-2.3.7$
```

Output 4.4 HBase import data information

We scan the table with `scan 'weatherdata'`. If we can see the sky, the table has been activated successfully.

```
hbase(main):049:0> scan 'weatherdata'
ROW                                              COLUMN+CELL
 12987                                           column=cf:CRX_VN, timestamp=2023-04-07T23:08:31.569, value=-98.06
 12987                                           column=cf:P_DAILY_CALC, timestamp=2023-04-07T23:08:31.569, value=13.86
 12987                                           column=cf:RH_DAILY_MAX, timestamp=2023-04-07T23:08:31.569, value=24.7
 12987                                           column=cf:RH_DAILY_MIN, timestamp=2023-04-07T23:08:31.569, value=70.0
 12987                                           column=cf:SOLARAD_DAILY, timestamp=2023-04-07T23:08:31.569, value=37.0
 12987                                           column=cf:SUR_TEMP_DAILY_AVG, timestamp=2023-04-07T23:08:31.569, value=98.1
 12987                                           column=cf:SUR_TEMP_DAILY_MAX, timestamp=2023-04-07T23:08:31.569, value=7.1
 12987                                           column=cf:SUR_TEMP_DAILY_MIN, timestamp=2023-04-07T23:08:31.569, value=18.4
 12987                                           column=cf:T_DAILY_AVG, timestamp=2023-04-07T23:08:31.569, value=0.0
 12987                                           column=cf:T_DAILY_MAX, timestamp=2023-04-07T23:08:31.569, value=8.9
 12987                                           column=cf:T_DAILY_MEAN, timestamp=2023-04-07T23:08:31.569, value=18.2
 12987                                           column=cf:T_DAILY_MIN, timestamp=2023-04-07T23:08:31.569, value=18.9
 12987                                           column=cf:WBANNO, timestamp=2023-04-07T23:08:31.569, value=20221231
 12987                                           column=cf:lLATITUDE, timestamp=2023-04-07T23:08:31.569, value=28.9
 12987                                           column=cf:lLONGITUDE, timestamp=2023-04-07T23:08:31.569, value=26.53
 12987                                           column=cf:lLST_DATE, timestamp=2023-04-07T23:08:31.569, value=2.622
 13301                                           column=cf:CRX_VN, timestamp=2023-04-07T23:08:31.569, value=-93.15
 13301                                           column=cf:P_DAILY_CALC, timestamp=2023-04-07T23:08:31.569, value=4.38
 13301                                           column=cf:RH_DAILY_MAX, timestamp=2023-04-07T23:08:31.569, value=53.0
 13301                                           column=cf:RH_DAILY_MIN, timestamp=2023-04-07T23:08:31.569, value=79.2
 13301                                           column=cf:SOLARAD_DAILY, timestamp=2023-04-07T23:08:31.569, value=18.0
 13301                                           column=cf:SUR_TEMP_DAILY_AVG, timestamp=2023-04-07T23:08:31.569, value=92.9
 13301                                           column=cf:SUR_TEMP_DAILY_MAX, timestamp=2023-04-07T23:08:31.569, value=-6.9
 13301                                           column=cf:SUR_TEMP_DAILY_MIN, timestamp=2023-04-07T23:08:31.569, value=2.1
 13301                                           column=cf:T_DAILY_AVG, timestamp=2023-04-07T23:08:31.569, value=0.0
 13301                                           column=cf:T_DAILY_MAX, timestamp=2023-04-07T23:08:31.569, value=-3.2
 13301                                           column=cf:T_DAILY_MEAN, timestamp=2023-04-07T23:08:31.569, value=3.9
 13301                                           column=cf:T_DAILY_MIN, timestamp=2023-04-07T23:08:31.569, value=4.3
 13301                                           column=cf:WBANNO, timestamp=2023-04-07T23:08:31.569, value=20221231
 13301                                           column=cf:lLATITUDE, timestamp=2023-04-07T23:08:31.569, value=11.8
 13301                                           column=cf:lLONGITUDE, timestamp=2023-04-07T23:08:31.569, value=39.87
 13301                                           column=cf:lLST_DATE, timestamp=2023-04-07T23:08:31.569, value=2.622
 21514                                           column=cf:CRX_VN, timestamp=2023-04-07T23:08:31.569, value=-155.58
 21514                                           column=cf:P_DAILY_CALC, timestamp=2023-04-07T23:08:31.569, value=19.85
 21514                                           column=cf:RH_DAILY_MAX, timestamp=2023-04-07T23:08:31.569, value=8.2
 21514                                           column=cf:RH_DAILY_MIN, timestamp=2023-04-07T23:08:31.569, value=23.0
 21514                                           column=cf:SOLARAD_DAILY, timestamp=2023-04-07T23:08:31.569, value=34.6
 21514                                           column=cf:SUR_TEMP_DAILY_AVG, timestamp=2023-04-07T23:08:31.569, value=54.9
 21514                                           column=cf:SUR_TEMP_DAILY_MAX, timestamp=2023-04-07T23:08:31.569, value=2.2
 21514                                           column=cf:SUR_TEMP_DAILY_MIN, timestamp=2023-04-07T23:08:31.569, value=11.5
 21514                                           column=cf:T_DAILY_AVG, timestamp=2023-04-07T23:08:31.569, value=0.0
 21514                                           column=cf:T_DAILY_MAX, timestamp=2023-04-07T23:08:31.569, value=6.3
 21514                                           column=cf:T_DAILY_MEAN, timestamp=2023-04-07T23:08:31.569, value=10.2
 21514                                           column=cf:T_DAILY_MIN, timestamp=2023-04-07T23:08:31.569, value=11.3
```

Output 4.5 HBase scan data

4.4.3 Apache Sqoop:

Sqoop is a tool that allows exporting data from relational database management systems (RDMS) to distributed HDFS' tables. It got its name from the combination of SQL and Hadoop. This allows Sqoop to act as a bridge from SQL to Hadoop.

Sqoop example usage:

To use Sqoop, we must first create a table in MySQL. Since we will use MySQL, let me talk about MySQL first.

MYSQL

MySQL is a widely used open-source relational database management system (RDBMS)

We open mysql with the `sudo mysql -u root -p` command.

We create a database named "weather_data_db_mysql" and then switch to this database.

```
create DATABASE weather_data_db_mysql;
```

We create a new user and password for Sqoop.

```
CREATE USER 'sqoop_test'@'localhost' IDENTIFIED BY
'BerfinTek';
```

We give all privileges of the "weather_data_db_mysql" database we created to the "sqoop_test" user.

```
GRANT ALL PRIVILEGES ON weather_data_db_mysql.* TO
'sqoop_test'@'localhost';
FLUSH PRIVILEGES;
```

We log in with the new user in MySQL and switch to the database we created.

```
sudo mysql -u sqoop_test -p
use weather_data_db_mysql;
```

We create a table called weather in this database and define our columns.

```
create table weather (WBANNO varchar(20), LSAT_DATE
varchar(20), CRX_VN float, LONGITUDE float, LATITUDE float,
T_DAILY_MAX float, T_DAILY_MIN float, T_DAILY_MEAN float,
T_DAILY_AVG float, P_DAILY_CALC float, SOLARAD_DAILY float,
SUR_TEMP_DAILY_MAX float, SUR_TEMP_DAILY_MIN float,
SUR_TEMP_DAILY_AVG float, RH_DAILY_MAX float, RH_DAILY_MIN
float, RH_DAILY_AVG float, YEAR int, MONTH int, DAY int);
```

We open a new command window and connect to mysql and import the table we created in it.

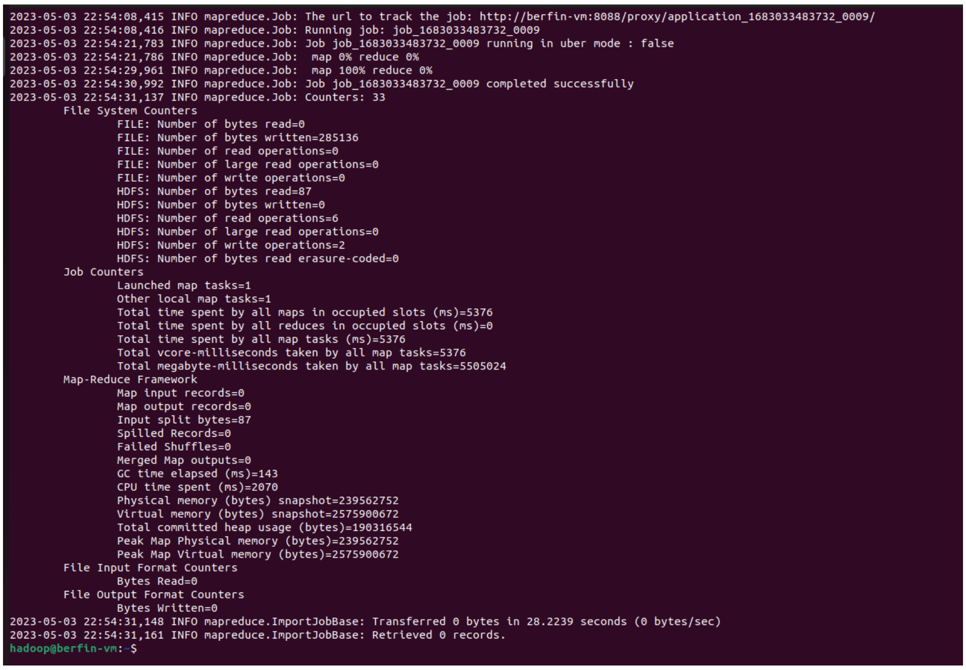
```
sqoop import --connect
jdbc:mysql://localhost/weather_data_db_mysql --username
sqoop_test --password BerfinTek --table weather -m 1
```

- With the **-connect** parameter, the IP address and instance name of the oracle database to which we will connect are given.

  `jdbc:mysql://localhost/weather_data_db_mysql`

- With the **-username** parameter, the username information that we will connect to the oracle database is given. `sqoop_test`
- With the **-password** parameter, the password information of the users to whom we will connect to the oracle database is entered. `BerfinTek`
- With the **-m** parameter, the number of parallel jobs running in the background is entered. `1`
- The name of the table to be read is entered with the **-table** parameter. `Weather`

When the import process is completed, a screen like the picture below will appear.



```
2023-05-03 22:54:08,415 INFO mapreduce.Job: The url to track the job: http://berfin-vm:8088/proxy/application_1683033483732_0009/
2023-05-03 22:54:08,416 INFO mapreduce.Job: Running job: job_1683033483732_0009
2023-05-03 22:54:21,783 INFO mapreduce.Job: Job job_1683033483732_0009 running in uber mode : false
2023-05-03 22:54:21,786 INFO mapreduce.Job:  map 0% reduce 0%
2023-05-03 22:54:29,961 INFO mapreduce.Job:  map 100% reduce 0%
2023-05-03 22:54:30,992 INFO mapreduce.Job: Job job_1683033483732_0009 completed successfully
2023-05-03 22:54:31,137 INFO mapreduce.Job: Counters: 33
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=285136
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=87
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
                Other local map tasks=1
                Total time spent by all maps in occupied slots (ms)=5376
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=5376
                Total vcore-milliseconds taken by all map tasks=5376
                Total megabyte-milliseconds taken by all map tasks=5505024
        Map-Reduce Framework
                Map input records=0
                Map output records=0
                Input split bytes=87
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=143
                CPU time spent (ms)=2070
                Physical memory (bytes) snapshot=239562752
                Virtual memory (bytes) snapshot=2575900672
                Total committed heap usage (bytes)=190316544
                Peak Map Physical memory (bytes)=239562752
                Peak Map Virtual memory (bytes)=2575900672
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=0
2023-05-03 22:54:31,148 INFO mapreduce.ImportJobBase: Transferred 0 bytes in 28.2239 seconds (0 bytes/sec)
2023-05-03 22:54:31,161 INFO mapreduce.ImportJobBase: Retrieved 0 records.
hadoop@berfin-vm:~$
```

Output 4.6 Sqoop import data information

When we check the HDFS cluster, we can see that the weather data has arrived successfully.

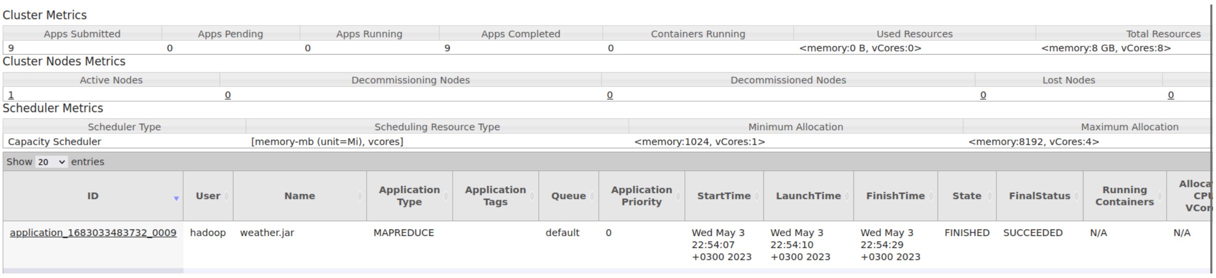


Figure 4.12 Sqoop data file in Hadoop cluster

4.4.4 RabbitMQ

RabbitMQ is a messaging tool written in the java programming language. Licensed under the Apache license.

In this section we will write two small programs in Python. phrases using RabbitMQ to message “I Love Big Data”. We will use the pika library for this.

```
python -m pip install pika -upgrade
```

First, we write codes to send file to “send.py” text message queue.

```
import pika

connection = pika.BlockingConnection(
pika.ConnectionParameters(host='localhost')) channel =
connection.channel()

channel.queue_declare(queue='bigdata')

channel.basic_publish(exchange='', routing_key='bigdata',
body='I Love Big Data!')
print(" [x] Sent 'I Love Big Data!'")
connection.close()
```

Second, we create the “receive.py” file and write the codes to send the message from the queue to the receiver.

```
import pika, sys, os

def main():
     connection =
pika.BlockingConnection(pika.ConnectionParameters(host='localh
ost'))     channel = connection.channel()

     channel.queue_declare(queue='bigdata')

     def callback(ch, method, properties, body):
          print(" [x] Received %r" % body)
```

```
    channel.basic_consume(queue='bigdata',
    on_message_callback=callback, auto_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    channel.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)
```

Finally, we open cmd in the folder with “receive.py” and “send.py” files are. First we download the pika library.

```
pip install pika
```

Then we run the “receive.py” script file.

```
python3 receive.py
```

Finally, we open a new cmd in the same folder and run the “send.py” code file in it.

```
python3 send.py
```

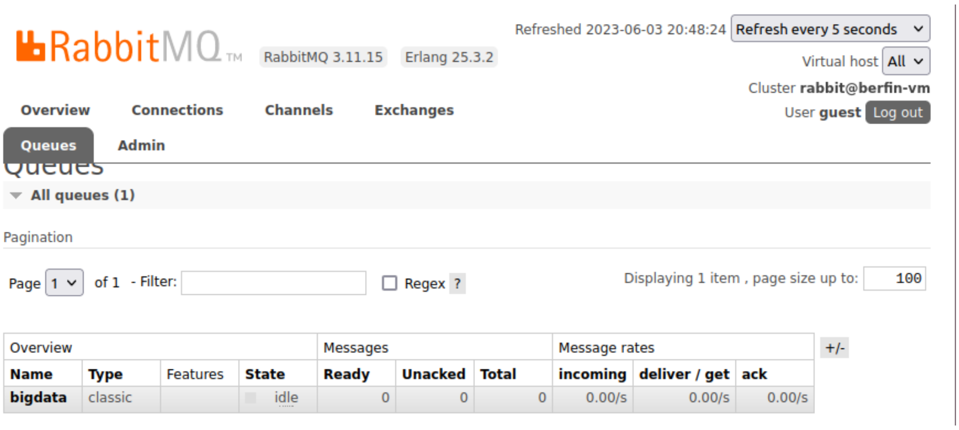


Figure 4.13 RabbitMQ message queue in browser

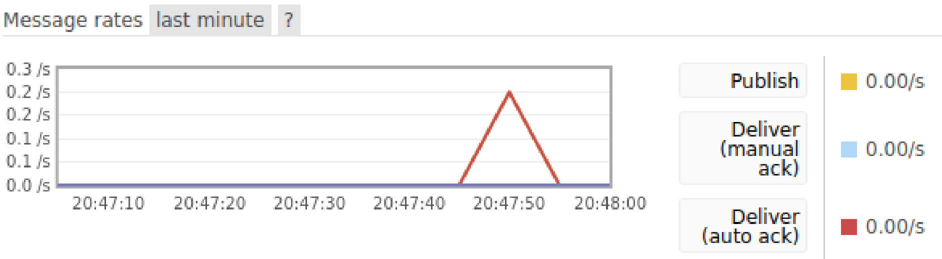


Figure 4.14 RabbitMQ message rates graph

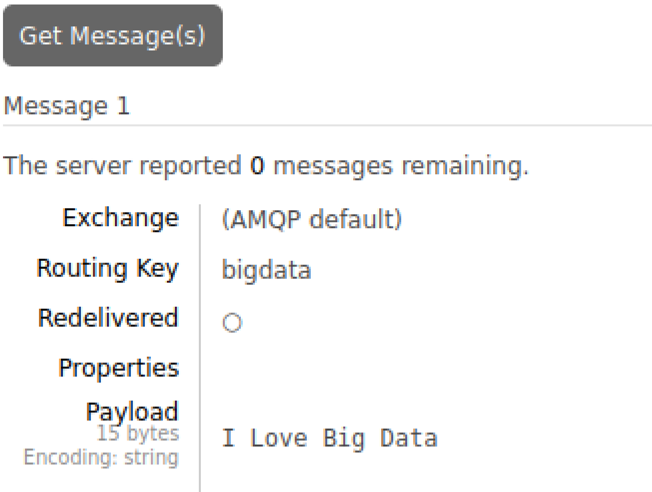


Figure 4.15 RabbitMQ message

4.5 Interactive Querying

4.5.1 Apache Hive

Hive is a data warehouse infrastructure for processing data in hadoop. It resides on Hadoop and makes it easy to query and analyze big data. For this, it provides an SQL-like query language called Hive Query Language. Hive provides a shell for creating tables.

Hive architecture:

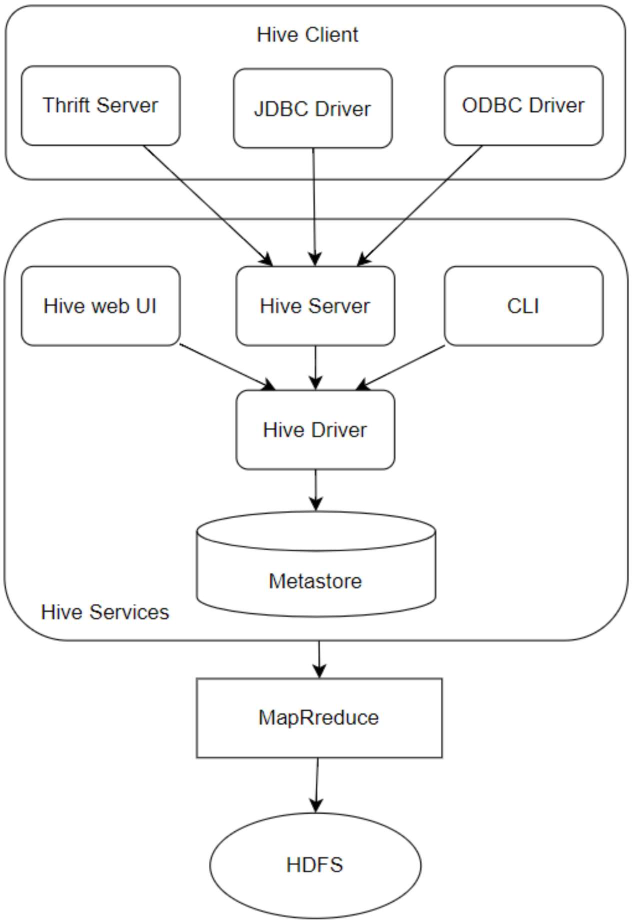


Figure 4.16 Hive architecture

- **Hive CLI** - Hive CLI (Command Line Interface) is a shell with which we can run Hive queries and commands.
- **Hive MetaStore** - It is a central repository that stores all the structure information of the various tables and partitions in the repository. It also includes the column's metadata and type information, the serializers and deserializers used to read and write the data, and the associated HDFS files where the data is stored.

We start this shell with the `hive` command. And we use the `create` command to create a new database.

```
create database weather_data_db;
show databases;
```

```
hive> SHOW databases;
OK
default
weather_data_db
Time taken: 1.241 seconds, Fetched: 2 row(s)
```

Output 4.7 Hive show databases

We enter the database we created with the `use` command.

```
use weather_data_db;
```

Now we need to create a table in this database and define the columns. We save the table we created in "/user/hive/warehouse" in HDFS.

```
create table weather_data( wbanno STRING, lsat_date STRING,
crx_vn FLOAT, longitude FLOAT, latitude FLOAT, t_daily_max
FLOAT, t_daily_min FLOAT, t_daily_mean FLOAT, t_daily_avg
FLOAT, p_daily_calc FLOAT, solarad_daily FLOAT,
sur_temp_daily_max FLOAT, sur_temp_daily_min FLOAT,
sur_temp_daily_avg FLOAT, rh_daily_max FLOAT, rh_daily_min
FLOAT, rh_daily_avg FLOAT, year INT, month INT, day INT) row
format delimited fields terminated by ',' stored as textfile
location '/user/hive/warehouse/weather_data';
```

```
show tables;
```

```
hive> show tables;
OK
books
weather_data
Time taken: 0.082 seconds, Fetched: 2 row(s)
```

Output 4.8 Hive show tables

As an example, I previously created a table called "books". We can delete this table with the drop command.

```
drop table books;
```

We load our data file in HDFS into this table we created in hive.

```
LOAD DATA INPATH '/user/hadoop/weather_data.csv' INTO TABLE
weather_data;
```

Now we can query and analyses our data file using hive commands (similar to SQL commands). Below are a few examples of using hive commands.

```
Select * from weather_data;
```

```
94082   2022-11-16      3       -109    40      0       -10     -5      -5      0       10      -3      -14     -10     72      48      59      2022    11      16
94082   2022-11-17      3       -109    40      1       -8      -4      -4      0       11      -2      -15     -9      73      42      59      2022    11      17
94082   2022-11-18      3       -109    40      -3      -11     -7      -7      0       11      -5      -17     -11     73      36      53      2022    11      18
94082   2022-11-19      3       -109    40      -2      -13     -7      -8      0       11      -4      -18     -12     73      41      56      2022    11      19
94082   2022-11-20      3       -109    40      2       -12     -5      -5      0       11      -3      -18     -12     73      38      55      2022    11      20
94082   2022-11-21      3       -109    40      2       -9      -3      -4      0       10      -2      -16     -10     70      43      55      2022    11      21
94082   2022-11-22      3       -109    40      2       -8      -3      -4      0       10      -1      -16     -10     67      42      57      2022    11      22
94082   2022-11-23      3       -109    40      2       -8      -3      -2      0       7       0       -16     -7      77      39      57      2022    11      23
94082   2022-11-24      3       -109    40      5       -3      1       0       0       10      0       -9      -5      64      43      54      2022    11      24
94082   2022-11-25      3       -109    40      5       -6      -1      -2      0       10      0       -12     -7      76      44      61      2022    11      25
94082   2022-11-26      3       -109    40      4       -7      -2      -1      0       7       1       -12     -5      92      45      69      2022    11      26
94082   2022-11-27      3       -109    40      2       -4      -1      -1      0       6       -1      -10     -5      95      45      71      2022    11      27
94082   2022-11-28      3       -109    40      3       -6      -2      -1      6       3       -1      -9      -5      91      40      53      2022    11      28
94082   2022-11-29      3       -109    40      -5      -11     -8      -7      1       10      -6      -21     -12     92      42      71      2022    11      29
94082   2022-11-30      3       -109    40      -2      -13     -8      -7      0       10      -7      -19     -13     64      37      52      2022    11      30
94082   2022-12-01      3       -109    40      4       -10     -3      -2      0       7       -2      -18     -8      64      33      46      2022    12      1
94082   2022-12-02      3       -109    40      4       -9      -3      -2      1       9       0       -16     -7      90      30      50      2022    12      2
94082   2022-12-03      3       -109    40      -3      -11     -7      -7      0       7       -4      -18     -9      70      44      58      2022    12      3
94082   2022-12-04      3       -109    40      2       -6      -2      -3      0       5       -2      -9      -4      76      50      65      2022    12      4
94082   2022-12-05      3       -109    40      3       -3      0       -1      7       2       0       -6      -2      95      65      85      2022    12      5
94082   2022-12-06      3       -109    40      -2      -8      -5      -4      1       3       -1      -18     -7      95      71      85      2022    12      6
94082   2022-12-07      3       -109    40      -5      -11     -8      -7      5       4       -3      -19     -8      91      79      88      2022    12      7
94082   2022-12-08      3       -109    40      -1      -12     -6      -5      0       9       -6      -18     -11     90      56      79      2022    12      8
94082   2022-12-09      3       -109    40      -1      -13     -7      -7      0       8       -3      -17     -12     89      64      83      2022    12      9
94082   2022-12-10      3       -109    40      -5      -16     -11     -11     0       8       -8      -22     -14     89      75      85      2022    12      10
94082   2022-12-11      3       -109    40      4       -11     -4      -5      0       6       -1      -17     -8      88      35      75      2022    12      11
94082   2022-12-12      3       -109    40      2       -7      -3      -3      4       4       0       -15     -5      92      55      84      2022    12      12
94082   2022-12-13      3       -109    40      -6      -9      -8      -7      2       5       -6      -14     -9      90      73      84      2022    12      13
94082   2022-12-14      3       -109    40      -4      -13     -9      -8      0       9       -8      -21     -15     89      60      81      2022    12      14
94082   2022-12-15      3       -109    40      -7      -15     -11     -10     1       8       -7      -23     -13     88      72      83      2022    12      15
94082   2022-12-16      3       -109    40      -9      -17     -13     -13     0       9       -13     -25     -20     87      56      73      2022    12      16
94082   2022-12-17      3       -109    40      -12     -19     -15     -15     0       9       -13     -28     -21     86      72      77      2022    12      17
94082   2022-12-18      3       -109    40      -11     -19     -15     -16     0       9       -13     -28     -22     84      73      80      2022    12      18
94082   2022-12-19      3       -109    40      -9      -18     -13     -14     0       9       -12     -26     -21     82      63      75      2022    12      19
94082   2022-12-20      3       -109    40      -9      -16     -12     -12     0       6       -8      -21     -13     82      65      75      2022    12      20
94082   2022-12-21      3       -109    40      0       -14     -7      -5      0       6       -3      -20     -10     79      49      65      2022    12      21
94082   2022-12-22      3       -109    40      -2      -19     -11     -15     1       9       -4      -23     -17     88      41      59      2022    12      22
94082   2022-12-23      3       -109    40      -8      -18     -13     -12     0       8       -9      -18     -14     68      41      56      2022    12      23
94082   2022-12-24      3       -109    40      -4      -13     -9      -9      0       5       -4      -17     -12     81      48      64      2022    12      24
94082   2022-12-25      3       -109    40      -2      -11     -7      -7      0       7       -5      -14     -10     81      59      73      2022    12      25
94082   2022-12-26      3       -109    40      -2      -9      -5      -5      0       6       -2      -13     -8      87      68      78      2022    12      26
94082   2022-12-27      3       -109    40      3       -9      -3      -2      16      3       0       -14     -5      94      59      85      2022    12      27
94082   2022-12-28      3       -109    40      1       -4      -2      -1      3       2       0       -10     -2      95      68      90      2022    12      28
94082   2022-12-29      3       -109    40      -4      -7      -5      -5      1       9       -4      -16     -10     93      62      76      2022    12      29
94082   2022-12-30      3       -109    40      -1      -8      -5      -4      2       5       -2      -9      -5      92      71      83      2022    12      30
94082   2022-12-31      3       -109    40      3       -4      0       0       7       4       1       -3      -1      94      77      87      2022    12      31
Time taken: 3.17 seconds, Fetched: 170272 row(s)
hive>
```

Output 4.9 Hive select command

LIMIT operator:

```
Select * From weather_data LIMIT 10;
```

```
hive> Select * From weather_data LIMIT 10;
OK
23802   2020-01-01      5       -88     32      14      -2      6       7       0       -9999   -9999   -9999   -9999   -9999   -9999   -9999   2020    1       1
23802   2020-01-02      5       -88     32      22      9       15      17      24      -9999   -9999   -9999   -9999   -9999   -9999   -9999   2020    1       2
23802   2020-01-03      5       -88     32      21      13      17      15      16      -9999   -9999   -9999   -9999   -9999   -9999   -9999   2020    1       3
23802   2020-01-04      5       -88     32      15      2       8       11      0       -9999   -9999   -9999   -9999   -9999   -9999   -9999   2020    1       4
23802   2020-01-05      5       -88     32      15      -2      7       6       0       -9999   -9999   -9999   -9999   -9999   -9999   -9999   2020    1       5
23802   2020-01-06      5       -88     32      20      2       11      11      0       -9999   -9999   -9999   -9999   -9999   -9999   -9999   2020    1       6
23802   2020-01-07      5       -88     32      16      0       8       10      0       -9999   -9999   -9999   -9999   -9999   -9999   -9999   2020    1       7
23802   2020-01-08      5       -88     32      17      -2      7       6       0       -9999   -9999   -9999   -9999   -9999   -9999   -9999   2020    1       8
23802   2020-01-09      5       -88     32      19      -1      9       11      0       -9999   -9999   -9999   -9999   -9999   -9999   -9999   2020    1       9
23802   2020-01-10      5       -88     32      23      15      19      19      2       -9999   -9999   -9999   -9999   -9999   -9999   -9999   2020    1       10
Time taken: 1.623 seconds, Fetched: 10 row(s)
hive>
```

Output 4.10 Hive limit command

MAX, MIN, AVG operators:

```
SELECT MAX(t_daily_max) AS maxTemp, MIN(t_daily_min) AS
minTmep, AVG(t_daily_avg) AS avgTemp FROM weather_data;
```

```
hive> SELECT MAX(t_daily_max) AS maxTemp, MIN(t_daily_min) AS minTmep, AVG(t_daily_avg) AS avgTemp FROM weather_data;
Query ID = hadoop_20230507233841_e1abf969-7d42-4982-9b0b-97522f1aa362
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1683033483732_0011, Tracking URL = http://berfin-vm:8088/proxy/application_1683033483732_0011/
Kill Command = /usr/local/hadoop/bin/mapred job  -kill job_1683033483732_0011
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2023-05-07 23:39:00,928 Stage-1 map = 0%,  reduce = 0%
2023-05-07 23:39:09,457 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.18 sec
2023-05-07 23:39:17,843 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.43 sec
MapReduce Total cumulative CPU time: 6 seconds 430 msec
Ended Job = job_1683033483732_0011
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.43 sec   HDFS Read: 18328347 HDFS Write: 118 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 430 msec
OK
54      -9999   -230.8543
Time taken: 37.735 seconds, Fetched: 1 row(s)
hive>
```

Output 4.11 Hive max, min, avg command

ORDER BY, DESC operator:

```
SELECT lsat_date , t_daily_mean FROM weather_data ORDER BY
t_daily_mean DESC LIMIT 10;
```

```
hive> SELECT lsat_date ,  t_daily_mean FROM weather_data ORDER BY t_daily_mean  DESC LIMIT 10;
Query ID = hadoop_20230507234250_5c0915e1-32ad-49db-96e1-b4bd8edb2995
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1683033483732_0012, Tracking URL = http://berfin-vm:8088/proxy/application_1683033483732_0012/
Kill Command = /usr/local/hadoop/bin/mapred job  -kill job_1683033483732_0012
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2023-05-07 23:43:01,190 Stage-1 map = 0%,  reduce = 0%
2023-05-07 23:43:08,507 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.45 sec
2023-05-07 23:43:15,824 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 5.63 sec
MapReduce Total cumulative CPU time: 5 seconds 630 msec
Ended Job = job_1683033483732_0012
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 5.63 sec   HDFS Read: 18321305 HDFS Write: 347 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 630 msec
OK
2021-07-11      48
2021-07-10      47
2020-08-17      46
2021-07-12      45
2020-08-18      45
2022-07-16      44
2021-06-18      44
2021-06-17      44
2021-07-08      44
2021-06-28      44
Time taken: 26.151 seconds, Fetched: 10 row(s)
hive>
```

Output 4.12 Hive desc command

4.6 Batch Analysis

4.6.1 Apache Pig

All scripts run on a single machine without requiring Hadoop MapReduce and HDFS. Native mode does not require Hadoop.

```
pig
```

```
hdoop@berfin-VirtualBox:~$ pig -x local
2023-01-04 23:43:25,973 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
2023-01-04 23:43:25,990 INFO pig.ExecTypeProvider: Picked LOCAL as the ExecType
2023-01-04 23:43:26,689 [main] INFO  org.apache.pig.Main - Apache Pig version 0.17.0 (r1797386) compiled Jun 02 2017, 15:41:58
2023-01-04 23:43:26,689 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/hdoop/pig_1672865006562.log
2023-01-04 23:43:27,117 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/hdoop/.pigbootup not found
2023-01-04 23:43:27,743 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2023-01-04 23:43:27,745 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: file:///
2023-01-04 23:43:28,091 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2023-01-04 23:43:28,216 [main] INFO  org.apache.pig.PigServer - Pig Script ID for the session: PIG-default-88267a3e-3796-4856-9de2-25e6bb0ddd90
2023-01-04 23:43:28,216 [main] WARN  org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabled set to false
grunt>
```

Output 4.13 Pig local script

CSVLoader is a PigStorage based loading function.

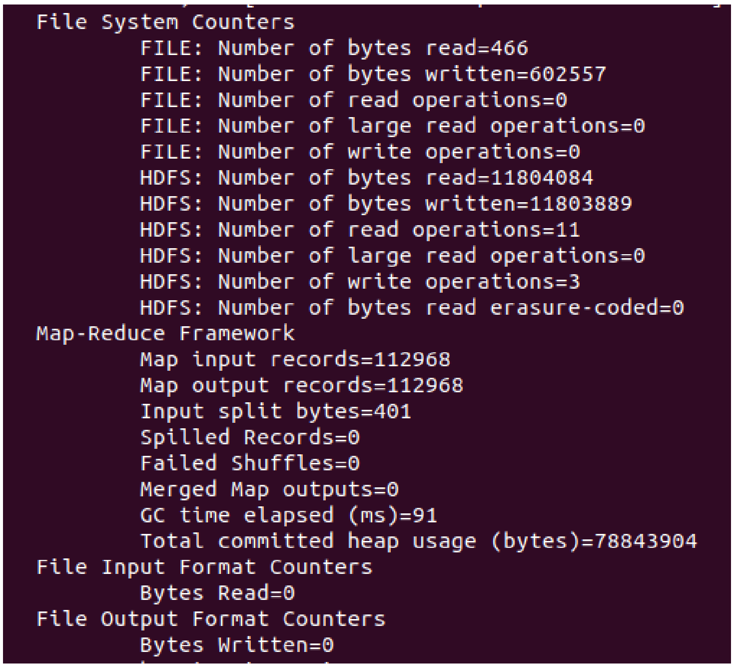
```
define CSVLoader org.apache.pig.piggybank.storage.CSVLoader();

data = LOAD '/pig/data/weather_data.csv' USING CSVLoader as
(WBANNO:chararray, LSAT_DATE:chararray, CRX_VN:float,
LONGITUDE:float, LATITUDE:float, T_DAILY_MAX:float,
T_DAILY_MIN:float, T_DAILY_MEAN:float, T_DAILY_AVG:float,
P_DAILY_CALC:float, SOLARAD_DAILY:float,
SUR_TEMP_DAILY_MAX:float, SUR_TEMP_DAILY_MIN:float,
SUR_TEMP_DAILY_AVG:float, RH_DAILY_MAX:float,
```

```
RH_DAILY_MIN:float, RH_DAILY_AVG:float, YEAR:int, MONTH:int,
DAY:int);
```

We can store the uploaded data in the HDFS file system using the Store operator.

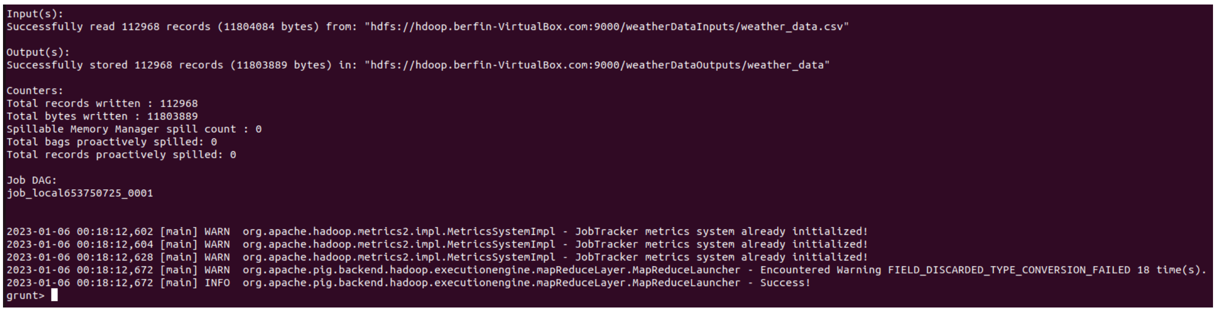
```
STORE data INTO '/pig/outputs/weather_data';
```

Script Statistics:



```
File System Counters
        FILE: Number of bytes read=466
        FILE: Number of bytes written=602557
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=11804084
        HDFS: Number of bytes written=11803889
        HDFS: Number of read operations=11
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=3
        HDFS: Number of bytes read erasure-coded=0
Map-Reduce Framework
        Map input records=112968
        Map output records=112968
        Input split bytes=401
        Spilled Records=0
        Failed Shuffles=0
        Merged Map outputs=0
        GC time elapsed (ms)=91
        Total committed heap usage (bytes)=78843904
File Input Format Counters
        Bytes Read=0
File Output Format Counters
        Bytes Written=0
```

Output 4.14 Pig store data script



```
Input(s):
Successfully read 112968 records (11804084 bytes) from: "hdfs://hdoop.berfin-VirtualBox.com:9000/weatherDataInputs/weather_data.csv"

Output(s):
Successfully stored 112968 records (11803889 bytes) in: "hdfs://hdoop.berfin-VirtualBox.com:9000/weatherDataOutputs/weather_data"

Counters:
Total records written : 112968
Total bytes written : 11803889
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local653750725_0001


2023-01-06 00:18:12,602 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-01-06 00:18:12,604 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-01-06 00:18:12,628 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-01-06 00:18:12,672 [main] WARN  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Encountered Warning FIELD_DISCARDED_TYPE_CONVERSION_FAILED 18 time(s).
2023-01-06 00:18:12,672 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
grunt>
```

Output 4.15 Pig statistic script 1

FILTER operator:

```
low_temp_daily_avg = FILTER data by T_DAILY_AVG<20.0;

STORE low_temp_daily_avg INTO
'/pig/outputs/low_temp_daily_avg';
```

```
Input(s):
Successfully read 112968 records (11804084 bytes) from: "hdfs://hdoop.berfin-VirtualBox.com:9000/weatherDataInputs/weather_data.csv"

Output(s):
Successfully stored 65251 records (6786196 bytes) in: "hdfs://hdoop.berfin-VirtualBox.com:9000/weatherDataOutputs/low_temp_daily_avg"

Counters:
Total records written : 65251
Total bytes written : 6786196
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local1954239232_0001
```

Output 4.16 Pig filter operator script

Execute and verify the data.

```
DUMP low_temp_daily_avg;
```

```
(4139,2021-11-14,2.623,-119.64,41.85,16.7,6.4,11.5,10.5,0.0,8.0,19.1,2.6,7.9,59.3,28.5,42.3,2021,11,14)
(4139,2021-11-15,2.623,-119.64,41.85,13.0,4.2,8.6,8.4,0.0,6.44,15.0,1.1,6.4,75.4,25.7,45.7,2021,11,15)
(4139,2021-11-16,2.623,-119.64,41.85,4.7,-8.3,-1.8,-1.5,0.0,8.77,10.0,-10.8,-1.3,94.9,30.1,68.0,2021,11,16)
(4139,2021-11-17,2.623,-119.64,41.85,3.9,-11.0,-3.5,-2.7,0.0,9.8,6.9,-12.5,-3.5,81.2,21.5,49.1,2021,11,17)
(4139,2021-11-18,2.623,-119.64,41.85,10.7,-2.1,4.3,3.5,0.3,4.79,14.3,-4.4,2.5,85.8,25.0,51.7,2021,11,18)
(4139,2021-11-19,2.623,-119.64,41.85,5.8,-5.2,0.3,2.2,0.7,5.3,7.3,-8.4,1.4,91.6,53.2,73.8,2021,11,19)
(4139,2021-11-20,2.623,-119.64,41.85,4.1,-8.9,-2.4,-2.6,0.0,9.8,7.0,-12.4,-5.7,94.0,29.0,64.7,2021,11,20)
(4139,2021-11-22,2.623,-119.64,41.85,11.3,-1.3,5.0,3.6,0.0,10.55,14.0,-4.9,0.5,52.4,13.6,36.8,2021,11,22)
(4139,2021-11-23,2.623,-119.64,41.85,3.0,-9.4,-3.2,-1.1,0.0,8.06,7.4,-12.8,-2.6,80.7,32.0,53.2,2021,11,23)
(4139,2021-11-24,2.623,-119.64,41.85,3.3,-10.3,-3.5,-3.2,0.0,9.84,7.8,-14.4,-4.7,91.7,18.5,58.1,2021,11,24)
(4139,2021-11-25,2.623,-119.64,41.85,10.5,-3.2,3.6,3.4,0.0,8.71,13.2,-5.2,1.3,45.6,12.3,22.2,2021,11,25)
(4139,2021-11-26,2.623,-119.64,41.85,12.3,0.3,6.3,5.9,0.0,8.31,17.0,-2.6,3.7,74.4,15.5,35.0,2021,11,26)
(4139,2021-11-27,2.623,-119.64,41.85,11.1,1.3,6.2,5.4,0.0,8.53,14.2,-2.5,2.7,90.0,51.0,73.2,2021,11,27)
(4139,2021-11-28,2.623,-119.64,41.85,14.3,3.6,9.0,8.2,0.0,8.74,17.5,-1.9,4.5,77.6,26.0,50.8,2021,11,28)
(4139,2021-11-29,2.623,-119.64,41.85,14.0,-1.0,6.5,6.4,0.0,9.64,16.2,-7.5,1.9,71.8,12.0,46.1,2021,11,29)
(4139,2021-11-30,2.623,-119.64,41.85,13.7,1.7,7.7,7.0,0.0,8.31,16.3,-3.9,2.5,55.9,18.2,31.0,2021,11,30)
(4139,2021-12-01,2.623,-119.64,41.85,15.1,2.6,8.9,8.3,0.0,8.47,18.1,-4.1,4.2,82.1,24.4,41.9,2021,12,1)
(4139,2021-12-02,2.623,-119.64,41.85,14.5,0.0,7.3,7.9,0.0,9.24,18.1,-4.7,3.8,62.7,22.0,35.5,2021,12,2)
(4139,2021-12-03,2.623,-119.64,41.85,12.8,-1.3,5.7,6.0,0.0,9.23,16.2,-5.8,1.6,70.5,21.2,43.6,2021,12,3)
(4139,2021-12-04,2.623,-119.64,41.85,11.7,0.3,6.0,5.7,0.0,8.82,14.1,-3.9,2.0,59.8,19.8,37.1,2021,12,4)
(4139,2021-12-05,2.623,-119.64,41.85,8.2,-2.4,2.9,3.1,0.0,8.61,12.4,-6.1,0.2,67.5,21.1,44.0,2021,12,5)
(4139,2021-12-06,2.623,-119.64,41.85,4.8,-2.5,1.2,2.5,0.0,1.35,5.2,-5.1,1.1,90.8,25.9,51.6,2021,12,6)
(4139,2021-12-07,2.623,-119.64,41.85,7.4,-2.9,2.2,2.3,0.0,7.52,11.5,-5.7,0.6,92.3,54.3,77.3,2021,12,7)
(4139,2021-12-08,2.623,-119.64,41.85,3.1,-1.8,0.6,0.9,0.0,2.5,4.8,-3.1,-0.1,85.3,56.2,75.2,2021,12,8)
(4139,2021-12-09,2.623,-119.64,41.85,-1.8,-8.4,-5.1,-5.4,0.0,7.56,2.5,-9.9,-4.8,91.0,43.5,70.1,2021,12,9)
(4139,2021-12-10,2.623,-119.64,41.85,-1.6,-11.6,-6.6,-6.3,0.0,8.51,4.0,-12.4,-6.8,89.9,32.9,63.3,2021,12,10)
(4139,2021-12-11,2.623,-119.64,41.85,1.8,-5.6,-1.9,-1.5,0.2,3.17,4.0,-5.8,-1.8,65.6,33.8,46.0,2021,12,11)
(4139,2021-12-12,2.623,-119.64,41.85,2.1,-1.7,0.2,-0.3,0.5,2.08,4.6,-2.5,-0.8,82.3,62.4,72.1,2021,12,12)
(4139,2021-12-13,2.623,-119.64,41.85,1.3,-5.3,-2.0,-0.8,14.0,2.68,0.2,-5.5,-1.4,93.7,67.2,86.5,2021,12,13)
(4139,2021-12-14,2.623,-119.64,41.85,-5.2,-13.5,-9.4,-8.4,3.8,2.25,-4.5,-18.9,-9.9,92.5,77.2,87.9,2021,12,14)
(4139,2021-12-15,2.623,-119.64,41.85,-3.8,-13.0,-8.4,-7.4,1.9,5.27,-4.6,-17.2,-8.3,90.6,76.7,83.4,2021,12,15)
(4139,2021-12-16,2.623,-119.64,41.85,-3.3,-8.3,-5.8,-5.1,1.2,4.89,-3.4,-13.7,-6.6,93.5,77.7,86.3,2021,12,16)
(4139,2021-12-17,2.623,-119.64,41.85,-0.8,-10.5,-5.7,-5.5,0.0,8.23,-3.5,-16.9,-10.6,90.1,32.0,63.8,2021,12,17)
(4139,2021-12-18,2.623,-119.64,41.85,2.3,-8.8,-3.2,-1.2,0.0,7.37,-1.0,-11.5,-5.6,81.4,30.7,52.0,2021,12,18)
(4139,2021-12-19,2.623,-119.64,41.85,3.2,-2.6,0.3,0.2,0.0,7.69,-0.1,-8.9,-4.2,80.2,28.6,53.6,2021,12,19)
(4139,2021-12-20,2.623,-119.64,41.85,2.6,-3.1,-0.2,0.0,0.0,7.19,-1.8,-10.6,-6.2,40.5,21.9,29.7,2021,12,20)
(4139,2021-12-21,2.623,-119.64,41.85,4.3,-3.4,0.4,0.7,0.0,6.3,0.6,-10.6,-4.8,64.5,22.7,34.6,2021,12,21)
(4139,2021-12-22,2.623,-119.64,41.85,1.7,-2.4,-0.3,-0.2,1.8,4.41,0.4,-3.6,-1.6,94.4,66.1,83.1,2021,12,22)
(4139,2021-12-23,2.623,-119.64,41.85,1.0,-2.5,-0.7,-0.9,8.4,2.63,-0.2,-2.6,-1.3,95.7,75.6,91.6,2021,12,23)
(4139,2021-12-24,2.623,-119.64,41.85,-2.2,-5.7,-3.9,-4.3,7.6,5.54,-2.3,-8.3,-4.9,94.6,72.7,86.0,2021,12,24)
(4139,2021-12-25,2.623,-119.64,41.85,-3.5,-8.2,-5.8,-5.8,20.1,3.61,-3.6,-10.6,-6.3,93.6,76.9,89.5,2021,12,25)
(4139,2021-12-26,2.623,-119.64,41.85,-6.3,-10.9,-8.6,-8.7,0.0,4.43,-6.5,-12.8,-8.9,90.9,78.3,87.1,2021,12,26)
(4139,2021-12-27,2.623,-119.64,41.85,-6.3,-12.7,-9.5,-10.0,0.0,6.46,-6.5,-16.6,-11.1,91.5,67.8,80.3,2021,12,27)
(4139,2021-12-28,2.623,-119.64,41.85,-7.5,-12.5,-10.0,-9.8,0.0,4.94,-7.8,-15.3,-10.4,91.1,80.9,86.2,2021,12,28)
(4139,2021-12-29,2.623,-119.64,41.85,-6.0,-10.8,-8.4,-7.4,0.6,4.18,-5.5,-15.1,-8.8,91.7,86.0,89.7,2021,12,29)
(4139,2021-12-30,2.623,-119.64,41.85,-4.4,-10.0,-7.2,-6.9,0.2,4.7,-5.9,-13.8,-8.4,92.3,58.5,80.7,2021,12,30)
(4139,2021-12-31,2.623,-119.64,41.85,-6.8,-15.6,-11.2,-8.8,3.8,6.43,-5.4,-25.9,-9.8,90.0,63.8,81.9,2021,12,31)
grunt>
```

Output 4.17 Pig dump operator script

GROUP operator:

```
t_daily_avg_group = GROUP data BY T_DAILY_AVG;
```

Used to display the schema of a relationship.

```
DESCRIBE t_daily_avg_group;
```

```
grunt> DESCRIBE t_daily_avg_group;
t_daily_avg_group: {group: float,data: {(WBANNO: chararray,LSAT_DATE: chararray,CRX_VN: float,LONGITUDE: float,LATITUDE: float,T_DAILY_MAX: float,T_DAILY_MIN: float,T_DAILY_MEAN: float,T_DAILY_AVG: float,P_DAILY_CALC: float,SOLARAD_DAILY: float,SUR_TEMP_DAILY_MAX: float,SUR_TEMP_DAILY_MIN: float,SUR_TEMP_DAILY_AVG: float,RH_DAILY_MAX: float,RH_DAILY_MIN: float,RH_DAILY_AVG: float,YEAR: int,MONTH: int,DAY: int)}}
grunt>
```

Output 4.18 Pig describe operator script

```
STORE t_daily_avg_group INTO '/pig/outputs/t_daily_avg_group
';
```

```
Counters:
Total records written : 858
Total bytes written : 25608438
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local769059157_0003
```

Output 4.19 Pig store operator script

LIMIT operator:

```
limit10_low_temp_daily_avg = LIMIT low_temp_daily_avg 10;
DUMP limit10_low_temp_daily_avg;
```

```
Counters:
Total records written : 10
Total bytes written : 37644484
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local1638998459_0004        ->      job_local1012153940_0005,
job_local1012153940_0005


2023-01-06 00:51:22,866 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-01-06 00:51:22,869 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-01-06 00:51:22,904 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-01-06 00:51:22,910 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-01-06 00:51:22,921 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-01-06 00:51:22,927 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2023-01-06 00:51:22,952 [main] WARN  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Encountered Warning
2023-01-06 00:51:22,954 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2023-01-06 00:51:22,955 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use
2023-01-06 00:51:22,958 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2023-01-06 00:51:22,972 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2023-01-06 00:51:22,989 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(54933,2020-01-01,2.622,-99.13,45.71,1.9,-5.9,-2.0,-0.8,7.8,4.7,0.1,-8.6,-2.4,94.2,66.1,78.0,2020,1,1)
(54933,2020-01-02,2.622,-99.13,45.71,0.8,-2.4,-0.8,-0.8,0.2,4.87,-0.1,-4.6,-1.3,93.9,85.0,88.8,2020,1,2)
(54933,2020-01-03,2.622,-99.13,45.71,1.0,-11.7,-5.4,-4.7,1.1,4.66,-0.1,-16.9,-6.3,93.8,71.5,83.2,2020,1,3)
(54933,2020-01-04,2.622,-99.13,45.71,0.8,-13.8,-6.5,-7.5,0.0,5.86,-1.6,-17.9,-9.4,90.7,76.2,86.3,2020,1,4)
(54933,2020-01-05,2.622,-99.13,45.71,4.3,-6.3,-1.0,-1.7,0.0,6.98,-0.1,-9.4,-4.4,84.6,59.1,74.4,2020,1,5)
(54933,2020-01-06,2.622,-99.13,45.71,-1.4,-10.1,-5.8,-5.5,0.0,6.81,-3.4,-14.1,-8.6,86.7,66.9,74.7,2020,1,6)
(54933,2020-01-07,2.622,-99.13,45.71,-4.8,-18.3,-11.5,-10.2,0.0,6.39,-6.4,-23.4,-12.2,86.1,57.0,72.7,2020,1,7)
(54933,2020-01-08,2.622,-99.13,45.71,-11.7,-18.2,-15.0,-14.4,0.0,4.67,-12.5,-22.3,-14.7,85.4,71.3,78.2,2020,1,8)
(54933,2020-01-09,2.622,-99.13,45.71,-5.2,-13.9,-9.6,-10.2,0.0,4.61,-4.8,-15.2,-11.0,92.5,71.0,82.5,2020,1,9)
(54933,2020-01-10,2.622,-99.13,45.71,-13.5,-22.7,-18.1,-17.8,0.0,6.39,-13.4,-28.0,-18.9,79.4,62.1,70.1,2020,1,10)
grunt>
```

Output 4.20 Pig limit operator script

```
STORE limit10_low_temp_daily_avg INTO
'/pig/outputs/limit10_low_temp_daily_avg';

months_group = GROUP data BY MONTH;
limit10_months_group = LIMIT months_group 10;
DUMP limit10_months_group;
```

```
STORE limit10_months_group INTO '/pig/outputs/
limit10_months_group ';
```

Used to display the logical, physical, and MapReduce execution plans of a relationship.

```
EXPLAIN data;
```

```
|    |    |
|    |    (Name: Cast Type: chararray Uid: 21)
|    |    |
|    |    |---WBANNO:(Name: Project Type: bytearray Uid: 21 Input: 0 Column: (*))
|    |    |
|    |    (Name: Cast Type: chararray Uid: 22)
|    |    |
|    |    |---LSAT_DATE:(Name: Project Type: bytearray Uid: 22 Input: 1 Column: (*))
|    |    |
|    |    (Name: Cast Type: float Uid: 23)
|    |    |
|    |    |---CRX_VN:(Name: Project Type: bytearray Uid: 23 Input: 2 Column: (*))
|    |    |
|    |    (Name: Cast Type: float Uid: 24)
|    |    |
|    |    |---LONGITUDE:(Name: Project Type: bytearray Uid: 24 Input: 3 Column: (*))
|    |    |
|    |    (Name: Cast Type: float Uid: 25)
|    |    |
|    |    |---LATITUDE:(Name: Project Type: bytearray Uid: 25 Input: 4 Column: (*))
|    |    |
|    |    (Name: Cast Type: float Uid: 26)
|    |    |
|    |    |---T_DAILY_MAX:(Name: Project Type: bytearray Uid: 26 Input: 5 Column: (*))
|    |    |
|    |    (Name: Cast Type: float Uid: 27)
|    |    |
|    |    |---T_DAILY_MIN:(Name: Project Type: bytearray Uid: 27 Input: 6 Column: (*))
|    |    |
|    |    (Name: Cast Type: float Uid: 28)
|    |    |
|    |    |---T_DAILY_MEAN:(Name: Project Type: bytearray Uid: 28 Input: 7 Column: (*))
|    |    |
|    |    (Name: Cast Type: float Uid: 29)
|    |    |
|    |    |---T_DAILY_AVG:(Name: Project Type: bytearray Uid: 29 Input: 8 Column: (*))
|    |    |
|    |    (Name: Cast Type: float Uid: 30)
|    |    |
|    |    |---P_DAILY_CALC:(Name: Project Type: bytearray Uid: 30 Input: 9 Column: (*))
|    |    |
|    |    (Name: Cast Type: float Uid: 31)
|    |    |
|    |    |---SOLARAD_DAILY:(Name: Project Type: bytearray Uid: 31 Input: 10 Column: (*))
|    |    |
|    |    (Name: Cast Type: float Uid: 32)
|    |    |
```

```
|       |       |
|       |       |---SUR_TEMP_DAILY_MAX:(Name: Project Type: bytearray Uid: 32 Input: 11 Column: (*))
|       |       |
|       |       (Name: Cast Type: float Uid: 33)
|       |       |
|       |       |---SUR_TEMP_DAILY_MIN:(Name: Project Type: bytearray Uid: 33 Input: 12 Column: (*))
|       |       |
|       |       (Name: Cast Type: float Uid: 34)
|       |       |
|       |       |---SUR_TEMP_DAILY_AVG:(Name: Project Type: bytearray Uid: 34 Input: 13 Column: (*))
|       |       |
|       |       (Name: Cast Type: float Uid: 35)
|       |       |
|       |       |---RH_DAILY_MAX:(Name: Project Type: bytearray Uid: 35 Input: 14 Column: (*))
|       |       |
|       |       (Name: Cast Type: float Uid: 36)
|       |       |
|       |       |---RH_DAILY_MIN:(Name: Project Type: bytearray Uid: 36 Input: 15 Column: (*))
|       |       |
|       |       (Name: Cast Type: float Uid: 37)
|       |       |
|       |       |---RH_DAILY_AVG:(Name: Project Type: bytearray Uid: 37 Input: 16 Column: (*))
|       |       |
|       |       (Name: Cast Type: int Uid: 38)
|       |       |
|       |       |---YEAR:(Name: Project Type: bytearray Uid: 38 Input: 17 Column: (*))
|       |       |
|       |       (Name: Cast Type: int Uid: 39)
|       |       |
|       |       |---MONTH:(Name: Project Type: bytearray Uid: 39 Input: 18 Column: (*))
|       |       |
|       |       (Name: Cast Type: int Uid: 40)
|       |       |
|       |       |---DAY:(Name: Project Type: bytearray Uid: 40 Input: 19 Column: (*))
|       |
|       |---(Name: LOInnerLoad[0] Schema: WBANNO#21:bytearray)
|       |
|       |---(Name: LOInnerLoad[1] Schema: LSAT_DATE#22:bytearray)
|       |
|       |---(Name: LOInnerLoad[2] Schema: CRX_VN#23:bytearray)
|       |
|       |---(Name: LOInnerLoad[3] Schema: LONGITUDE#24:bytearray)
|       |
|       |---(Name: LOInnerLoad[4] Schema: LATITUDE#25:bytearray)
```

```
|       |
|       |---(Name: LOInnerLoad[5] Schema: T_DAILY_MAX#26:bytearray)
|       |
|       |---(Name: LOInnerLoad[6] Schema: T_DAILY_MIN#27:bytearray)
|       |
|       |---(Name: LOInnerLoad[7] Schema: T_DAILY_MEAN#28:bytearray)
|       |
|       |---(Name: LOInnerLoad[8] Schema: T_DAILY_AVG#29:bytearray)
|       |
|       |---(Name: LOInnerLoad[9] Schema: P_DAILY_CALC#30:bytearray)
|       |
|       |---(Name: LOInnerLoad[10] Schema: SOLARAD_DAILY#31:bytearray)
|       |
|       |---(Name: LOInnerLoad[11] Schema: SUR_TEMP_DAILY_MAX#32:bytearray)
|       |
|       |---(Name: LOInnerLoad[12] Schema: SUR_TEMP_DAILY_MIN#33:bytearray)
|       |
|       |---(Name: LOInnerLoad[13] Schema: SUR_TEMP_DAILY_AVG#34:bytearray)
|       |
|       |---(Name: LOInnerLoad[14] Schema: RH_DAILY_MAX#35:bytearray)
|       |
|       |---(Name: LOInnerLoad[15] Schema: RH_DAILY_MIN#36:bytearray)
|       |
|       |---(Name: LOInnerLoad[16] Schema: RH_DAILY_AVG#37:bytearray)
|       |
|       |---(Name: LOInnerLoad[17] Schema: YEAR#38:bytearray)
|       |
|       |---(Name: LOInnerLoad[18] Schema: MONTH#39:bytearray)
|       |
|       |---(Name: LOInnerLoad[19] Schema: DAY#40:bytearray)
```

Output 4.21 Pig statistic scripts 2

Hive	Pig
Hive is commonly used by Data Analysts.	Pig is commonly used by programmers.
It follows SQL-like queries.	It follows the data-flow language.
It can handle structured data.	It can handle semi-structured data.
It works on server-side of HDFS cluster.	It works on client-side of HDFS cluster.
Hive is slower than Pig.	Pig is comparatively faster than Hive.

Figure 4.17 Pig and Hive comparison table

4.6.2 Apache Spark

Apache Spark is an open-source library that allows us to process big data in parallel. Parallel processing allows for faster processing such as analytics and machine learning by dividing large data into clusters. Divided placements are processed and calculated with a calculation such as the average.

Apache Spark's main feature and processing speed boost is in-memory cluster computing.

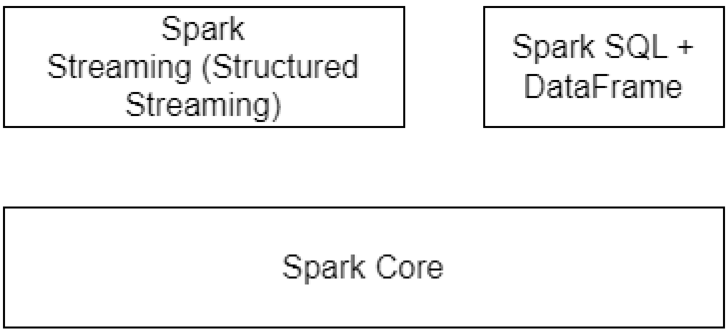


Figure 4.18 Spark ecosystem I use

Spark Core: It is the essential spark engine for large-scale parallel and distributed data processing. The kernel has low-level APIs (RDD) and structured APIs (dataframe, dataset).

Spark Streaming: One of the spark components used to process and analyze real-time data. Analysis results can be exported to HDFS, database and dashboards. It divides the incoming data into groups and is analyzed as an RDD array. Analysis results are transferred to the target system in batches.

Spark SQL: It is the SQL component that works with Spark's structured data. It is used to build Spark-specific queries.

Weather Data Analysis with Spark Streaming

```
~/.local/bin/jupyter-notebook
```

I imported the libraries I will use

```
pip install pyspark

from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

I am creating a base with the `SparkSession` class.

```
spark =
SparkSession.builder.appName("WeatherDataAnalysis").getOrCreat
e()
```

I created a collection of `StructField` objects with `StructType. StructField` objects contain the column names and types in the dataset.

```
schema = StructType([StructField('WBANNO', IntegerType(),
True),
            StructField('LST_DATE', StringType(), True),
            StructField('CRX_VN', FloatType(), True),
            StructField('LONGITUDE', FloatType(), True),
            StructField('LATITUDE', FloatType(), True),
            StructField('T_DAILY_MAX', FloatType(), True),
            StructField('T_DAILY_MIN', FloatType(), True),
            StructField('T_DAILY_MEAN', FloatType(), True),
            StructField('T_DAILY_AVG', DoubleType(), True),
            StructField('P_DAILY_CALC', FloatType(), True),
            StructField('SOLARAD_DAILY', FloatType(), True),
            StructField('SUR_TEMP_DAILY_MAX', FloatType(),
True),
            StructField('SUR_TEMP_DAILY_MIN', FloatType(),
True),
            StructField('SUR_TEMP_DAILY_AVG', FloatType(),
True),
            StructField('RH_DAILY_MAX', FloatType(), True),
            StructField('RH_DAILY_MIN', FloatType(), True),
            StructField('RH_DAILY_AVG', FloatType(), True),
            StructField('YEAR', IntegerType(), True),
            StructField('MONTH', IntegerType(), True),
            StructField('DAY', IntegerType(), True),
           ])
```

I read the data set with .csv extension, which we stored in HDFS, with the `Spark.readStream` function. The important point here is to give the data file the correct HDFS path.

```
streaming = spark\
    .readStream\
    .option("header", "true")\
```

```
    .schema(schema)\
.csv("http://localhost:9870/explorer.html#//weather_data*.csv"
)
```

I did some data analysis. The purpose of this analysis process is to find the average air temperature by month and state ID. I used the "T_DAILY_MEAN" column because the daily data in our data are used in separate columns in the format "min, max, average and mean" due to their changes during the day.

```
wbenno_month_mean =
streaming.groupBy("MONTH","WBANNO").mean("T_DAILY_MEAN")
```

I need to create a query to see my analysis. I saved the content of the stream query to external storage with the `writeStream` function. I preferred the **console** format to see my output in the console. The **outputMode("complete")** operation writes all rows in the stream data when there are some updates.

```
activityQuery = (

wbenno_month_mean.writeStream.queryName("wbenno_month_mean")

.option("checkpointLocation","/home/berfin/WeatherDataAnalysis
") \
    .format("console")
    .outputMode("complete")
    .start()
)
```

```
-------------------------------------------
Batch: 0
-------------------------------------------
+-----+------+------------------+
|MONTH|WBANNO|  avg(T_DAILY_MEAN)|
+-----+------+------------------+
|    6| 53974| 25.028333282470705|
|   12| 96404|-19.932258032021984|
|    9| 22016| 24.395000076293947|
|    1|  4136| 0.6209677293175652|
|    5| 54856|  14.17580646084201|
|    7| 92827| 27.701612903225808|
|   11| 94075|-0.1899999901652336|
|    3|  4128|  2.282258065477494|
|    6|  4127| 18.241666650772096|
|   11| 53878|   8.18666667342186|
|    7|  4990| 22.545161308780795|
|    9| 25630|   9.02666668097178|
|   11| 26494|  -3.92833360756437|
|    3|  3061| 2.8145161194186055|
|    2| 73801|    9.2280701864184|
|   10|  3728|  20.91451615671958|
|    8|  3758| 25.469354844862416|
|   11| 63895|  9.745000020662944|
|    7|  4222|  27.48064514898485|
|   12| 53151| 13.925806383932791|
+-----+------+------------------+
only showing top 20 rows
```

Output 4.22 Spark notebook output of the analysis

```
-------------------------------------------
Batch: 0
-------------------------------------------
+-----+------+-------------------+
|MONTH|WBANNO|  avg(T_DAILY_MEAN)|
+-----+------+-------------------+
|    6| 53974| 25.028333282470705|
|   12| 96404|-19.932258032021984|
|    9| 22016| 24.395000076293947|
|    1|  4136| 0.6209677293175652|
|    5| 54856|  14.17580646084201|
|    7| 92827| 27.701612903225808|
|   11| 94075|-0.1899999901652336|
|    3|  4128|  2.282258065477494|
|    6|  4127| 18.241666650772096|
|   11| 53878|   8.18666667342186|
|    7|  4990| 22.545161308780795|
|    9| 25630|   9.02666668097178|
|   11| 26494|  -3.92833360756437|
|    3|  3061| 2.8145161194186055|
|    2| 73801|    9.2280701864184|
|   10|  3728|  20.91451615671958|
|    8|  3758| 25.469354844862416|
|   11| 63895|  9.745000020662944|
|    7|  4222|  27.48064514898485|
|   12| 53151| 13.925806383932791|
+-----+------+-------------------+
only showing top 20 rows
```

Output 4.23 Spark console output of the analysis

Word count operation with Spark streaming:

We will implement a MapReduce operation, word count example using Spark Streaming and HDFS. First, let's start with what MapReduce means. The MapReduce model consists of two functions, Map and Reduce. In the map phase, the data is read in HDFS. Data is partitioned and sent to each partition as a key-value pair. These data pairs are stored as intermediate data. When the map function is complete, the reduction function starts. The reduction process starts with the step of sorting and grouping the key-value pairs of the intermediate data. The result of the operation is stored as a reduced and combined result output. We can now proceed to the proceedings.

First of all, we need to have text data. We create a text file named “word_count_data.txt” and insert the data into it. I will not use weather data in this section. We will perform all operations via cmd. Let's be careful to open the files from the terminal in the folder you created.

```
#to create a file
touch word_count_data.txt

# to open the created file
nano word_count_data.txt
```

We paste the data text into the opened file. We should do the same for mapper.py and reducer.py files.

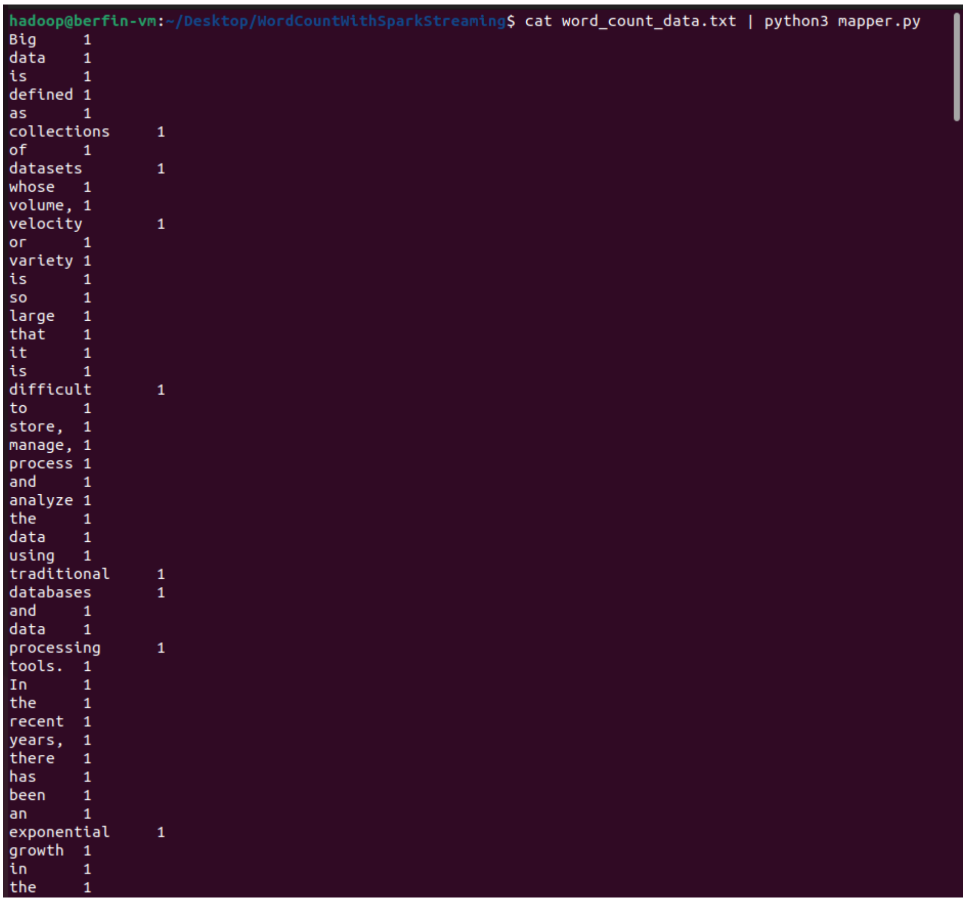
```
touch mapper.py
nano mapper.py
```

Let's add the following lines of code into the “mapper.py” file

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

for line in sys.stdin:
     line = line.strip()
    words = line.split()

  for word in words:
        print( '%s\t%s' % (word, 1))
```

I run the file to check if the map function is working.

```
cat word_count_data.txt | python3 mapper.py
```



```
hadoop@berfin-vm:~/Desktop/WordCountWithSparkStreaming$ cat word_count_data.txt | python3 mapper.py
Big	1
data	1
is	1
defined	1
as	1
collections	1
of	1
datasets	1
whose	1
volume,	1
velocity	1
or	1
variety	1
is	1
so	1
large	1
that	1
it	1
is	1
difficult	1
to	1
store,	1
manage,	1
process	1
and	1
analyze	1
the	1
data	1
using	1
traditional	1
databases	1
and	1
data	1
processing	1
tools.	1
In	1
the	1
recent	1
years,	1
there	1
has	1
been	1
an	1
exponential	1
growth	1
in	1
the	1
```

Output 4.24 Map function in word count operation

```
touch reducer.py
nano reducer.py
```

Let's add the following lines of code into the “reducer.py” file.

```
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print( '%s\t%s' % (current_word, current_count))
        current_count = count
        current_word = word

if current_word == word:
    print( '%s\t%s' % (current_word, current_count))
```

Now let's check if the map and shrink functions work together and harmoniously.

```
cat word_count_data.txt | python3 mapper.py | sort -k1,1 |
python3 reducer.py
```

```
hadoop@berfin-vm:~/Desktop/WordCountWithSparkStreaming$ cat word_count_data.txt | python3 mapper.py |
sort -k1,1 |  python3 reducer.py
(1)     1
(2)     1
(3)     1
(5)     1
a       3
an      1
analysis        4
analytics       6
analytics.      1
analyze 2
analyzed        1
and     22
applications    2
Applications    1
architectures   1
are     4
as      5
backend.        1
banking 1
Based   1
(batch  1
be      4
been    1
big     2
Big     7
both    1
by      2
can     2
cleansing,      1
cloud   1
cluster 1
collected       1
collection      1
collection,     1
collections     1
computational   1
computing,      2
cyber-physical  1
data    25
data.   1
Data    3
databases       1
databases)      1
databases,      1
```

Output 4.25 Map-reduce function in word count operation

NOTE: sort=sort lines of text files, -k,1= start a key on origin and end on last line.

We have completed the word counting process. But our goal is to take the data file to be used for word counting from hdfs and save the output to HDFS. For this, we first transfer the data to HDFS. I will do this using HDFS commands.

```
hdfs dfs -mkdir /word_count
hdfs dfs -copyFromLocal
/home/hadoop/Desktop/WordCountWithSparkStreaming/word_count_da
ta.txt /word_count
```

Now let's give executable permissions to “mapper.py” and “reducer.py” files.

```
chmod 777 mapper.py reducer.py
```

```
hadoop@berfin-vm:~/Desktop/WordCountWithSparkStreaming$ ls -l
total 152
-rw-rw-rw- 1 hadoop hadoop 141265 May 26 14:49 hadoop-streaming-3.3.5.jar
-rwxrwxrwx 1 hadoop hadoop    189 May 26 18:10 mapper.py
-rwxrwxrwx 1 hadoop hadoop    593 May 26 18:10 reducer.py
-rw-rw-r-- 1 hadoop hadoop   2490 May 25 22:54 word_count_data.txt
```

Output 4.26 HDFS list of files

We need to download the hadoop-streaming jar file so that we can do Hadoop and streaming together.[15]

NOTE: Make sure the file version is the same or compatible with the hadoop version you are using.

Now let's run python with Hadoop release utility using another cmd.

```
hadoop jar
/home/hadoop/Desktop/WordCountWithSparkStreaming/hadoop-stream
ing-3.3.5.jar -input /word_count/word_count_data.txt -output
/word_count/output_data -mapper
/home/hadoop/Desktop/WordCountWithSparkStreaming/mapper.py
-reducer
/home/hadoop/Desktop/WordCountWithSparkStreaming/reducer.py
```

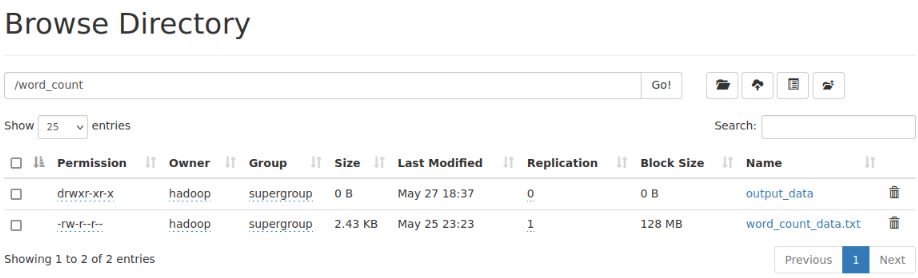


Figure 4.19 Word-count data in HDFS 1

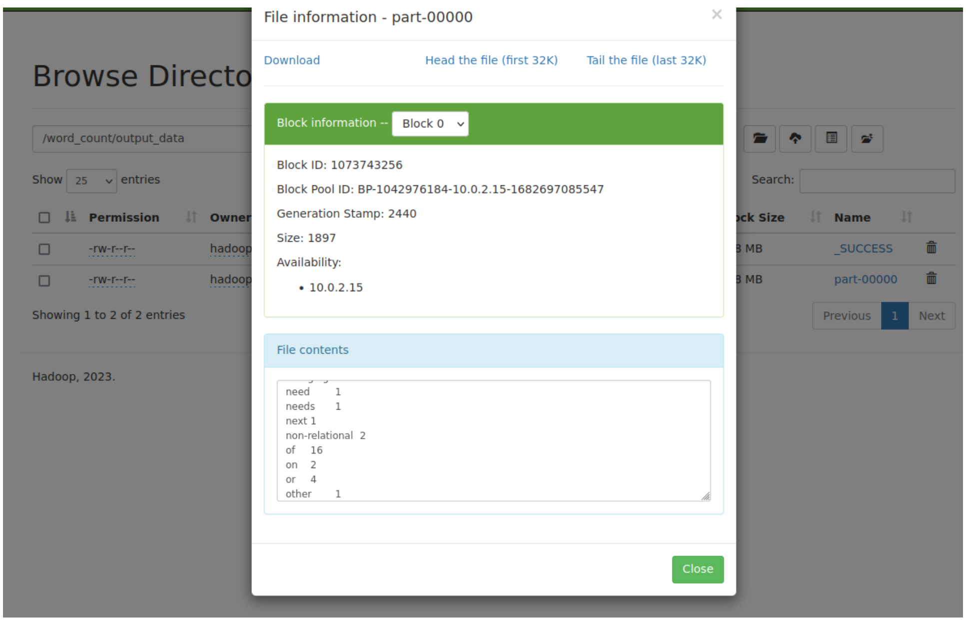


Figure 4.20 Word-count data in HDFS 2

References

[1] https://hadoop.apache.org/

[2] https://pig.apache.org/

[3] https://spark.apache.org/

[4] https://flume.apache.org

[5] https://hive.apache.org/

[6] https://sqoop.apache.org/

[7] https://www.rabbitmq.com/

[8] https://hbase.apache.org/

[9] Book of Big Data: Big Data Analytics: A Hands-On Approach Copyright © 2019 by Arshdeep Bahga & Vijay Madisetti

[10] https://ubuntu.com/download/desktop/thank-you?version=22.04.1&architecture=amd64

[11] https://www.jetbrains.com/pycharm/download/other.html

[12] http://127.0.1.1:15672

[13] https://www.ncei.noaa.gov/pub/data/uscrn/products/daily01/

[14] https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[15]
https://jar-download.com/artifacts/org.apache.hadoop/hadoop-streaming/3.3.5/source-code

APPENDIX A: SOURCE CODE

1 Python Code for Weather Data Preprocessing

```
import os.path
import pandas as pd

def data_preparation(year):
    file_path = f"./RawData/{year}"
    new_data = open(f"./RawData/{year}_data.txt", "w",
encoding='utf-8')
    files = os.listdir(file_path)
    for file in files:
        cleaned = ""
        with open(os.path.join(file_path, file), "r",
encoding='utf-8') as f:
            data = f.readlines()
            for line in data:
                cleaned = " ".join(line.split())
                cleaned = cleaned.replace(" ", ",") + "\n"
                new_data.write(cleaned)
    new_data.close()
    df = pd.read_csv(f'./RawData/{year}_data.txt',
                     names=['WBANNO', 'LST_DATE', 'CRX_VN',
'LONGITUDE', 'LATITUDE', 'T_DAILY_MAX', 'T_DAILY_MIN',
'T_DAILY_MEAN', 'T_DAILY_AVG', 'P_DAILY_CALC',
                            'SOLARAD_DAILY',
'SUR_TEMP_DAILY_TYPE', 'SUR_TEMP_DAILY_MAX',
'SUR_TEMP_DAILY_MIN','SUR_TEMP_DAILY_AVG','RH_DAILY_MAX',
'RH_DAILY_MIN',
                            'RH_DAILY_AVG',
'SOIL_MOISTURE_5_DAILY', 'SOIL_MOISTURE_10_DAILY',
'SOIL_MOISTURE_20_DAILY', 'SOIL_MOISTURE_50_DAILY',
'SOIL_MOISTURE_100_DAILY',
                            'SOIL_TEMP_5_DAILY',
'SOIL_TEMP_10_DAILY', 'SOIL_TEMP_20_DAILY',
'SOIL_TEMP_50_DAILY', 'SOIL_TEMP_100_DAILY'])
    df =
df.drop(['SUR_TEMP_DAILY_TYPE','SOIL_MOISTURE_5_DAILY','SOIL_M
OISTURE_10_DAILY','SOIL_MOISTURE_20_DAILY','SOIL_MOISTURE_20_D
AILY','SOIL_MOISTURE_50_DAILY',

'SOIL_MOISTURE_100_DAILY','SOIL_TEMP_5_DAILY','SOIL_TEMP_10_DA
ILY','SOIL_TEMP_20_DAILY','SOIL_TEMP_50_DAILY','SOIL_TEMP_100_
DAILY'], axis=1)
    df.to_csv(f'./RawData/raw_weather_data_{year}.csv',
index=False)

def data_preprocessing(df):
    df = df.copy()
```

```
    for col in df.columns:
        for row_i, _ in df.loc[(df[col] == -9999.0) | (df[col]
== -99.000) , col].items():
            df[col][row_i] = df[col].tolist()[row_i - 1]

    df['LST_DATE'] = pd.to_datetime(df['LST_DATE'],
format='%Y%m%d')
    df[["YEAR", "MONTH", "DAY"]] =
df["LST_DATE"].astype(str).str.split("-", expand=True)
    df['YEAR'] = pd.to_numeric(df['YEAR'])
    df['MONTH'] = pd.to_numeric(df['MONTH'])
    df['DAY'] = pd.to_numeric(df['DAY'])

    return df

if __name__ == '__main__':
    data_preparation(2020)
    data_preparation(2021)
    df_2020 =
pd.read_csv('./RawData/raw_weather_data_2020.csv')
    df_2021 =
pd.read_csv('./RawData/raw_weather_data_2021.csv')
    #concat 2020's dataframe and 2021's dataframe
    df = pd.concat([df_2020, df_2021])
    #df = data_preprocessing(df)
    #save dataframe as csv file
    df.to_csv('./OutputData/weather_data.csv', index=False)
    #save dataframe as txt file
    df.to_csv('./OutputData/weather_data.txt', header=None,
index=None, sep=' ', mode='a')
    df = pd.read_csv('./OutputData/weather_data.csv')
    #df = pd.read_csv('./OutputData/weather_data.txt')
    print(df.info())
```

2 Python Code for RabbitMQ Message Queue

receive.py

```
import pika, sys, os

def main():
     connection =
pika.BlockingConnection(pika.ConnectionParameters(host='localh
ost'))    channel = connection.channel()

     channel.queue_declare(queue='bigdata')

     def callback(ch, method, properties, body):
          print(" [x] Received %r" % body)

     channel.basic_consume(queue='bigdata',
     on_message_callback=callback, auto_ack=True)

     print(' [*] Waiting for messages. To exit press CTRL+C')
     channel.start_consuming()

if __name__ == '__main__':
     try:
          main()
     except KeyboardInterrupt:
          print('Interrupted')
          try:
               sys.exit(0)
          except SystemExit:
               os._exit(0)
```

send.py

```
import pika

connection = pika.BlockingConnection(
pika.ConnectionParameters(host='localhost')) channel =
connection.channel()

channel.queue_declare(queue='bigdata')

channel.basic_publish(exchange='', routing_key='bigdata',
body='I Love Big Data!')
print(" [x] Sent 'I Love Big Data!'")
connection.close()
```