

Project Report
On
“Descriptive Analysis of Weather Data”

CPSC 531-01 13480

Advanced Database Management

Spring, 2023

Under Guidance of

Prof. Tseng-Ching James Shen

Department of Computer Science



California State University

Fullerton, CA - 92831

May 2023

Prepared By:

Anirudh Venkatesh (885174318) anirudh.venkatesh@csu.fullerton.edu

Vedita Deshpande (885177469) vedita@csu.fullerton.edu

Revision History

Version Number	Date of release	Remarks
Version 1.0	14 May 2023	First Version

Intended Audience

This document is intended for:

1. System Engineers
2. Solution Integrators

Table of Contents

1) Introduction.....	3
2) Functionalities.....	3
3) Dataset.....	5
4) Architecture & Design (Technology and Tools Used).....	6
5) Minimum System Requirements.....	7
6) GitHub Location of Code.....	8
7) Deployment Instructions.....	8
8) Steps to Run the Application.....	17
9) Steps to safely shutdown the system.....	18
10) Test Results.....	18
11) Conclusion.....	20
12) References.....	21

1) Introduction

Weather is an integral part, having a significant effect on our day-to-day activities. Due to the vast array of tasks and operations ranging from personal tasks to more large-scale industrial tasks that get affected by weather, weather analysis has been a highly significant and impactful field of study.

There has been intense research conducted to perform weather-related analysis and judgements that can act as guides for taking actions in the future.

Weather experts have systems that collect huge amounts of data about everyday weather, consisting of information on factors such as rain, humidity, day length, wind speed, and direction, etc. This data is collected throughout the day and fed into the data storage systems designed to store it. This generates the need for more efficient systems to store and analyze this huge amount of data over a long period of time and for a large number of areas under analysis.

The project aims to solve this problem by using the big data technologies like Hadoop, Hive and Spark SQL for weather-analysis tasks. The project proposes two methods for the analysis from the point of view of a weather expert –

Graphical Analysis and Command Line analysis are developed in a way so that the weather-expert can use both the tools to gain meaningful insights from a huge amount of weather data for a long time period for a wide range of attributes under consideration.

2) Functionalities

The design includes input from online transaction processing systems in different cities in the United States. This implementation mimics the actual data storage systems in the real world which have independent storage servers that collect data specific to a city.

The data from multiple such storage servers are gathered and injected into the Hadoop Distributed File System using the tool 'Sqoop.' Sqoop collects data from different cities and creates a secure connection to deliver the daily data collection into the standard HDFS cluster for analyzing this weather data.

The example used in the project uses data collected from 5 different cities – San Francisco, New York City, Austin, Miami, and Chicago. This model can be extended to multiple cities to be implemented in the real world further to store the big data into a Hadoop distributed file system.

After the data is stored into the Hadoop Distributed File System, Hive queries can be run on the system to get the desired output for analyzing the required weather data.

The project consists of two implementations for this –

1. Hive CLI & Spark CLI

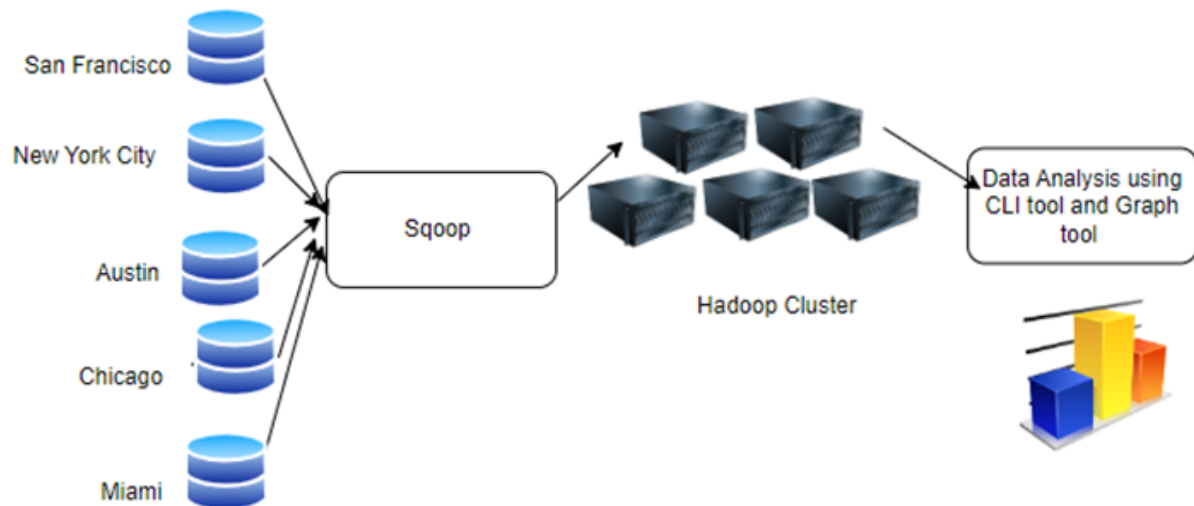
- a. The custom CLI tool developed, uses Hive queries in the background that will generate MapReduce jobs that are automatically on top of HDFS and Yarn. Thus the user can run custom queries and have more flexibility with the data analysis. The output from these queries can generate meaningful data that can be further used to gain good insight into the weather pattern and can also be used for prediction activities.

- b. The application also offers Spark CLI as a tool for running Hive queries on top of HDFS and Yarn. This is implemented using Hive Integration offered as part of the Spark framework used to run SQL queries for the data stored in Hadoop. Queries running on spark are faster than the queries running using the Hive CLI which can give the additional advantage of speed to the user to perform a quicker analysis.

2. Graphical Analysis

The application consists of a python app that can be directly triggered by the weather expert to generate graphical reports of the required weather-related fields stored in Hadoop. This is done using MatPlot library in python which is

a powerful tool to generate a variety of graphs depending on the type of analysis required by the weather-expert using the application.



3) Dataset

Reference Link - <https://www.wunderground.com/history>

- For testing purposes we used the data collected from - <https://www.wunderground.com/history> to mimic independent OLTP systems that can be used as input for the project. This design can be easily scaled to support multiple cities further.
- The database systems for the cities include several weather parameters collected throughout the day that are stored in the form of rows and column. Parameters like high temp , low temp, dew point, day length, humidity , wind etc are used for the project.
- We used independent datasets for a sample of 5 cities (New York City, Chicago, Miami, Austin and San Francisco) for the project that are used as input for Sqoop to inject the data into the Hadoop systems

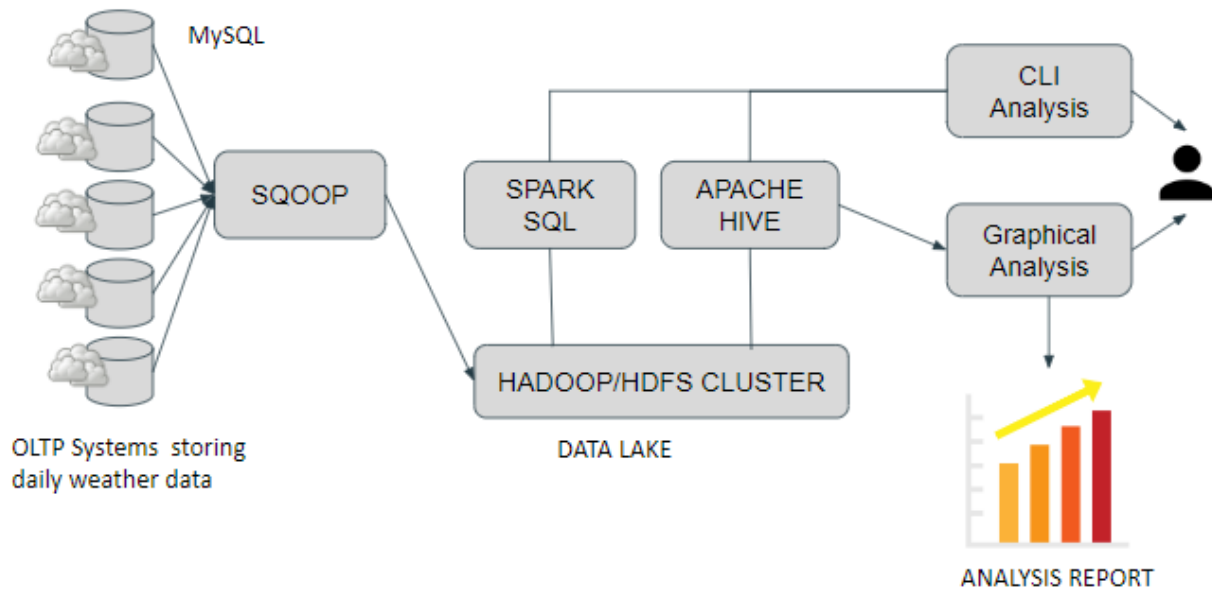
Sample Dataset used -

Date	High Temp	Low Temp	Precipitation	Dew Point	Wind	Humidity	Sea Level Pressure	Day Length
3/1/2022	48	31	0	38	18	76	30.3	680
3/2/2022	52	41	0	35	16	74	30	681
3/3/2022	47	26	0.03	36	25	73	30.4	685
3/4/2022	38	22	0	12	15	48	30.6	687
3/5/2022	45	31	0	32	14	67	30.5	688
3/6/2022	68	39	0	56	18	84	30.2	689
3/7/2022	74	51	0.05	55	32	67	29.9	700
3/8/2022	47	40	0.05	28	33	48	30.2	703
3/9/2022	40	33	0.01	32	15	92	30.2	705
3/10/2022	49	32	0.57	34	10	92	30.2	705

3) **Architecture & Design**

Tools and technologies used -

- Apache Hadoop 2.6.5
- Apache Hive 2.3.5
- Sqoop 1.4.7 for Hadoop 2.6.0
- MySQL 8.0.32
- MySQL connector JAVA 8.0.17
- OpenJDK 11.0.18
- Python 3.10
- Paramiko
- Matplotlib
- Spark 2.4.6 for Hadoop 2.6
- OpenSSH
- Oracle Virtualbox
- Scala



The system consists of input data sources as part of independent OLTP systems running using MySQL databases. Using Sqoop as an ingestion tool, these data sources are combined in a uniform manner in order to be injected into a common cluster comprising the Hadoop nodes.

Once the data is injected into the hadoop data nodes, Apache Hive and Spark SQL can be used to run SQL-like queries on top of the running Hadoop Cluster to get the weather-related data for analysis.

We initially developed our application using Apache Hive to run the SQL-like queries but later found out about the implementation use case using Spark SQL, which improved the query time significantly making it easier to get timely weather reports. Spark SQL can be easily integrated as part of an existing Hadoop System to leverage the Spark features without the need for a complete Apache Spark setup.

After getting the results from Apache Hive and Spark SQL we gather the data to present it in the form of graphs that can be used by weather experts to do further analysis on the weather data.

This feature enables non-technical people to leverage the advantages of big data technologies using the graphical analysis tool developed primarily in Python, for analyzing the weather data.

5) Minimum System Requirements

Every system that is planned to be a part of the cluster must satisfy the following hardware requirements:

- 1.5 GB RAM (2GB recommended)

- 20 GB Disk Space
- 2 CPU Cores (3 CPU Cores recommended)
- Ethernet
- Hypervisor to support virtualization

6) GitHub Location of Code



[Descriptive Analysis of Weather Data \(GitHub\)](#)

7) Deployment Instructions

The following steps are to be followed to deploy the application:

1. Setup Virtual Machines with requirements specified in Section 4. Number of virtual machines depends on the number of Datanodes required.
2. Fetch the disc Image file of ubuntu 20.04 LTS from the official [Ubuntu](#) website.
3. Load the disk image file onto the virtual machine's and run the VMs. Start the VMs and follow instructions on screen to install Ubuntu on the machine.

Note: Basic Installation is sufficient. Complete Installation is not required.

4. Obtain the ip address of each of the VM by using the following command:
ifconfig
5. Modify the /etc/hosts file with the ip address of each of the VMs. Format is:
ip-address Hostname-of-that-vm

A sample is given below:

192.168.4.11	Namenode
192.168.4.12	Datanode-1
192.168.4.13	Datanode-2

6. Test if each machine is able to ping each other.
ping <ip-address>

If machines are able to ping each other, then the machines are able to communicate with each other.

If ping is not installed, then the same can be downloaded by using the command:
`sudo apt-get install -y iputils-ping`

7. Disable firewall on each of the VM

`ufw disable`

8. Install openssh-server and wget on each machine. Command is:

`sudo apt-get install -y openssh-server wget`

9. Create directories for Hadoop, Java, Hive, Spark. You may choose your own path but for the purposes of guidance, the instructions shall use the following paths for installation.

Hadoop	/usr/local/
Spark	/usr/local/
Hive	/usr/local/
Java	/usr/lib/

10. Download the Java SE Development kit 1.8 from the official [Oracle](#) website. Also download the [Apache Hadoop](#), [Apache Hive](#) and [Apache Spark](#) .

11. Untar all the downloaded files onto the respective paths. First, navigate to the target directory and execute the following command to untar the files:

`sudo tar -xvf <your-download-path>/jdk-8u201-linux-x64.tar.gz`

Follow the same for Hadoop, Hive and Spark as well.

12. Create a link folder for all the folders. This will help in changing versions in the future.

`sudo ln -s /usr/lib/jdk1.8.0_201 java`

Repeat for Hadoop, Spark and Hive.

13. Change ownership of all the directories

`sudo chown -R <your-user>:<you-user> java`

`sudo chown -R <your-user>:<your-user> jdk1.8.0_201`

Repeat for Hadoop, Hive and Spark.

14. Open the .bashrc file (located in /home/<user> directory) and append the following lines in it:

```
export JAVA_HOME=/usr/lib/java
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:$JAVA_HOME/sbin
```

```
export HADOOP_HOME=/usr/lib/hadoop
```

```
export PATH=$PATH:$HADOOP_HOME/bin
```

```
export HIVE_HOME=/usr/lib/hive  
export PATH=$PATH:$HIVE_HOME/bin
```

```
export SPARK_HOME=/usr/lib/spark  
export PATH=$PATH:$SPARK_HOME/bin
```

15. Run the bashrc file so that Ubuntu is aware of the paths.

```
source .bashrc
```

16. MySQL Server Configuration

- a. Install MySQL-Server on the Namenode

```
sudo apt-get install -y mysql-server
```
- b. Login to the MySQL-Server

```
sudo mysql -u root -p
```
- c. Configure MySQL to run without sudo
 - I.

```
use mysql;
```
 - II.

```
update user set plugin='mysql_native_password' where user='root';
```
 - III.

```
flush privileges;
```
- d. Configure MySQL Password
 - I.

```
use mysql;
```
 - II.

```
alter user 'root'@'localhost' IDENTIFIED BY 'NewPassword';
```
- e. Configure Timezone

```
SET GLOBAL time_zone = '+3:00';
```
- f. Configure remote access of MySQL Server
 - I.

```
GRANT ALL ON *.* to root@'%' IDENTIFIED BY 'root';
```
 - II.

```
flush privileges;
```

17. Import data in csv into MySQL Server

- a. Copy csv file into the secure mysql path

```
sudo cp <your-csv-file>.csv /var/lib/mysql-files
```
- b. Login to MySQL server

```
mysql -u root -p
```
- c. Create new database

```
create database <your-database-name>;
```

- d. Use database
Use <your-database-name>;
- e. Create MySQL table
create table <table-name>(<columns and datatypes>;
- f. Load data into the table
*LOAD DATA INFILE '/var/lib/mysql-files/<your-file-name>.csv' INTO TABLE
<table-name> FIELDS TERMINATED BY ',' IGNORE 1 LINES;*

18. Download mysql-jdbc driver from [MySQL](#)

19. Configure Hadoop (\$HADOOP_HOME/conf directory)

- a. hdfs-site.xml

For Namenode:

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

```
<property>
<name>dfs.namenode.name.dir</name>
<value>/adbms/name</value>
</property>
```

```
<property>
<name>dfs.datanode.data.dir</name>
<value>/adbms/data1</value>
<final>true</final>
</property>
```

```
<property>
<name>dfs.namenode.http-address</name>
<value>Namenode:50070</value>
</property>
```

```
<property>
<name>dfs.namenode.secondary.http-address</name>
<value>Datanode-1:50090</value>
</property>
```

For Datanode:

```

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

<property>
<name>dfs.datanode.data.dir</name>
<value>/adbms/data2</value>
<final>true</final>
</property>

<property>
<name>dfs.namenode.http-address</name>
<value>Namenode:50070</value>
</property>

<property>
<name>dfs.namenode.secondary.http-address</name>
<value>Datanode-1:50090</value>
</property>

```

b. yarn-site.xml

```

<property>
<name>yarn.resourcemanager.address</name>
<value>Namenode:9001</value>
</property>

<property>
<name>yarn.resourcemanager.resource-tracker.address</name>
<value>Namenode:8031</value>
</property>

<property>
<name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>

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

```

```
<property>
<name>yarn.nodemanager.pmem-check-enabled</name>
<value>>false</value>
</property>
```

```
<property>
<name>yarn.nodemanager.vmem-check-enabled</name>
<value>>false</value>
</property>
```

c. core-site.xml

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

d. mapred-site.xml

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

e. slaves

```
Namenode
Datanode-1
Datanode-2
```

20. Configure HIVE

a. Copy mysql-jdbc driver to \$HIVE_HOME/lib

b. hive-site.xml

For Namenode:

```
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://localhost/metastore?createDatabaseIfNotExist=true</value>
</property>
```

```
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.cj.jdbc.Driver</value>
</property>
```

```
<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>root</value>
</property>
```

```
<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>root</value>
</property>
```

```
<property>
  <name>hive.metastore.schema.validation</name>
  <value>>false</value>
</property>
```

For Datanode (add all above configurations and add additional configuration below)

```
<property>
  <name>hive.metastore.uris</name>
  <value>thrift://Namenode:9083</value>
</property>
```

21. Configure SPARK (\$SPARK_HOME)

- a. Copy mysql-jdbc driver to \$SPARK_HOME/lib
- b. Copy hive-site.xml from \$HIVE_HOME/conf to \$SPARK_HOME/conf
- c. Slaves (inside \$SPARK_HOME/conf)
 - Namenode
 - Datanode-1
 - Datanode-2
- d. spark-env.sh
 - export HADOOP_CONF_DIR=\$HADOOP_HOME/etc/hadoop
 - export SPARK_HOME=/usr/local/spark
 - export JAVA_HOME=/usr/lib/jdk1.8.0_201
 - export SPARK_MASTER_IP=Namenode

22. Perform HDFS Namenode format

Note: To be done only on Namenode and must be done only once.

hdfs namenode -format

```
Activities Terminal May 6 01:40 adbms@NameNode: ~
/.../lib/jersey-guice-1.9.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/lib/leveldbjni-all-1.8.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/lib/commons-io-2.4.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/lib/jackson-asl-1.9.13.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/lib/hamcrest-core-1.3.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/lib/log4j-1.2.17.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/lib/asm-3.2.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/lib/guice-3.0.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/lib/snappy-java-1.0.4.1.jar:/usr/local/hadoop-2.6.5/sha
re/hadoop/mapred
uce/lib/jackson-core-asl-1.9.13.jar:/usr/local/hadoop-2.6.5/sha
re/hadoop/mapred
uce/lib/junit-4.11.jar:/usr/local/hadoop-2.6.5/share/hadoop/map
red
uce/lib/xz-1.0.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/hadoop-map
red
uce-client-core-2.6.5.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/hadoop-map
red
uce-client-jobclient-2.6.5-tests.jar:/usr/local/hadoop-2.6.5/share/ha
doo
p/mapred
uce/hadoop-mapred
uce-client-common-2.6.5.jar:/usr/local/hadoop-2.6.5
/share/hadoop/mapred
uce/hadoop-mapred
uce-client-shuffle-2.6.5.jar:/usr/local/ha
doo
p-2.6.5/share/hadoop/mapred
uce/hadoop-mapred
uce-client-jobclient-2.6.5.jar:/
usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/hadoop-mapred
uce-examples-2.6.5.j
ar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/hadoop-mapred
uce-client-hs-2.
6.5.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/hadoop-mapred
uce-client-hs-
plugins-2.6.5.jar:/usr/local/hadoop-2.6.5/share/hadoop/mapred
uce/hadoop-mapr
educe-client-app-2.6.5.jar:/usr/local/hadoop/contrib/capacity-scheduler/*.jar
STARTUP_MSG: build = https://github.com/apache/hadoop.git -r e8c9fe0b4c252caf
2ebf1464220599650f119997; compiled by 'sjlee' on 2016-10-02T23:43Z
STARTUP_MSG: java = 1.8.0_201
*****
23/05/06 01:40:03 INFO namenode.NameNode: registered UNIX signal handlers for [
TERM, HUP, INT]
23/05/06 01:40:03 INFO namenode.NameNode: createNameNode [-format]
```

```
Activities Terminal May 6 01:40 adbms@NameNode: ~
= 30000
23/05/06 01:40:05 INFO namenode.FSNamesystem: Retry cache on namenode is enable
d
23/05/06 01:40:05 INFO namenode.FSNamesystem: Retry cache will use 0.03 of tota
l heap and retry cache entry expiry time is 600000 millis
23/05/06 01:40:05 INFO util.GSet: Computing capacity for map NameNodeRetryCache
23/05/06 01:40:05 INFO util.GSet: VM type = 64-bit
23/05/06 01:40:05 INFO util.GSet: 0.029999999329447746% max memory 966.7 MB = 2
97.0 KB
23/05/06 01:40:05 INFO util.GSet: capacity = 2^15 = 32768 entries
23/05/06 01:40:05 INFO namenode.NNConf: ACLs enabled? false
23/05/06 01:40:05 INFO namenode.NNConf: XAttr enabled? true
23/05/06 01:40:05 INFO namenode.NNConf: Maximum size of an xattr: 16384
23/05/06 01:40:05 INFO namenode.FSImage: Allocated new BlockPoolId: BP-18337360
14-192.168.4.3-1683351605480
23/05/06 01:40:05 INFO common.Storage: Storage directory /adbms/name has been s
uccessfully formatted.
23/05/06 01:40:05 INFO namenode.FSImageFormatProtobuf: Saving image file /adbms
/name/current/fsimage.ckpt_000000000000000000 using no compression
23/05/06 01:40:05 INFO namenode.FSImageFormatProtobuf: Image file /adbms/name/c
urrent/fsimage.ckpt_000000000000000000 of size 322 bytes saved in 0 seconds.
23/05/06 01:40:05 INFO namenode.NNStorageRetentionManager: Going to retain 1 im
ages with txid >= 0
23/05/06 01:40:05 INFO util.ExitUtil: Exiting with status 0
23/05/06 01:40:05 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at NameNode/192.168.4.3
*****
adbms@NameNode:~$
```

23. Start Hadoop for the first time. This will create data files as mentioned in the `hdfs-site.xml` configuration file.
start-all.sh

```

clusterID=CID-a0afe39c-6717-4d6f-a907-5426f5e7ced7
cTime=0
storageType=NAME_NODE
blockpoolID=BP-1833736014-192.168.4.3-1683351605480
layoutVersion=-60
adbms@Namenode:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [Namenode]
Namenode: starting namenode, logging to /usr/local/hadoop-2.6.5/logs/hadoop-adbms-namenode-Namenode.out
Namenode: starting datanode, logging to /usr/local/hadoop-2.6.5/logs/hadoop-adbms-datanode-Namenode.out
Datanode-1: starting datanode, logging to /usr/local/hadoop-2.6.5/logs/hadoop-adbms-datanode-Datanode-1.out
Datanode-2: starting datanode, logging to /usr/local/hadoop-2.6.5/logs/hadoop-adbms-datanode-Datanode-2.out
Starting secondary namenodes [Datanode-1]
Datanode-1: starting secondarynamenode, logging to /usr/local/hadoop-2.6.5/logs/hadoop-adbms-secondarynamenode-Datanode-1.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop-2.6.5/logs/yarn-adbms-resourcemanager-Namenode.out
Datanode-1: starting nodemanager, logging to /usr/local/hadoop-2.6.5/logs/yarn-adbms-nodemanager-Datanode-1.out
Datanode-2: starting nodemanager, logging to /usr/local/hadoop-2.6.5/logs/yarn-adbms-nodemanager-Datanode-2.out
Namenode: starting nodemanager, logging to /usr/local/hadoop-2.6.5/logs/yarn-adbms-nodemanager-Namenode.out
adbms@Namenode:~$

```

24. Perform Hive Schematool Initialization

schematool -dbType mysql -initSchema

```

adbms@Namenode:~$ schematool -dbType mysql -initSchema
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/apache-hive-2.3.5-bin/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.6.5/share/hadoop/common/lib/slf4j-log4j12-1.7.5.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.apache.logging.slf4j.Log4jLoggerFactory]
Metastore connection URL: jdbc:mysql://localhost/metastore?createDatabaseIfNotExist=true
Metastore Connection Driver : com.mysql.cj.jdbc.Driver
Metastore connection User: root
Starting metastore schema initialization to 2.3.0
Initialization script hive-schema-2.3.0.mysql.sql
Initialization script completed
schematool completed
adbms@Namenode:~$

```

25. On a new Terminal, run the Hive Metastore service

hive --service metastore

26. Run Spark Services

- Navigate to the Spark conf directory (\$SPARK_HOME/conf)
- Enter the Hadoop, Spark, Java paths and configure Spark Master IP on the spark-env.sh file


```
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export SPARK_HOME=/usr/local/spark
export JAVA_HOME=/usr/lib/jdk1.8.0_201
export SPARK_MASTER_IP=Namenode
adbms@Namenode:/usr/local/spark/conf$ S
```

- c. Enter the worker nodes on the slaves file
Namenode
Datanode-1
Datanode-2
- d. Navigate to the Spark sbin directory
`cd $SPARK_HOME/sbin`
- e. Run the start-all.sh
`./start-all.sh`

27. Install Python libraries

```
pip install paramiko
pip install matplotlib
```

8) Steps to Run the Application

Once deployed, running the application is fairly simple.

1. Start Hadoop
`start-all.sh`
2. Start Hive metastore service (to be run on new terminal)
`hive - -service metastore`
3. Navigate to spark home directory and start spark
`cd $SPARK_HOME/sbin`
`./start-all.sh`
4. Launch the Resource Manager GUI
<http://namenode:8088/cluster/apps>
5. Launch Hadoop GUI
<http://namenode:50070/>
6. Graphical Tool can be launched remotely by entering the machine ip address, username, password and entering the desired parameters.

7. Custom HiveUI sessions can also be created by entering ip address, username and password.

9) **Steps to safely shutdown the system**

The following steps are to be followed to safely shutdown the application:

1. Stop Spark Services
`cd $SPARK_HOME/sbin`
`./stop-all.sh`
2. Terminate HIVE metadata services
Navigate to the terminal running the hive metadata service and press CTRL + C to terminate the session.
3. Stop Hadoop
`stop-all.sh`

10) **Test Results**

HIVE QL running a successful Hadoop MapReduce job

Application application_1683865605880_0001

Not secure | namenode:8088/cluster/app/application_1683865605880_0001

Set up a WSL devel... CSU Student Login CSUF Pantry Home | reMarkable Online Rubik's Cub... Sequence Diagram Job Descriptions | S...

hadoop

Application application_1683865605880_0001

Cluster

- About
- Nodes
- Applications
 - NEW
 - NEW SAVING
 - SUBMITTED
 - ACCEPTED
 - RUNNING
 - FINISHED
 - FAILED
 - KILLED
- Scheduler

Tools

Kill Application

Application Overview

User: adbrms

Name: SELECT MAX(High_Temp)...Date'<=2023-03-31'(Stage-1)

Application Type: MAPREDUCE

Application Tags:

YarnApplicationState: FINISHED

FinalStatus Reported by AM: SUCCEEDED

Started: 12-May-2023 00:32:33

Elapsed: 44sec

Tracking URL: History

Diagnostics:

Application Metrics

Total Resource Preempted: <memory:0, vCores:0>

Total Number of Non-AM Containers Preempted: 0

Total Number of AM Containers Preempted: 0

Resource Preempted from Current Attempt: <memory:0, vCores:0>

Number of Non-AM Containers Preempted from Current Attempt: 0

Aggregate Resource Allocation: 133201 MB-seconds, 79 vcore-seconds

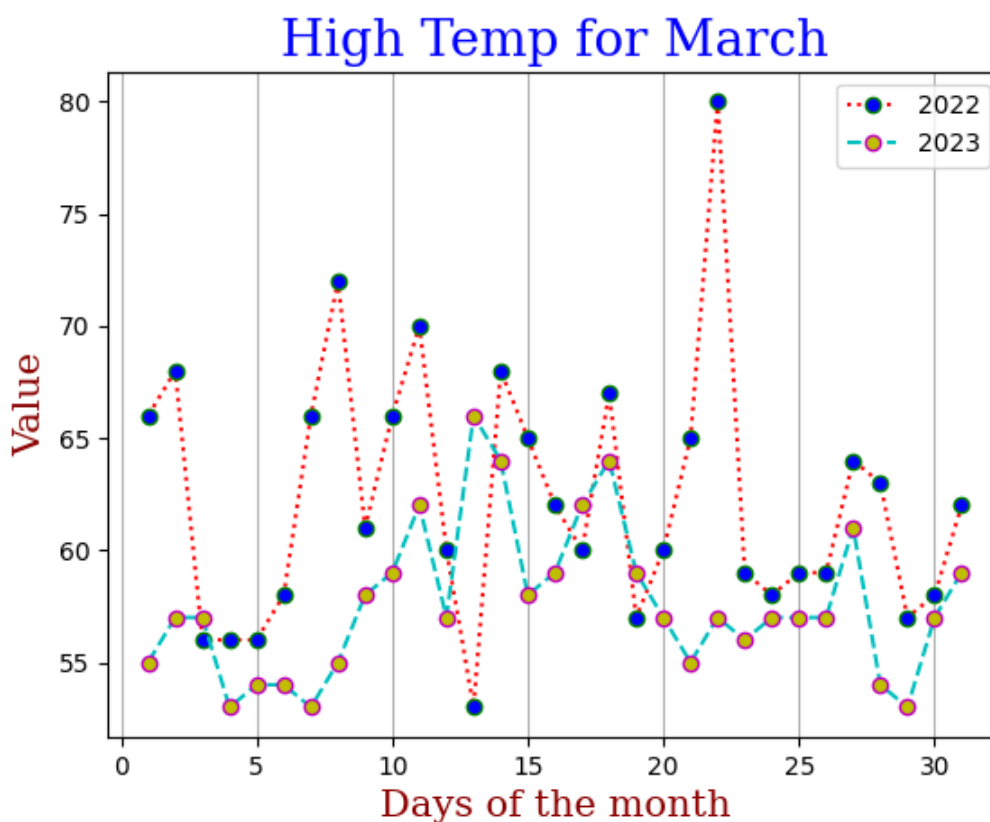
Show 20 entries

Attempt ID	Started	Node	Logs	Blacklisted Nodes
appattempt_1683865605880_0001_000001	Fri, 12 May 2023 04:32:33 GMT	http://Datanode-1:8042	Logs	N/A

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

Graphical Analysis using Matplotlib



Hadoop Node Status

Overview 'NameNode:9000' (active)

Started:	Fri May 12 00:26:22 EDT 2023
Version:	2.6.5, re8c9fe0b4c252caf2ebf1464220599650/119997
Compiled:	2016-10-02T23:43Z by sjlee from branch-2.6.5
Cluster ID:	CID-8b68784-e49a-4da7-85f3-e4097623a96
Block Pool ID:	BP-1567734073-192.168.4.13-1682978507294

Summary

Security is off.
 Safemode is off.
 286 files and directories, 254 blocks = 540 total filesystem object(s).
 Heap Memory used 28.15 MB of 56.47 MB Heap Memory. Max Heap Memory is 966.69 MB.
 Non Heap Memory used 43.46 MB of 44.13 MB Committed Non Heap Memory. Max Non Heap Memory is -1 B.

Configured Capacity:	52.28 GB
----------------------	----------

Sqoop Data Ingestion

```

23/05/12 02:07:46 INFO mapreduce.Job: Job job_1683865605880_0002 running in uber mode : false
23/05/12 02:07:46 INFO mapreduce.Job: map 0% reduce 0%
23/05/12 02:08:01 INFO mapreduce.Job: map 100% reduce 0%
23/05/12 02:08:01 INFO mapreduce.Job: Job job_1683865605880_0002 completed successfully
23/05/12 02:08:01 INFO mapreduce.Job: Counters: 30
  File System Counters
    FILE: Number of bytes read=0
    FILE: Number of bytes written=126334
    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=2376
    HDFS: Number of read operations=4
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  Job Counters
    Launched map tasks=1
    Other local map tasks=1
    Total time spent by all maps in occupied slots (ms)=11322
    Total time spent by all reduces in occupied slots (ms)=0
    Total time spent by all map tasks (ms)=11322
    Total vcore-milliseconds taken by all map tasks=11322
    Total megabyte-milliseconds taken by all map tasks=11593728
  Map-Reduce Framework
    Map input records=62
    Map output records=62
    Input split bytes=87
    Spilled Records=0
    Failed Shuffles=0
    Merged Map outputs=0
    GC time elapsed (ms)=107
    CPU time spent (ms)=1120
    Physical memory (bytes) snapshot=112054272
    Virtual memory (bytes) snapshot=1929478144
    Total committed heap usage (bytes)=244444928
  File Input Format Counters
    Bytes Read=0
  
```

11) Conclusion

Thus the weather analysis tools (CLI and Graphical Analysis tool) developed as part of the project can be used by a weather expert to perform descriptive analysis on the weather data collected from various sources.

This application is scalable for adding more cities and weather fields to the storage and leverages parallel processing as part of the MapReduce framework to run queries to analyze a huge amount of data to gain meaningful insights into the weather patterns for the cities used.

12) References

<https://spark.apache.org/sql/>

<https://hadoop.apache.org/>

<https://www.wunderground.com/history>

<https://www.youtube.com/@dataengineeringvideos>

<https://www.youtube.com/@SimplilearnOfficial>