

Project Report

Assignment - EC7205 Cloud Computing
University of Ruhuna

Group No:	35
Student Name 1:	Galpayage G. D. T. G. – EG/2020/3935
Student Name 2:	Vihanga V. M. B. P. – EG/2020/4252
Student Name 3:	Weerasekara W.M.N.S. – EG/2020/4266
Project Title:	Analyzing Bike-Sharing System Usage Patterns

1 Project Introduction

This project aims to identify the most popular Citi Bike stations in New York City based on the frequency of trips originating or ending at each station. Subsequently, it analyzes the busiest hours for these top stations. By leveraging the distributed computing capabilities of Hadoop MapReduce, the project processes large volumes of trip data to determine high demand stations and peak usage times, which can inform urban transportation planning, resource allocation and service optimization.

2 Dataset Introduction

2.1 Source

The dataset used in this project is obtained from the official Citi Bike NYC system data portal: <https://citibikenyc.com/system-data>. This open dataset provides detailed information about individual bike trips taken with Citi Bike, New York City's bike-sharing service.

2.2 Format and Structure

The data is provided in CSV (Comma-Separated Values) format, with each file representing one month of trip data. File names typically follow the format: `YYYYMM-citibike-tripdata.csv`.

2.3 Attributes

Each record in the dataset corresponds to a single trip and includes the following key attributes:

- **ride_id**: Unique identifier assigned to each trip.
- **rideable_type**: Type of bicycle used (e.g., classic_bike, electric_bike).
- **started_at**: Timestamp indicating when the ride began.
- **ended_at**: Timestamp indicating when the ride ended.
- **start_station_name**: Name of the station where the ride originated.
- **start_station_id**: Unique ID of the starting station.
- **end_station_name**: Name of the station where the ride ended.
- **end_station_id**: Unique ID of the destination station.
- **start_lat, start_lng**: Geographic coordinates (latitude and longitude) of the starting station.
- **end_lat, end_lng**: Geographic coordinates of the destination station.
- **member_casual**: Indicates whether the user is a registered member or a casual rider.

2.4 Size and Volume

The dataset is updated monthly, and each file contains a large volume of records, often ranging from hundreds of thousands to millions of rows, depending on the month. This makes it an ideal candidate for processing using distributed computing frameworks such as Hadoop MapReduce. For instance, the dataset from the year 2023 which is used in this project comprises approximately 35,107,070 records, highlighting the scale and suitability of this data for parallel processing.

3 Installation and Setup

3.1 Prerequisites

- Operating System: Ubuntu (WSL)
- Java Development Kit (JDK 8)
- Hadoop (version 3.3.6 - Pseudo-distributed mode configuration)
- Git (for code management)

3.2 Java Installation (JDK 8)

1. Update the package index:

```
sudo apt update
```

2. Install OpenJDK 8:

```
sudo apt install openjdk-8-jdk
```

3. Verify installation:

```
java -version
javac -version
```

4. Configure Passwordless SSH:

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

3.3 Hadoop Installation and Configuration

1. Download Hadoop binary (e.g., version 3.3.6) and extract

```
cd ~ #
wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
tar -xzf hadoop-3.3.6.tar.gz
```

2. Set environment variables (JAVA_HOME, HADOOP_HOME, PATH) in .bashrc.

```
nano ~/.bashrc
```

Add following to .bashrc:

```
# Give actual path where the hadoop is extracted
export HADOOP_HOME=/home/testuser/hadoop-3.3.6
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"

# If you didn't set JAVA_HOME system-wide or for this user already:
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$PATH:$JAVA_HOME/bin
```

Source the file:

```
source ~/.bashrc
```

3. Configure Hadoop XML files (core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml) for pseudo-distributed mode.

- **core-site.xml:** Specify `fs.defaultFS`.
 - Edit the file (e.g., using nano):


```
nano $HADOOP_HOME/etc/hadoop/core-site.xml
```
 - Add the following content between the `<configuration>` and `</configuration>` tags:


```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
    <description>The name of the default file system. A URI whose
                  scheme and authority determine the FileSystem
                  implementation.</description>
  </property>
</configuration>
```
- **hdfs-site.xml:** Specify `dfs.replication` (set to 1 for pseudo-distributed) and paths for namenode/datanode directories.
 - Before editing, create directories for NameNode and DataNode data (adjust path for your Hadoop user, e.g., `/home/hadoopuser/hadoop_data/...`):


```
mkdir -p ~/hadoop_data/hdfs/namenode
mkdir -p ~/hadoop_data/hdfs/datanode
```
 - Edit the file:


```
nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```
 - Add the following content between the `<configuration>` and `</configuration>` tags (adjust paths if you chose different locations):


```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>Default block replication. The actual number of
                  replications can be specified when the file is
                  created.</description>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/home/testuser/hadoop_data/hdfs/namenode</value>
  </property>
</configuration>
```

```

        <name>dfs.datanode.data.dir</name>
        <value>file:/home/testuser/hadoop_data/hdfs/datanode</value>
    </property>
</configuration>

```

- **mapred-site.xml:** Specify `mapreduce.framework.name` to `yarn`.

– If `mapred-site.xml` doesn't exist, copy it from the template:

```
cp $HADOOP_HOME/etc/hadoop/mapred-site.xml.template $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

– Edit the file:

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

– Add the following content between the `<configuration>` and `</configuration>` tags:

```

<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
    <description>The runtime framework for executing MapReduce jobs.
                  Can be one of local, classic or yarn.</description>
  </property>
  <property>
    <name>mapreduce.jobhistory.address</name>
    <value>localhost:10020</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>localhost:19888</value>
  </property>
</configuration>

```

- **yarn-site.xml:** Configure `yarn.nodemanager.aux-services` to `mapreduce_shuffle` and other YARN properties.

– Edit the file:

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

– Add the following content between the `<configuration>` and `</configuration>` tags:

```

<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
    <description>A comma separated list of services that need to be
                  auxiliary to the NodeManager. For example,
                  mapreduce_shuffle.</description>
  </property>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>localhost</value>
  </property>
  <property>
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>2048</value> <!-- Adjust based on your system's RAM -->
  </property>
  <property>
    <name>yarn.scheduler.minimum-allocation-mb</name>
    <value>256</value> <!-- Minimum allocation for a container -->
  </property>
  <property>
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>1024</value> <!-- Maximum allocation for a container -->
  </property>
  <property>

```

```
        <name>yarn.nodemanager.vmem-pmem-ratio</name>
        <value>2.1</value>
    </property>
</configuration>
```

4. Format the HDFS NameNode:

- Navigate to your Hadoop installation directory and execute the format command:

```
cd $HADOOP_HOME
hdfs namenode -format
```

3.4 Starting and Verifying Hadoop Services

1. Navigate to Hadoop's `sbin` directory and start services:

```
cd $HADOOP_HOME/sbin
./start-dfs.sh
./start-yarn.sh
```

(Or use `./start-all.sh` if preferred, though it's often recommended to start DFS and YARN separately).

2. Verify the services using `jps` (Java Virtual Machine Process Status Tool). You should see processes like NameNode, DataNode, ResourceManager, NodeManager.

```
jps
```

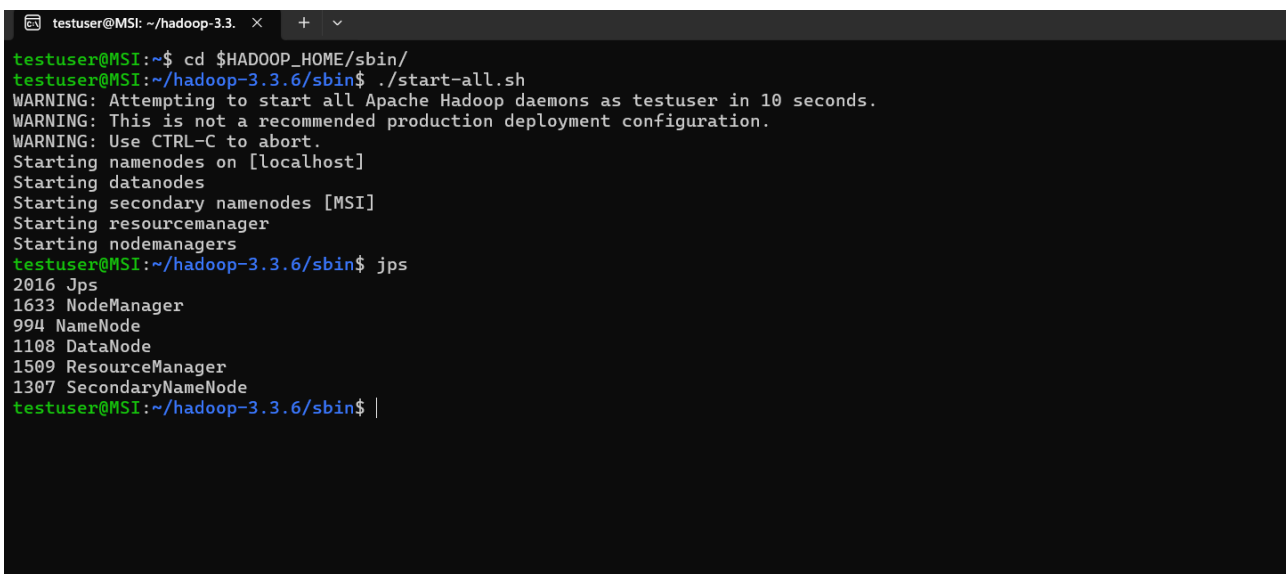
A terminal window titled 'testuser@MSI: ~/hadoop-3.3.' shows the execution of Hadoop startup scripts. The user runs 'cd \$HADOOP_HOME/sbin/' and then './start-all.sh'. The script outputs several warnings and status messages: 'WARNING: Attempting to start all Apache Hadoop daemons as testuser in 10 seconds.', 'WARNING: This is not a recommended production deployment configuration.', 'WARNING: Use CTRL-C to abort.', 'Starting namenodes on [localhost]', 'Starting datanodes', 'Starting secondary namenodes [MSI]', 'Starting resourcemanager', and 'Starting nodemanagers'. After the scripts complete, the user runs 'jps', which outputs a list of running processes with their PIDs: '2016 Jps', '1633 NodeManager', '994 NameNode', '1108 DataNode', '1509 ResourceManager', and '1307 SecondaryNameNode'. The terminal prompt returns to 'testuser@MSI:~/hadoop-3.3.6/sbin\$'.

Figure 1: Output of `jps` showing running Hadoop daemons.

3. Verify NameNode UI (Browse <http://localhost:9870/>)

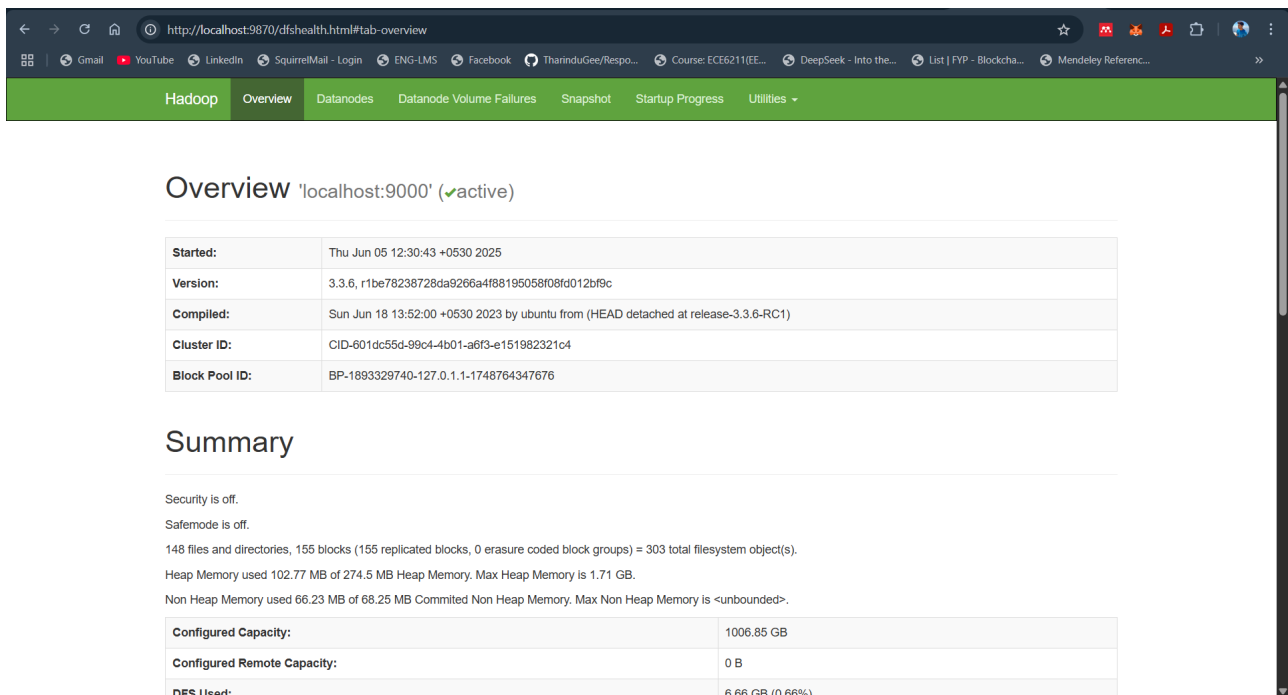


Figure 2: HDFS NameNode UI

3.5 Dataset Preparation on HDFS

1. Download the 2023 Citi Bike dataset from <https://s3.amazonaws.com/tripdata/index.html>.
2. Extract all monthly CSV files from their zipped archives into a single local directory.

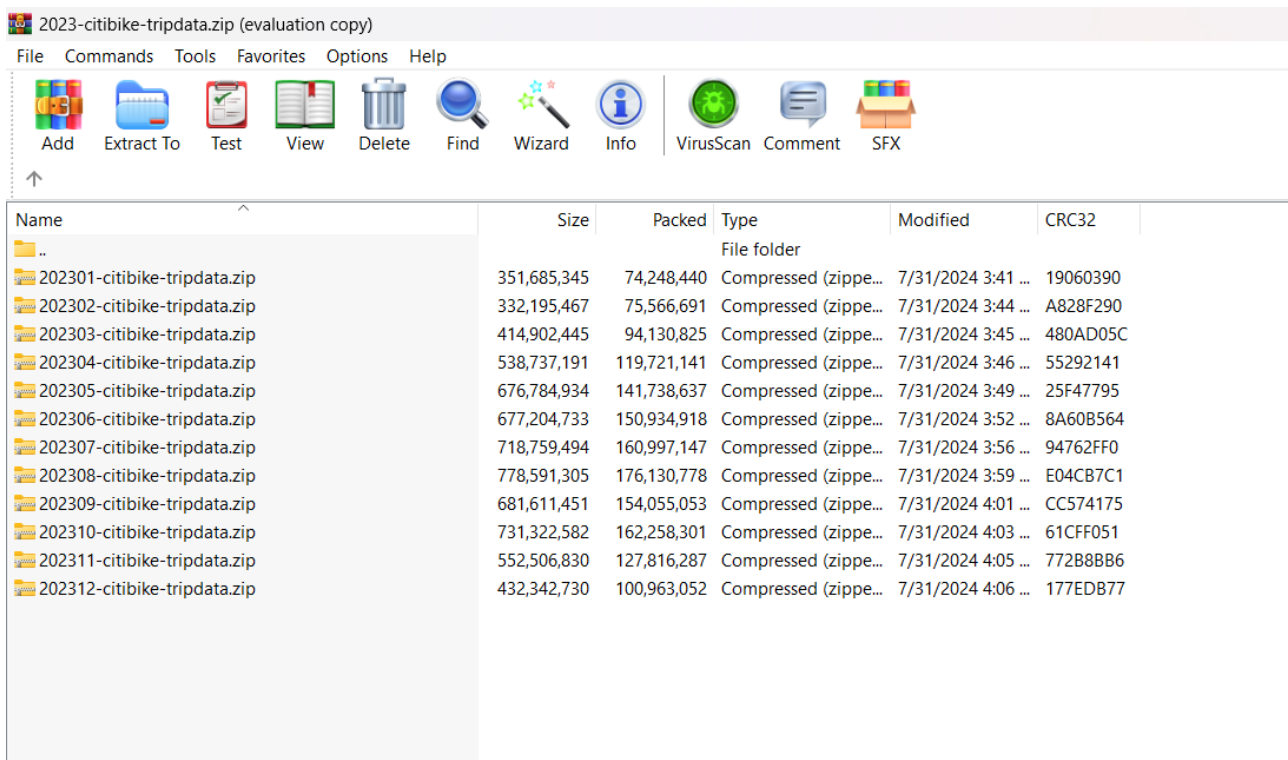


Figure 3: View of zipped monthly data files.

```
testuser@MSI: ~/2023-trip-data/
testuser@MSI:~/2023-trip-data$ ls
202301-citibike-tripdata_1.csv      202307-citibike-tripdata_2.csv:Zone.Identifier  202310-citibike-tripdata_1.csv
202301-citibike-tripdata_2.csv      202307-citibike-tripdata_3.csv:Zone.Identifier  202310-citibike-tripdata_2.csv
202302-citibike-tripdata_1.csv      202307-citibike-tripdata_4.csv                  202310-citibike-tripdata_3.csv:Zone.Identifier
202302-citibike-tripdata_2.csv      202307-citibike-tripdata_1.csv:Zone.Identifier  202310-citibike-tripdata_4.csv
202303-citibike-tripdata_1.csv      202308-citibike-tripdata_1.csv                  202311-citibike-tripdata_1.csv
202303-citibike-tripdata_2.csv      202308-citibike-tripdata_2.csv:Zone.Identifier  202311-citibike-tripdata_2.csv
202303-citibike-tripdata_3.csv      202308-citibike-tripdata_3.csv                  202311-citibike-tripdata_3.csv:Zone.Identifier
202304-citibike-tripdata_1.csv      202308-citibike-tripdata_4.csv:Zone.Identifier  202311-citibike-tripdata_4.csv
202304-citibike-tripdata_2.csv      202309-citibike-tripdata_1.csv                  202312-citibike-tripdata_1.csv
202304-citibike-tripdata_3.csv      202309-citibike-tripdata_2.csv:Zone.Identifier  202312-citibike-tripdata_2.csv
202305-citibike-tripdata_1.csv      202309-citibike-tripdata_3.csv:Zone.Identifier  202312-citibike-tripdata_3.csv:Zone.Identifier
202305-citibike-tripdata_2.csv      202309-citibike-tripdata_4.csv                  202312-citibike-tripdata_4.csv
202305-citibike-tripdata_3.csv      202309-citibike-tripdata_1.csv:Zone.Identifier  202312-citibike-tripdata_1.csv:Zone.Identifier
202306-citibike-tripdata_1.csv      202309-citibike-tripdata_2.csv                  202312-citibike-tripdata_2.csv:Zone.Identifier
202306-citibike-tripdata_2.csv      202309-citibike-tripdata_3.csv:Zone.Identifier  202312-citibike-tripdata_3.csv
202306-citibike-tripdata_3.csv      202309-citibike-tripdata_4.csv:Zone.Identifier  202312-citibike-tripdata_4.csv
202307-citibike-tripdata_1.csv      202309-citibike-tripdata_1.csv:Zone.Identifier  202312-citibike-tripdata_1.csv:Zone.Identifier
202307-citibike-tripdata_2.csv      202309-citibike-tripdata_2.csv                  202312-citibike-tripdata_2.csv:Zone.Identifier
202307-citibike-tripdata_3.csv      202309-citibike-tripdata_3.csv                  202312-citibike-tripdata_3.csv:Zone.Identifier
202307-citibike-tripdata_4.csv      202309-citibike-tripdata_4.csv:Zone.Identifier  202312-citibike-tripdata_4.csv:Zone.Identifier
testuser@MSI:~/2023-trip-data$
```

Figure 4: Extracted CSV files for all months of 2023 in a local directory.

3. Create an input directory on HDFS and upload the CSV files:

```
hdfs dfs -mkdir popular_stations_input
hdfs dfs -put path/to/local/citibike_data_2023/*.csv popular_stations_input
hdfs dfs -ls popular_stations_input
```

4 MapReduce Job Implementation and Execution

This section details the process of compiling the Java source code for the MapReduce jobs, packaging them into JAR files, and executing them on the Hadoop cluster.

4.1 Cloning the Project Repository

The Java source code for the MapReduce jobs is available in a Git repository.

```
git clone https://github.com/TharinduGee/Hadoop_MapReduce.git
cd Hadoop_MapReduce
```

It is assumed that the Java source files (e.g., PopularStationsDriver.java, PopularStationsMapper.java, PopularStationsReducer.java, and similarly for BusiestHour) are located in the root of this cloned directory.

4.2 Job 1: Identifying Popular Stations

This job processes the trip data to count the usage frequency of each station.

4.2.1 Compiling and Packaging Job 1 (popularStations.jar)

1. Set the Hadoop classpath:

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

2. Create a directory for compiled class files:

```
mkdir classes_popular_stations
```

3. Compile the Java source files for the Popular Stations job (adjust filenames as necessary):

```
javac -cp $HADOOP_CLASSPATH -d classes_popular_stations \
    PopularStationsDriver.java PopularStationsMapper.java \
    PopularStationsReducer.java
```

4. Create the JAR file `popularStations.jar`:

```
jar -cvf popularStations.jar -C classes_popular_stations .
```

This creates `popularStations.jar` in the current directory (`Hadoop_MapReduce`).

5. (Optional) Verify the JAR contents:

```
jar -tf popularStations.jar
```

Note: If a pre-compiled `popularStations.jar` is already available in the repository, the compilation and packaging steps can be skipped.

4.2.2 Running Job 1 and Retrieving Results

1. Execute the MapReduce job:

```
hadoop jar popularStations.jar PopularStationsDriver \  
popular_stations_input popular_stations_output
```

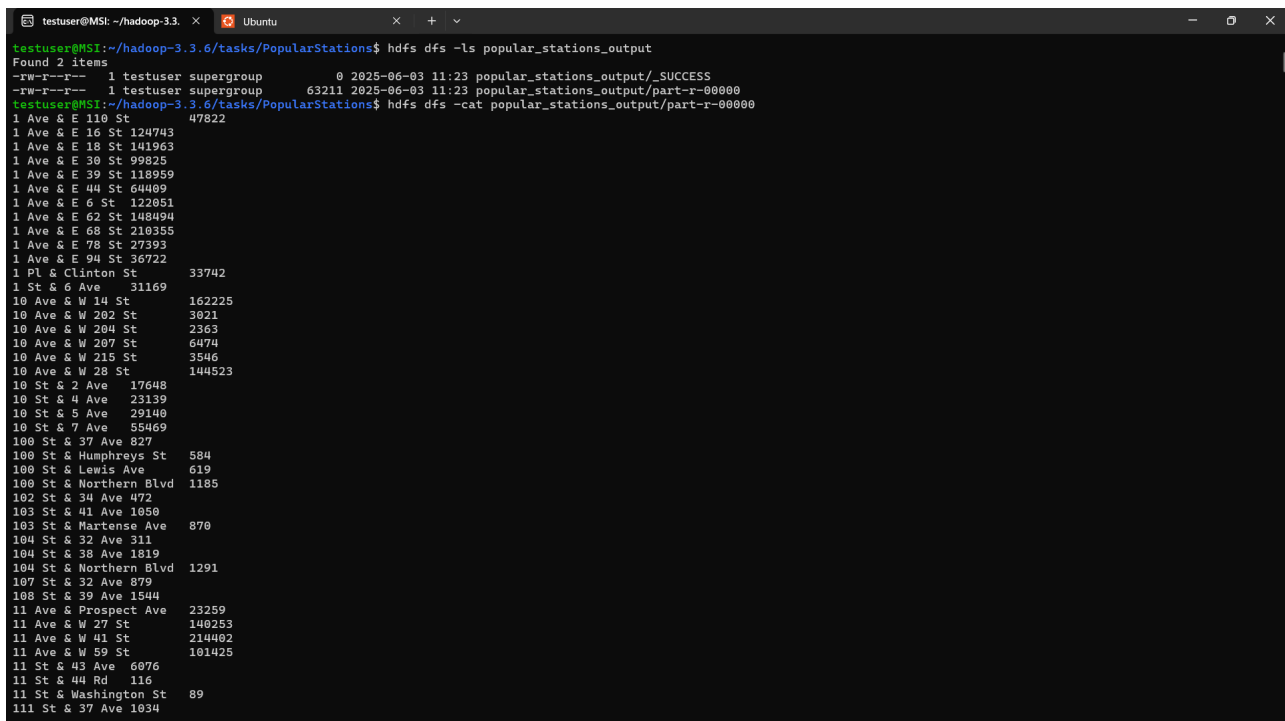
The input path is `/popular_stations_input` on HDFS and the output will be stored in `/popular_stations_output` on HDFS.

2. View the output files:

```
hdfs dfs -ls popular_stations_output
```

3. Display the contents of the output file (typically `part-r-00000`):

```
hdfs dfs -cat popular_stations_output/part-r-00000
```



```
testuser@MSI: ~/hadoop-3.3.6/tasks/PopularStations$ hdfs dfs -ls popular_stations_output
Found 2 items
-rw-r--r-- 1 testuser supergroup 0 2025-06-03 11:23 popular_stations_output/_SUCCESS
-rw-r--r-- 1 testuser supergroup 63211 2025-06-03 11:23 popular_stations_output/part-r-00000
testuser@MSI: ~/hadoop-3.3.6/tasks/PopularStations$ hdfs dfs -cat popular_stations_output/part-r-00000
1 Ave & E 110 St 47822
1 Ave & E 16 St 124743
1 Ave & E 18 St 141063
1 Ave & E 30 St 99825
1 Ave & E 39 St 118959
1 Ave & E 44 St 64409
1 Ave & E 6 St 122051
1 Ave & E 62 St 148494
1 Ave & E 68 St 210355
1 Ave & E 78 St 27393
1 Ave & E 94 St 36722
1 Pl & Clinton St 33742
1 St & 6 Ave 31169
10 Ave & W 14 St 162225
10 Ave & W 202 St 3021
10 Ave & W 204 St 2363
10 Ave & W 207 St 6474
10 Ave & W 215 St 3546
10 Ave & W 28 St 144523
10 St & 2 Ave 17648
10 St & 4 Ave 23139
10 St & 5 Ave 29140
10 St & 7 Ave 55469
100 St & 37 Ave 827
100 St & Humphreys St 584
100 St & Lewis Ave 619
100 St & Northern Blvd 1185
102 St & 34 Ave 472
103 St & 41 Ave 1050
103 St & Martense Ave 870
104 St & 32 Ave 311
104 St & 38 Ave 1819
104 St & Northern Blvd 1291
107 St & 32 Ave 879
108 St & 39 Ave 1644
11 Ave & Prospect Ave 23259
11 Ave & W 27 St 140253
11 Ave & W 41 St 214402
11 Ave & W 59 St 101425
11 St & 43 Ave 6076
11 St & 44 Rd 116
11 St & Washington St 89
111 St & 37 Ave 1034
```

Figure 5: Sample output of Job 1, showing station names and their usage counts.

The first column represents the station name, and the second column represents the total number of trips starting or ending at that station.

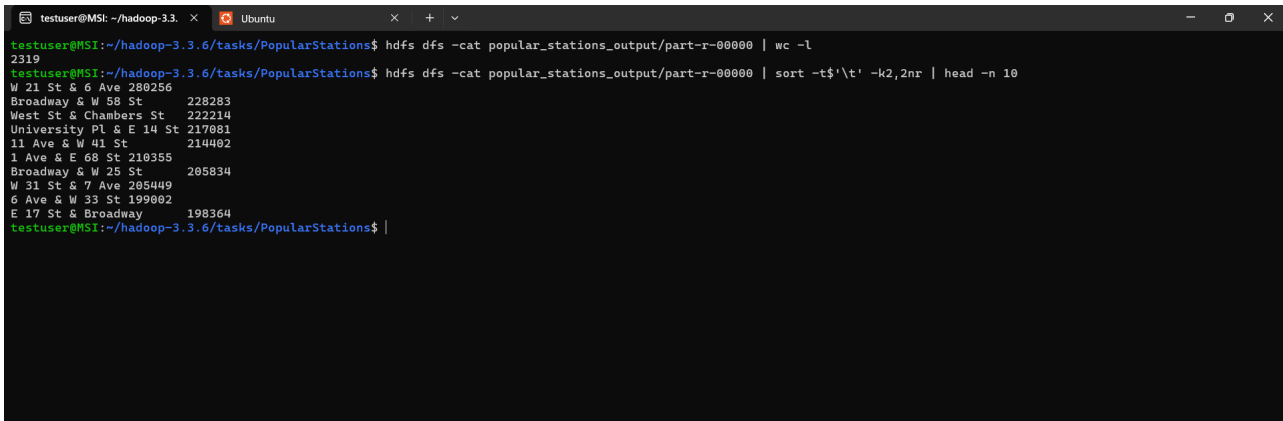
4.2.3 Analyzing Job 1 Results

1. Get the count of unique stations processed:

```
hdfs dfs -cat popular_stations_output/part-r-00000 | wc -l
```

2. Get the top 10 most popular stations:

```
hdfs dfs -cat popular_stations_output/part-r-00000 | sort -t$'\t' -k2,2nr | head -n 10
```



The screenshot shows a terminal window with the following commands and output:

```
testuser@MSI:~/hadoop-3.3.6/tasks/PopularStations$ hdfs dfs -cat popular_stations_output/part-r-00000 | wc -l
2319
testuser@MSI:~/hadoop-3.3.6/tasks/PopularStations$ hdfs dfs -cat popular_stations_output/part-r-00000 | sort -t$'\t' -k2,2nr | head -n 10
W 21 St & 6 Ave 280256
Broadway & W 58 St 228283
West St & Chambers St 222214
University Pl & E 14 St 217081
11 Ave & W 41 St 214402
1 Ave & E 68 St 210355
Broadway & W 25 St 205834
W 31 St & 7 Ave 205449
6 Ave & W 33 St 199002
E 17 St & Broadway 198364
testuser@MSI:~/hadoop-3.3.6/tasks/PopularStations$ |
```

Figure 6: Unique station count and the top 10 most popular stations with their counts.

4.3 Job 2: Finding Busiest Hours for Top Stations

This job identifies the busiest hours for the top 10 most popular stations (determined by Job 1) using Hadoop's Distributed Cache feature.

4.3.1 Preparing Data for Distributed Cache

The list of top 10 popular stations needs to be provided to Job 2 via the Distributed Cache.

1. Create a local file containing the names of the top 10 stations:

```
hdfs dfs -cat popular_stations_output/part-r-00000 | sort -t$'\t' -k2,2nr | \
head -n 10 | awk -F'\t' '{print $1}' > /tmp/top_stations.txt
```

2. Create a directory on HDFS for cache files and upload the station list:

```
hdfs dfs -mkdir -p job2_cache_data
hdfs dfs -put tmp/top_stations.txt job2_cache_data/top_stations.txt
```

4.3.2 Compiling and Packaging Job 2 (busiestHour.jar)

Follow a similar process as for Job 1:

1. Ensure Hadoop classpath is set: `export HADOOP_CLASSPATH=$(hadoop classpath)`
2. Create a directory for compiled class files:

```
mkdir classes_busiest_hour
```

3. Compile the Java source files for the Busiest Hour job (adjust filenames as necessary):

```
javac -cp $HADOOP_CLASSPATH -d classes_busiest_hour \
BusiestHourDriver.java BusiestHourMapper.java BusiestHourReducer.java
```

4. Create the JAR file busiestHour.jar:

```
jar -cvf busiestHour.jar -C classes_busiest_hour .
```

This creates busiestHour.jar in the current directory (Hadoop.MapReduce).

5. (Optional) Verify the JAR contents: `jar -tf busiestHour.jar`

Note: If a pre-compiled busiestHour.jar is available, these steps can be skipped.

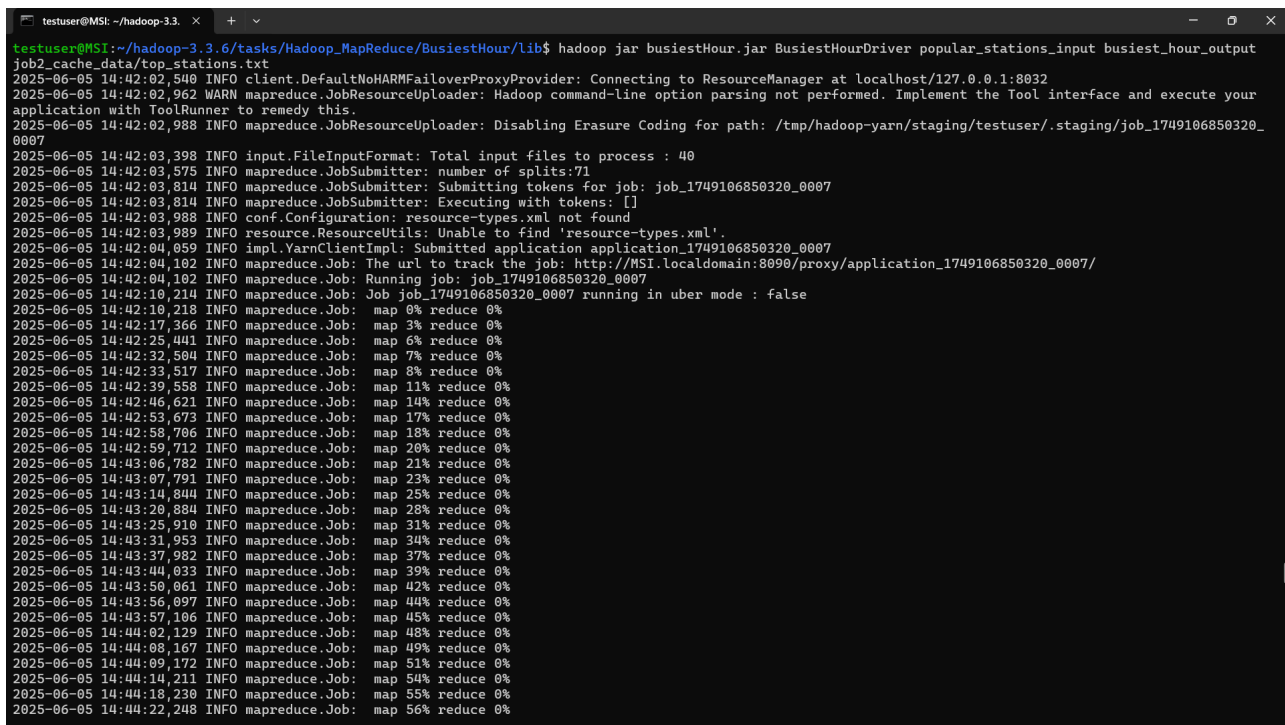
4.3.3 Running Job 2 and Retrieving Results

1. Execute the MapReduce job, providing the cache file path:

```
hadoop jar busiestHour.jar BusiestHourDriver popular_stations_input \
busiest_hour_output job2_cache_data/top_stations.txt
```

The input is the same main dataset, output is to /busiest_hour_output, and the third argument is the HDFS path to the cached file.

2. View the output (this may show intermediate logging from the job):



```
testuser@MSI: ~/hadoop-3.3.6/tasks/Hadoop_MapReduce/BusiestHour/Lib$ hadoop jar busiestHour.jar BusiestHourDriver popular_stations_input
job2_cache_data/top_stations.txt
2025-06-05 14:42:02,540 INFO client.DefaultNoHARMAFailoverProxyProvider: Connecting to ResourceManager at localhost/127.0.0.1:8032
2025-06-05 14:42:02,962 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your
application with ToolRunner to remedy this.
2025-06-05 14:42:02,988 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/testuser/.staging/job_1749106850320_
0007
2025-06-05 14:42:03,398 INFO input.FileInputFormat: Total input files to process : 40
2025-06-05 14:42:03,575 INFO mapreduce.JobSubmitter: number of splits:71
2025-06-05 14:42:03,814 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1749106850320_0007
2025-06-05 14:42:03,814 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-06-05 14:42:03,988 INFO conf.Configuration: resource-types.xml not found
2025-06-05 14:42:03,989 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'
2025-06-05 14:42:04,059 INFO impl.YarnClientImpl: Submitted application application_1749106850320_0007
2025-06-05 14:42:04,102 INFO mapreduce.Job: The url to track the job: http://MSI.localdomain:8090/proxy/application_1749106850320_0007/
2025-06-05 14:42:04,102 INFO mapreduce.Job: Running job: job_1749106850320_0007
2025-06-05 14:42:10,214 INFO mapreduce.Job: Job job_1749106850320_0007 running in uber mode : false
2025-06-05 14:42:10,218 INFO mapreduce.Job: map 0% reduce 0%
2025-06-05 14:42:17,366 INFO mapreduce.Job: map 3% reduce 0%
2025-06-05 14:42:25,441 INFO mapreduce.Job: map 6% reduce 0%
2025-06-05 14:42:32,504 INFO mapreduce.Job: map 7% reduce 0%
2025-06-05 14:42:33,517 INFO mapreduce.Job: map 8% reduce 0%
2025-06-05 14:42:39,558 INFO mapreduce.Job: map 11% reduce 0%
2025-06-05 14:42:46,621 INFO mapreduce.Job: map 14% reduce 0%
2025-06-05 14:42:53,673 INFO mapreduce.Job: map 17% reduce 0%
2025-06-05 14:42:58,706 INFO mapreduce.Job: map 18% reduce 0%
2025-06-05 14:42:59,712 INFO mapreduce.Job: map 20% reduce 0%
2025-06-05 14:43:06,782 INFO mapreduce.Job: map 21% reduce 0%
2025-06-05 14:43:07,791 INFO mapreduce.Job: map 23% reduce 0%
2025-06-05 14:43:14,844 INFO mapreduce.Job: map 25% reduce 0%
2025-06-05 14:43:20,884 INFO mapreduce.Job: map 28% reduce 0%
2025-06-05 14:43:25,910 INFO mapreduce.Job: map 31% reduce 0%
2025-06-05 14:43:31,953 INFO mapreduce.Job: map 34% reduce 0%
2025-06-05 14:43:37,982 INFO mapreduce.Job: map 37% reduce 0%
2025-06-05 14:43:44,033 INFO mapreduce.Job: map 39% reduce 0%
2025-06-05 14:43:50,061 INFO mapreduce.Job: map 42% reduce 0%
2025-06-05 14:43:56,097 INFO mapreduce.Job: map 44% reduce 0%
2025-06-05 14:43:57,106 INFO mapreduce.Job: map 45% reduce 0%
2025-06-05 14:44:02,129 INFO mapreduce.Job: map 48% reduce 0%
2025-06-05 14:44:08,167 INFO mapreduce.Job: map 49% reduce 0%
2025-06-05 14:44:09,172 INFO mapreduce.Job: map 51% reduce 0%
2025-06-05 14:44:14,211 INFO mapreduce.Job: map 54% reduce 0%
2025-06-05 14:44:18,230 INFO mapreduce.Job: map 55% reduce 0%
2025-06-05 14:44:22,248 INFO mapreduce.Job: map 56% reduce 0%
```

Figure 7: MapReduce job execution log/status for Job 2 (Part 1).

```
testuser@MSI: ~/hadoop-3.3.  +  v
2025-06-05 14:46:17,787 INFO mapreduce.Job: map 96% reduce 32%
2025-06-05 14:46:20,282 INFO mapreduce.Job: map 99% reduce 32%
2025-06-05 14:46:24,824 INFO mapreduce.Job: map 99% reduce 32%
2025-06-05 14:46:28,838 INFO mapreduce.Job: map 100% reduce 32%
2025-06-05 14:46:29,868 INFO mapreduce.Job: map 100% reduce 100%
2025-06-05 14:46:30,868 INFO mapreduce.Job: Job job_1749106858320_0007 completed successfully
2025-06-05 14:46:31,085 INFO mapreduce.Job: Counters: 54
File System Counters
  FILE: Number of bytes read=3514727
  FILE: Number of bytes written=26904172
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=6886776713
  HDFS: Number of bytes written=56018
  HDFS: Number of read operations=218
  HDFS: Number of large read operations=8
  HDFS: Number of write operations=2
  HDFS: Number of bytes read erasure-coded=0
Job Counters
  Launched map tasks=71
  Launched reduce tasks=1
  Data-local map tasks=71
  Total time spent by all maps in occupied slots (ms)=308334
  Total time spent by all reduces in occupied slots (ms)=134061
  Total time spent by all map tasks (ms)=308334
  Total time spent by all reduce tasks (ms)=134061
  Total vcore-millisecods taken by all map tasks=308334
  Total vcore-millisecods taken by all reduce tasks=134061
  Total megabyte-millisecods taken by all map tasks=315734816
  Total megabyte-millisecods taken by all reduce tasks=137278464
Map-Reduce Framework
  Map input records=35187070
  Map output records=1089874
  Map output bytes=36617848
  Map output materialized bytes=3515147
  Input split bytes=10934
  Combine input records=1089874
  Combine output records=98993
  Reduce input groups=1680
  Reduce shuffle bytes=3515147
  Reduce input records=98993
  Reduce output records=1680
  Spilled Records=197986
  Shuffled Maps=71
  Failed Shuffles=0
  Merged Map outputs=71
  GC time elapsed (ms)=13859
  CPU time spent (ms)=366466
  Physical memory (bytes) snapshot=36118736896
  Virtual memory (bytes) snapshot=18532883392
  Total committed heap usage (bytes)=32146718728
Peak Map Physical memory (bytes)=52189568
Peak Map Virtual memory (bytes)=2583416832
Peak Reduce Physical memory (bytes)=216737744
Peak Reduce Virtual memory (bytes)=2577893616
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=6886765779
File Output Format Counters
  Bytes Written=56018
```

Figure 8: MapReduce job execution log/status for Job 2 (Part 2).

3. Display the contents of the output file:

```
hdfs dfs -cat busiest_hour_output/part-r-00000
```

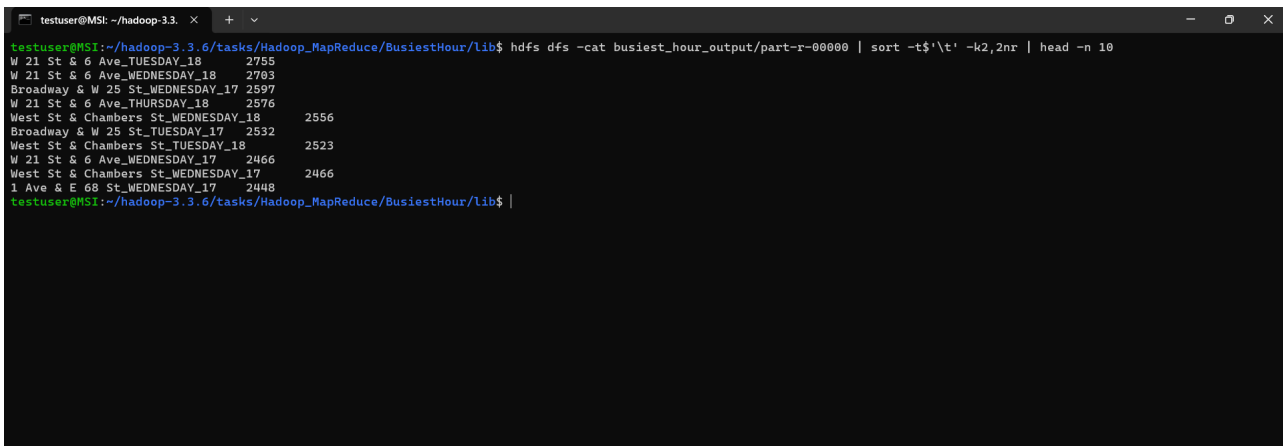
```
testuser@MSI: ~/hadoop-3.3.  +  v
testuser@MSI: ~/hadoop-3.3.6/tasks/Hadoop_MapReduce/BusiestHour/lib$ hdfs dfs -cat busiest_hour_output/part-r-00000
1 Ave 6 E 60 St-FRIDAY_00 152
1 Ave 6 E 60 St-FRIDAY_01 56
1 Ave 6 E 60 St-FRIDAY_02 42
1 Ave 6 E 60 St-FRIDAY_03 25
1 Ave 6 E 60 St-FRIDAY_04 28
1 Ave 6 E 60 St-FRIDAY_05 128
1 Ave 6 E 60 St-FRIDAY_06 643
1 Ave 6 E 60 St-FRIDAY_07 964
1 Ave 6 E 60 St-FRIDAY_08 1267
1 Ave 6 E 60 St-FRIDAY_09 792
1 Ave 6 E 60 St-FRIDAY_10 588
1 Ave 6 E 60 St-FRIDAY_11 635
1 Ave 6 E 60 St-FRIDAY_12 727
1 Ave 6 E 60 St-FRIDAY_13 896
1 Ave 6 E 60 St-FRIDAY_14 926
1 Ave 6 E 60 St-FRIDAY_15 1201
1 Ave 6 E 60 St-FRIDAY_16 1866
1 Ave 6 E 60 St-FRIDAY_17 2173
1 Ave 6 E 60 St-FRIDAY_18 1294
1 Ave 6 E 60 St-FRIDAY_19 830
1 Ave 6 E 60 St-FRIDAY_20 432
1 Ave 6 E 60 St-FRIDAY_21 291
1 Ave 6 E 60 St-FRIDAY_22 231
1 Ave 6 E 60 St-FRIDAY_23 233
1 Ave 6 E 60 St-MONDAY_00 102
1 Ave 6 E 60 St-MONDAY_01 56
1 Ave 6 E 60 St-MONDAY_02 22
1 Ave 6 E 60 St-MONDAY_03 8
1 Ave 6 E 60 St-MONDAY_04 20
1 Ave 6 E 60 St-MONDAY_05 88
1 Ave 6 E 60 St-MONDAY_06 585
1 Ave 6 E 60 St-MONDAY_07 940
1 Ave 6 E 60 St-MONDAY_08 1250
1 Ave 6 E 60 St-MONDAY_09 764
1 Ave 6 E 60 St-MONDAY_10 523
1 Ave 6 E 60 St-MONDAY_11 683
1 Ave 6 E 60 St-MONDAY_12 695
1 Ave 6 E 60 St-MONDAY_13 753
1 Ave 6 E 60 St-MONDAY_14 847
1 Ave 6 E 60 St-MONDAY_15 1137
1 Ave 6 E 60 St-MONDAY_16 1667
1 Ave 6 E 60 St-MONDAY_17 2065
1 Ave 6 E 60 St-MONDAY_18 1221
1 Ave 6 E 60 St-MONDAY_19 951
1 Ave 6 E 60 St-MONDAY_20 510
1 Ave 6 E 60 St-MONDAY_21 319
1 Ave 6 E 60 St-MONDAY_22 283
1 Ave 6 E 60 St-MONDAY_23 186
1 Ave 6 E 60 St-SATURDAY_00 179
1 Ave 6 E 60 St-SATURDAY_01 79
1 Ave 6 E 60 St-SATURDAY_02 68
1 Ave 6 E 60 St-SATURDAY_03 35
1 Ave 6 E 60 St-SATURDAY_04 26
1 Ave 6 E 60 St-SATURDAY_05 42
1 Ave 6 E 60 St-SATURDAY_06 174
1 Ave 6 E 60 St-SATURDAY_07 555
1 Ave 6 E 60 St-SATURDAY_08 491
1 Ave 6 E 60 St-SATURDAY_09 556
1 Ave 6 E 60 St-SATURDAY_10 465
1 Ave 6 E 60 St-SATURDAY_11 631
1 Ave 6 E 60 St-SATURDAY_12 638
1 Ave 6 E 60 St-SATURDAY_13 646
1 Ave 6 E 60 St-SATURDAY_14 614
1 Ave 6 E 60 St-SATURDAY_15 569
1 Ave 6 E 60 St-SATURDAY_16 556
1 Ave 6 E 60 St-SATURDAY_17 435
```

Figure 9: Sample output of Job 2, showing station-hour combinations and their counts.

4.3.4 Analyzing Job 2 Results

Get the top 10 busiest station-hour combinations:

```
hdfs dfs -cat busiest_hour_output/part-r-00000 | sort -t$'\t' -k2,2nr | head -n 10
```



The image shows a terminal window with a dark background. The prompt is `testuser@MSI: ~/hadoop-3.3.6`. The command executed is `hdfs dfs -cat busiest_hour_output/part-r-00000 | sort -t$'\t' -k2,2nr | head -n 10`. The output is a list of station-hour combinations with their corresponding counts, sorted in descending order. The top 10 results are:

Station	Hour	Count
W 21 St & 6 Ave	TUESDAY_18	2755
W 21 St & 6 Ave	WEDNESDAY_18	2703
Broadway & W 25 St	WEDNESDAY_17	2597
W 21 St & 6 Ave	THURSDAY_18	2576
West St & Chambers St	WEDNESDAY_18	2556
Broadway & W 25 St	TUESDAY_17	2532
West St & Chambers St	TUESDAY_18	2523
W 21 St & 6 Ave	WEDNESDAY_17	2466
West St & Chambers St	WEDNESDAY_17	2466
1 Ave & E 68 St	WEDNESDAY_17	2448

Figure 10: Top 10 busiest hours for the top 10 popular stations.

5 Expanding the Model

5.1 Dashboard Integration

To visualize the results of MapReduce jobs (e.g., popular stations, busiest times):

1. **Data Serving Layer:** After MapReduce jobs complete, their aggregated output (typically from `part-r-` files in HDFS) should be transferred to a database for dashboard querying.
The custom script or Apache spark jobs can be used in order to perform above improvement easily.
2. **Visualization Tools:** Show the analytics in the dashboard
The Power BI, Grafana with Elastic search or custom frontend can be used to show the result.

5.2 Continuous and Autonomous Job Triggering

To keep the analysis up-to-date as new data arrives(e.g., daily or hourly CSV files):

1. **Workflow Orchestration:** Employ a workflow management system to schedule and monitor the data pipeline.
Simple cron jobs or Apache Airflow can be used to achieve this.
2. **Incremental Data Processing Strategy:** To maintain up-to-date analysis as new data arrives (e.g., daily CSV files), an automated pipeline is essential. This involves structuring raw input data in HDFS by time partitions (e.g. `/raw_data/year=YYYY/month=MM/day=DD/`) and choosing a processing scope: either full reprocessing of the entire dataset, which is simpler but less efficient, or more complex incremental processing where jobs only act on new data partitions, with results merged into existing aggregates. Regardless of the processing scope, after each scheduled pipeline run, the aggregated results in the serving database must be updated, either by overwriting existing data or by performing update operations for incremental changes.

6 Conclusion

This project successfully demonstrated the use of Hadoop MapReduce for analyzing large-scale bike-sharing data. Job 1 identified the most frequently used Citi Bike stations, providing insights into high-traffic areas. Job 2, utilizing the Distributed Cache, further pinpointed the peak operational hours for these popular stations. The results, such as the top 10 popular stations and their busiest hours, offer valuable information for optimizing bike distribution, station capacity planning, and enhancing user experience in New York City's bike-sharing system. The methodology showcases the power of distributed computing for deriving actionable intelligence from extensive datasets.