Practical File
on
Big Data using Hadoop Lab
(Course Code: AIEU4211)

Department of Computer Science and Engineering

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

SUBMITTED TO: -

Dr. Jyoti Chaudhary

SUBMITTED BY: -

NAME: Kanishk

ENROLLMENT NO: A501144824006

COURSE: Mtech AIML

SEMESTER: 3rd Semester

INDEX

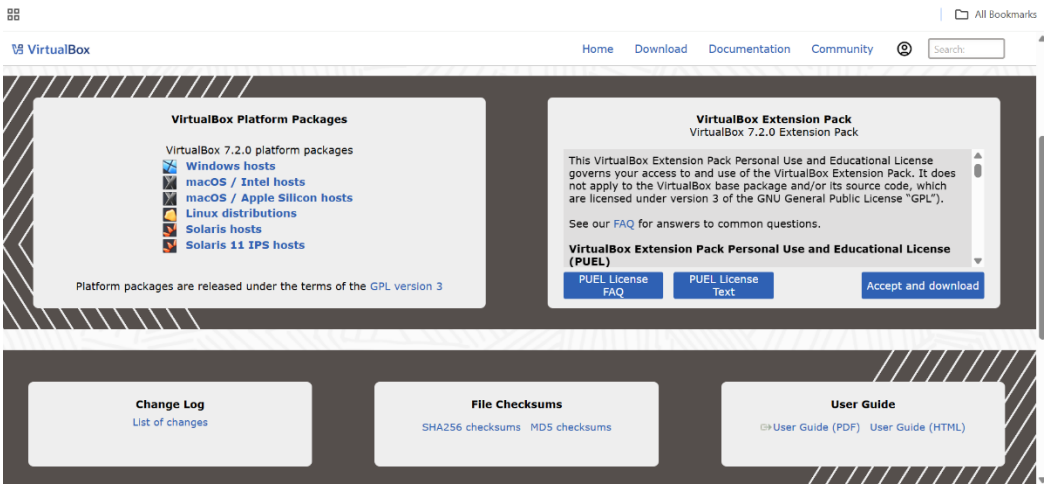
Experiment No: 1

Objective :

To install Oracle VirtualBox and Cloudera Quickstart VM with proper configuration steps and illustrations.

Program Code:

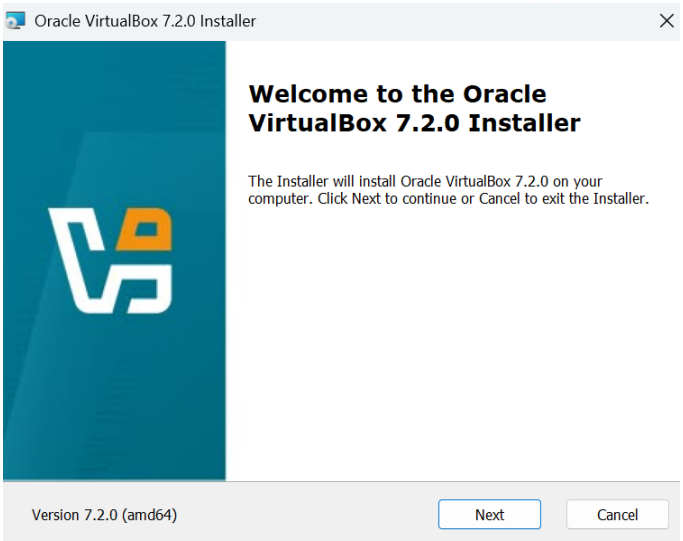
Step 1: To download the Oracle virtual box https://www.virtualbox.org/wiki/Downloads

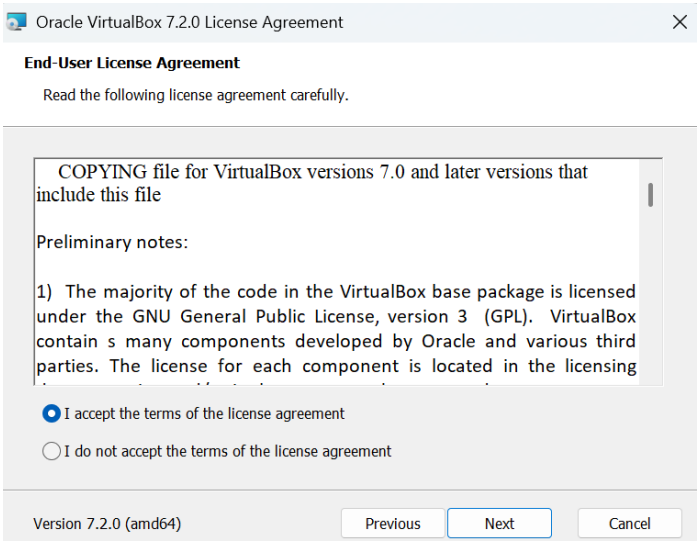


Step2: Go to the Downloads

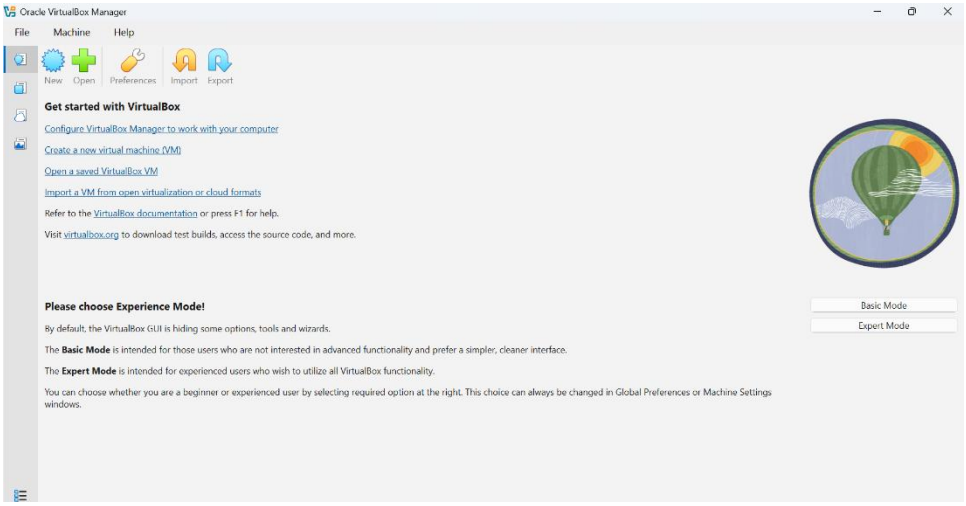
- Double click on the oracle virtual box installer
- Select yes

Note: If (visual studio C++ 2019 distribution package is not then goto)
https://download.visualstudio.microsoft.com/download/pr/9565895b-35a6-434b-a881-11a6f4beec76/EE84FED2552E018E854D4CD2496DF4DD516F30733A27901167B8A9882119E57C/VC_redist.x64.exe

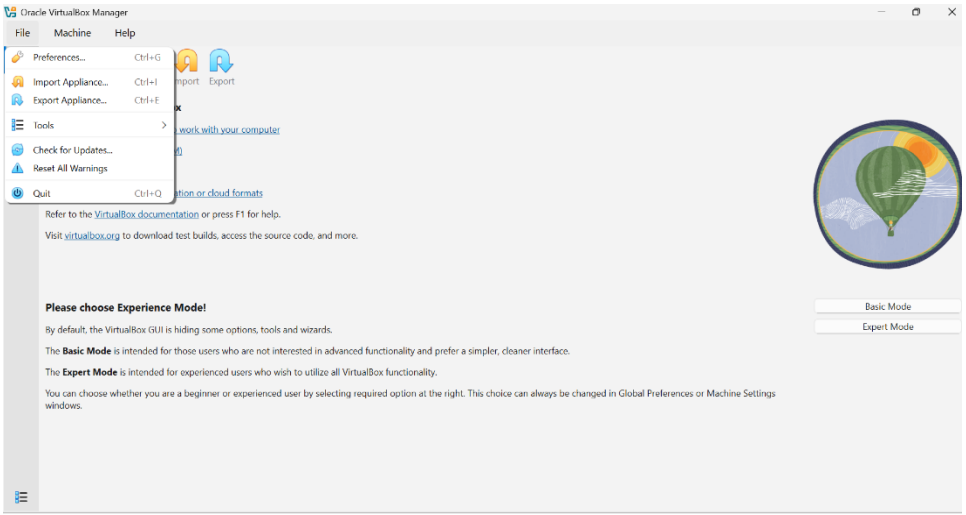




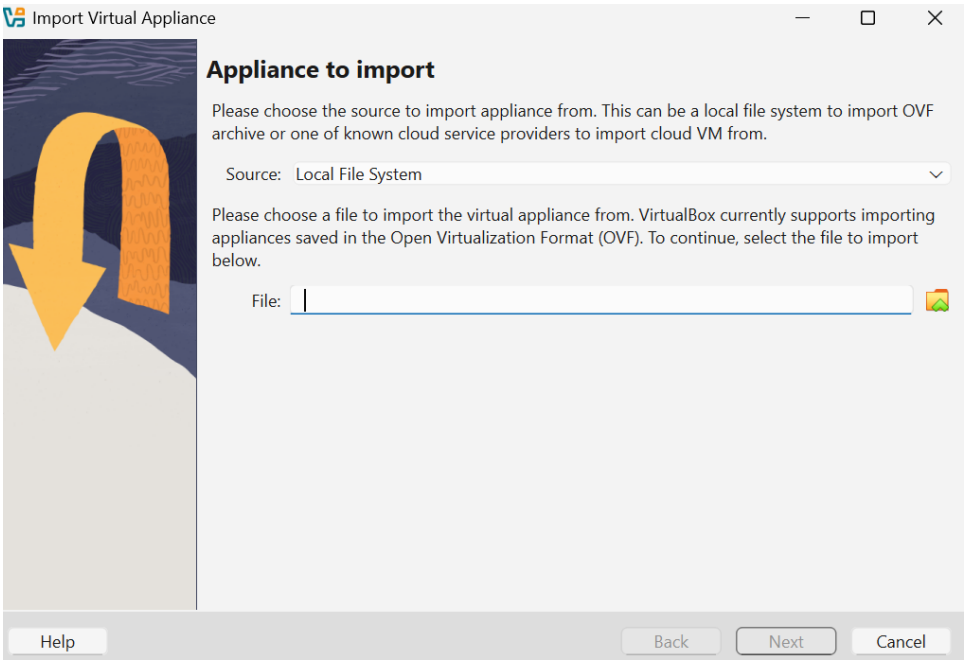
Step 4: Now, To download Cloudera quickstart vm, unzip the provided file.
To run the cloudera, goto the oracle vm
1. Goto the file



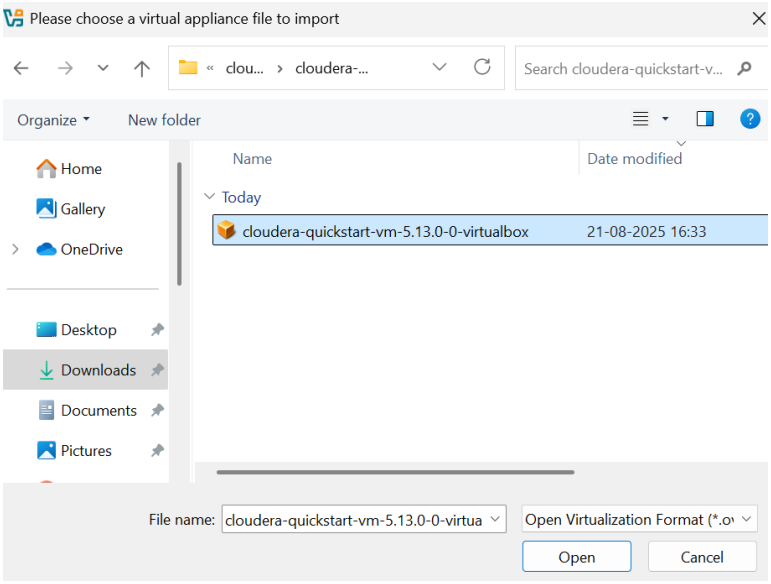
Step 5: Select Import Appliance



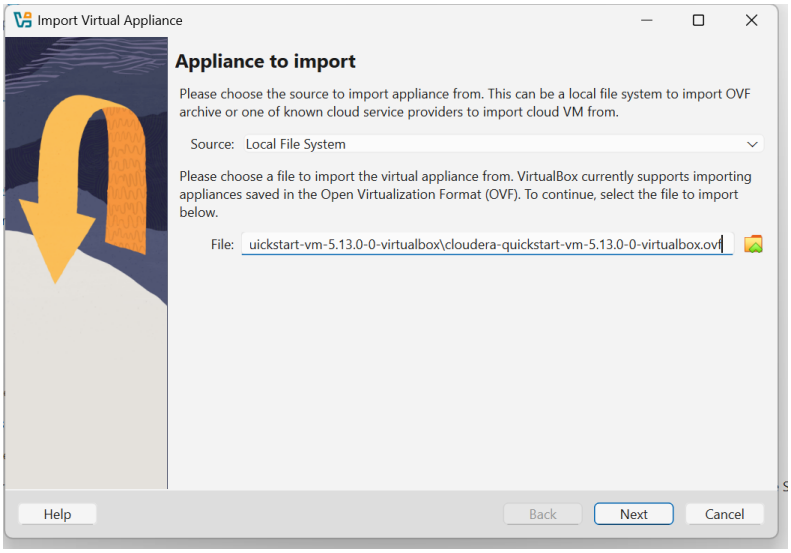
Step 6: Browse the path of cloudera quickstart vm where the file is unzipped



Step 7: Now open till the below screenshot output achieve.

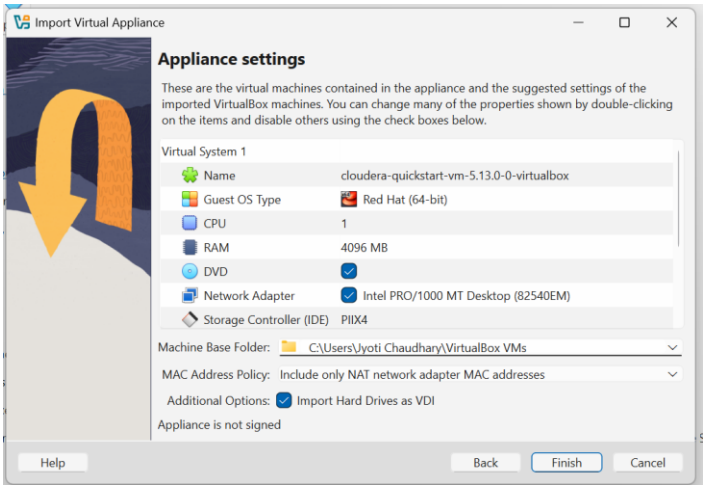


Step 8: So, the following output will achieve.

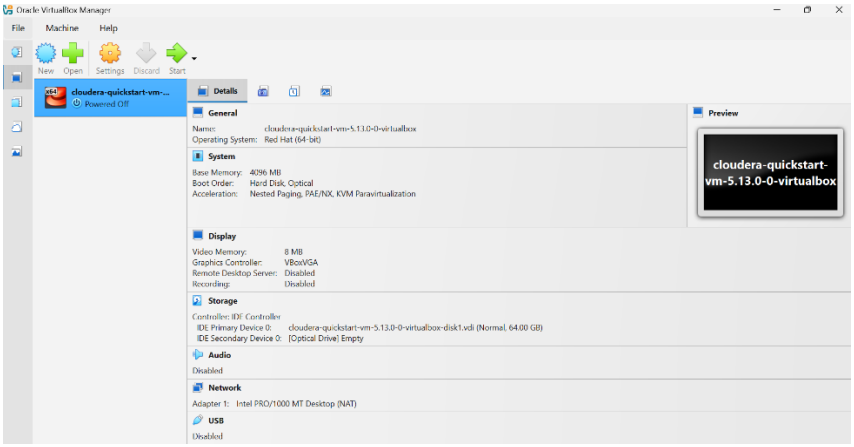


Now click on the next.

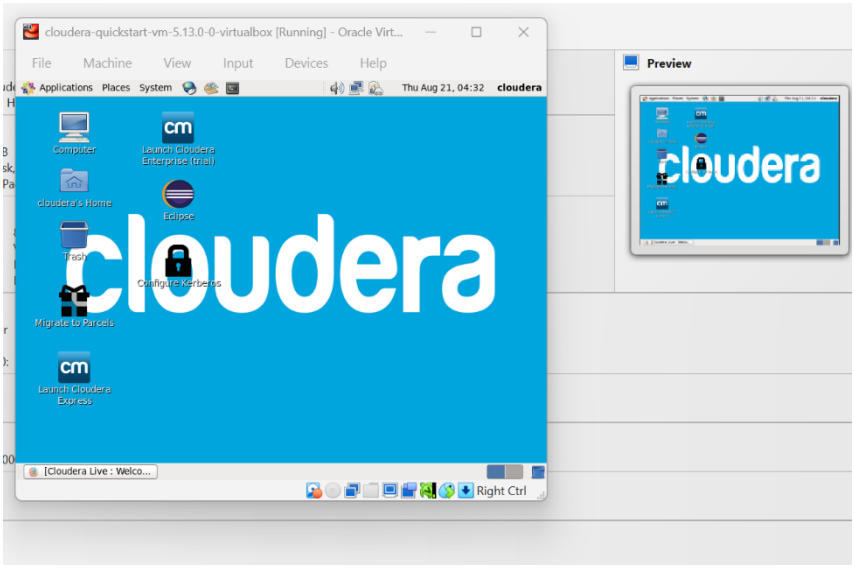
Step9: By continuing to the default configuration, finish the import process.



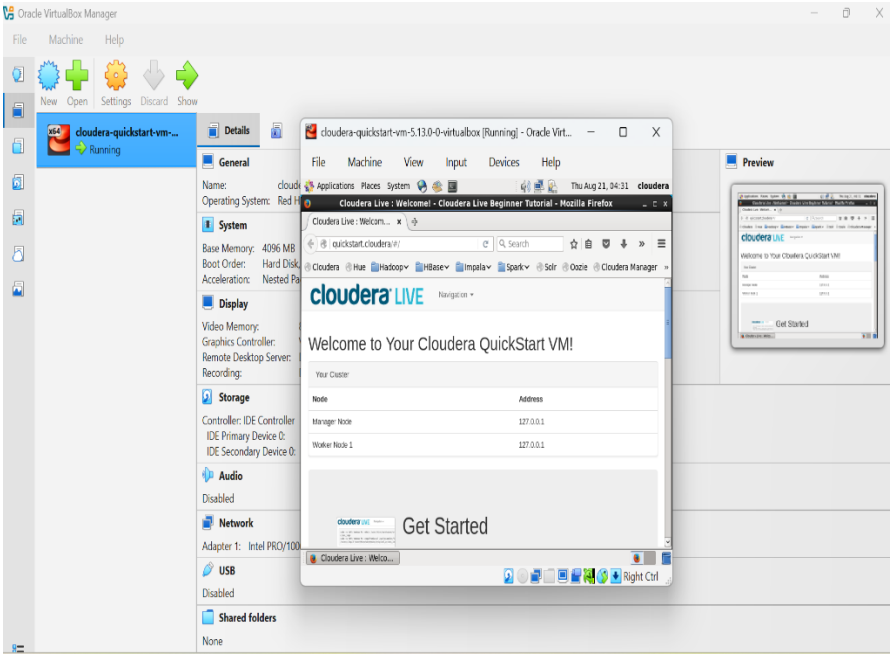
Step 10:



Step 11:



Step 12:



Experiment No: 2

Objective

• Interact with Hadoop using the command-line application.

• Copy files into and out of the Hadoop Distributed File System (HDFS).

Program Code:

1. Open a browser. Open the browser by clicking on the browser icon on the top left of the screen.
2. Download the Sample text file. We are going to download a text file to copy into HDFS. Enter the following link in the browser: http://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.sample-3.txt
   a. Once the page is loaded, click on the Open menu button.
   b. Click on Save Page.
   c. Change the output to sample-3.txt and click Save.
3. Open a terminal shell. Open a terminal shell by clicking on the square black box on the top left of the screen.
   - Run the following command to change to the Downloads directory:

     ```
     [cloudera@quickstart ~]$ cd Downloads
     ```

   - Run the following command to see that words.txt was saved:

     ```
     [cloudera@quickstart Downloads]$ ls
     sample-3.txt
     ```

4. Copy file to HDFS. Run the following command to copy the text file to HDFS:

   ```
   [cloudera@quickstart Downloads]$ hadoop fs -copyFromLocal sample-3.txt
   ```

5. Verify the file was copied to HDFS. Run the following command to verify the file was copied to HDFS:

   ```
   [cloudera@quickstart Downloads]$ hadoop fs -ls
   Found 1 items
   -rw-r--r--   1 cloudera cloudera       1162 2025-09-11 03:55 sample-3.txt
   ```

6. Copy a file within HDFS. You can make a copy of a file in HDFS. Run the following command to make a copy of words.txt called words2.txt:

   ```
   [cloudera@quickstart Downloads]$ hadoop fs -cp sample-3.txt words.txt
   ```

   - We can see the new file by running the following command:

     ```
     [cloudera@quickstart Downloads]$ hadoop fs -ls
     Found 2 items
     -rw-r--r--   1 cloudera cloudera       1162 2025-09-11 03:55 sample-3.txt
     -rw-r--r--   1 cloudera cloudera       1162 2025-09-11 03:57 words.txt
     ```

7. Copy a file from HDFS. We can also copy a file from HDFS to the local file system. Run the following command to copy words2.txt to the local directory:

   ```
   [cloudera@quickstart Downloads]$
   [cloudera@quickstart Downloads]$ hadoop fs -copyToLocal words.txt
   ```

   - Let's run the following command to see that the file was copied and confirm that words2.txt is there:

     ```
     [cloudera@quickstart Downloads]$ ls
     sample-3.txt  words.txt
     ```

8. Delete a file in HDFS. Let's delete words2.txt in HDFS. Run the following command:

   ```
   [cloudera@quickstart Downloads]$ hadoop fs -rm sample-3.txt
   Deleted sample-3.txt
   ```

- Run the following command to see that the file is gone:

```
[cloudera@quickstart Downloads]$ hadoop fs -ls
Found 1 items
-rw-r--r--   1 cloudera cloudera       1162 2025-09-11 03:57 words.txt
[cloudera@quickstart Downloads]$
```

```
bash: cd: downloads: No such file or directory
[cloudera@quickstart ~]$ cd Downloads
[cloudera@quickstart Downloads]$ ls
sample-3.txt
[cloudera@quickstart Downloads]$ hadoop fs -copyFromLocal sample-3.txt
25/09/11 03:55:34 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1281)
        at java.lang.Thread.join(Thread.java:1355)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DF
SOutputStream.java:967)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutpu
tStream.java:705)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStre
am.java:894)
[cloudera@quickstart Downloads]$ hadoop fs -ls
Found 1 items
-rw-r--r--   1 cloudera cloudera       1162 2025-09-11 03:55 sample-3.txt
[cloudera@quickstart Downloads]$ hadoop fs -cp sample-3.txt words.txt
25/09/11 03:57:08 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1281)
        at java.lang.Thread.join(Thread.java:1355)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DF
SOutputStream.java:967)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutpu
tStream.java:705)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStre
am.java:894)
[cloudera@quickstart Downloads]$ ls
sample-3.txt
[cloudera@quickstart Downloads]$ hadoop fs -ls
Found 2 items
-rw-r--r--   1 cloudera cloudera       1162 2025-09-11 03:55 sample-3.txt
-rw-r--r--   1 cloudera cloudera       1162 2025-09-11 03:57 words.txt
[cloudera@quickstart Downloads]$
[cloudera@quickstart Downloads]$ hadoop fs -copyToLocal words.txt
[cloudera@quickstart Downloads]$ ls
sample-3.txt  words.txt
[cloudera@quickstart Downloads]$
[cloudera@quickstart Downloads]$ hadoop fs -rm sample-3.txt
Deleted sample-3.txt
[cloudera@quickstart Downloads]$ hadoop fs -ls
Found 1 items
-rw-r--r--   1 cloudera cloudera       1162 2025-09-11 03:57 words.txt
[cloudera@quickstart Downloads]$
```

Experiment No: 3

Objective

• Execute the WordCount application.

• Copy the results from WordCount out of HDFS.

Program Code:

1. **Open a terminal shell.** Start the Cloudera VM in VirtualBox, if not already running, and open a terminal shell. Detailed instructions for these steps can be found in the previous readings.
2. **See example MapReduce programs.** Hadoop comes with several example MapReduce applications. You can see a list of them by running the following command:

```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-e
xamples.jar
An example program must be given as the first argument.
Valid program names are:
  aggregatewordcount: An Aggregate based map/reduce program that counts the word
s in the input files.
  aggregatewordhist: An Aggregate based map/reduce program that computes the his
togram of the words in the input files.
  bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact di
gits of Pi.
  dbcount: An example job that count the pageview counts from a database.
  distbbp: A map/reduce program that uses a BBP-type formula to compute exact bi
ts of Pi.
  grep: A map/reduce program that counts the matches of a regex in the input.
  join: A job that effects a join over sorted, equally partitioned datasets
  multifilewc: A job that counts words from several files.
  pentomino: A map/reduce tile laying program to find solutions to pentomino pro
blems.
  pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
  randomtextwriter: A map/reduce program that writes 10GB of random textual data
 per node.
  randomwriter: A map/reduce program that writes 10GB of random data per node.
  secondarysort: An example defining a secondary sort to the reduce.
  sort: A map/reduce program that sorts the data written by the random writer.
  sudoku: A sudoku solver.
  teragen: Generate data for the terasort
  terasort: Run the terasort
  teravalidate: Checking results of terasort
  wordcount: A map/reduce program that counts the words in the input files.
  wordmean: A map/reduce program that counts the average length of the words in
the input files.
  wordmedian: A map/reduce program that counts the median length of the words in
 the input files.
  wordstandarddeviation: A map/reduce program that counts the standard deviation
 of the length of the words in the input files.
[cloudera@quickstart ~]$ S
```

   We are interested in running WordCount. The output will indicate that WordCount takes the name of one or more input files and the name of the output directory. Note that these files are in HDFS, not the local file system.
3. Verify input file exists. In the previous reading, we downloaded the complete works of Shakespeare and copied them into HDFS. Let's make sure this file is still in HDFS so we can run WordCount on it. Run the following command:

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 11 items
drwxr-xr-x   - cloudera supergroup          0 2025-09-25 23:40 /bdalab_line
drwxrwxrwx   - hdfs     supergroup          0 2017-10-23 09:15 /benchmarks
drwxr-xr-x   - hbase    supergroup          0 2025-11-19 01:27 /hbase
drwxr-xr-x   - cloudera supergroup          0 2025-09-15 23:53 /lab_multi
-rw-r--r--   1 cloudera supergroup       1520 2025-09-11 23:10 /lab_multiarchived_file2.txt
-rw-r--r--   2 cloudera supergroup       1152 2025-09-11 23:32 /rep_test
drwxr-xr-x   - cloudera supergroup          0 2025-09-15 23:49 /sample_test
drwxr-xr-x   - solr     solr                0 2017-10-23 09:18 /solr
drwxrwxrwt   - hdfs     supergroup          0 2025-09-03 11:00 /tmp
drwxr-xr-x   - hdfs     supergroup          0 2017-10-23 09:17 /user
drwxr-xr-x   - hdfs     supergroup          0 2017-10-23 09:17 /var
[cloudera@quickstart ~]$
```

4. See WordCount command line arguments. We can learn how to run WordCount by examining its command-line arguments. Run the following command:

```
bash: /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar: Permission denied
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar
An example program must be given as the first argument.
Valid program names are:
  aggregatewordcount: An Aggregate based map/reduce program that counts the words in the input files.
  aggregatewordhist: An Aggregate based map/reduce program that computes the histogram of the words in the input file
s.
  bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
  dbcount: An example job that count the pageview counts from a database.
  distbbp: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.
  grep: A map/reduce program that counts the matches of a regex in the input.
  join: A job that effects a join over sorted, equally partitioned datasets
  multifilewc: A job that counts words from several files.
  pentomino: A map/reduce tile laying program to find solutions to pentomino problems.
  pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
  randomtextwriter: A map/reduce program that writes 10GB of random textual data per node.
  randomwriter: A map/reduce program that writes 10GB of random data per node.
  secondarysort: An example defining a secondary sort to the reduce.
  sort: A map/reduce program that sorts the data written by the random writer.
  sudoku: A sudoku solver.
  teragen: Generate data for the terasort
  terasort: Run the terasort
  teravalidate: Checking results of terasort
  wordcount: A map/reduce program that counts the words in the input files.
  wordmean: A map/reduce program that counts the average length of the words in the input files.
  wordmedian: A map/reduce program that counts the median length of the words in the input files.
  wordstandarddeviation: A map/reduce program that counts the standard deviation of the length of the words in the in
put files.
[cloudera@quickstart ~]$
```

5. Run WordCount. Run WordCount for words.txt by executing the following command:
As WordCount executes, Hadoop prints the progress in terms of Map and Reduce. When WordCount is complete, both will show 100%.

```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-e
xamples.jar wordcount /file.txt /out
25/11/19 04:42:53 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0
:8032
25/11/19 04:43:51 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1281)
        at java.lang.Thread.join(Thread.java:1355)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DF
SOutputStream.java:967)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutpu
tStream.java:705)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStre
am.java:894)
25/11/19 04:43:51 INFO input.FileInputFormat: Total input paths to process : 1
25/11/19 04:43:51 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1281)
        at java.lang.Thread.join(Thread.java:1355)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DF
SOutputStream.java:967)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutpu
tStream.java:705)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStre
am.java:894)
25/11/19 04:43:51 INFO mapreduce.JobSubmitter: number of splits:1
25/11/19 04:43:52 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_17
63544405865_0006
25/11/19 04:43:54 INFO impl.YarnClientImpl: Submitted application application_17
63544405865_0006
25/11/19 04:43:54 INFO mapreduce.Job: The url to track the job: http://quickstar
t.cloudera:8088/proxy/application_1763544405865_0006/
25/11/19 04:43:54 INFO mapreduce.Job: Running job: job_1763544405865_0006
25/11/19 04:44:44 INFO mapreduce.Job: Job job_1763544405865_0006 running in uber mode : false
25/11/19 04:44:44 INFO mapreduce.Job:  map 0% reduce 0%
25/11/19 04:45:18 INFO mapreduce.Job:  map 100% reduce 0%
25/11/19 04:45:40 INFO mapreduce.Job:  map 100% reduce 100%
25/11/19 04:45:42 INFO mapreduce.Job: Job job_1763544405865_0006 completed successfully
25/11/19 04:45:43 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=43
                FILE: Number of bytes written=287259
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
```

```
        FILE: Number of write operations=0
        HDFS: Number of bytes read=130
        HDFS: Number of bytes written=25
        HDFS: Number of read operations=6
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
Job Counters
        Launched map tasks=1
        Launched reduce tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=31135
        Total time spent by all reduces in occupied slots (ms)=18694
        Total time spent by all map tasks (ms)=31135
        Total time spent by all reduce tasks (ms)=18694
        Total vcore-milliseconds taken by all map tasks=31135
        Total vcore-milliseconds taken by all reduce tasks=18694
        Total megabyte-milliseconds taken by all map tasks=31882240
        Total megabyte-milliseconds taken by all reduce tasks=19142656
Map-Reduce Framework
        Map input records=1
        Map output records=4
        Map output bytes=41
        Map output materialized bytes=43
        Input split bytes=105
        Combine input records=4
        Combine output records=3
        Reduce input groups=3
        Reduce shuffle bytes=43
        Reduce input records=3
        Reduce output records=3
        Spilled Records=6
        Shuffled Maps =1
        Failed Shuffles=0
        Merged Map outputs=1
        GC time elapsed (ms)=739
        CPU time spent (ms)=1440
        Physical memory (bytes) snapshot=361984000
        Virtual memory (bytes) snapshot=3015593984
        Total committed heap usage (bytes)=226365440
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=25
File Output Format Counters
                Bytes Written=25
```

6. See WordCount output directory. Once WordCount is finished, verify that the output was created. First, check if the output directory, out, was created in HDFS by running the following command:

```
[cloudera@quickstart ~]$ hadoop fs -ls
Found 1 items
-rw-r--r--   1 cloudera cloudera       1162 2025-09-11 03:57 words.txt
[cloudera@quickstart ~]$
```

You should see two items in HDFS: words.txt (the text file we previously created) and out (the directory created by WordCount).

7. Look inside output directory. The directory created by WordCount contains several files. Look inside the directory by running the following command:

```
[cloudera@quickstart ~]$ hadoop fs -ls /out
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2025-11-19 04:45 /out/_SUCCESS
-rw-r--r--   1 cloudera supergroup         25 2025-11-19 04:45 /out/part-r-00000
[cloudera@quickstart ~]$
```

8. Copy WordCount results to local file system. Copy part-r-00000 to the local file system by running the following command:

```
[cloudera@quickstart ~]$ hadoop fs -copyToLocal /out/part-r-00000 local.txt
```

9. View the WordCount results. View the contents of the results file by running the following command:

```
[cloudera@quickstart ~]$ hadoop fs -copyToLocal /out/part-r-00000 local.txt
[cloudera@quickstart ~]$ more local.txt
hadoop  1
hello   2
world   1
```

Experiment No: 4

Objective

Running Hadoop MapReduce Programs.

Theory:

Hadoop comes with several MapReduce applications, which are available in the Cloudera VM at /usr/jars/hadoop-examples.jar. You can use these applications to perform various data processing tasks. To figure out how to run these programs, follow the instructions below.

Program Code:

1. To see a list of all available MapReduce applications in Hadoop, you can run the following command in the terminal:

```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-e
xamples-2.6.0-cdh5.13.0.jar
An example program must be given as the first argument.
Valid program names are:
  aggregatewordcount: An Aggregate based map/reduce program that counts the word
s in the input files.
  aggregatewordhist: An Aggregate based map/reduce program that computes the his
togram of the words in the input files.
  bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact di
gits of Pi.
  dbcount: An example job that count the pageview counts from a database.
  distbbp: A map/reduce program that uses a BBP-type formula to compute exact bi
ts of Pi.
  grep: A map/reduce program that counts the matches of a regex in the input.
  join: A job that effects a join over sorted, equally partitioned datasets
  multifilewc: A job that counts words from several files.
  pentomino: A map/reduce tile laying program to find solutions to pentomino pro
blems.
  pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
  randomtextwriter: A map/reduce program that writes 10GB of random textual data
 per node.
  randomwriter: A map/reduce program that writes 10GB of random data per node.
  secondarysort: An example defining a secondary sort to the reduce.
  sort: A map/reduce program that sorts the data written by the random writer.
  sudoku: A sudoku solver.
  teragen: Generate data for the terasort
  terasort: Run the terasort
  teravalidate: Checking results of terasort
  wordcount: A map/reduce program that counts the words in the input files.
  wordmean: A map/reduce program that counts the average length of the words in
the input files.
  wordmedian: A map/reduce program that counts the median length of the words in
 the input files.
  wordstandarddeviation: A map/reduce program that counts the standard deviation
 of the length of the words in the input files.
```

Get Usage Instructions for a Specific Application

2. To get detailed usage instructions for a specific MapReduce application, append the application name to the end of the command line. For example, to see how to run the wordcount application, execute:

```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-2.6.0-cdh5.13.0.jar wordcount
Usage: wordcount <in> [<in>...] <out>
```

Understanding the Usage Pattern

3. The usage pattern provided in the output explains the command-line arguments for the application:
   - <in> denotes the names of the input files. You can specify one or more input files.

- The square brackets around the second <1>mean that the second input is optional.
- The ellipsis (...) indicates that you can specify multiple input files.
- <out> denotes the name of the output directory.

4. The wordcount application is run with one or more input files specified first and the output directory specified last. For example:

```
[cloudera@quickstart Downloads]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-2.6.0-cdh5.13.0.jar w
ordcount sample-1.txt words.txt output_dir
25/11/19 06:53:53 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
25/11/19 06:53:54 INFO input.FileInputFormat: Total input paths to process : 2
25/11/19 06:53:54 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1281)
        at java.lang.Thread.join(Thread.java:1355)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DFSOutputStream.java:967)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutputStream.java:705)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:894)
25/11/19 06:53:54 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1281)
        at java.lang.Thread.join(Thread.java:1355)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DFSOutputStream.java:967)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutputStream.java:705)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:894)
25/11/19 06:53:54 INFO mapreduce.JobSubmitter: number of splits:2
25/11/19 06:53:55 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1763544405865_0007
25/11/19 06:53:55 INFO impl.YarnClientImpl: Submitted application application_1763544405865_0007
25/11/19 06:53:55 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_176
3544405865_0007/
25/11/19 06:53:55 INFO mapreduce.Job: Running job: job_1763544405865_0007
25/11/19 06:54:05 INFO mapreduce.Job: Job job_1763544405865_0007 running in uber mode : false
25/11/19 06:54:05 INFO mapreduce.Job:  map 0% reduce 0%
25/11/19 06:54:24 INFO mapreduce.Job:  map 50% reduce 0%
25/11/19 06:54:26 INFO mapreduce.Job:  map 100% reduce 0%
25/11/19 06:54:33 INFO mapreduce.Job:  map 100% reduce 100%
25/11/19 06:54:34 INFO mapreduce.Job: Job job_1763544405865_0007 completed successfully
25/11/19 06:54:34 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=2289
                FILE: Number of bytes written=435656
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2557
                HDFS: Number of bytes written=1508
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Killed map tasks=1
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=34808
                Total time spent by all reduces in occupied slots (ms)=5945
                Total time spent by all map tasks (ms)=34808
                Total time spent by all reduce tasks (ms)=5945
                Total vcore-milliseconds taken by all map tasks=34808
                Total vcore-milliseconds taken by all reduce tasks=5945
                Total megabyte-milliseconds taken by all map tasks=35643392
                Total megabyte-milliseconds taken by all reduce tasks=6087680
        Map-Reduce Framework
                Map input records=45
                Map output records=342
                Map output bytes=3619
                Map output materialized bytes=2295
                Input split bytes=243
                Combine input records=342
                Combine output records=176
                Reduce input groups=166
                Reduce shuffle bytes=2295
                Reduce input records=176
                Reduce output records=166
                Spilled Records=352
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=517
                CPU time spent (ms)=5570
                Physical memory (bytes) snapshot=587415552
                Virtual memory (bytes) snapshot=4520235008
                Total committed heap usage (bytes)=391979008
        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=2314
        File Output Format Counters
                Bytes Written=1508
```

Experiment No: 5

Objective

Customizing MapReduce Execution in Cloudera

Program Code:

Step 1: Writing the Mapper class

The Mapper class reads the input text and outputs key-value pairs, where the key is the word and the value is the count (1).

WordCountReducer.java | WordCountDriver.java | *WordCountMapper.java

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException{

        String[] words = value.toString().split("\\s+");

        for (String w : words) {
            word.set(w);
            context.write(word, one);

        }
    }
}

```

Step 2:Writing the Reducer class

The Reducer class receives the output from the Mapper and aggregates the word counts.

WordCountReducer.java | WordCountDriver.java | *WordCountMapper.java

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable>values, Context context)
    throws IOException, InterruptedException{
         int sum = 0;
         for (IntWritable val : values) {
             sum += val.get();
         }

         context.write(key,  new IntWritable(sum));
    }
}

```

Driver Code:

WordCountReducer.java | *WordCountDriver.java | *WordCountMapper.java

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

public class WordCountDriver {

    public static void main(String[] args) throws Exception {

        if (args.length != 2) {
            System.err.println("Usage: wordcount <input> <output>");
            System.exit(-1);
        }

        Job job = Job.getInstance(new Configuration());
        job.setJarByClass(WordCountDriver.class);

        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

Step 3: Runing the Program on Cloudera

Once the Java code is written, you compile and package it into a JAR file, then run it using Hadoop on Cloudera.

1. Upload input files to HDFS

```
[cloudera@quickstart ~]$ echo "Hello Cloudera Hadoop" > Text.txt
[cloudera@quickstart ~]$ echo "Hello Hadoop Hello Cloudera " >Text.txt
[cloudera@quickstart ~]$ hdfs dfs -put Text.txt /user/cloudera/input/
```

2. Run the MapReduce job:

```
[cloudera@quickstart ~]$ hadoop jar wordcount.jar WordCountDriver /user/cloudera
/input /user/cloudera/output
25/11/19 10:21:59 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0
:8032
25/11/19 10:21:59 WARN mapreduce.JobResourceUploader: Hadoop command-line option
 parsing not performed. Implement the Tool interface and execute your applicatio
n with ToolRunner to remedy this.
25/11/19 10:22:00 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1281)
        at java.lang.Thread.join(Thread.java:1355)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DF
SOutputStream.java:967)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutpu
tStream.java:705)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStre
am.java:894)
25/11/19 10:22:00 INFO input.FileInputFormat: Total input paths to process : 2
25/11/19 10:22:00 INFO mapreduce.JobSubmitter: number of splits:2
25/11/19 10:22:00 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_17
63544405865_0008
25/11/19 10:22:01 INFO impl.YarnClientImpl: Submitted application application_17
63544405865_0008
25/11/19 10:22:01 INFO mapreduce.Job: The url to track the job: http://quickstar
t.cloudera:8088/proxy/application_1763544405865_0008/
25/11/19 10:22:01 INFO mapreduce.Job: Running job: job_1763544405865_0008
25/11/19 10:22:19 INFO mapreduce.Job: Job job_1763544405865_0008 running in uber
 mode : false
25/11/19 10:22:19 INFO mapreduce.Job:  map 0% reduce 0%
25/11/19 10:22:44 INFO mapreduce.Job:  map 100% reduce 0%
25/11/19 10:22:57 INFO mapreduce.Job:  map 100% reduce 100%
25/11/19 10:22:57 INFO mapreduce.Job: Job job_1763544405865_0008 completed succe
ssfully
25/11/19 10:22:57 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=98
                FILE: Number of bytes written=429732
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=301
                HDFS: Number of bytes written=28
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
```

3. Check the output:

```
[cloudera@quickstart ~]$ hdfs dfs -cat /user/cloudera/output/part-r-00000
Cloudera        2
Hadoop  2
Hello   3
```

Experiment No: 6

Objective

Building Reliable MapReduce Applications

Program Code:

Step 1: Enable Speculative Execution

Speculative execution helps by launching duplicate tasks for slow-running tasks to improve performance. This can be configured in Cloudera Manager or by modifying mapred-site.xml.

```
</configuration>

<property>
<name>mapreduce.map.speclative</name>
<value>tru</name>
</property>

<property>
<name>mapreduce.reduce.speclative</name>
<value>true</value>
</property>

<property>
<name>mapreduce.task.timeout</name>
<value>600000</value>
</property>

<property>
<name>mapreduce.map.maxattempts</name>
<value>4</value>
</property>


sudo service hadoop-yarn-resourcemanager restart
sudo service hadoop-yarn-nodemanager restart
sudo service hadoop-mapreduce-historyserver restart
```

Step 2: Set Task Timeout and Retries

Configuring task timeouts and retries ensures that tasks that take too long or fail are automatically retried.

```
</configuration>

<property>
<name>mapreduce.map.speclative</name>
<value>tru</name>
</property>

<property>
<name>mapreduce.reduce.speclative</name>
<value>true</value>
</property>

<property>
<name>mapreduce.task.timeout</name>
<value>600000</value>
</property>

<property>
<name>mapreduce.map.maxattempts</name>
<value>4</value>
</property>


sudo service hadoop-yarn-resourcemanager restart
sudo service hadoop-yarn-nodemanager restart
sudo service hadoop-mapreduce-historyserver restart
```

Step 3: Run a Long-Running MapReduce Job

Now, you can run a long-running job to test fault tolerance.

This will calculate the value of Pi using MapReduce, and speculative execution will help speed up any slow tasks**.**

```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-2.6.0-cdh5.13.0.jar pi 100 10
00
Number of Maps  = 100
Samples per Map = 1000
Wrote input for Map #0
25/11/19 23:31:59 INFO mapreduce.Job:  map 100% reduce 33%
25/11/19 23:32:03 INFO mapreduce.Job:  map 100% reduce 100%
25/11/19 23:32:06 INFO mapreduce.Job: Job job_1763622694853_0001 completed successfully
25/11/19 23:32:08 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=2206
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=11800
        File Output Format Counters
                Bytes Written=97
Job Finished in 833.867 seconds
Estimated value of Pi is 3.14120000000000000000
[cloudera@quickstart ~]$
```

Experiment No: 7

Objective

Distributed Cache and Joins in Hadoop

Program Code:

1 Exercise 1: Using Distributed Cache in MapReduce

1.1 Step 1: Upload a File to HDFS

Upload a file to HDFS that will be used in the Distributed Cache.

```
[cloudera@quickstart ~]$ echo "1,John,Manager" >emp-details.txt
[cloudera@quickstart ~]$ hdfs dfs -mkdir -p /user/cloudera/cache
[cloudera@quickstart ~]$ hdfs dfs -put emp-details.txt /usr/cloudera/cache
put: `/usr/cloudera/cache': No such file or directory
[cloudera@quickstart ~]$ hdfs dfs -put emp-details.txt /user/cloudera/cache
25/11/20 10:36:09 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
```

1.2 Step 2: Modify the MapReduce Code to Use the Distributed Cache

In your MapReduce job, modify the Mapper class to read the cached file.

```
import org.apache.hadoop.io.LongWritable;
import java.util.HashMap;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.BufferedReader;
import java.io.FileReader;
import java.net.URI;

public class MapperDC extends Mapper<LongWritable, Text, Text, Text> {

private HashMap<String, String> empMap = new HashMap<>();

protected void setup(Context context){
try{
URI[] cacheFiles = context.getCacheFiles();

if(cacheFiles != null && cacheFiles.length > 0) {

BufferedReader Reader = new BufferedReader(
new FileReader("/mnt/hdfs"+cacheFiles[0].getPath()));


String line;
while ((line = reader.readLine()) != null){
String[] parts = line.split(",");
empMap.put(parts[0], parts[1]);
}
}
}catch(Exception e) {}
}

protected void map(LongWritable key, Text value, Context context)
throws java.io.IOException, InterruptedException{
String[] fields  = value.toString().split(",");
String empID = parts[0].trim();

String empId= fields[0];
String empNmae = empMap.get(empID);

if (empName != null){
context.write(new Text(empId), new Text(details));
}
```

1.3 Step 3: Run the Job with Distributed Cache

Run the MapReduce job with the Distributed Cache file:

```
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

public class ReducerDC extends Reducer<Text, Text, Text, Text> {
protected void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {

for (Text v: values) {
context.write(key, v);
}
}
}
```

Experiment No: 8

Objective

Running Hadoop Applications on Cloudera

Program Code:

1.1 Step 1: Create and Package the Hadoop Application

First, you need to create your custom MapReduce application. This typically involves writing the Mapper and Reducer classes in Java.

```
[cloudera@quickstart ~]$ mkdir wordcount
[cloudera@quickstart ~]$ cd wordcount
[cloudera@quickstart wordcount]$ WordCountMapper.java
bash: WordCountMapper.java: command not found
[cloudera@quickstart wordcount]$ pwd
/home/cloudera/wordcount
[cloudera@quickstart wordcount]$ ^C
[cloudera@quickstart wordcount]$ ls
[cloudera@quickstart wordcount]$ ls
[cloudera@quickstart wordcount]$ -ls
bash: -ls: command not found
[cloudera@quickstart wordcount]$ ls/
bash: ls/: No such file or directory
[cloudera@quickstart wordcount]$  ls/
bash: ls/: No such file or directory
[cloudera@quickstart wordcount]$
[cloudera@quickstart wordcount]$
[cloudera@quickstart wordcount]$
[cloudera@quickstart wordcount]$ nano WordCountMapper.java
[cloudera@quickstart wordcount]$ nano WordCountReducer.java
[cloudera@quickstart wordcount]$ nano WordCountDriver.java
[cloudera@quickstart wordcount]$ ls
WordCountDriver.java  WordCountMapper.java  WordCountReducer.java
```

Explanation: This command packages your compiled Mapper and Reducer classes into a JAR file named myhadoopapp.jar, which can be executed by Hadoop.

1.2 Step 2: Upload Input Files to HDFS

Next, create a directory in HDFS and upload your input files.

```
[cloudera@quickstart wordcount]$ javac -classpath `hadoop classpath` -d . WordCountMapper.java WordCountReducer.java
WordCountDriver.java
[cloudera@quickstart wordcount]$ jar -cf wordcount.jar *.class
[cloudera@quickstart wordcount]$ hdfs dfs -mkdir /user/cloudera/wordcount_input
mkdir: `/user/cloudera/wordcount_input': File exists
[cloudera@quickstart wordcount]$ hdfs dfs -put input.txt /user/cloudera/wordcount_input
```

Explanation: The first command creates a new directory in HDFS where your input files will reside. The second command uploads the file input.txt from your local filesystem to HDFS.

1.3 Step 3: Run the Application

Now, you can run your custom Hadoop application using the following command:

```
[cloudera@quickstart wordcount]$ hadoop jar wordcount.jar WordCountDriver /user/cloudera/wordcount_input /user/cloude
ra/wordcount_output
25/11/20 09:46:41 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
25/11/20 09:46:56 INFO mapreduce.Job: Running job: job_1763635503529_0001
25/11/20 09:47:53 INFO mapreduce.Job: Job job_1763635503529_0001 running in uber mode : false
25/11/20 09:47:53 INFO mapreduce.Job:  map 0% reduce 0%
25/11/20 09:48:26 INFO mapreduce.Job:  map 100% reduce 0%
25/11/20 09:48:51 INFO mapreduce.Job:  map 100% reduce 100%
25/11/20 09:48:52 INFO mapreduce.Job: Job job_1763635503529_0001 completed successfully
25/11/20 09:48:53 INFO mapreduce.Job: Counters: 49
```

Explanation: In this command, MyMainClass is the entry point of your MapReduce job. The command specifies the input and output paths in HDFS. The job will read data from /user/cloudera/input and write the output to /user/cloudera/output.

1.4 Step 4: Check the Output

After the job completes, you can check the output stored in HDFS.

```
[cloudera@quickstart wordcount]$ hdfs dfs -ls /user/cloudera/wordcount_output
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2025-11-20 09:48 /user/cloudera/wordcount_output/_SUCCESS
-rw-r--r--   1 cloudera cloudera         37 2025-11-20 09:48 /user/cloudera/wordcount_output/part-r-00000
[cloudera@quickstart wordcount]$
[cloudera@quickstart wordcount]$ hdfs dfs -cat /user/cloudera/wordcount_output/part-r-00000
hadoop  1
hello   2
mapreduce       1
world   1
[cloudera@quickstart wordcount]$
```

Explanation: The first command lists the files in the output directory, and the second command displays the content of the output file (typically named part-r-00000).

1.5 Step 5: Monitor the Job Execution

You can monitor the execution of your MapReduce job using the Cloudera Manager web interface.

1. Open Cloudera Manager in your web browser.
2. Navigate to the YARN Applications section.
3. Find your job in the list to see its status and resource usage.

Explanation: Cloudera Manager provides an easy way to track the performance and resource consumption of your running Hadoop applications.