

CS4830 Big Data Laboratory

Jan – May 2021

Lab 4 - Assignment



ME17B158 - Omkar Nath

1. Write a spark code for executing the Hash example provided in slide 14 on Hashing from Lab 1 Presentation, on the public file: *gs://bucket_two_2/hash_file.txt*. You would have to find the number of user clicks between 0-6, 6-12, 12-18, and 18-24, as was discussed in the first class.

a. Submit the python file with your code.

b. Also, provide the text file containing your output. [6 marks]

Attached Code: “count_clicks.py”

Attached Output: “output_lab4_part-00000.txt”

Assumption:

The group from 0 to 6 implies from 0 to 5:59. 6-12 implies 6:00 to 11:59.

Sequence flow for the execution of the code:

Defining Variables:

```
me17b158@instance-1:~$ PROJECT=splendid-sector-305218
me17b158@instance-1:~$ BUCKET_NAME=me17b158_cs4830
me17b158@instance-1:~$ CLUSTER=lab4
me17b158@instance-1:~$ REGION=us-central1
me17b158@instance-1:~$
```

Creating Cluster:

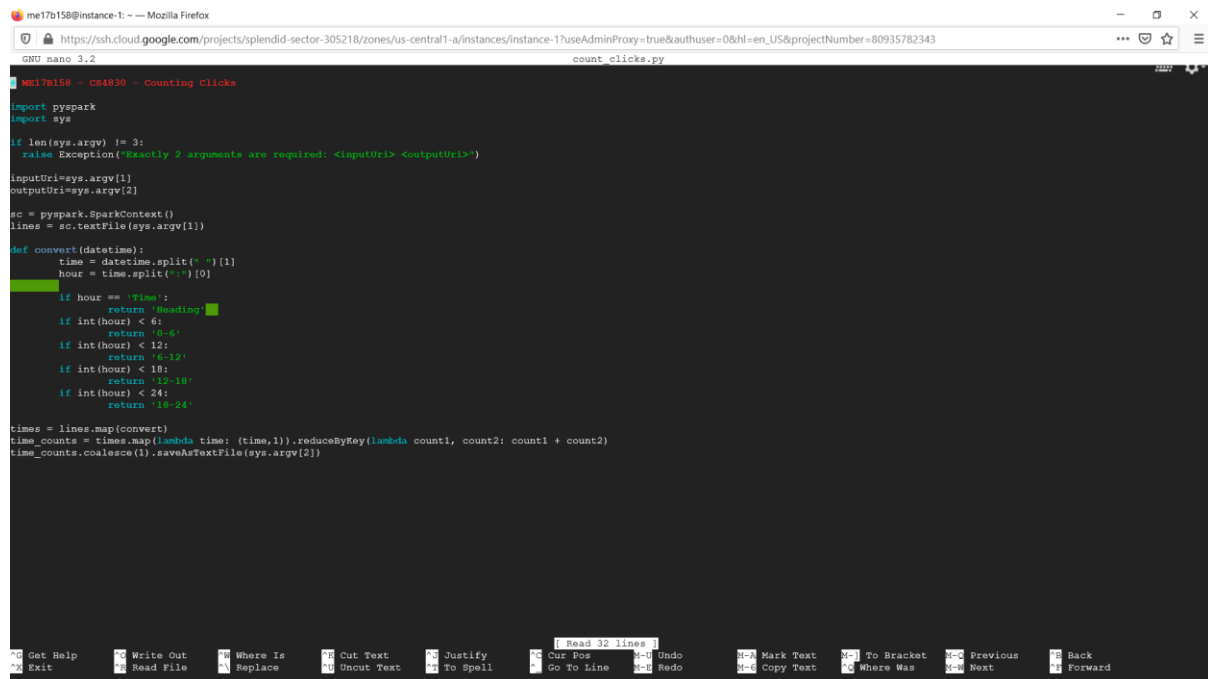
```
me17b158@instance-1:~$ gcloud dataproc clusters create ${CLUSTER} \
> --project=${PROJECT} \
> --region=${REGION} \
> --single-node

Waiting on operation [projects/splendid-sector-305218/regions/us-central1/operations/166014b4-ed6d-350b-8333-eff99dfc3fe5].
Waiting for cluster creation operation...
WARNING: No image specified. Using the default image version. It is recommended to select a specific image version in production, as the default image version may change at any time.
WARNING: For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consistently high I/O performance. See https://cloud.google.com/compute/docs/disks/performance for information on disk I/O performance.
Waiting for cluster creation operation...done.
Created [https://dataproc.googleapis.com/v1/projects/splendid-sector-305218/regions/us-central1/clusters/lab4] Cluster placed in zone [us-central1-f].
me17b158@instance-1:~$
```

Copying Data:

```
me17b158@instance-1:~$ gsutil cp gs://bucket_two_2/hash_file.txt \
> gs://me17b158_cs4830/input/hash_file.txt
Copying gs://bucket_two_2/hash_file.txt [Content-Type=text/plain]...
/ [1 files][ 40.0 MiB/ 40.0 MiB]
Operation completed over 1 objects/40.0 MiB.
me17b158@instance-1:~$ nano count_clicks.py
```

Code to count clicks:



```
me17b158@instance-1: ~ -- Mozilla Firefox
https://ssh.cloud.google.com/projects/splendid-sector-305218/zones/us-central1-a/instances/instance-17useAdminProxy=true&authuser=0&hl=en_US&projectNumber=80935782343
GNU nano 3.2 count_clicks.py
me17b158 - CS4830 - Counting Clicks
import pyspark
import sys

if len(sys.argv) != 3:
    raise Exception("Exactly 2 arguments are required: <inputUri> <outputUri>")

inputUri=sys.argv[1]
outputUri=sys.argv[2]

sc = pyspark.SparkContext()
lines = sc.textFile(sys.argv[1])

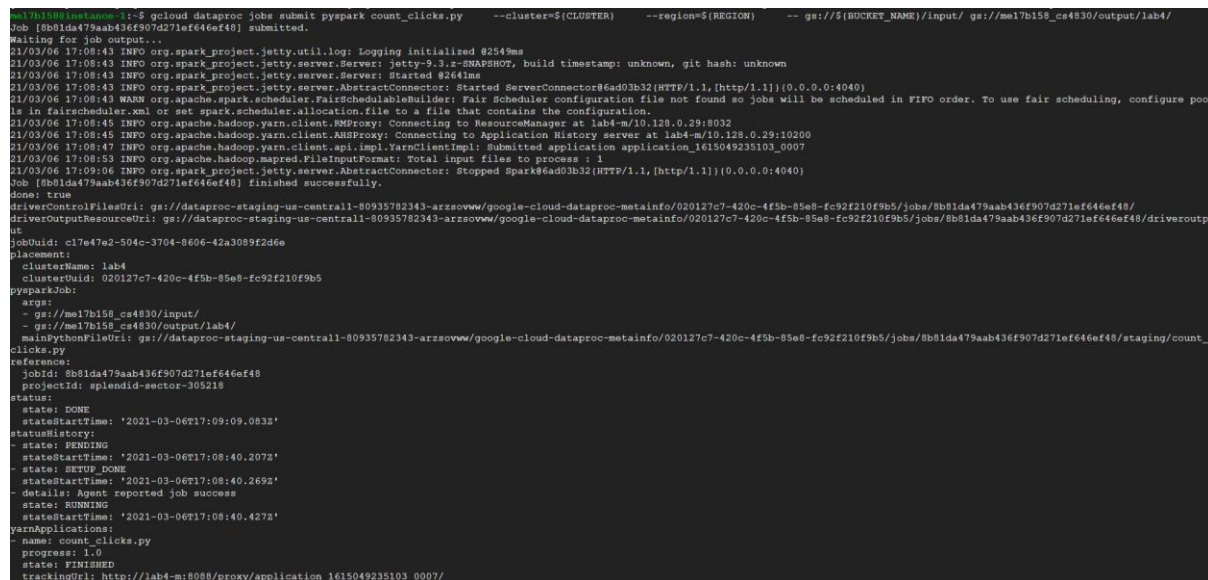
def convert(datetime):
    time = datetime.split(" ")[1]
    hour = time.split(":")[0]

    if hour == "Time":
        return "Heading"
    if int(hour) < 6:
        return "0-6"
    if int(hour) < 12:
        return "6-12"
    if int(hour) < 18:
        return "12-18"
    if int(hour) < 24:
        return "18-24"

times = lines.map(convert)
time_counts = times.map(lambda time: (time,1)).reduceByKey(lambda count1, count2: count1 + count2)
time_counts.coalesce(1).saveAsTextFile(sys.argv[2])

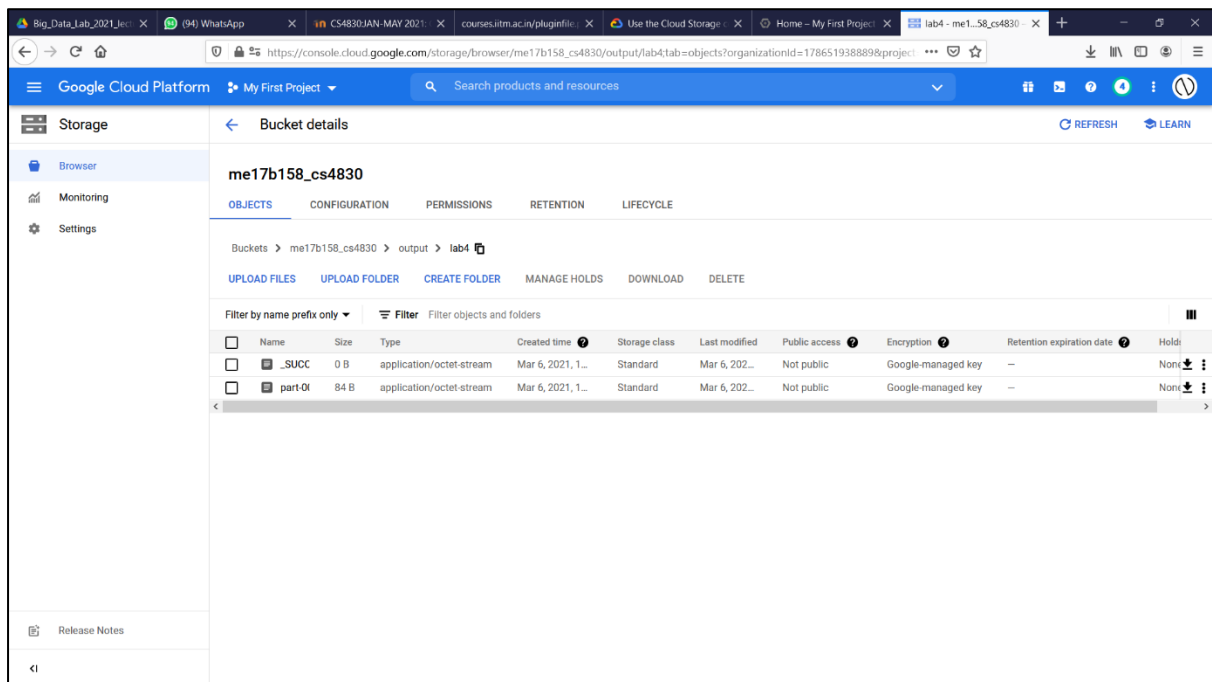
Get Help Write Out Where Is Cut Text Justify Cur Pos Mark Text To Bracket Previous Back
Exit Read File Replace Uncut Text To Spell Go To Line Redo Copy Text Where Was Next Forward
```

Executing File:



```
me17b158@instance-1: ~$ gcloud dataproc jobs submit pyspark count_clicks.py --cluster=$(CLUSTER) --region=$(REGION) --gs://$(BUCKET_NAME)/input/ gs://me17b158_cs4830/output/lab4/
Job [8b81da479aab436f907d271ef646ef48] submitted.
Waiting for job output...
21/03/06 17:08:43 INFO org.spark.project.jetty.util.log: Logging initialized $254ms
21/03/06 17:08:43 INFO org.spark.project.jetty.server.Server: jetty-9.3.2-SNAPSHOT, build timestamp: unknown, git hash: unknown
21/03/06 17:08:43 INFO org.spark.project.jetty.server.Server: Started $264ms
21/03/06 17:08:43 INFO org.spark.project.jetty.server.AbstractConnector: Started ServerConnector$86ad03b32(HTTP/1.1,[http/1.1])(0.0.0.0:4040)
21/03/06 17:08:43 WARN org.apache.spark.scheduler.FairSchedulerBuilder: Fair Scheduler configuration file not found so jobs will be scheduled in FIFO order. To use fair scheduling, configure poo
is in fairscheduler.xml or set spark.scheduler.allocation.file to a file that contains the configuration.
21/03/06 17:08:45 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at lab4-m/10.128.0.29:8032
21/03/06 17:08:45 INFO org.apache.hadoop.yarn.client.AMRProxy: Connecting to Application History server at lab4-m/10.128.0.29:10200
21/03/06 17:08:47 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1615049235103_0007
21/03/06 17:08:53 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
21/03/06 17:09:06 INFO org.spark.project.jetty.server.AbstractConnector: Stopped Spark$86ad03b32(HTTP/1.1,[http/1.1])(0.0.0.0:4040)
Job [8b81da479aab436f907d271ef646ef48] finished successfully.
Done! true
driverControlFilesUri: gs://dataproc-staging-us-central1-80935782343-arzsovw/google-cloud-dataproc-metainfo/020127c7-420c-4f5b-85e8-fc92f210f9b5/jobs/8b81da479aab436f907d271ef646ef48/
driverOutputResourceUri: gs://dataproc-staging-us-central1-80935782343-arzsovw/google-cloud-dataproc-metainfo/020127c7-420c-4f5b-85e8-fc92f210f9b5/jobs/8b81da479aab436f907d271ef646ef48/driveroutp
ut
jobUuid: c17e47e2-504c-3704-8606-42a3089f2d6e
placement:
  clusterName: lab4
  clusterUuid: 020127c7-420c-4f5b-85e8-fc92f210f9b5
pysparkJob:
  args:
  - gs://me17b158_cs4830/input/
  - gs://me17b158_cs4830/output/lab4/
  mainPythonFileUri: gs://dataproc-staging-us-central1-80935782343-arzsovw/google-cloud-dataproc-metainfo/020127c7-420c-4f5b-85e8-fc92f210f9b5/jobs/8b81da479aab436f907d271ef646ef48/staging/count_
clicks.py
reference:
  jobId: 8b81da479aab436f907d271ef646ef48
  projectId: splendid-sector-305218
  status:
    state: DONE
    stateStartTime: "2021-03-06T17:09:08.083z"
    stateHistory:
      - state: PENDING
        stateStartTime: "2021-03-06T17:08:40.207z"
      - state: SETUP_DONE
        stateStartTime: "2021-03-06T17:08:40.269z"
      - details: Agent reported job success
        state: RUNNING
        stateStartTime: "2021-03-06T17:08:40.427z"
  yarnApplications:
  - name: count_clicks.py
    progress: 1.0
    state: FINISHED
  trackingUri: http://lab4-m:8088/proxy/application_1615049235103_0007/
```

Output file in the bucket:



Results:



2. Provide a brief description of the functionality of the following services:

- a. HDFS
- b. Hive
- c. Pig
- d. Yarn [4 marks]

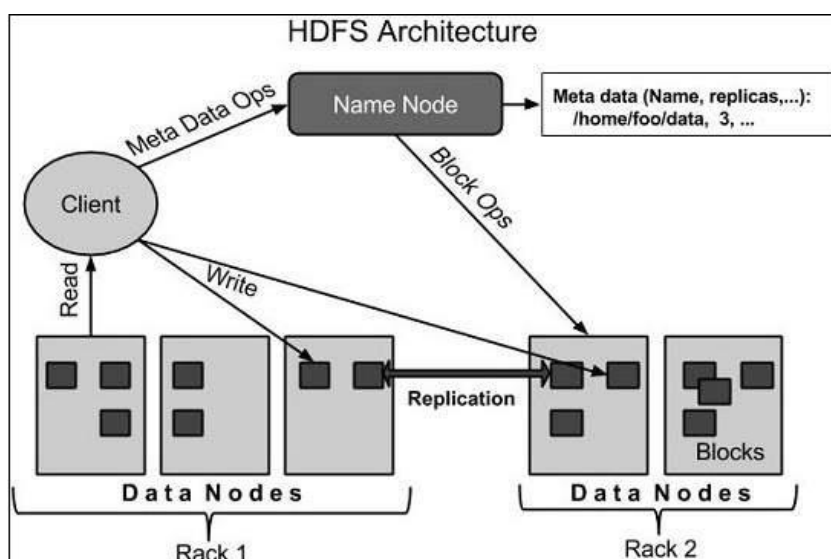
a. HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is now an Apache Hadoop subproject. The project URL is <https://hadoop.apache.org/hdfs/>.

The key features of HDFS are:

- 1. Cost-effective
- 2. Large Datasets/ Variety and volume of data
- 3. Replication
- 4. Fault Tolerance and reliability
- 5. High Availability
- 6. Scalability
- 7. Data Integrity
- 8. High Throughput
- 9. Data Locality

HDFS Architecture:



b. Hive

The Apache Hive™ data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL. Structure can be projected onto data already in storage. A command line tool and JDBC driver are provided to connect users to Hive.

Apache Hive supports analysis of large datasets stored in Hadoop's [HDFS](#) and compatible file systems such as [Amazon S3](#) filesystem and [Alluxio](#). It provides a [SQL](#)-like query language called HiveQL^[8] with schema on read and transparently converts queries to [MapReduce](#), Apache Tez^[9] and [Spark](#) jobs. All three execution engines can run in [Hadoop](#)'s resource negotiator, YARN (Yet Another Resource Negotiator).

Other features of Hive include:

- Different storage types such as plain text, [RCFile](#), [HBase](#), ORC, and others.
- Metadata storage in a [relational database management system](#), significantly reducing the time to perform semantic checks during query execution.
- Operating on compressed data stored into the Hadoop ecosystem using algorithms including [DEFLATE](#), [BWT](#), [snappy](#), etc.
- Built-in [user-defined functions](#) (UDFs) to manipulate dates, strings, and other data-mining tools. Hive supports extending the UDF set to handle use-cases not supported by built-in functions.
- SQL-like queries (HiveQL), which are implicitly converted into MapReduce or Tez, or Spark jobs.

c. Pig

Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

At the present time, Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist (e.g., the Hadoop subproject). Pig's language layer currently consists of a textual language called Pig Latin, which has the following key properties:

- **Ease of programming.** It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain.
- **Optimization opportunities.** The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.
- **Extensibility.** Users can create their own functions to do special-purpose processing.

d. Yarn

YARN stands for “Yet Another Resource Negotiator “. It was introduced in Hadoop 2.0 to remove the bottleneck on Job Tracker which was present in Hadoop 1.0. YARN was described as a “Redesigned Resource Manager” at the time of its launching, but it has now evolved to be known as large-scale distributed operating system used for Big Data processing. YARN architecture basically separates resource management layer from the processing layer. In Hadoop 1.0 version, the responsibility of Job tracker is split between the resource manager and application manager.

YARN also allows different data processing engines like graph processing, interactive processing, stream processing as well as batch processing to run and process data stored in HDFS (Hadoop Distributed File System) thus making the system much more efficient. Through its various components, it can dynamically allocate various resources and schedule the application processing. For large volume data processing, it is quite necessary to manage the available resources properly so that every application can leverage them.

Key Features of YARN:

1. **Multi-tenancy:** YARN has allowed access to multiple data processing engines such as batch processing engine, stream processing engine, interactive processing engine, graph processing engine and much more. This has given the benefit of multi-tenancy to the company.
2. **Cluster Utilization:** Clusters are utilized in an optimized way because clusters are used dynamically in Hadoop with the help of YARN.
3. **Compatibility:** YARN is also compatible with the first version of Hadoop, i.e. Hadoop 1.0, because it uses the existing map-reduce apps. So YARN can also be used with Hadoop 1.0.
4. **Scalability:** Thousands of clusters and nodes are allowed by the scheduler in Resource Manager of YARN to be managed and extended by Hadoop.