For this project, a hadoop cluster had to be set up which consists of a master and at least two worker nodes that are interconnected together by a network, be it private or internet. Hadoop is popular for being able to store large amount of data in distributed fashion called HDFS and they can also process huge amount of data in parallel using MapReduce. For this project, we are going to deploy a Hadoop cluster in Google Cloud using *Dataproc* that has one master and two slaves. We are also going to run two jobs, one being counting number of words in a large text file, and another being estrimating value of PI using Monte Carlo methods.

**Setting Up Hadoop/Spark Service using DataProc:**

First step in creating a Hadoop cluster is creating a yaml file which describes the properties of our cluster. Since I already did have a cluster running, I wanted to simply copy the configuration yaml from the running cluster to my local machine.

#Copying already running cluster configuration into local

gcloud dataproc clusters export cluster-898a --destination cluster.yaml --region us-central1

#Config file for our cluster

config:

configBucket: dataproc-staging-us-central1-475931246002-ptsog0z7

encryptionConfig: {}

endpointConfig: {}

gceClusterConfig:

networkUri: https://www.googleapis.com/compute/v1/projects/codingrant/global/networks/default

serviceAccountScopes:

- https://www.googleapis.com/auth/bigquery

- https://www.googleapis.com/auth/bigtable.admin.table

- https://www.googleapis.com/auth/bigtable.data

- https://www.googleapis.com/auth/cloud.useraccounts.readonly

- https://www.googleapis.com/auth/devstorage.full\_control

- https://www.googleapis.com/auth/devstorage.read\_write

- https://www.googleapis.com/auth/logging.write

shieldedInstanceConfig: {}

zoneUri: https://www.googleapis.com/compute/v1/projects/codingrant/zones/us-central1-a

masterConfig:

diskConfig:

bootDiskSizeGb: 500

bootDiskType: pd-standard

imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20210311-093551-rc01

machineTypeUri: https://www.googleapis.com/compute/v1/projects/codingrant/zones/us-central1-a/machineTypes/n1-standard-4

minCpuPlatform: AUTOMATIC

numInstances: 1

preemptibility: NON\_PREEMPTIBLE

securityConfig:

kerberosConfig: {}

softwareConfig:

imageVersion: 1.5.34-ubuntu18

optionalComponents:

- HIVE\_WEBHCAT

- ANACONDA

- DOCKER

properties:

capacity-scheduler:yarn.scheduler.capacity.root.default.ordering-policy: fair

core:fs.gs.block.size: '134217728'

core:fs.gs.metadata.cache.enable: 'false'

core:hadoop.ssl.enabled.protocols: TLSv1,TLSv1.1,TLSv1.2

distcp:mapreduce.map.java.opts: -Xmx768m

distcp:mapreduce.map.memory.mb: '1024'

distcp:mapreduce.reduce.java.opts: -Xmx768m

distcp:mapreduce.reduce.memory.mb: '1024'

hdfs:dfs.datanode.address: 0.0.0.0:9866

hdfs:dfs.datanode.http.address: 0.0.0.0:9864

hdfs:dfs.datanode.https.address: 0.0.0.0:9865

hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867

hdfs:dfs.namenode.handler.count: '20'

hdfs:dfs.namenode.http-address: 0.0.0.0:9870

hdfs:dfs.namenode.https-address: 0.0.0.0:9871

hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868

hdfs:dfs.namenode.secondary.https-address: 0.0.0.0:9869

hdfs:dfs.namenode.service.handler.count: '10'

hive:hive.fetch.task.conversion: none

mapred-env:HADOOP\_JOB\_HISTORYSERVER\_HEAPSIZE: '3840'

mapred:mapreduce.job.maps: '21'

mapred:mapreduce.job.reduce.slowstart.completedmaps: '0.95'

mapred:mapreduce.job.reduces: '7'

mapred:mapreduce.jobhistory.recovery.store.class: org.apache.hadoop.mapreduce.v2.hs.HistoryServerLeveldbStateStoreService

mapred:mapreduce.map.cpu.vcores: '1'

mapred:mapreduce.map.java.opts: -Xmx2457m

mapred:mapreduce.map.memory.mb: '3072'

mapred:mapreduce.reduce.cpu.vcores: '1'

mapred:mapreduce.reduce.java.opts: -Xmx2457m

mapred:mapreduce.reduce.memory.mb: '3072'

mapred:mapreduce.task.io.sort.mb: '256'

mapred:yarn.app.mapreduce.am.command-opts: -Xmx2457m

mapred:yarn.app.mapreduce.am.resource.cpu-vcores: '1'

mapred:yarn.app.mapreduce.am.resource.mb: '3072'

spark-env:SPARK\_DAEMON\_MEMORY: 3840m

spark:spark.driver.maxResultSize: 1920m

spark:spark.driver.memory: 3840m

spark:spark.executor.cores: '2'

spark:spark.executor.instances: '2'

spark:spark.executor.memory: 5586m

spark:spark.executorEnv.OPENBLAS\_NUM\_THREADS: '1'

spark:spark.scheduler.mode: FAIR

spark:spark.sql.cbo.enabled: 'true'

spark:spark.ui.port: '0'

spark:spark.yarn.am.memory: 640m

yarn-env:YARN\_NODEMANAGER\_HEAPSIZE: '3840'

yarn-env:YARN\_RESOURCEMANAGER\_HEAPSIZE: '3840'

yarn-env:YARN\_TIMELINESERVER\_HEAPSIZE: '3840'

yarn:yarn.nodemanager.address: 0.0.0.0:8026

yarn:yarn.nodemanager.resource.cpu-vcores: '4'

yarn:yarn.nodemanager.resource.memory-mb: '12288'

yarn:yarn.resourcemanager.nodemanager-graceful-decommission-timeout-secs: '86400'

yarn:yarn.scheduler.maximum-allocation-mb: '12288'

yarn:yarn.scheduler.minimum-allocation-mb: '1024'

tempBucket: dataproc-temp-us-central1-475931246002-1uwyswuw

workerConfig:

diskConfig:

bootDiskSizeGb: 500

bootDiskType: pd-standard

imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20210311-093551-rc01

machineTypeUri: https://www.googleapis.com/compute/v1/projects/codingrant/zones/us-central1-a/machineTypes/n1-standard-4

minCpuPlatform: AUTOMATIC

numInstances: 2

preemptibility: NON\_PREEMPTIBLE

#Running a cluster using the yaml file

gcloud dataproc clusters import my-hadoop-cluster --source cluster.yaml --region us-central1

This command enables us to successfully run a spark/Hadoop cluster with one master and 2 slaves. It has HDFS and spark service running in all the nodes in the cluster.

**Running my first spark Job- Counting Words:**

Since, master and slaves we created earlier has everything set up for us, we can now start working on trying to get out first job running in the pyspark. First let’s SSH into the master node and then download the file whose texts we will count by the end of this section.

#Downloading the text version of our text file

pawanbhatta178@cluster-898a-m:~$ wget <https://www.gutenberg.org/files/1342/1342-0.txt>

#Copying the file into a folder that I want to share with my HDFS cluster

pawanbhatta178@cluster-898a-m:~$ sudo cp 1342-0.txt ./hadoopFiles/prideAndPrejudice.txt

#Checking if the file was successfully created

pawanbhatta178@cluster-898a-m:~$ cd hadoopFiles/

pawanbhatta178@cluster-898a-m:~/hadoopFiles$ ls

prideAndPrejudice.txt

Now, it is time to upload the folder from the local file system to HDFS. We will do it using Hadoop commands.

#Saving the text file into HDFS cluster

pawanbhatta178@cluster-898a-m:~$ hadoop fs -copyFromLocal ./hadoopFiles /

pawanbhatta178@cluster-898a-m:~$ hadoop fs -ls /

Found 3 items

drwxr-xr-x - pawanbhatta178 hadoop 0 2021-04-18 17:10 /hadoopFiles

drwxrwxrwt - hdfs hadoop 0 2021-04-18 13:49 /tmp

drwxrwxrwt - hdfs hadoop

Then I used python to write some code that reads the text from HDFS, performs MapReduce and prints the output to a file.

#Python code that returns top 20 frequently used word using " " delimiter.

import sys

from pyspark import SparkContext, SparkConf

if \_\_name\_\_ == "\_\_main\_\_":

# create Spark context with necessary configuration

sc = SparkContext("local","PySpark Word Count Exmaple")

# read data from text file and split each line into words

words = sc.textFile("/hadoopFiles/prideAndPrejudice.txt").flatMap(lambda line: line.split(" "))

# count the occurrence of each word

wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)

sortedCount=wordCounts.sortBy(lambda a:a[1], ascending=False)

top20=sortedCount.take(20)

# save the counts to output

sc.parallelize(top20).saveAsTextFile("/A9")

Now it’s time to run our code and see if it works. We use spark-submit command to submit the job which will be run by the Spark and it will write to a output file which we can read.

#Running the python code

pawanbhatta178@cluster-898a-m:~$ spark-submit wordCount2.py

#Output

pawanbhatta178@cluster-898a-m:~$ hadoop fs -cat /A9/part-00000

('', 73700)

('the', 4216)

('to', 4123)

('of', 3667)

('and', 3309)

('a', 1944)

('her', 1856)

('in', 1817)

('was', 1796)

('I', 1725)

('that', 1417)

('not', 1363)

('she', 1303)

('be', 1209)

('his', 1166)

('had', 1125)

('as', 1119)

('with', 1040)

('he', 1039)

('for', 1003)

**Running Monte Carlo method to estimate PI:**

We can write simple naïve code that runs on a single computer to estimate the value of PI. It is much easier to write and understand the code we write that way, but it may not always be the preferred way especially when the task is very large. It might even take couple of days to run a single huge processing job in a single computer. That’s where Spark comes in. We can use Spark to run independent functions, and we can combine or reduce the result obtained from each function to get the result we want much faster. Task that takes couple of days will now be completed in some hours or even minutes.

Here we estimate value of PI harnessing the power of spark. We run a function *isPointInsideCircle* a million times. Since the function is independent from each other, we can run them in parallel harnessing the power of Spark. Since each of the function will return whether the random point falls inside a circle or not, we can add the number of times the function returns 1 using reduce function. This happens all at once in different nodes. That way, we will have our result much faster.

Now let’s see the code for the method.

from time import time

import numpy as np

from random import random

from operator import add

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('PiEstimator').getOrCreate()

sc = spark.sparkContext

#n is the number of times the sample will be taken, as n increases accuracy of PI incre$

n = 1000000

def isPointInsideCircle(p):

x, y = random(), random()

return 1 if x\*x + y\*y < 1 else 0

startingTime = time()

# parallelize creates a spark Resilient Distributed Dataset (RDD)

# its values are useless in this case

# but allows us to distribute our calculation (inside function) across multiple nodes

# count stores how many times a randomly sampled point falls in unit circle.

# We get the value of PI by taking the ratio of count to n and multiplying it with 4.

# It is because areaOfUnitCircle/areaOfUnitCircle = 4\* PI.

# We divide both sides by 4 and we can estimate PI.

count = sc.parallelize(range(0, n)).map(isPointInsideCircle).reduce(add)

print(np.round(time()-startingTime, 3), "seconds elapsed with n=", n)

print("Pi as estimated by Monte Carlo Method is %f" % (4.0 \* count / n))

spark.stop()

#Result of Monte Carlo Method for estimating Pi

pawanbhatta178@cluster-898a-m:~$ spark-submit piEstimator.py

21/04/18 22:55:08 INFO org.spark\_project.jetty.util.log: Logging initialized @3000ms

21/04/18 22:55:08 INFO org.spark\_project.jetty.server.Server: jetty-9.3.z-SNAPSHOT, build timestamp: unknown, git hash: unknown

21/04/18 22:55:08 INFO org.spark\_project.jetty.server.Server: Started @3113ms

21/04/18 22:55:08 INFO org.spark\_project.jetty.server.AbstractConnector: Started ServerConnector@6f6720c6{HTTP/1.1,[http/1.1]}{0.0.0.0:36841}

21/04/18 22:55:09 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at cluster-898a-m/10.128.0.7:8032

21/04/18 22:55:09 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at cluster-898a-m/10.128.0.7:10200

21/04/18 22:55:09 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found

21/04/18 22:55:09 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.

21/04/18 22:55:09 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE

21/04/18 22:55:09 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE

21/04/18 22:55:11 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted

application application\_1618753755784\_0005

5.34 seconds elapsed with n= 1000000

Pi as estimated by Monte Carlo Method is 3.141352

21/04/18 22:55:24 INFO org.spark\_project.jetty.server.AbstractConnector: Stopped Spark@6

f6720c6{HTTP/1.1,[http/1.1]}{0.0.0.0:0}