

EECS 6893
HW1
Yutao Zhou
UNI: yz4395

1. Iterative K-means clustering on Spark

(1)

```
1 import operator
2 import sys
3 from pyspark import SparkConf, SparkContext
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from scipy import linalg
7 from pyspark.mllib.feature import Normalizer
8 from sklearn.manifold import TSNE
9
10 # Macros.
11 MAX_ITER = 20
12 DATA_PATH = "gs://eecs6893_data/q1/data.txt"
13 C1_PATH = "gs://eecs6893_data/q1/c1.txt"
14 C2_PATH = "gs://eecs6893_data/q1/c2.txt"
15 NORM = 2
16
17 # Helper functions.
18 def closest(p, centroids, norm):
19     """
20         Compute closest centroid for a given point.
21         Args:
22             p (numpy.ndarray): input point
23             centroids (list): A list of centroids points
24             norm (int): 1 or 2
25         Returns:
26             int: The index of closest centroid.
27     """
28     closest_c = min([(i, linalg.norm(p - c, norm))
29                      for i, c in enumerate(centroids)],
30                      key=operator.itemgetter(1))[0]
31     return closest_c
32
33 # K-means clustering
```

```

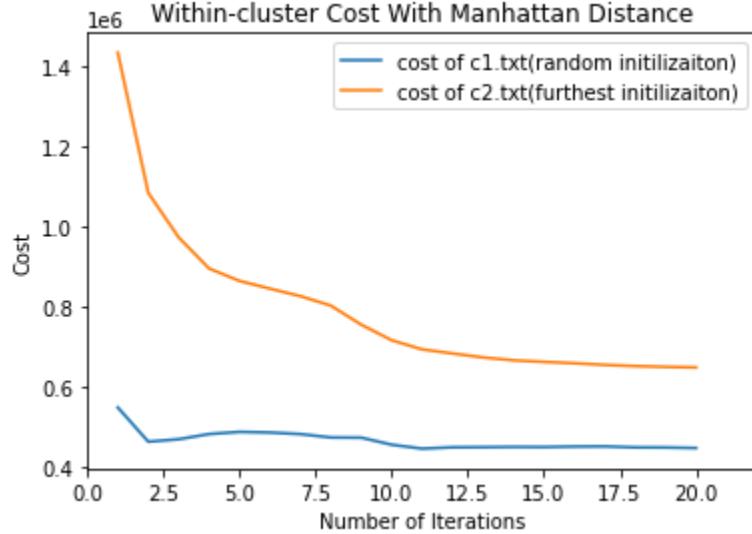
34     def getClosestCentroid(pt, centroids, norm=2):
35         c_index = closest(pt, centroids, norm)
36         cost = linalg.norm(pt - centroids[c_index], norm)
37         return (c_index, (pt, 1, cost))
38
39     def calculateCost(costData):
40         totalCost, numPoints = 0, 0
41         for cluster in costData:
42             totalCost += cluster[1][2]
43             numPoints += cluster[1][1]
44         return totalCost
45
46     def kmeans(data, centroids, norm=2):
47         """
48             Conduct k-means clustering given data and centroid.
49             This is the basic version of k-means, you might need more
50             code to record cluster assignment to plot TSNE, and more
51             data structure to record cost.
52             Args:
53                 data (RDD): RDD of points
54                 centroids (list): A list of centroids points
55                 norm (int): 1 or 2
56             Returns:
57                 RDD: assignment information of points, a RDD of (centroid, (point, 1))
58                 list: a list of centroids
59                 and define yourself...
60         """
61         # iterative k-means
62         cost = []
63         for _ in range(MAX_ITER):
64             # Transform each point to a combo of point, closest centroid, count=1
65             # point -> (closest_centroid, (point, 1))
66             clusters = data.map(lambda pt: getClosestCentroid(pt, centroids, norm)).reduceByKey(lambda count1, count2: \
67
67             (count1[0] + count2[0], count1[1] + count2[1], count1[2] + count2[2]))
68             # Re-compute cluster center
69             # For each cluster center (key), aggregate its values
70             centroids = clusters.map(lambda x: x[1][0]/x[1][1]).collect()
71             # by summing up points and count
72             # Average the points for each centroid: divide sum of points by count
73             # Use collect() to turn RDD into list
74             cost.append(calculateCost(clusters.collect()))
75         return cost, centroids
76
77     def plotCost(c1, c2, distanceType, MAX_ITER):
78         fig,ax1 = plt.subplots()
79         x = list(range(1, MAX_ITER + 1))
80         c1l = ax1.plot(x,c1,'-', label = f"cost of c1.txt(random initilizaiton)")
81         c2l = ax1.plot(x,c2,'-', label = f"cost of c2.txt(furthest initilizaiton)")
82         ax1.set_xlim(0, MAX_ITER + 2)
83         ax1.set_xlabel("Number of Iterations")
84         ax1.set_ylabel("Cost")
85         ax1.legend()
86         ax1.set_title(f"Within-cluster Cost With {distanceType} Distance")
87         fig.show()
88
89     def tSNE(data, centroids, norm, plotTitle):
90         def findCluster(pt, centroids, norm=2):
91             c_index = closest(pt, centroids, norm)
92             return (c_index)
93         def plotTSNE(embeddedData, plotTitle, c=None):
94             x = embeddedData[:, 0]
95             y = embeddedData[:, 1]
96             plt.scatter(x, y, c=c, cmap=plt.cm.get_cmap("jet", 10))
97             plt.title(plotTitle)
98             plt.show()
99             dataClusters = data.map(lambda pt: findCluster(pt, centroids, norm))

```

```

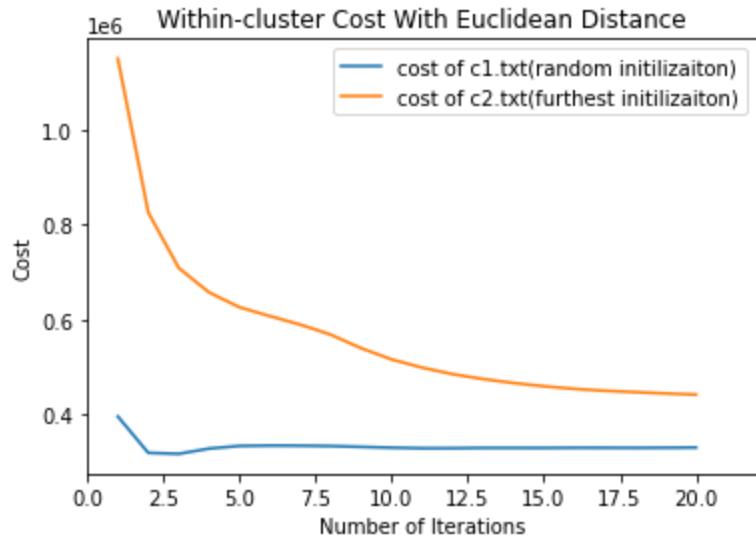
100     cluster = dataClusters.collect()
101     originalData = np.array(data.collect())
102     originalEmbeddedData = TSNE(n_components=2, random_state=100).fit_transform(originalData)
103     plotTSNE(originalEmbeddedData, plotTitle)
104     plotTSNE(originalEmbeddedData, plotTitle, cluster)
105
106 def main():
107     # Spark settings
108     conf = SparkConf()
109     sc = SparkContext(conf=conf)
110
111     # Load the data, cache this since we're accessing this each iteration
112     data = sc.textFile(DATA_PATH).map(
113         lambda line: np.array([float(x) for x in line.split(' ')]))
114         .cache()
115     # Load the initial centroids c1, split into a list of np arrays
116     centroids1 = sc.textFile(C1_PATH).map(
117         lambda line: np.array([float(x) for x in line.split(' ')]))
118         .collect()
119     # Load the initial centroids c2, split into a list of np arrays
120     centroids2 = sc.textFile(C2_PATH).map(
121         lambda line: np.array([float(x) for x in line.split(' ')]))
122         .collect()
123
124     # TODO: Run the kmeans clustering and complete the HW
125     c1, centroids1Out = kmeans(data, centroids1, NORM)
126     c2, centroids2Out = kmeans(data, centroids2, NORM)
127     if NORM == 1:
128         distanceType = "Manhattan"
129     elif NORM == 2:
130         distanceType = "Euclidean"
131     plotCost(c1, c2, distanceType, MAX_ITER)
132     # tSNE(data, centroids1Out, NORM, "t-SNE results of c1.txt using L2 distance and random_state=100")
133
134     # tSNE(data, centroids2Out, NORM, "t-SNE results of c2.txt using L2 distance and random_state=100")
135
136 if __name__ == "__main__":
137     main()

```



(2)

We simply change the NORM in part (1) to 2 and re-run the program. The output graph is as follows.



(3)

```

1 import operator
2 import sys
3 from pyspark import SparkConf, SparkContext
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from scipy import linalg
7 from pyspark.mllib.feature import Normalizer
8 from sklearn.manifold import TSNE
9
10 # Macros.
11 MAX_ITER = 20
12 DATA_PATH = "gs://eecs6893_data/q1/data.txt"
13 C1_PATH = "gs://eecs6893_data/q1/c1.txt"
14 C2_PATH = "gs://eecs6893_data/q1/c2.txt"
15 NORM = 2
16
17 # Helper functions.
18 def closest(p, centroids, norm):
19     """
20         Compute closest centroid for a given point.
21     Args:
22         p (numpy.ndarray): input point
23         centroids (list): A list of centroids points
24         norm (int): 1 or 2
25     Returns:
26         int: The index of closest centroid.
27     """
28     closest_c = min([(i, linalg.norm(p - c, norm))
29                      for i, c in enumerate(centroids)],
30                      key=operator.itemgetter(1))[0]
31     return closest_c
32
33 # K-means clustering
34
35 def getClosestCentroid(pt, centroids, norm=2):
36     c_index = closest(pt, centroids, norm)
37     cost = linalg.norm(pt - centroids[c_index], norm)
38     return (c_index, (pt, 1, cost))
39
40 def calculateCost(costData):
41     totalCost, numPoints = 0, 0
42     for cluster in costData:
43         totalCost += cluster[1][2]
44         numPoints += cluster[1][1]
45     return totalCost
46
47 def kmeans(data, centroids, norm=2):
48     """
49         Conduct k-means clustering given data and centroid.
50         This is the basic version of k-means, you might need more
51         code to record cluster assignment to plot TSNE, and more
52         data structure to record cost.
53     Args:
54         data (RDD): RDD of points
55         centroids (list): A list of centroids points
56         norm (int): 1 or 2
57     Returns:
58         RDD: assignment information of points, a RDD of (centroid, (point, 1))
59         list: a list of centroids
60         and define yourself...
61     """
62     # iterative k-means
63     cost = []
64     for _ in range(MAX_ITER):
65         # Transform each point to a combo of point, closest centroid, count=1
66         # point -> (closest_centroid, (point, 1))
67         clusters = data.map(lambda pt: getClosestCentroid(pt, centroids, norm)).reduceByKey(lambda count1, count2: \

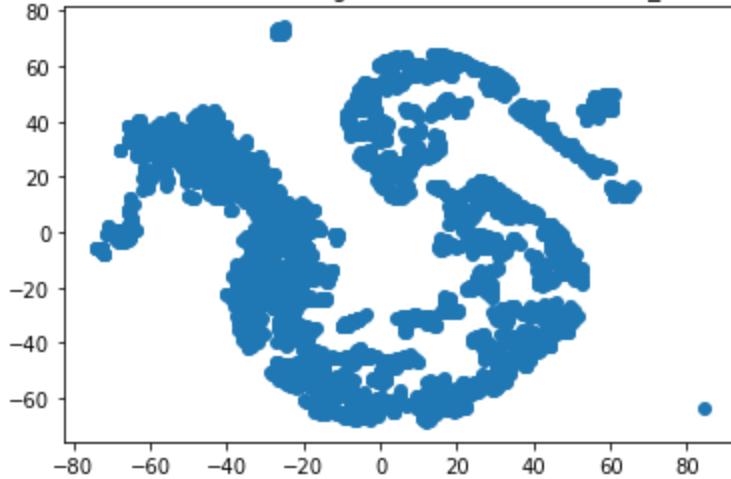
```

```

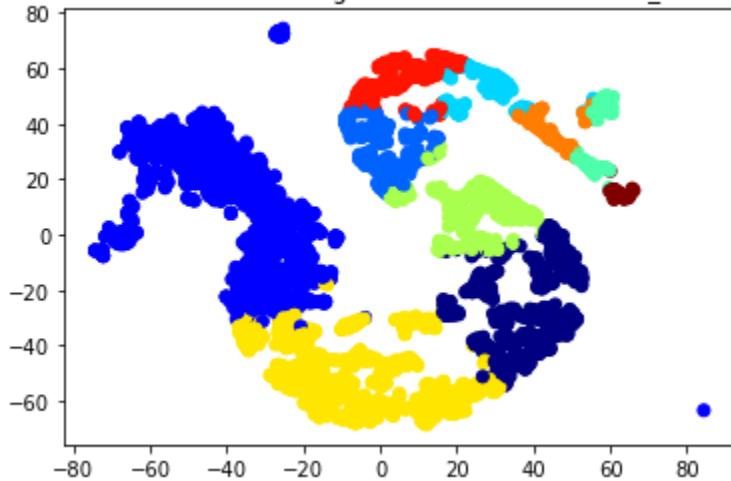
67     (count1[0] + count2[0], count1[1] + count2[1], count1[2] + count2[2]))
68     # Re-compute cluster center
69     # For each cluster center (key), aggregate its values
70     centroids = clusters.map(lambda x: x[1][0]/x[1][1].collect())
71     # by summing up points and count
72     # Average the points for each centroid: divide sum of points by count
73     # Use collect() to turn RDD into list
74     cost.append(calculateCost(clusters.collect()))
75
76     return cost, centroids
77
78 def plotCost(c1, c2, distanceType, MAX_ITER):
79     fig,ax1 = plt.subplots()
80     x = list(range(1, MAX_ITER + 1))
81     c1l = ax1.plot(x,c1, '-', label = f"cost of c1.txt(random initilizaiton)")
82     c2l = ax1.plot(x,c2, '--', label = f"cost of c2.txt(furthest initilizaiton)")
83     ax1.set_xlim(0, MAX_ITER + 2)
84     ax1.set_xlabel("Number of Iterations")
85     ax1.set_ylabel("Cost")
86     ax1.legend()
87     ax1.set_title(f"Within-cluster Cost With {distanceType} Distance")
88     fig.show()
89
90 def tSNE(data, centroids, norm, plotTitle):
91     def findCluster(pt, centroids, norm=2):
92         c_index = closest(pt, centroids, norm)
93         return (c_index)
94     def plotTSNE(embeddedData, plotTitle, c=None):
95         x = embeddedData[:, 0]
96         y = embeddedData[:, 1]
97         plt.scatter(x, y, c=c, cmap=plt.cm.get_cmap("jet", 10))
98         plt.title(plotTitle)
99         plt.show()
100    dataClusters = data.map(lambda pt: findCluster(pt, centroids, norm))
101
102    cluster = dataClusters.collect()
103    originalData = np.array(data.collect())
104    originalEmbeddedData = TSNE(n_components=2, random_state=100).fit_transform(originalData)
105    plottSNE(originalEmbeddedData,plotTitle)
106    plottSNE(originalEmbeddedData, plotTitle, cluster)
107
108 def main():
109     # Spark settings
110     conf = SparkConf()
111     sc = SparkContext(conf=conf)
112
113     # Load the data, cache this since we're accessing this each iteration
114     data = sc.textFile(DATA_PATH).map(
115         lambda line: np.array([float(x) for x in line.split(' ')]))
116     .cache()
117
118     # Load the initial centroids c1, split into a list of np arrays
119     centroids1 = sc.textFile(C1_PATH).map(
120         lambda line: np.array([float(x) for x in line.split(' ')]))
121     .collect()
122
123     # Load the initial centroids c2, split into a list of np arrays
124     centroids2 = sc.textFile(C2_PATH).map(
125         lambda line: np.array([float(x) for x in line.split(' ')]))
126     .collect()
127
128     # TODO: Run the kmeans clustering and complete the HW
129     c1, centroids1Out = kmeans(data, centroids1, NORM)
130     c2, centroids2Out = kmeans(data, centroids2, NORM)
131
132     if NORM == 1:
133         distanceType = "Manhattan"
134     elif NORM == 2:
135         distanceType = "Euclidean"
136
137     # plotCost(c1, c2, distanceType, MAX_ITER)
138     tSNE(data, centroids1Out, NORM, "t-SNE results of c1.txt using L2 distance and random_state=100")
139
140     tSNE(data, centroids2Out, NORM, "t-SNE results of c2.txt using L2 distance and random_state=100")
141
142 if __name__ == "__main__":
143     main()

```

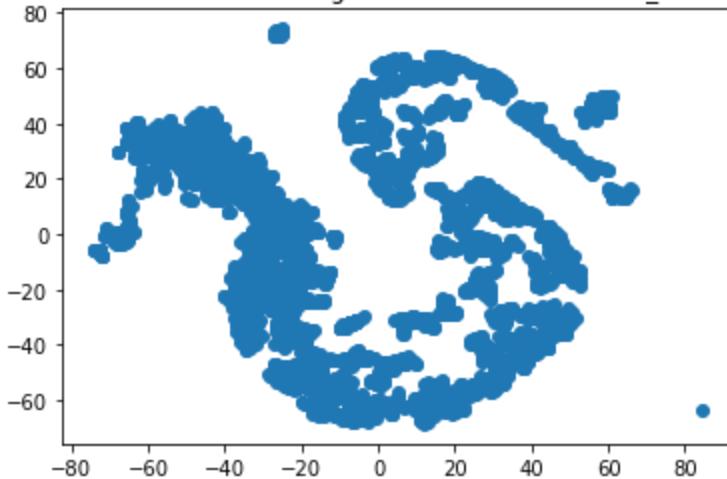
t-SNE results of c1.txt using L2 distance and random_state=100



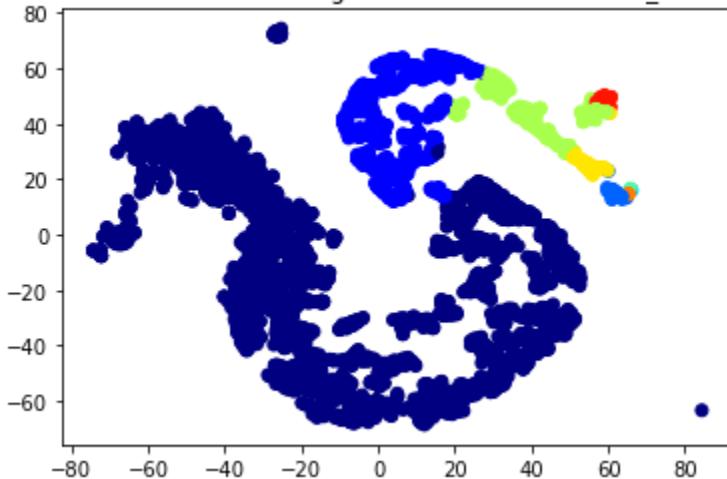
t-SNE results of c1.txt using L2 distance and random_state=100



t-SNE results of c2.txt using L2 distance and random_state=100



t-SNE results of c2.txt using L2 distance and random_state=100



(4)

For L2 and L1 distance, random initialization of K-means using c1.txt is better than initialization using c2.txt in terms of cost. As we can see from my plot before. The initial and final within-cluster cost of c1.txt is better than c2.txt. This make sense because if we initialize cluster centers to be as far away from each other as possible then all points would be far away from these points. As a result it have a higher initial cost. The cost keep decreasing in those 20 interactions. However, it is flatten out and the cost is not decreasing. The final cost for c2.txt is still higer than c1.txt. c2.txt is trying to have 10 independent clusters that are not far from boundary in this data set. However, c1.txt is more like grouping them together in relatively dense area. The cost of c1.txt is not decreasing much because there are simply too much data point around the center of those clusters and is it hard for the clusters to move to a beeter center. All in all, the cost of c1.txt is still lower than c2.txt at all time.

(5)

The time complexity of the iterative K-means is $O(\text{number of points} * \text{number of centroids} * \text{number of iterations})$

2. Monitoring Hadoop metrics

(1)

```
$ ssh localhost  
hadoop@localhost's password:  
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-48-generic x86_64)  
  
 * Documentation: https://help.ubuntu.com  
 * Management: https://landscape.canonical.com  
 * Support: https://ubuntu.com/advantage  
  
0 updates can be applied immediately.  
  
Last login: Mon Oct  3 10:51:53 2022 from 127.0.0.1
```

```
hadoop@yutao-GL502VSK:/usr/local$ hadoop version  
Hadoop 3.3.1  
Source code repository https://github.com/apache/hadoop.git -r a3b9c37a397ad4188041dd80621bdeefc46885f2  
Compiled by ubuntu on 2021-06-15T05:13Z  
Compiled with protoc 3.7.1  
From source with checksum 88a4ddb2299aca054416d6b7f81ca55  
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-3.3.1.jar
```

```
2022-10-03 11:51:52,247 INFO util.GSet: capacity      = 2^17 = 131072 entries  
Re-format filesystem in Storage Directory root= /home/hadoop/hadoopinfra/hdfs/namenode; location= null ?  
(Y or N) y  
2022-10-03 11:51:59,985 INFO namenode.FSImage: Allocated new BlockPoolId: BP-832349331-127.0.1.1-16648123  
19979  
2022-10-03 11:51:59,985 INFO common.Storage: Will remove files: [/home/hadoop/hadoopinfra/hdfs/namenode/c  
urrent/fsimage_00000000000000000000, /home/hadoop/hadoopinfra/hdfs/namenode/current/edits_0000000000000000  
001-000000000000000002, /home/hadoop/hadoopinfra/hdfs/namenode/current/edits_000000000000000003-000000  
000000000003, /home/hadoop/hadoopinfra/hdfs/namenode/current/edits_inprogress_000000000000000006, /home/  
hadoop/hadoopinfra/hdfs/namenode/current/VERSION, /home/hadoop/hadoopinfra/hdfs/namenode/current/seen_tx  
id, /home/hadoop/hadoopinfra/hdfs/namenode/current/fsimage_00000000000000000000.md5, /home/hadoop/hadoopinf  
ra/hdfs/namenode/current/edits_000000000000000004-000000000000000005]  
2022-10-03 11:51:59,996 INFO common.Storage: Storage directory /home/hadoop/hadoopinfra/hdfs/namenode has  
been successfully formatted.  
2022-10-03 11:52:00,018 INFO namenode.FSImageFormatProtobuf: Saving image file /home/hadoop/hadoopinfra/h  
dfs/namenode/current/fsimage.ckpt_000000000000000000 using no compression  
2022-10-03 11:52:00,095 INFO namenode.FSImageFormatProtobuf: Image file /home/hadoop/hadoopinfra/hdfs/nam  
enode/current/fsimage.ckpt_000000000000000000 of size 401 bytes saved in 0 seconds .  
2022-10-03 11:52:00,102 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0  
2022-10-03 11:52:00,126 INFO namenode.FSNamesystem: Stopping services started for active state  
2022-10-03 11:52:00,126 INFO namenode.FSNamesystem: Stopping services started for standby state  
2022-10-03 11:52:00,129 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.  
2022-10-03 11:52:00,129 INFO namenode.NameNode: SHUTDOWN_MSG:  
*****  
SHUTDOWN_MSG: Shutting down NameNode at yutao-GL502VSK/127.0.1.1  
*****
```

```
hadoop@yutao-GL502VSK:~$ start-dfs.sh  
Starting namenodes on [localhost]  
Starting datanodes  
Starting secondary namenodes [yutao-GL502VSK]
```

```
hadoop@yutao-GL502VSK:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
```

(2) HDFS metrics monitoring:

I chose to use Hadoop on GCP in the future because my windows keep messing up with my ubuntu somehow. Therefore, all of the following screenshots are from GCP.

Started:	Thu Oct 06 14:34:15 -0400 2022
Version:	3.2.3, rb85070eb738c0d1fbf983f438199bfab3558d7b0
Compiled:	Wed Jun 29 04:08:00 -0400 2022 by bigtop from (no branch)
Cluster ID:	CID-2edfa704-5abe-4175-bd29-785d2ae255c7
Block Pool ID:	BP-362613439-10.142.0.44-1665064025394

Overview 'hadoop-m:8020' (active)

Started:	Thu Oct 06 14:34:15 -0400 2022
Version:	3.2.3, rb85070eb738c0d1fbf983f438199bfab3558d7b0
Compiled:	Wed Jun 29 04:08:00 -0400 2022 by bigtop from (no branch)
Cluster ID:	CID-2edfa704-5abe-4175-bd29-785d2ae255c7
Block Pool ID:	BP-362613439-10.142.0.44-1665064025394

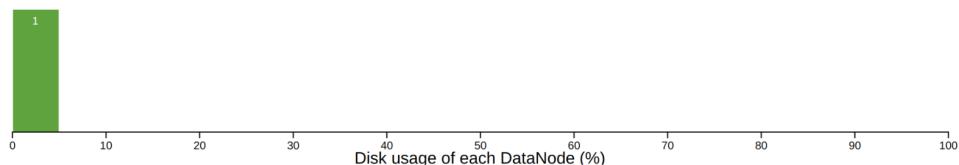
Summary

Security is off.
Safemode is off.
42 files and directories, 7 blocks (7 replicated blocks, 0 erasure coded block groups) = 49 total filesystem object(s).

Datanode Information

✓ In service
 ✗ Down
 ✗ Decommissioning
 ✗ Decommissioned
 ✗ Decommissioned & dead
↗ Entering Maintenance
 ⚡ In Maintenance
 ⚡ In Maintenance & dead

Datanode usage histogram



<https://dstt7midd5fvdj5jcjb55mpydu-dot-us-east1.dataproc.googleusercontent.com/hdfs/dfshealth.html#tab-datanode>

```
{
  "beans": [
    {
      "name": "Hadoop:service=NameNode,name=JvmMetrics",
      "modelerType": "JvmMetrics",
      "tag.Context": "jvm",
      "tag.ProcessName": "NameNode",
      "tag.SessionId": null,
      "tag.Hostname": "hadoop-m",
      "MemNonHeapUsedM": 59.27276,
      "MemNonHeapCommittedM": 60.492188,
      "MemNonHeapMaxM": -1.0,
      "MemHeapUsedM": 118.13485,
      "MemHeapCommittedM": 228.1875,
      "MemHeapMaxM": 2966.75,
      "MemMaxM": 2966.75,
      "GcCountParNew": 9,
      "GcTimeMillisParNew": 603,
      "GcCountConcurrentMarkSweep": 2,
      "GcTimeMillisConcurrentMarkSweep": 203,
      "GcCount": 11,
      "GcTimeMillis": 806,
      "GcNumWarnThresholdExceeded": 0,
      "GcNumInfoThresholdExceeded": 0,
      "GcTotalExtraSleepTime": 134,
      "ThreadsNew": 0,
      "ThreadsRunnable": 14,
      "ThreadsBlocked": 0,
      "ThreadsWaiting": 6,
      "ThreadsTimedWaiting": 55,
      "ThreadsTerminated": 0,
      "LogFatal": 0,
      "LogError": 0,
      "LogWarn": 2,
      "LogInfo": 160
    },
    {
      "name": "Hadoop:service=NameNode,name=RpcActivityForPort8051",
      "modelerType": "RpcActivityForPort8051",
      "tao.port": "8051"
    }
  ]
}
```

Five most important metrics:

1. StartupProgress

This metric is used to show startup statistics for NameNode. Some people might say this is not important. But, I believe this is a metric that I will first check every time after I start the instance. This metric is important because we have to make sure the instance had started completely and properly then we could do calculations. Without checking this debugging might be painful. I will always make sure the PercentComplete is 1.0.

2. JvmMetrics

This is a java virtual machine metric that is used to monitor system resources. This is important because it could let us know how much resource is our job using and we could infer what is the job status from this usage information(e.g. If it is not using resources it probably did not start running correctly).

3. RetryCache.NameNodeRetryCache

This is a metric that is used to monitor the fail-over of NameNode. This is important because it can let us know how many times NameNode had NameNode hit RetryCache to check for failure.

4. FSNamesystem

This is a metric that is used to check some of the system's status for NameNode. This is important because it could give us some statistics on the usage of the system like disk usage which can be handy if we want to know whether we need to expand our hard disk.

5. rpc

These metrics record rpc statistics like the number of bytes sent or received. This is important because we could see the total number of RPC calls and authentication or authorization failures. Therefore, we could see the healthiness of RPC.

(3) MapReduce counters monitoring:

The screenshot shows a web browser window with multiple tabs open. The active tab is titled "MapReduce Job job_1665064043571_0002". The page displays the "Job Overview" for a job named "QuasiMonteCarlo" submitted by user "yz4359". The job state is "SUCCEEDED". The "Diagnostics" section provides average times for map, shuffle, merge, and reduce tasks. Below this, the "ApplicationMaster" table lists one attempt with details like start time and node. The "Task Type" table shows 4 Map and 1 Reduce tasks, all completed successfully. The "Attempt Type" table shows 0 failed, 0 killed, and 4 successful attempts.

Attempt Number	Start Time	Node	Logs
1	Thu Oct 06 14:09:45 UTC 2022	hadoop-m.c.silken-water-362100.internal:8042	/gateway/default/jobhistory/logs

Task Type	Total	Complete
Map	4	4
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	4

The screenshot shows a web browser window with multiple tabs open. The active tab is titled "Counters for job_1665064043571_0002". The left sidebar has sections for Application, Job (Overview, Counters, Configuration, Map.tasks, Reduce.tasks), and Tools. The main content area displays two tables of counters.

Counter Group	Name	Counters		
		Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	94	94
	FILE: Number of bytes written	987,540	246,896	1,234,436
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	1,040	0	1,040
	HDFS: Number of bytes read erasure-coded	0	0	0
	HDFS: Number of bytes written	0	215	215
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	16	5	21
Job Counters	HDFS: Number of write operations	0	4	4
	Data-local map tasks	0	0	4
	Launched map tasks	0	0	4
	Launched reduce tasks	0	0	1
	Total megabyte-milliseconds taken by all map tasks	0	0	70,924,788
	Total megabyte-milliseconds taken by all reduce tasks	0	0	9,796,224
	Total time spent by all map tasks (ms)	0	0	22,473
Total time spent by all maps in occupied slots (ms)	0	0	70,924,788	
Total time spent by all reduce tasks (ms)	0	0	3,104	

I could monitor the execution of map-reduced tasks by looking at job counters. The “Launched map tasks” will increment when it starts to map and the “Launched reduce tasks” will increment when it starts to reduce. Also, the respective totals spend for maps or reduce tasks will increment. Also, we could check the “Map-reduce framework” for CPU time spent for map and reduce respectively.

(4) YARN metrics monitoring:

```
{
  "beans": [
    {
      "name": "Hadoop:service=ResourceManager,name=RMNMInfo",
      "modelerType": "org.apache.hadoop.yarn.server.resourcemanager.RMNMInfo",
      "LiveNodeManagers": "[{"HostName": "\nhadoop-m.c.silken-water-362100.internal\", \"Rack\": \"/default-rack\", \"State\": \"RUNNING\", \"NodeId\": \"hadoop-m.c.silken-water-362100.internal:8026\", \"NodeHTTPAddress\": \"hadoop-m.c.silken-water-362100.internal:8042\", \"LastHealthUpdate\": 1665082226676, \"HealthReport\": \"\", \"NodeManagerVersion\": \"3.2.3\", \"NumContainers\": 0, \"UsedMemoryMB\": 0, \"AvailableMemoryMB\": 12624}]"
    },
    {
      "name": "Hadoop:service=ResourceManager,name=RpcActivityForPort8033",
      "modelerType": "RpcActivityForPort8033",
      "tag.port": "8033",
      "tag.serverName": "ResourceManagerAdministrationProtocolService",
      "tag.Context": "rpc",
      "tag.NumOpenConnectionsPerUser": "{}",
      "tag.Hostname": "hadoop-m",
      "ReceivedBytes": 0,
      "SentBytes": 0,
      "RpcQueueTimeNumOps": 0,
      "RpcQueueTimeAvgTime": 0.0,
      "RpcClockWaitTimeNumOps": 0,
      "RpcClockWaitTimeAvgTime": 0.0,
      "RpcProcessingTimeNumOps": 0,
      "RpcProcessingTimeAvgTime": 0.0,
      "DeferredRpcProcessingTimeNumOps": 0,
      "DeferredRpcProcessingTimeAvgTime": 0.0,
      "RpcAuthenticationFailures": 0,
      "RpcAuthenticationSuccesses": 0,
      "RpcAuthorizationFailures": 0,
      "RpcAuthorizationSuccesses": 0,
      "RpcClientBackoff": 0,
      "RpcSlowCalls": 0,
      "NumOpenConnections": 0,
      "CallQueueLength": 0,
      "NumDroppedConnections": 0
    },
    {
      "name": "Hadoop:service=ResourceManager,name=RpcActivityForPort8031",
      "modelerType": "RpcActivityForPort8031"
    }
  ]
}
```

Five most important metrics:

1. QueueMetrics

This metric allows us to see the application queue from the resources manager's perspective. This is important because it could show us how many applications have been submitted and are running. Also, it would show current resource usage. In other words, this will allow us to know where we are in the job queue in detail (how many jobs are running, pending, failed, killed, finished, etc).

2. JvmMetrics

This is a java virtual machine metric that is used to monitor system resources. This is important because it could let us know how much resource is our job using and we could infer what is the job status from this usage information(e.g. If it is not using resources it probably did not start running correctly).

3. ClusterMetrics

This metric is used to show the YARN cluster's metric. This is important because it could give us information about NodeManagers' status. We could know the system status in terms of nodes (e.g. number of NodeManager that are active, unhealthy, lost, etc).

4. NodeManagerMetrics

This metric is used to show general statistics for containers in general in the node. This is important because it could give us information about how many containers are there in that node, and how are they doing (e.g. whether they are running, etc).

5. ContainerMetrics

This metric is used to show resource statistics for a specific container in the node. This is important because it could give us information about one specific container's status. In other words, we zoomed in from NodeManager to a container.