EECS6893 Big Data Analytics HW1

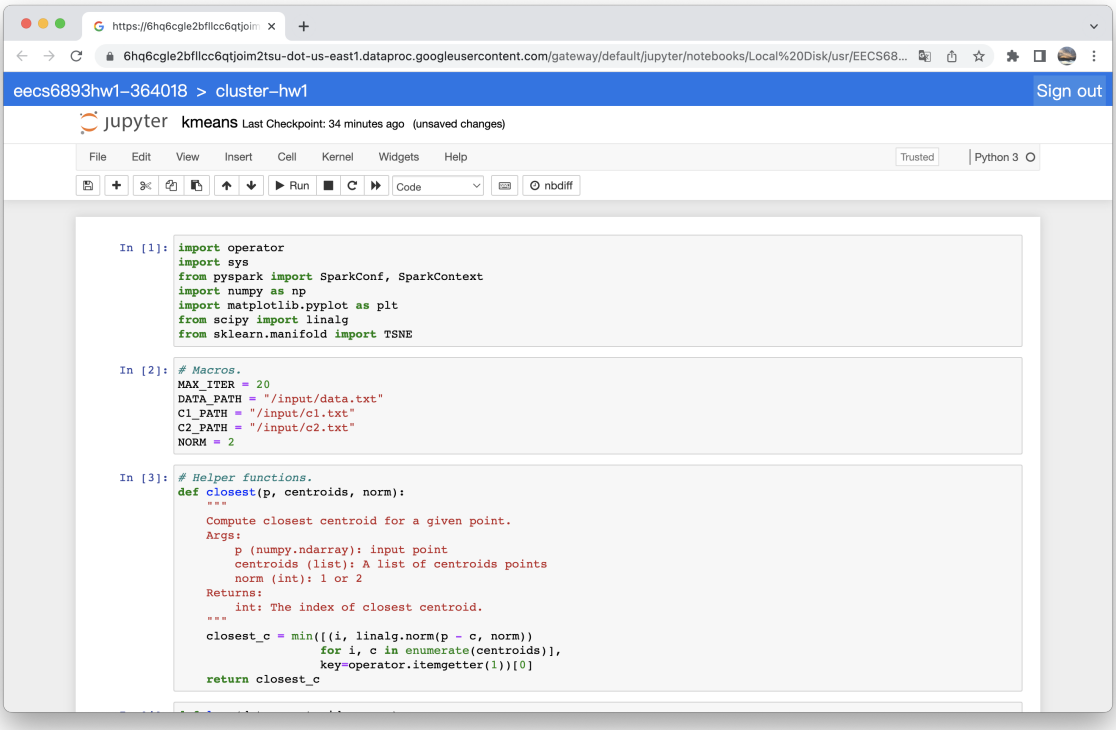
Lijie Huang

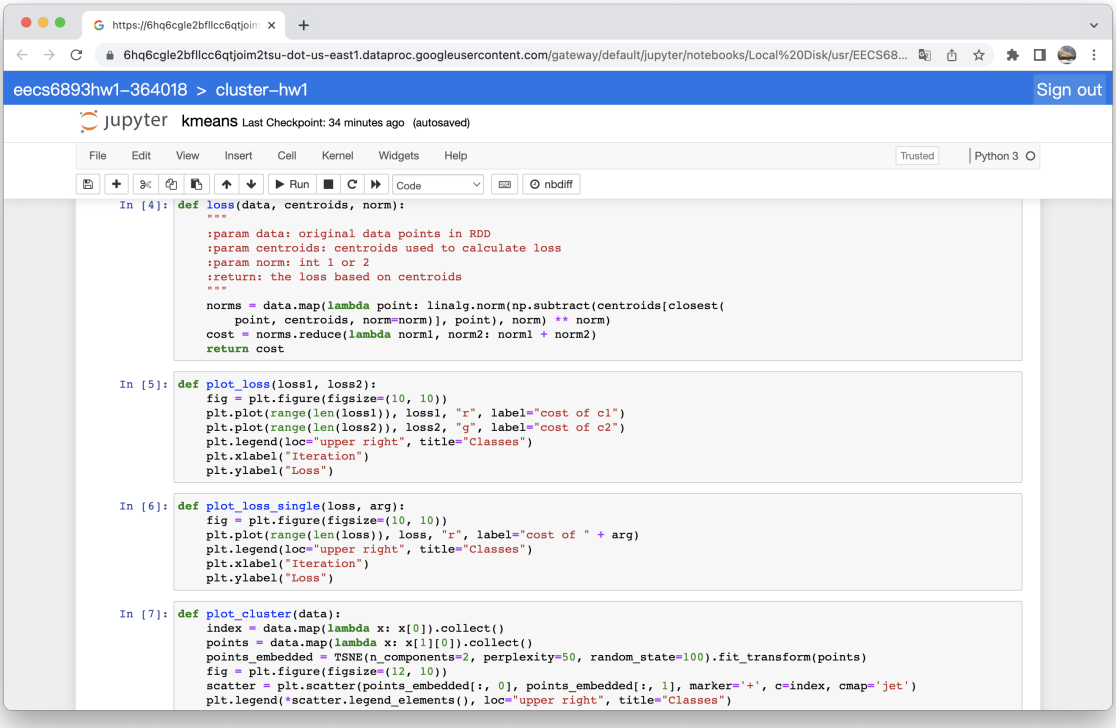
UNI: lh3158

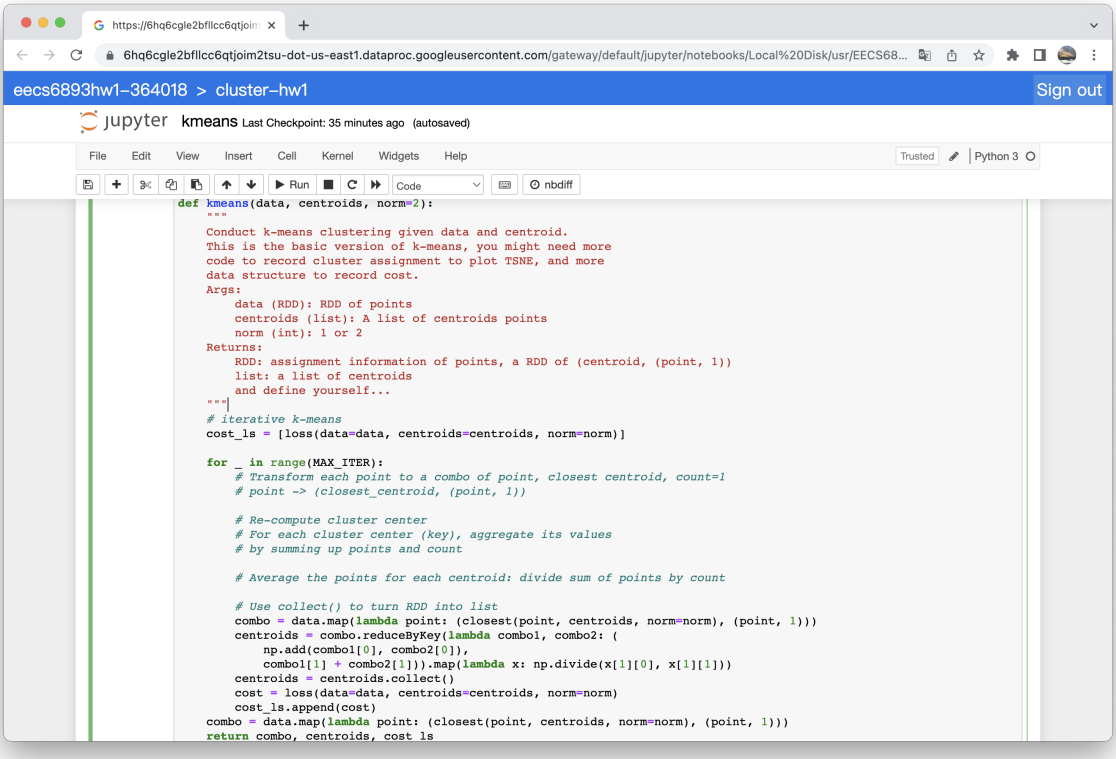
Oct 7th, 2022

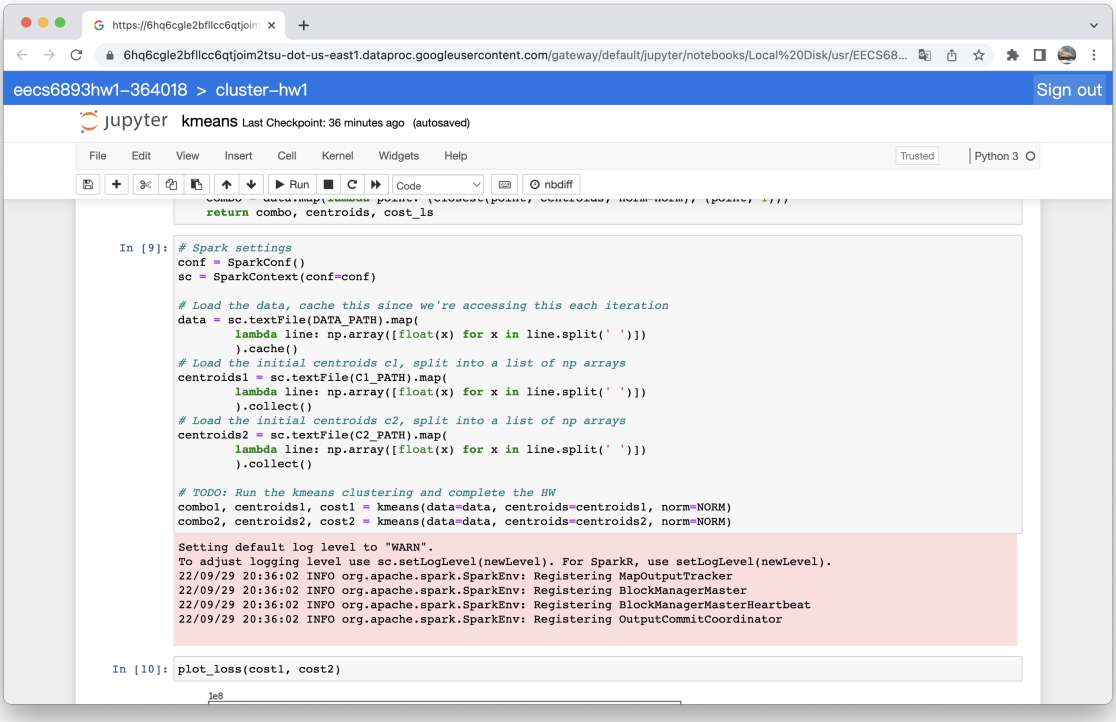
1. Question One

Implement iterative K-means in Spark. We've provided you with starter code kmeans.py on Canvas, which takes care of data loading. Complete the logic inside the for loop, and run the code with different initialization strategies and loss functions. Feel free to change the code if needed, or paste the code into a Jupyter notebook. Take a screenshot of your code and results in your report.
My codes in jupyter notebook are in screenshots below.



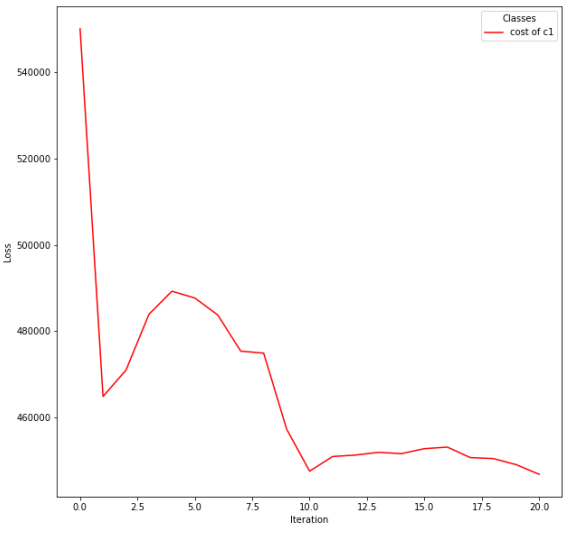

https://6hq6cgle2bfllcc6qtjoim
6hq6cgle2bfllcc6qtjoim2tsu-dot-us-east1.dataproc.googleusercontent.com/gateway/default/jupyter/notebooks/Local%20Disk/usr/EECS68...
eecs6893hw1-364018 > cluster-hw1
Sign out
jupyter kmeans Last Checkpoint: 34 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
nbdiff
In [4]:
def loss(data, centroids, norm):
"""
:param data: original data points in RDD
:param centroids: centroids used to calculate loss
:param norm: int 1 or 2
:return: the loss based on centroids
"""
norms = data.map(lambda point: linalg.norm(np.subtract(centroids[closest(
point, centroids, norm=norm)], point), norm) ** norm)
cost = norms.reduce(lambda norm1, norm2: norm1 + norm2)
return cost
In [5]:
def plot_loss(loss1, loss2):
fig = plt.figure(figsize=(10, 10))
plt.plot(range(len(loss1)), loss1, "r", label="cost of c1")
plt.plot(range(len(loss2)), loss2, "g", label="cost of c2")
plt.legend(loc="upper right", title="Classes")
plt.xlabel("Iteration")
plt.ylabel("Loss")
In [6]:
def plot_loss_single(loss, arg):
fig = plt.figure(figsize=(10, 10))
plt.plot(range(len(loss)), loss, "r", label="cost of " + arg)
plt.legend(loc="upper right", title="Classes")
plt.xlabel("Iteration")
plt.ylabel("Loss")
In [7]:
def plot_cluster(data):
index = data.map(lambda x: x[0]).collect()
points = data.map(lambda x: x[1][0]).collect()
points_embedded = TSNE(n_components=2, perplexity=50, random_state=100).fit_transform(points)
fig = plt.figure(figsize=(12, 10))
scatter = plt.scatter(points_embedded[:, 0], points_embedded[:, 1], marker='+', c=index, cmap='jet')
plt.legend(*scatter.legend_elements(), loc="upper right", title="Classes")

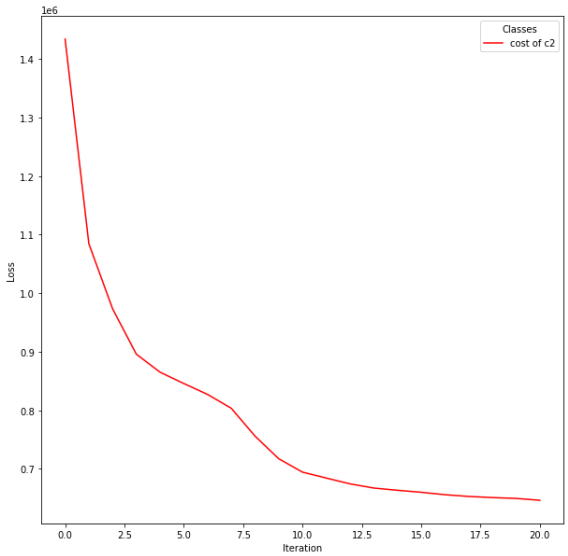

https://6hq6cgle2bfllcc6qtjoim
6hq6cgle2bfllcc6qtjoim2tsu-dot-us-east1.dataproc.googleusercontent.com/gateway/default/jupyter/notebooks/Local%20Disk/usr/EECS68...
eecs6893hw1-364018 > cluster-hw1
Sign out
jupyter kmeans Last Checkpoint: 35 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
nbdiff
def kmeans(data, centroids, norm=2):
"""
Conduct k-means clustering given data and centroid.
This is the basic version of k-means, you might need more
code to record cluster assignment to plot TSNE, and more
data structure to record cost.
Args:
data (RDD): RDD of points
centroids (list): A list of centroids points
norm (int): 1 or 2
Returns:
RDD: assignment information of points, a RDD of (centroid, (point, 1))
list: a list of centroids
and define yourself...
"""
# iterative k-means
cost_ls = [loss(data=data, centroids=centroids, norm=norm)]
for _ in range(MAX_ITER):
# Transform each point to a combo of point, closest centroid, count=1
# point -> (closest_centroid, (point, 1))
# Re-compute cluster center
# For each cluster center (key), aggregate its values
# by summing up points and count
# Average the points for each centroid: divide sum of points by count
# Use collect() to turn RDD into list
combo = data.map(lambda point: (closest(point, centroids, norm=norm), (point, 1)))
centroids = combo.reduceByKey(lambda combo1, combo2: (
np.add(combo1[0], combo2[0]),
combo1[1] + combo2[1])).map(lambda x: np.divide(x[1][0], x[1][1]))
centroids = centroids.collect()
cost = loss(data=data, centroids=centroids, norm=norm)
cost_ls.append(cost)
combo = data.map(lambda point: (closest(point, centroids, norm=norm), (point, 1)))
return combo, centroids, cost_ls

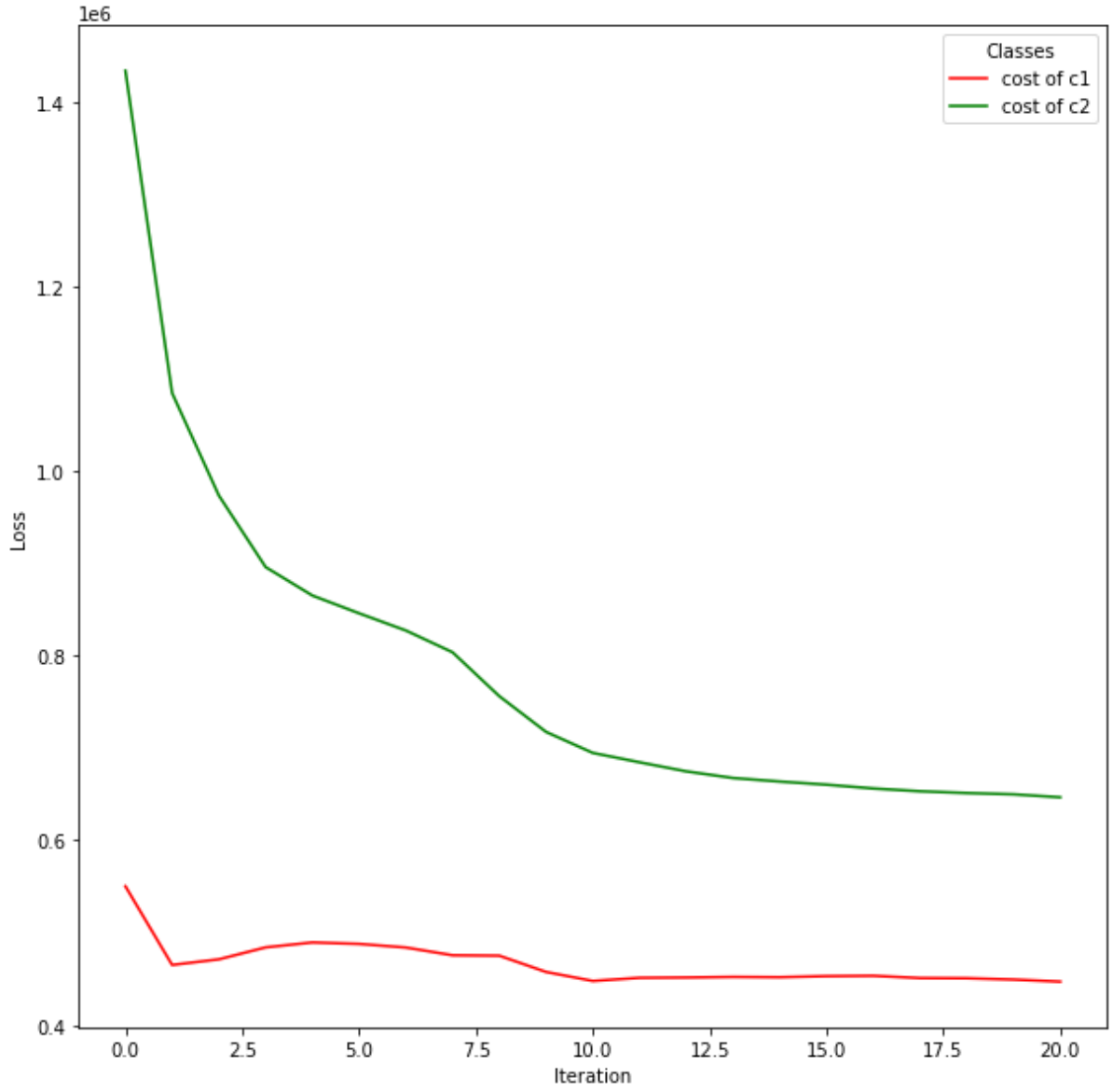


(1) L1 Within-Cluster Cost

Run clustering on data.txt with c1.txt and c2.txt as initial centroids and use L1 distance as similarity measurement. Compute and plot the within-cluster cost for each iteration. You'll need to submit two graphs here. (20%)

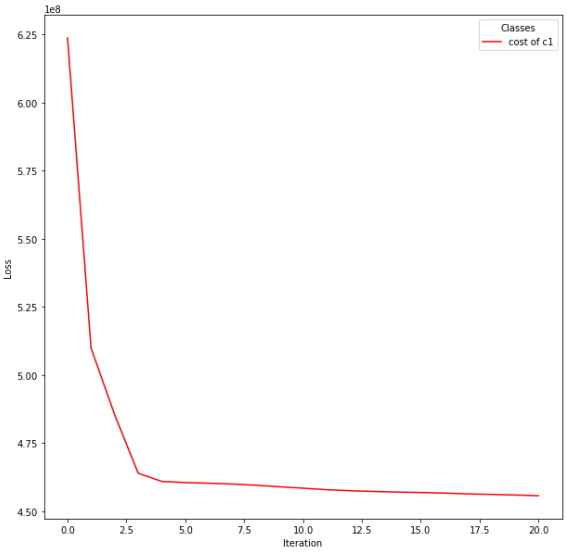


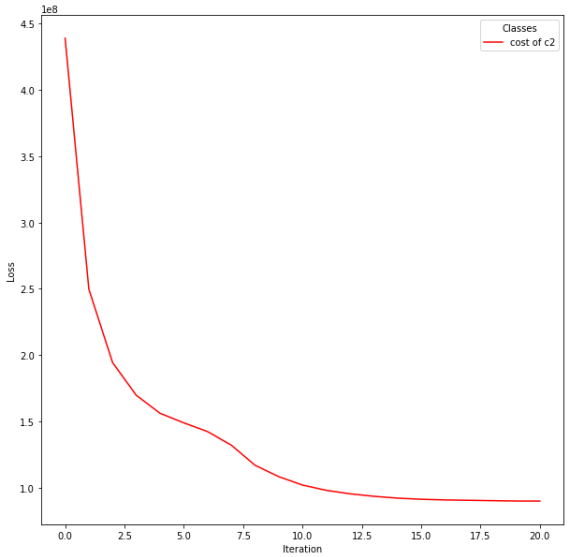


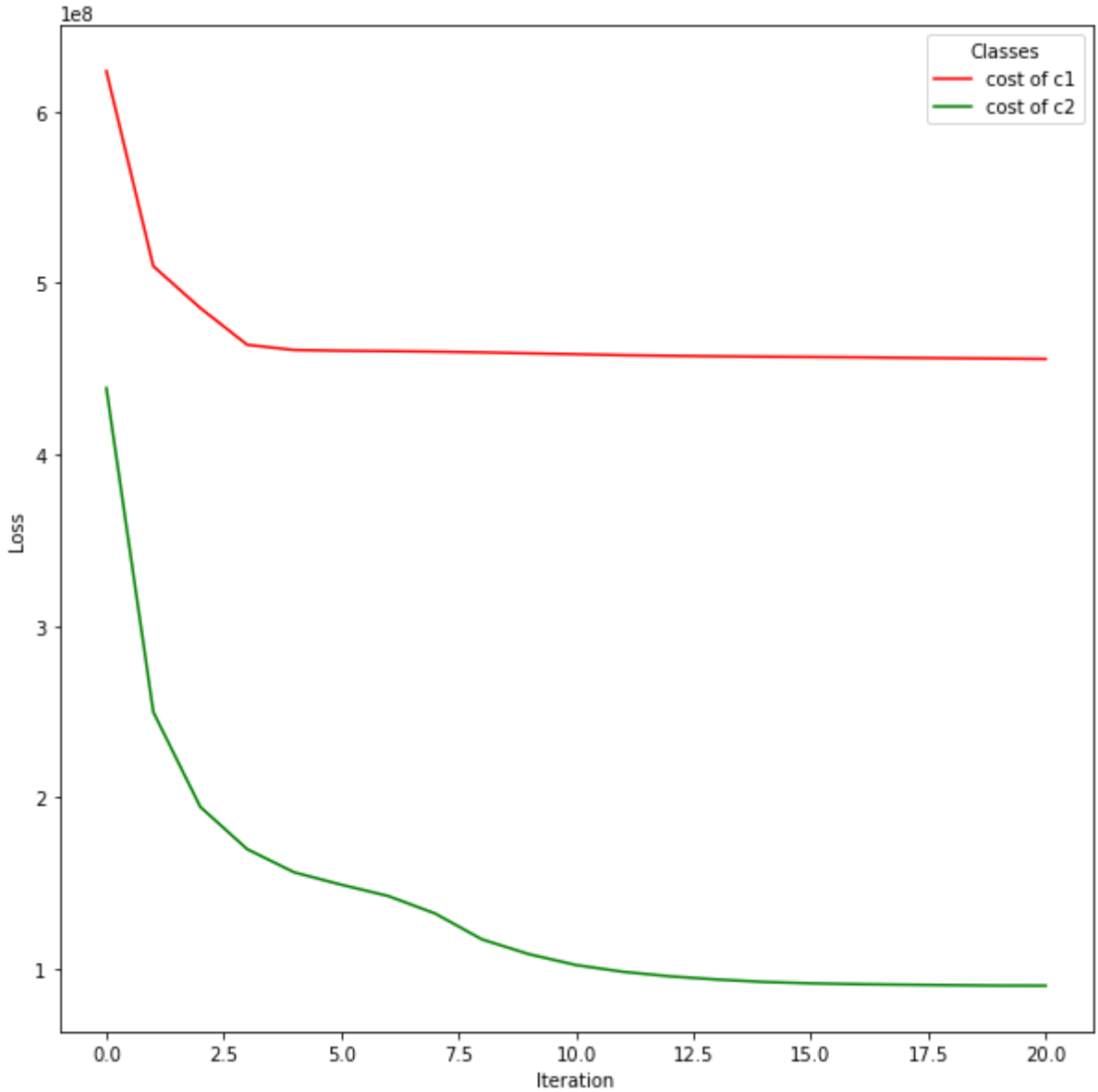


(2) L2 Within-Cluster Cost

Run clustering on data.txt with c1.txt and c2.txt as initial centroids and use L2 distance as similarity measurement. Compute and plot the within-cluster cost for each iteration. You'll need to submit two graphs here. (20%)


1e8
Classes
cost of c1
Loss
Iteration

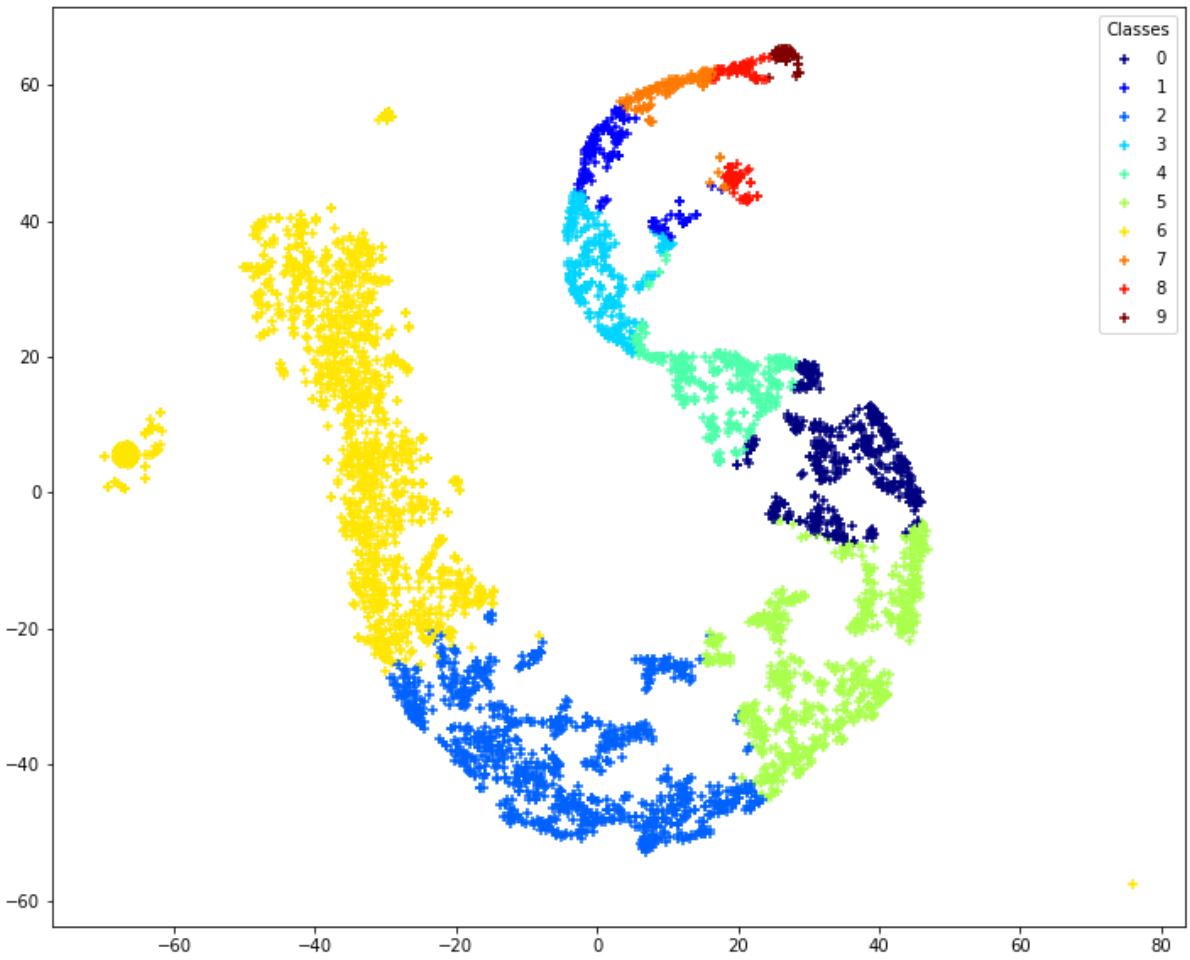

1e8
Classes
cost of c2
Loss
Iteration


1e8
Classes
cost of c1
cost of c2
Loss
Iteration

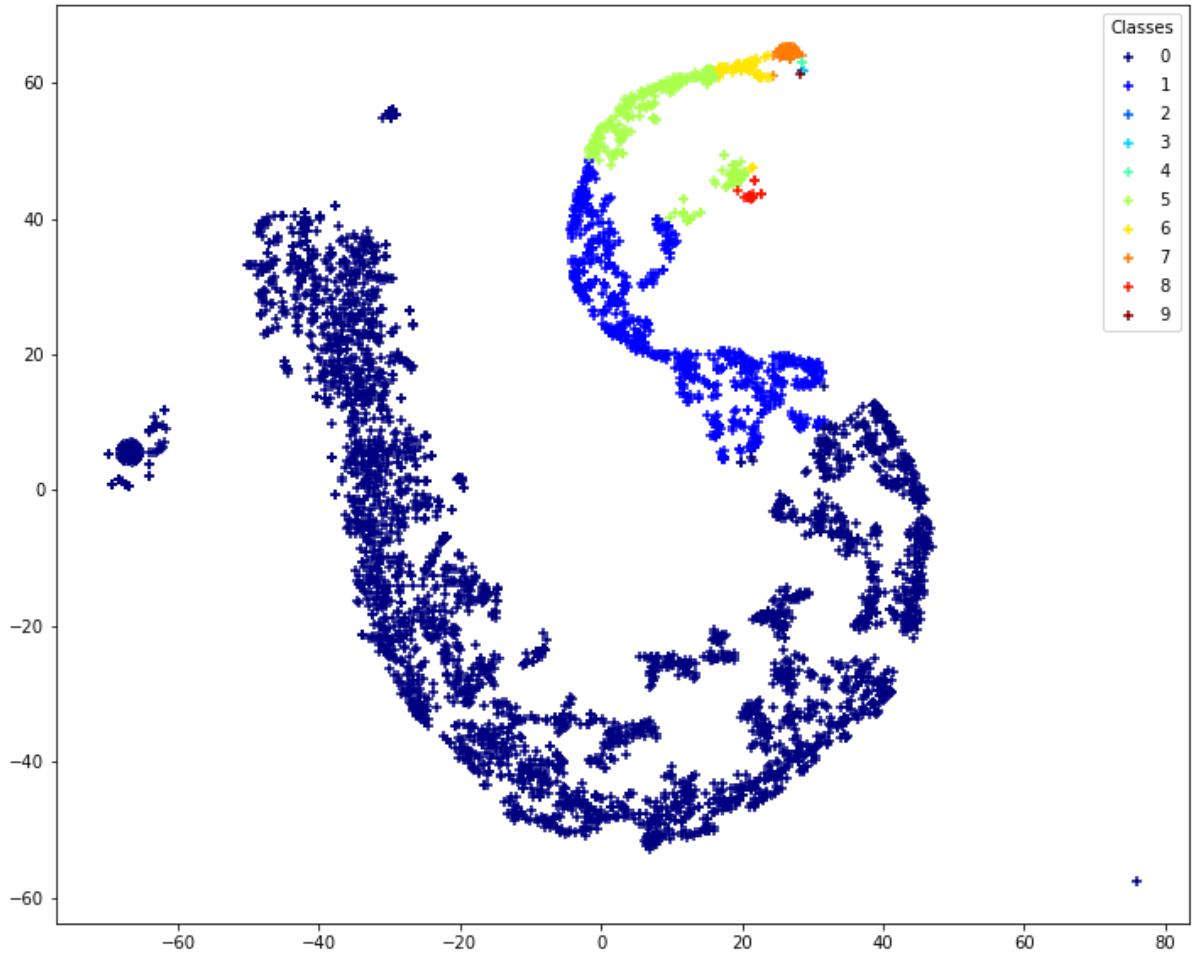
(3) t-SNE Visualization

t-SNE Visualization is a dimensionality reduction method particularly suitable for visualization of high-dimensional data. Visualize your clustering assignment result of (2) by reducing the dimension to a 2D space. You'll need to submit two graphs here. (10%)

c1:



c2:



(4)

For L2 and L1 distance, are random initialization of K-means using c1.txt better than initialization using c2.txt in terms of cost? Explain your reasoning. (5%)

It is clear to tell from the cost figure in (2) that initialization using c2.txt can reduce the cost rapidly and remains at a lower level for L2 distance. Although the order of lines in figures in (1) is the opposite, it is due to the fact that using L1 performs worse than using L2 as the distance function. The reason why initialization using c2.txt is better is that making origin centroids as far away to each other as possible makes the iteration to update centroids much more easier and quickly. While using random selected centroids may end in letting outliers weight too much to update centroids thus making the converge of cost function slow and leading to inappropriate centroids.

(5)

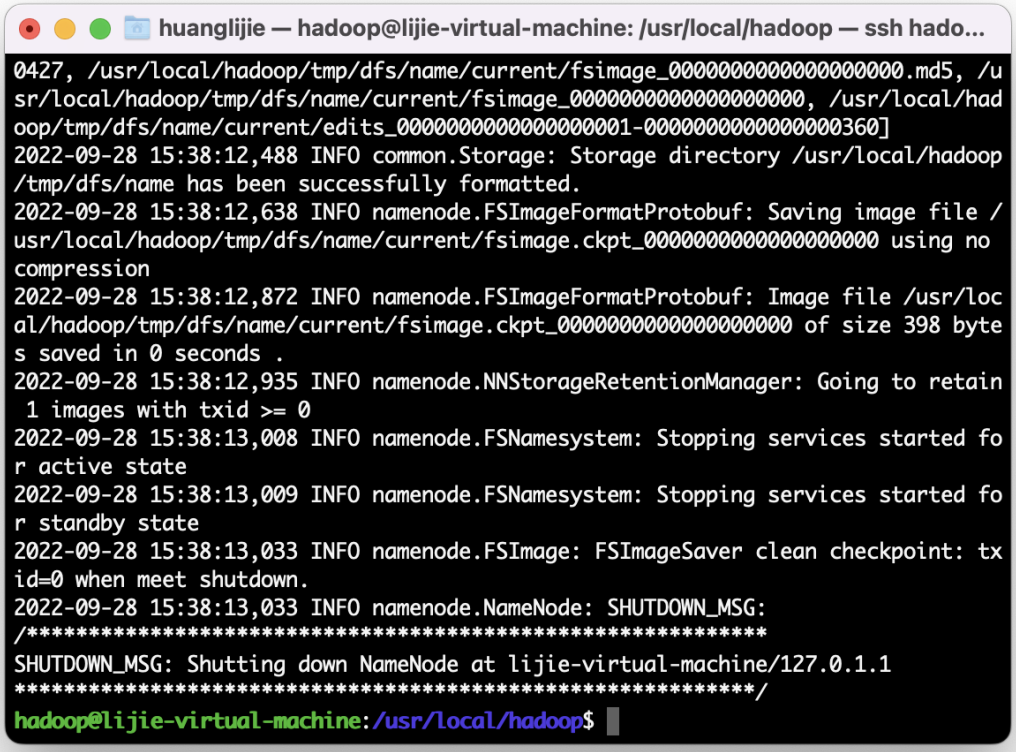
What is the time complexity of the iterative K-means? (5%)

The time complexity is $O(i * k * n * d)$, where i is the number of iterations, k is the number of centroids, n is the total number of all points, d is the dimension.

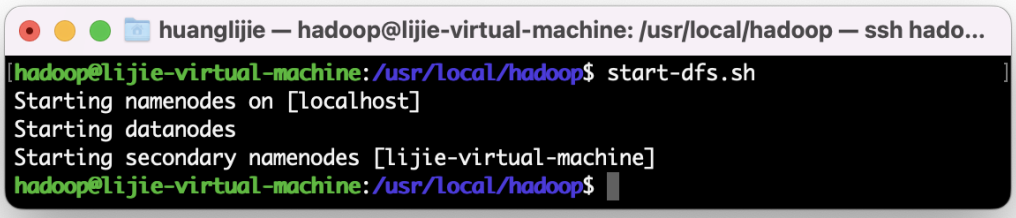
2. Question2

(1) Download and setup Hadoop

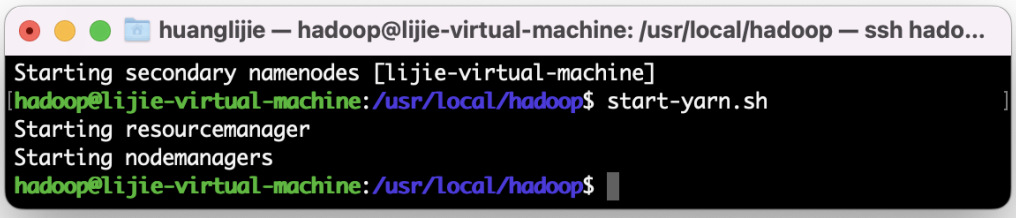
Set up the namenode



verify hadoop dfs

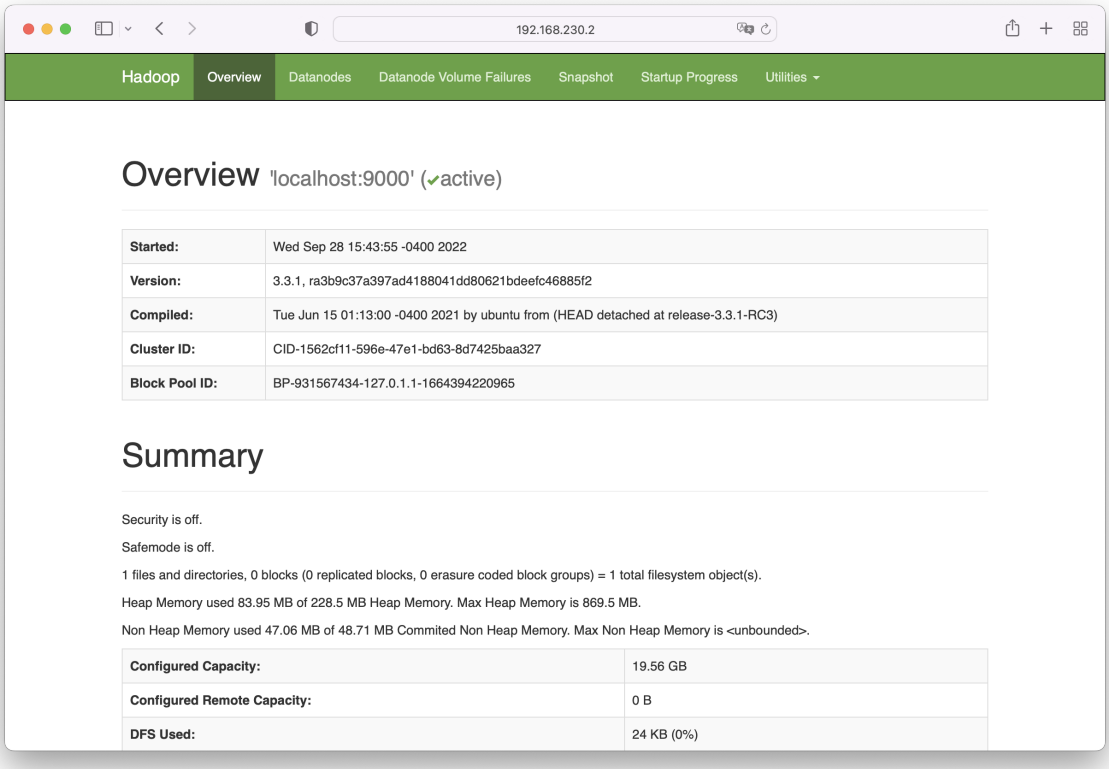


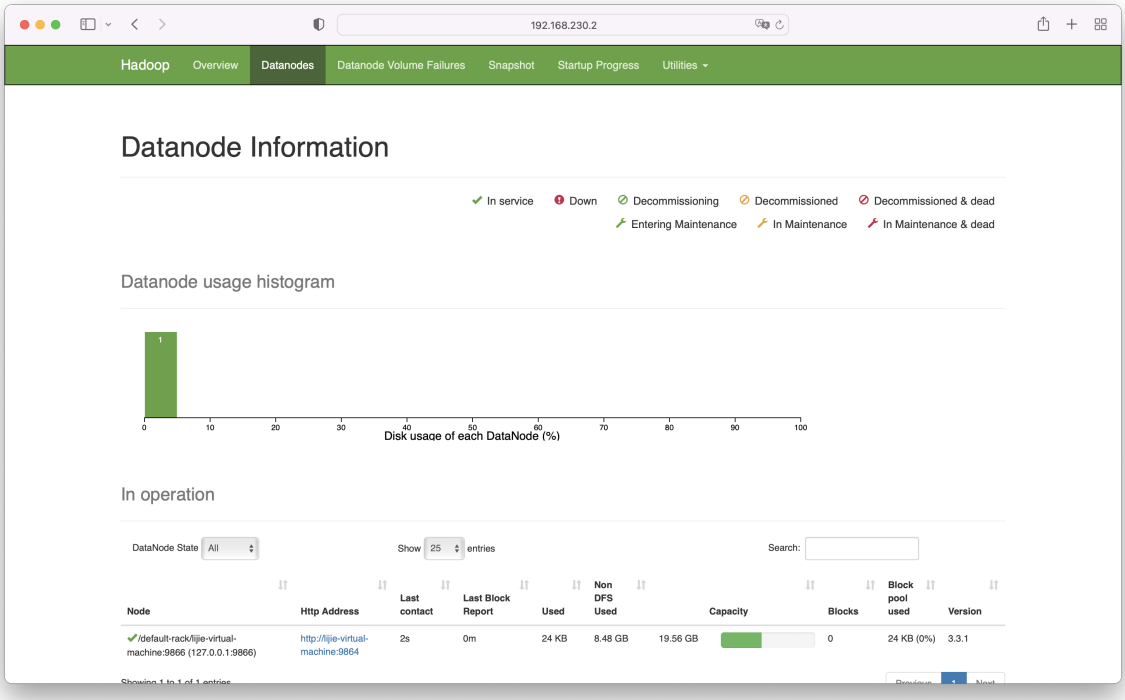
verify yarn scrpit



(2) HDFS metrics monitoring

- Monitor HDFS metrics through HTTP API. Provide a screenshot.





- Select five metrics that you think are most important. Explain their usages and why they are important.

1) CapacityRemaining. it represents the total available capacity remaining across the entire HDFS cluster. Running out of disk space in HDFS will cause bad things to happen. Any running jobs which write out of memory data may fail due to lack of capacity.

2) MissingBlocks. A missing block is far worse than a corrupt block, because a missing block cannot be recovered by copying a replica. A missing block represents a block for which no known copy exists in the cluster. That's not to say that the block no longer exists—if a series of DataNodes were taken offline for maintenance, missing blocks could be reported until they are brought back up.

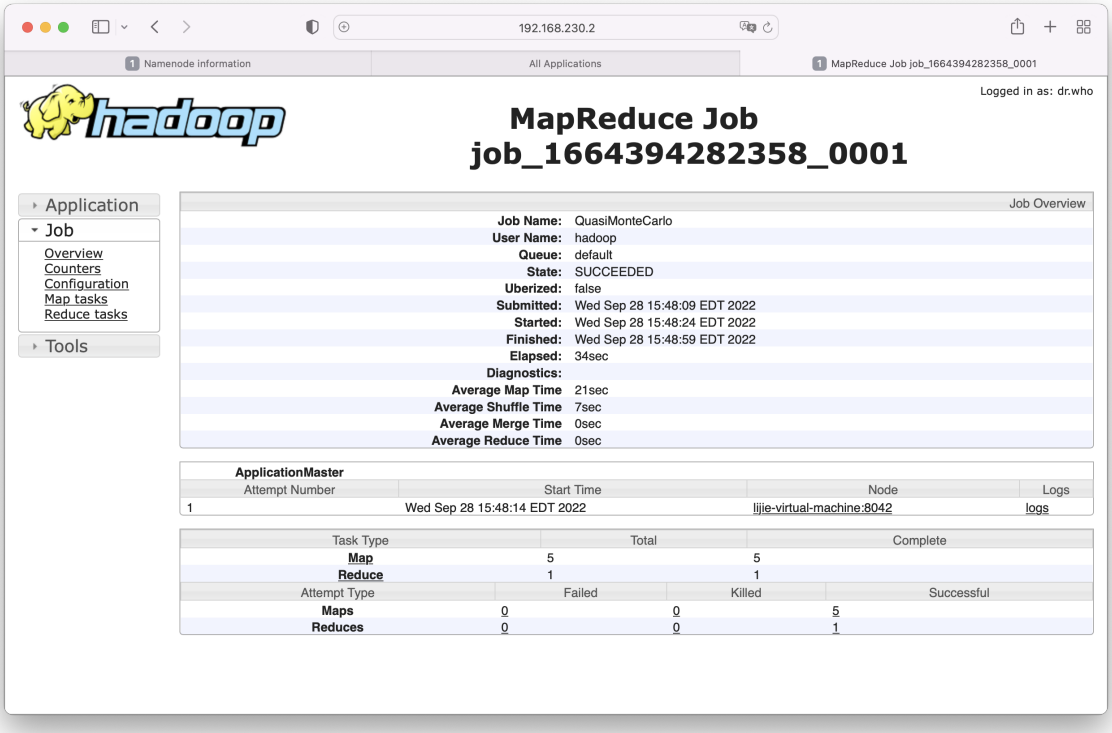
3) VolumeFailureTotal. Though a failed volume will not bring your cluster to grinding halt, you most likely want to know when hardware failures occur, if only so that you can replace the failed hardware.

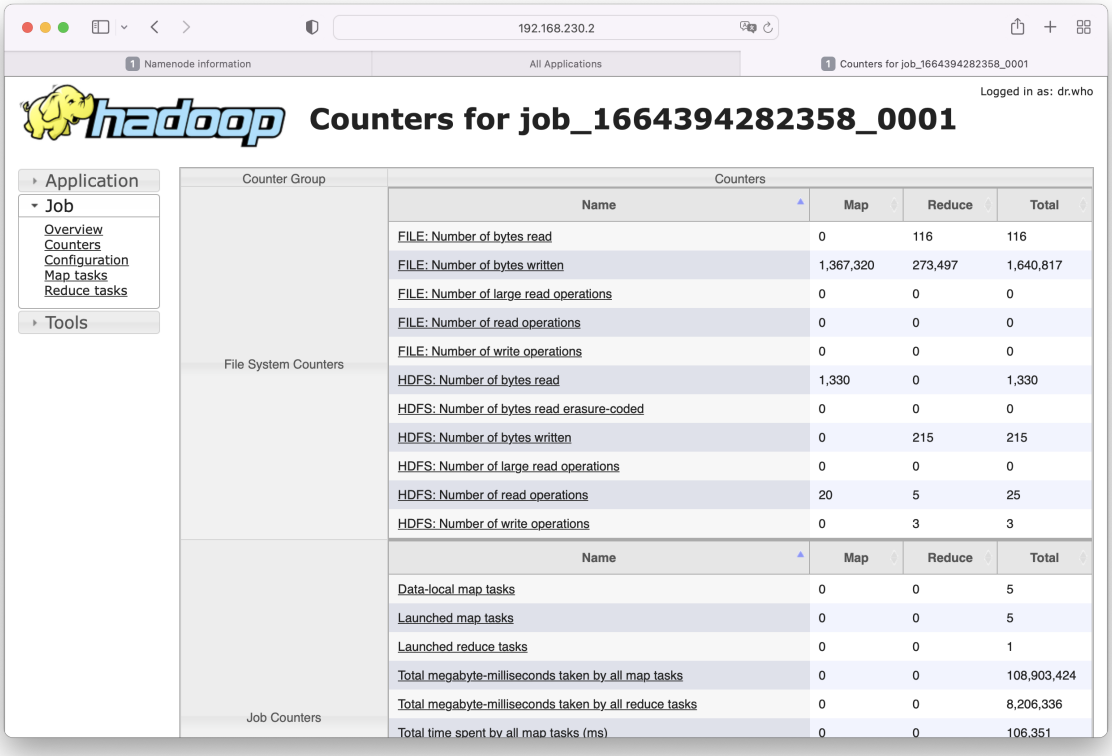
4) NumDeadDataNodes. The death of a DataNode causes a flurry of network activity, as the NameNode initiates replication of blocks lost on the dead nodes. Though the loss of a single DataNode may not impact performance much, losing multiple DataNodes will start to be very taxing on cluster resources, and could result in data loss.

5) Remaining: If left uncertified, a single DataNode running out of space could quickly cascade into failures across the entire cluster as data is written to an increasingly-shrinking pool of available DataNodes. Tracking this metric over time is essential to maintain a healthy cluster

(3) MapReduce counters monitoring

- Collect MapReduce counters related information through the web UI. Provide a screenshot.





- Describe how to monitor the execution of MapReduce tasks through related metrics. (5%)

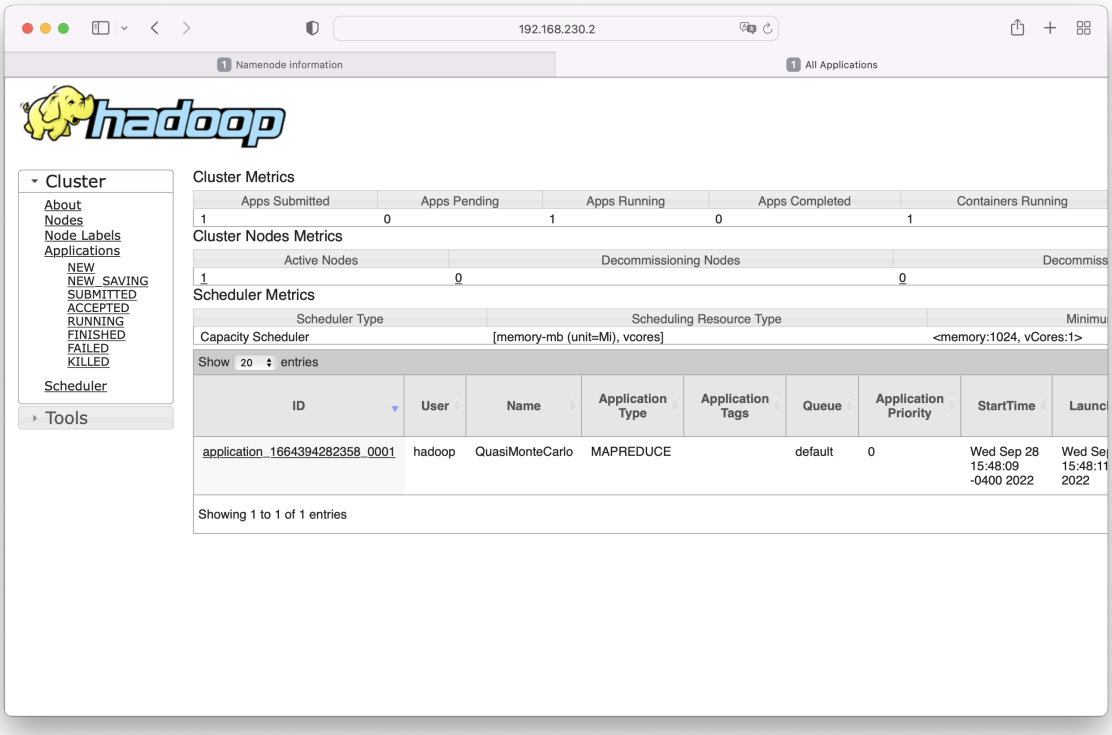
For jobs counters. MILLIS_MAPS/MILLIS_REDUCES tracks the processing time for maps/reduces. NUM_FAILED_MAPS/NUM_FAILED_REDUCES tracks number of failed maps/reduces. RACK_LOCAL_MAPS/DATA_LOCAL_MAPS/OTHER_LOCAL_MAPS tracks where map tasks were executed.

For task counters. REDUCE_INPUT_RECORDS keeps an eye on the number of input records for reduce tasks. SPILLED_RECORDS is the number of records spilled to disk. GC_TIME_MILLS shows the processing time spend in garbage collection.

For custom counters, Custom counters can be used to track more fine-grained counts, such as counting the counting the number of malformed or missing records.

(4) YARN metrics monitoring

- Monitor YARN metrics through HTTP API. Provide a screenshot in your report.



- Select five metrics that you think are most important from the returned json. Explain their usages and why they are important.

1) unhealthyNodes. Once a node is marked unhealthy, other nodes have to fill the void, which can cause a cascade of alerts as node after node nears full disk utilization.

2) activeNodes/lostNodes. The activeNodes metric tracks the current count of active, or normally operating, nodes. This metric should remain static in the absence of anticipated maintenance. If a NodeManager fails to maintain contact with the ResourceManager, it will eventually be marked as “lost” and its resources will become unavailable for allocation.

3) appsFailed. if the percentage of failed map or reduce tasks exceeds a specific threshold the application as a whole will fail.

4) progress. tracking progress alongside other metrics can better help you to determine the cause of any performance degradation.

5) containersFailed. This tracks the number of containers that failed to launch on that particular NodeManager. If the NodeManager’s disk is full, any container it launches will fail.