COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

EECS E6893 Big Data Analytics

# Homework #1 Report

Name: Yi Zhang

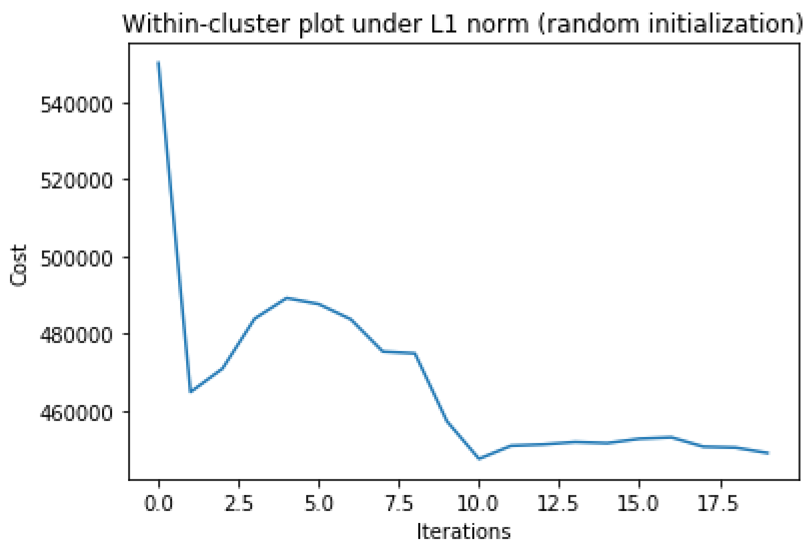
Uni: yz4130

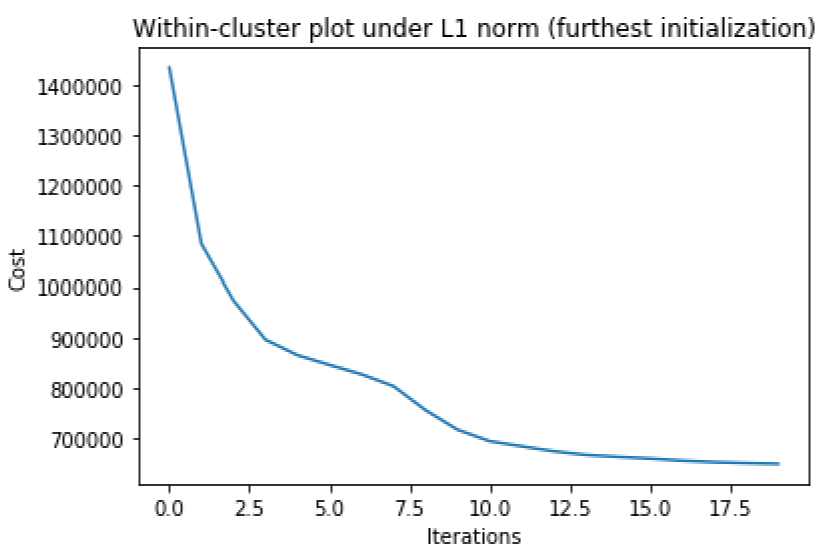
October 2022

## Question 1

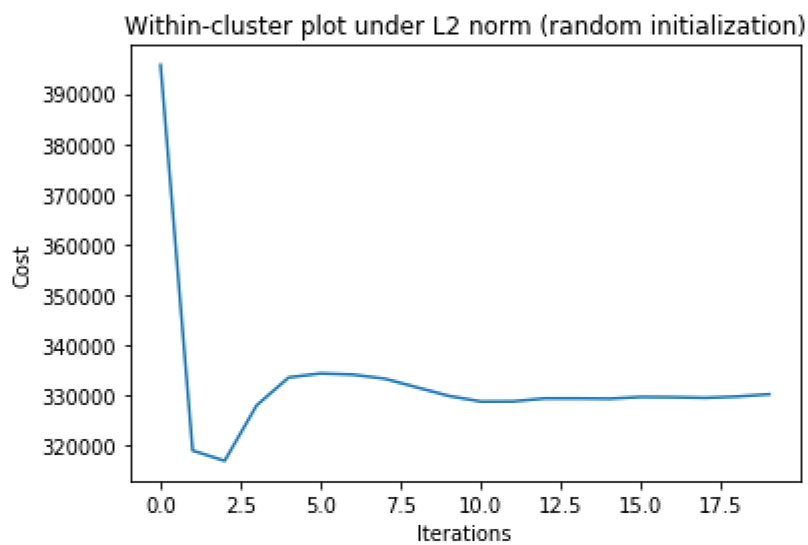
(1) The results are as follows. For random initialization, it can reach a better result within 20 iterations compared to the furthest initialization (compared to the cost function value at iteration 20).

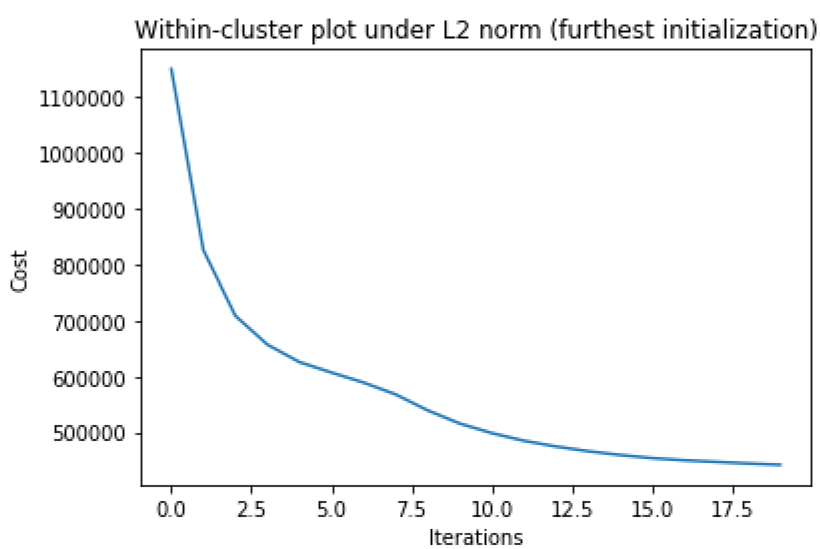
However, the furthest initialization's cost function is always decreasing within 20 iterations while random initialization sometimes failed to minimize the cost function. It might be an indication that the furthest initialization is a bad choice to start, thus leaving lots of room to improve.



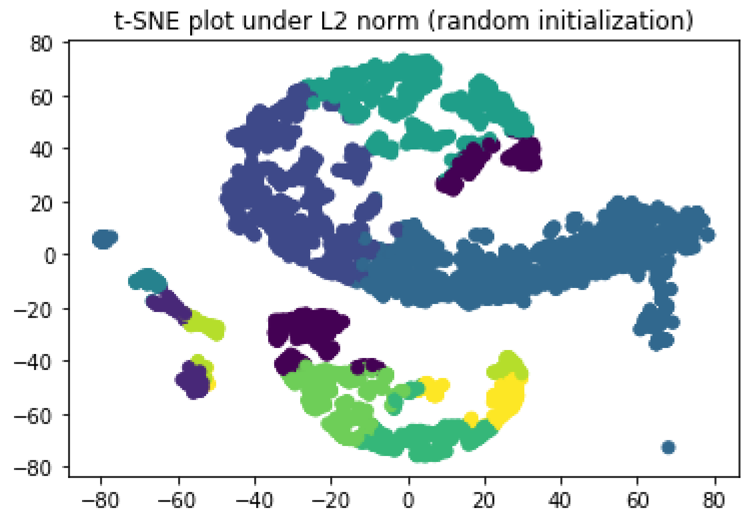



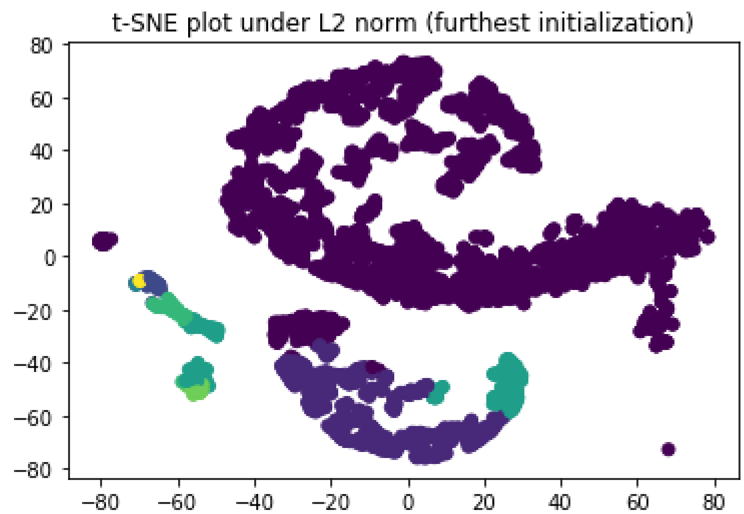
(2) The results are as follows. Still, the situations are very similar to (1). Random initialization can achieve better results but is not able to constantly minimize the cost function.




(3) The results are as follows. It can be seen that the furthest initialization is worse because its plot has fewer colors



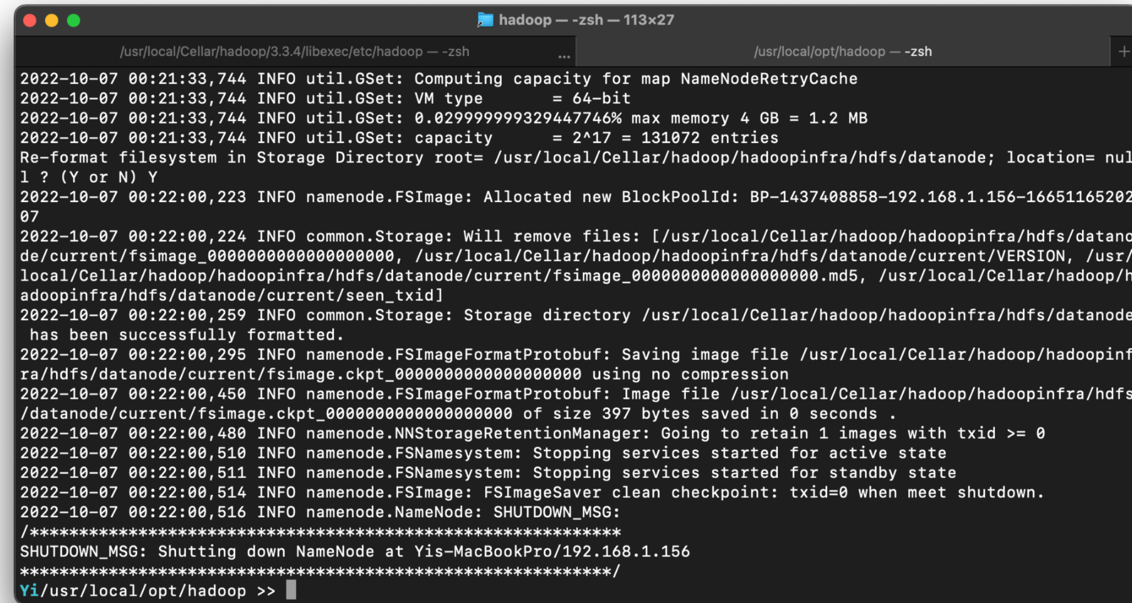




(4) Yes. Regardless of the similarity measure, random initialization is better in terms of cost. As mentioned before, the *c2.txt* gives us the furthest cluster setting. However, it is a bad starting point because the underlying data clusters are not necessarily far away from each other. Given the limited iterations, a bad starting point will not give us good results, which is the reason for the outperformance of the random initialization.

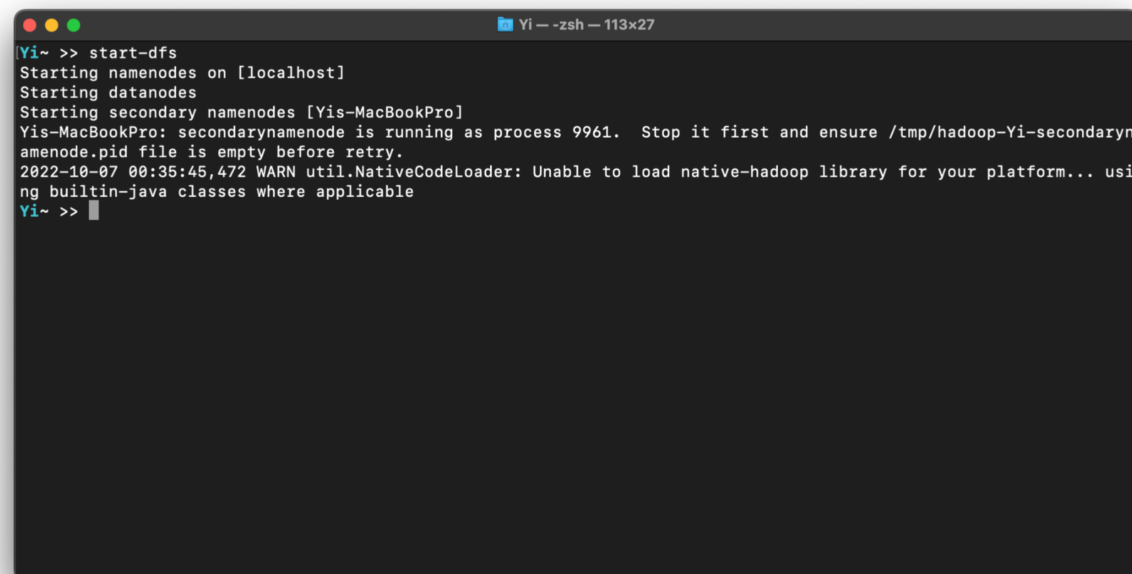
(5) Given a fixed number of centroids K and iterations T, suppose we have N data points to cluster, the time complexity is $O(KTN)$

## Question 2

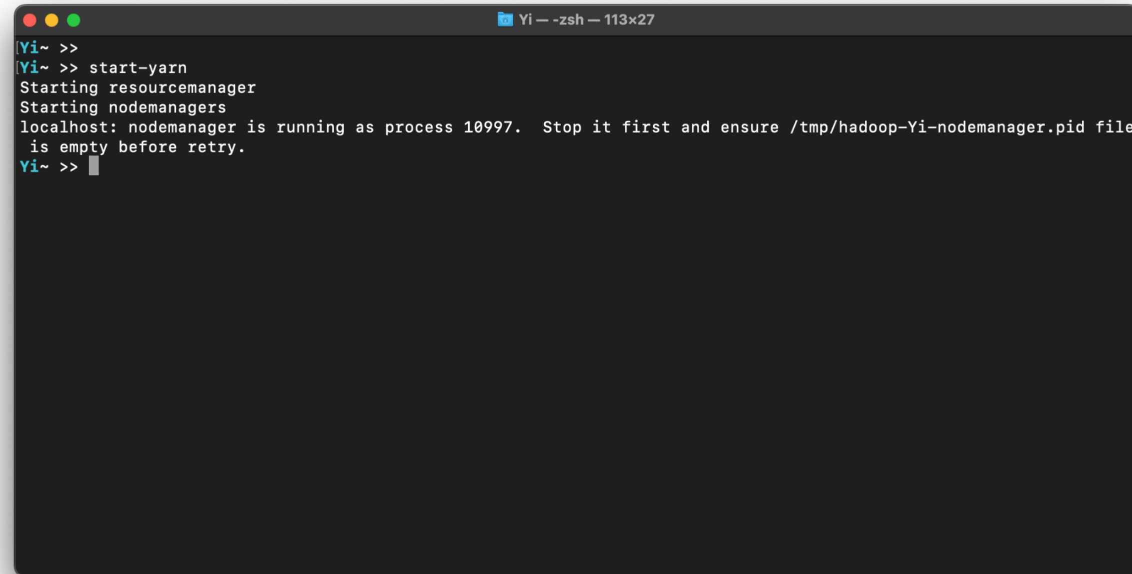
(1) The installation results are as follows.



```
hadoop — -zsh — 113×27
/usr/local/Cellar/hadoop/3.3.4/libexec/etc/hadoop — -zsh    /usr/local/opt/hadoop — -zsh
2022-10-07 00:21:33,744 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2022-10-07 00:21:33,744 INFO util.GSet: VM type       = 64-bit
2022-10-07 00:21:33,744 INFO util.GSet: 0.029999999329447746% max memory 4 GB = 1.2 MB
2022-10-07 00:21:33,744 INFO util.GSet: capacity      = 2^17 = 131072 entries
Re-format filesystem in Storage Directory root= /usr/local/Cellar/hadoop/hadoopinfra/hdfs/datanode; location= nul
l ? (Y or N) Y
2022-10-07 00:22:00,223 INFO namenode.FSImage: Allocated new BlockPoolId: BP-1437408858-192.168.1.156-16651165202
07
2022-10-07 00:22:00,224 INFO common.Storage: Will remove files: [/usr/local/Cellar/hadoop/hadoopinfra/hdfs/datano
de/current/fsimage_0000000000000000000, /usr/local/Cellar/hadoop/hadoopinfra/hdfs/datanode/current/VERSION, /usr/
local/Cellar/hadoop/hadoopinfra/hdfs/datanode/current/fsimage_0000000000000000000.md5, /usr/local/Cellar/hadoop/h
adoopinfra/hdfs/datanode/current/seen_txid]
2022-10-07 00:22:00,259 INFO common.Storage: Storage directory /usr/local/Cellar/hadoop/hadoopinfra/hdfs/datanode
 has been successfully formatted.
2022-10-07 00:22:00,295 INFO namenode.FSImageFormatProtobuf: Saving image file /usr/local/Cellar/hadoop/hadoopinf
ra/hdfs/datanode/current/fsimage.ckpt_0000000000000000000 using no compression
2022-10-07 00:22:00,450 INFO namenode.FSImageFormatProtobuf: Image file /usr/local/Cellar/hadoop/hadoopinfra/hdfs
/datanode/current/fsimage.ckpt_0000000000000000000 of size 397 bytes saved in 0 seconds .
2022-10-07 00:22:00,480 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2022-10-07 00:22:00,510 INFO namenode.FSNamesystem: Stopping services started for active state
2022-10-07 00:22:00,511 INFO namenode.FSNamesystem: Stopping services started for standby state
2022-10-07 00:22:00,514 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2022-10-07 00:22:00,516 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at Yis-MacBookPro/192.168.1.156
************************************************************/
Yi/usr/local/opt/hadoop >>
```



```
Yi — -zsh — 113×27
Yi~ >> start-dfs
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [Yis-MacBookPro]
Yis-MacBookPro: secondarynamenode is running as process 9961.  Stop it first and ensure /tmp/hadoop-Yi-secondaryn
amenode.pid file is empty before retry.
2022-10-07 00:35:45,472 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... usi
ng builtin-java classes where applicable
Yi~ >>
```




(2) The HTTP API results are as follows. I think the following metrics are the most important ones.

**DataNodes usages**

This metrics gives us how heavily we are utilizing the datanodes, with min, max, median and standard deviation, we can have a comprehensive knowledge of the usages.

**Missing blocks** (obtained through the JSON output using jmx suffix)

A missing block metric by its name, documents the number of meaning blocks in the cluster.  It is important for us to evaluate the health of our cluster because we cannot recover the data in that block anymore if it is missed, so it's crucial to keep track of this metric.

**Dead Nodes**

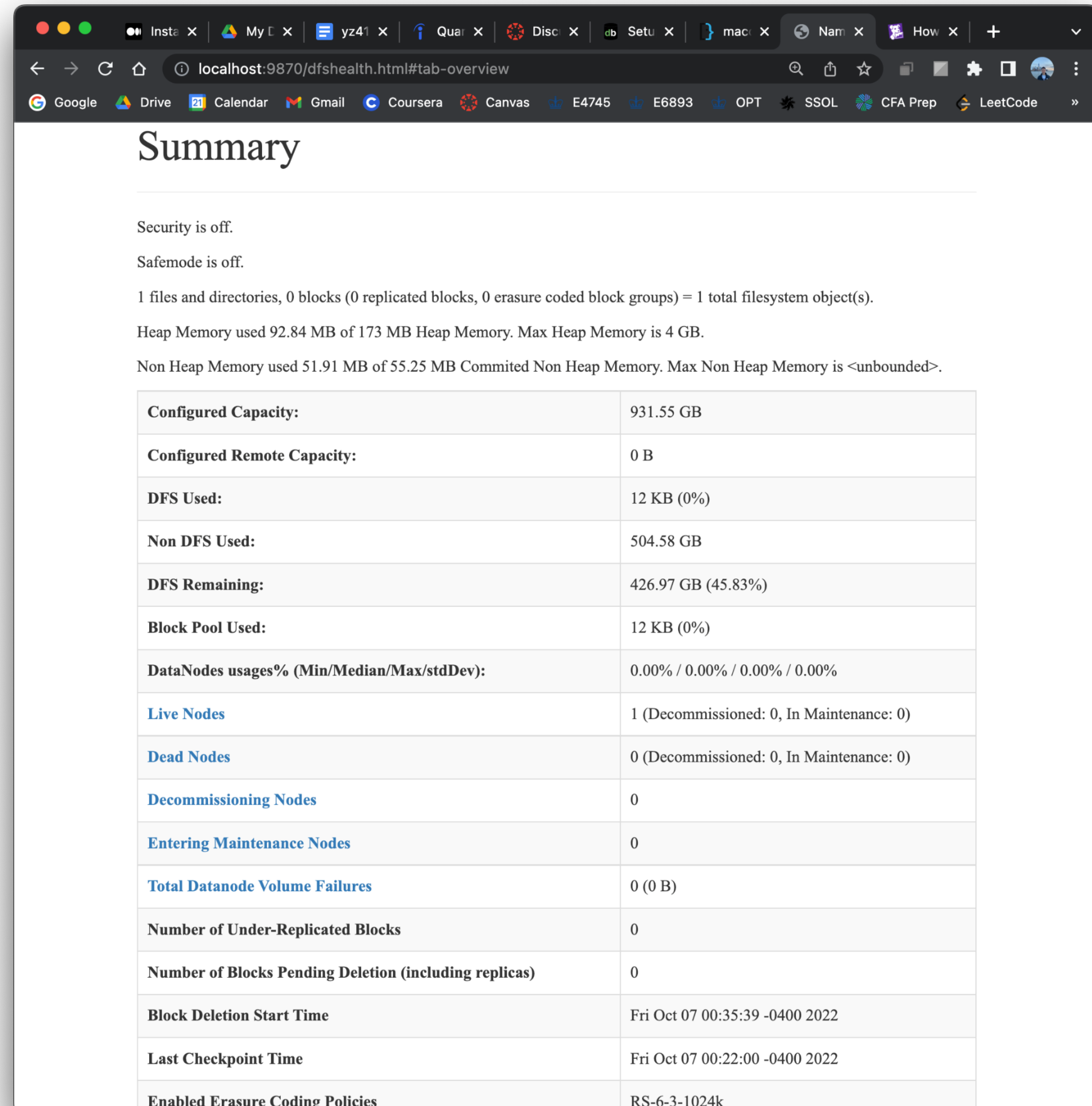
This metric simply count the number of dead nodes, which are nodes that have not communicated with namenode for a while. We need to monitor this because too many dead nodes will result in data loss and will consume lots of cluster recources.

**Block Pool Used**

This metrics measures the percentage of the the blocks we are using. In order to ensure the continuity of the cluster, we should make sure enough blocks are available at any time.

**Total Datanode Volume Failure**

It simply tracks the volume of failure when a disk is down. The new version of Hadoop is able to handle the disk failure, i.e. the datanode will continue to work even if disks are failed (up to a preset threshold). Monitoring this metric can help us understand how the failed volume behaves over time, based on this information, we can choose our threshold to maximize our computing power.




# Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks (0 replicated blocks, 0 erasure coded block groups) = 1 total filesystem object(s).

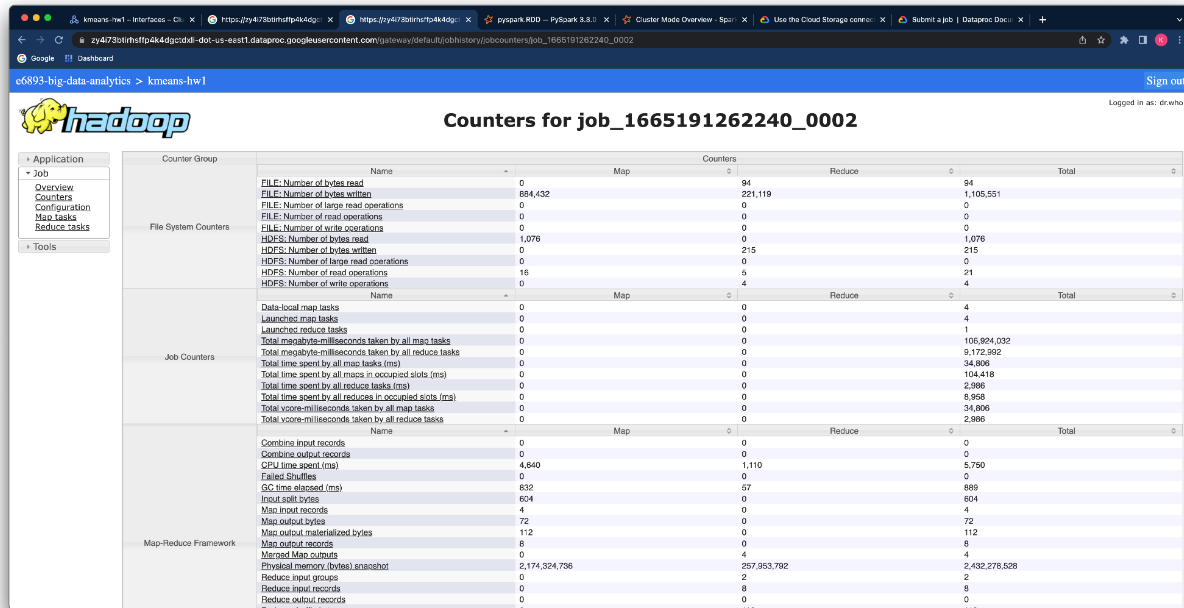
Heap Memory used 92.84 MB of 173 MB Heap Memory. Max Heap Memory is 4 GB.

Non Heap Memory used 51.91 MB of 55.25 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

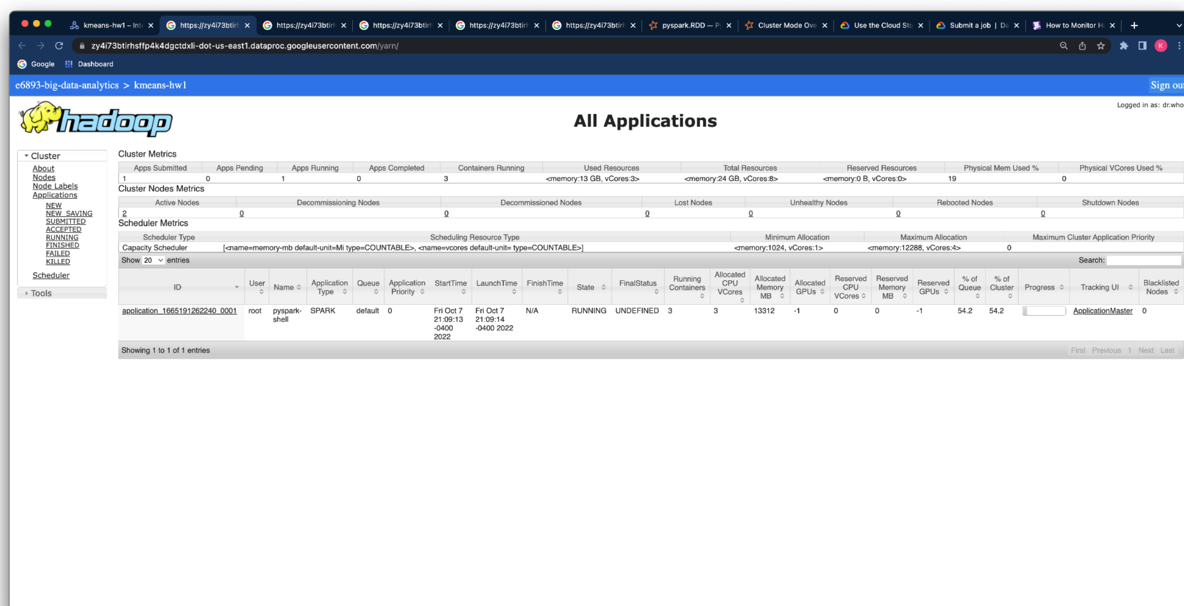
| | |
|---|---|
| **Configured Capacity:** | 931.55 GB |
| **Configured Remote Capacity:** | 0 B |
| **DFS Used:** | 12 KB (0%) |
| **Non DFS Used:** | 504.58 GB |
| **DFS Remaining:** | 426.97 GB (45.83%) |
| **Block Pool Used:** | 12 KB (0%) |
| **DataNodes usages% (Min/Median/Max/stdDev):** | 0.00% / 0.00% / 0.00% / 0.00% |
| **Live Nodes** | 1 (Decommissioned: 0, In Maintenance: 0) |
| **Dead Nodes** | 0 (Decommissioned: 0, In Maintenance: 0) |
| **Decommissioning Nodes** | 0 |
| **Entering Maintenance Nodes** | 0 |
| **Total Datanode Volume Failures** | 0 (0 B) |
| **Number of Under-Replicated Blocks** | 0 |
| **Number of Blocks Pending Deletion (including replicas)** | 0 |
| **Block Deletion Start Time** | Fri Oct 07 00:35:39 -0400 2022 |
| **Last Checkpoint Time** | Fri Oct 07 00:22:00 -0400 2022 |
| **Enabled Erasure Coding Policies** | RS-6-3-1024k |

(3) To monitor the tasks, we clicked the counters for a specific Job to see more details. For file system counter, there are numbers regarding read/written for map and reduce operations.
For jobs counter, the API provides total time for mapping and reducing. Through these metrics, we can know which part of the job is harder and takes more time.

For map-reduce framework, we can have information about memory usages, heap usages etc.
For shuffle error counters, we can see if anything goes wrong in the course of shuffling.




(4) The API gives us the following.



There are 3 metric classes shown as cluster metrics, cluster nodes metrics, scheduler metrics. I will discuss the important ones from each category.

**Total Resources & Used Resources**
It tells the number of cores and the memory we have. Combining these two metrics, we can get a glimpse of the overall performance cluster resources.

**Unhealthy Nodes**
It tells the number of unhealthy nodes. A node is unhealthy if its disk utilization level is beyond some preset threshold. Because other nodes will allocate their resources to make up for his unhealthy brother, they might become unhealthy later, causing catastrophic chain reaction. Thus, it is important to monitor this metrics.

**Minimum & Maximum Allocation**
It measures the maximum memory and number of cores used for a job. By monitoring this number, we can learn what jobs consume most of our resources.

**Lost Nodes**
The usage is as its name suggests. We can look at this metric and add new nodes if number of lost nodes is too large.

**Scheduling Resource Type**
This metric tells us what kind of resources is being allocated to our jobs. It is again useful for us since we can learn the resources that our jobs consume.

In [1]:
```
import operator
import sys, os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import linalg
from sklearn.manifold import TSNE
from pyspark import SparkConf, SparkContext
```

```
/opt/conda/anaconda/lib/python3.7/site-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. U
se the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [2]:
```
# Macros.
MAX_ITER = 20
DATA_PATH = "gs://6893_course_data/hw1_q1/data.txt"
C1_PATH = "gs://6893_course_data/hw1_q1/c1.txt"
C2_PATH = "gs://6893_course_data/hw1_q1/c2.txt"
L1 = 1
L2 = 2
```

In [10]:
```
# Helper functions
def closest(p, centroids, norm):
    """
    Compute closest centroid for a given point.
    Args:
        p (numpy.ndarray): input point
        centroids (list): A list of centroids points
        norm (int): 1 or 2
    Returns:
        int: The index of closest centroid.
    """
    closest_c = min([(i, linalg.norm(p - c, norm))
                    for i, c in enumerate(centroids)],
                    key=operator.itemgetter(1))[0]
    return closest_c


def within_cluster_cost(data, centroids, norm):
    """
    Compute within-cluster cost.
    Args:
        data (RDD): a RDD of the form (centroid, (point, 1))
        centroids (list): A list of centroids points
        norm (int): 1 or 2
    Returns:
        Float: Within-cluster cost of the current classification.
    """
    cost = sum(data.map(lambda pt: linalg.norm(pt[1][0] - centroids[pt[0]], norm)).collect())
    return cost


def tSNE_vis(data, norm, init):
    """
    Produce a t-SNE 2D dimensional clustering result.
    Args:
        data (RDD): a RDD of (centroid, (point, 1))
        norm (int): 1 or 2
        init (str): 'random' or 'furthest'
    Returns:
        None
    """
    arr = np.array(data.map(lambda pt: list(pt[1][0])).collect())
    labels = data.keys().collect()
    embedded = TSNE(n_components=2, random_state=666).fit_transform(arr)
    x = embedded[:, 0]
    y = embedded[:, 1]
    plt.scatter(x, y, c=labels)
    plt.title('t-SNE plot under L%d norm (%s initialization)'%(norm, init))
    plt.show()
```

In [4]:

```python
# K-means clustering
def kmeans(data, centroids, norm, init):
    """
    Conduct k-means clustering given data and centroid.
    Args:
        data (RDD): RDD of points
        centroids (list): A list of centroids points
        norm (int): 1 or 2
        init (str): 'random' or 'furthest'
    Returns:
        RDD: assignment information of points, a RDD of (centroid, (point, 1))
        list: a list of centroids
    """

    cost_list = np.zeros(MAX_ITER)

    # iterative k-means
    for i in range(MAX_ITER):

        # Points assignment
        points = data.map(lambda p: (closest(p, centroids, norm), (p, 1)))

        # Cost Calculation
        cost_list[i] = within_cluster_cost(points, centroids, norm)

        # Updata centroids
        reduced_pts = points.reduceByKey(lambda p1, p2: (p1[0] + p2[0], p1[1] + p2[1]))
        centroids = reduced_pts.values().map(lambda c: c[0] / c[1]).collect()

    # cost plot
    plt.plot(cost_list)
    plt.xlabel('Iterations')
    plt.ylabel('Cost')
    plt.title('Within-cluster plot under L%d norm (%s initialization)'%(norm, init))
    plt.show()

    return points, centroids
```

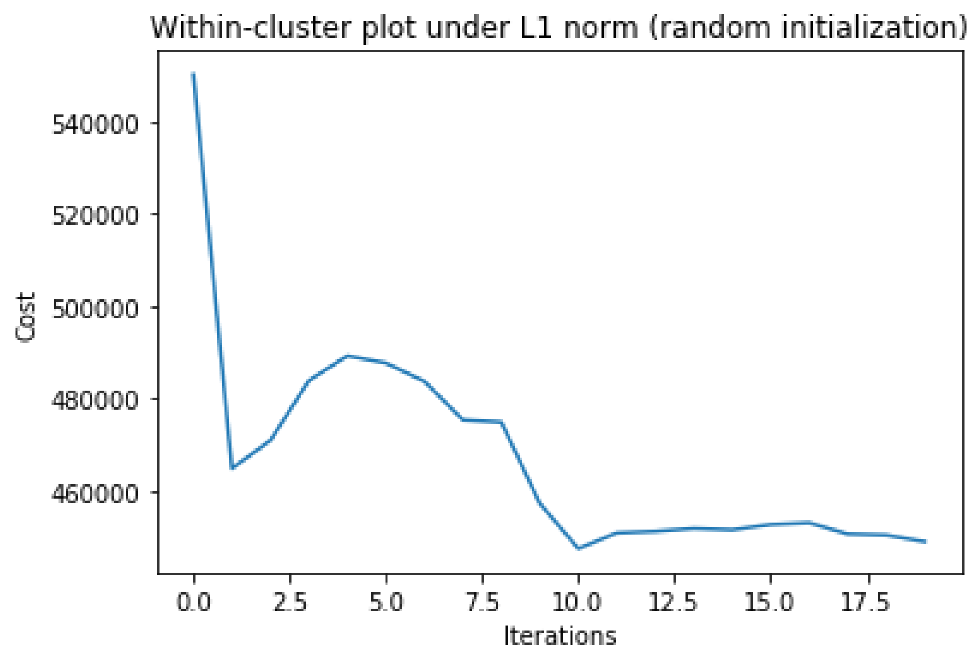
In [5]:

```python
# Spark settings
/gateway/default/node/conf?host&port = SparkConf()
sc = SparkContext(/gateway/default/node/conf?host&port=/gateway/default/node/conf?host&port)

# Load the data, cache this since we're accessing this each iteration
data = sc.textFile(DATA_PATH).map(
        lambda line: np.array([float(x) for x in line.split(' ')])
        ).cache()
# Load the initial centroids c1, split into a list of np arrays
centroids1 = sc.textFile(C1_PATH).map(
        lambda line: np.array([float(x) for x in line.split(' ')])
        ).collect()
# Load the initial centroids c2, split into a list of np arrays
centroids2 = sc.textFile(C2_PATH).map(
        lambda line: np.array([float(x) for x in line.split(' ')])
        ).collect()
```

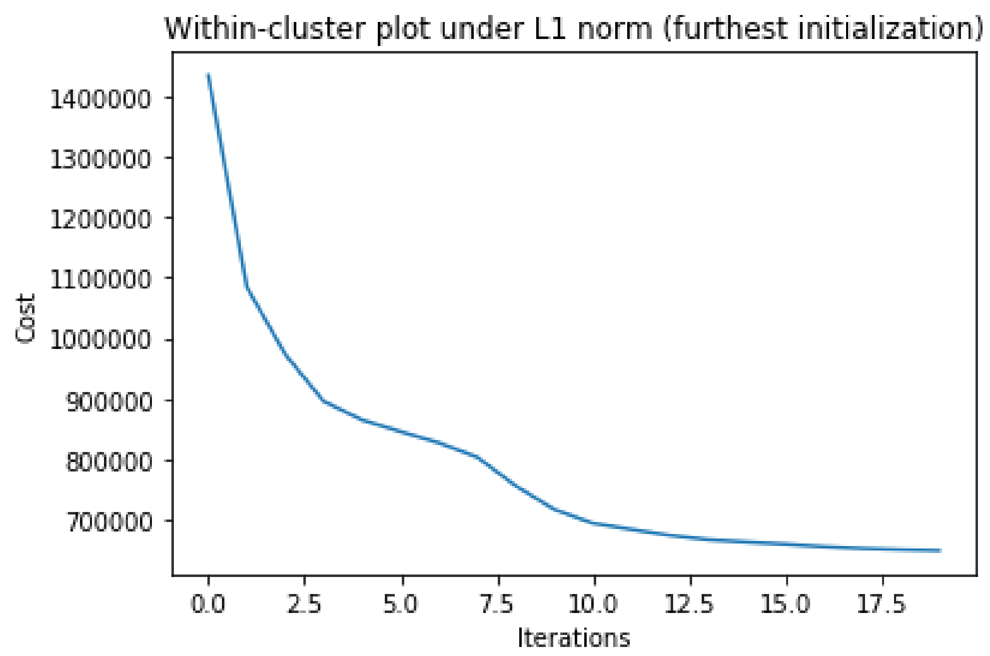
In [11]:

```python
# run kmeans
clustered_1, updated_centroids_1 = kmeans(data, centroids1, L1, 'random')
```



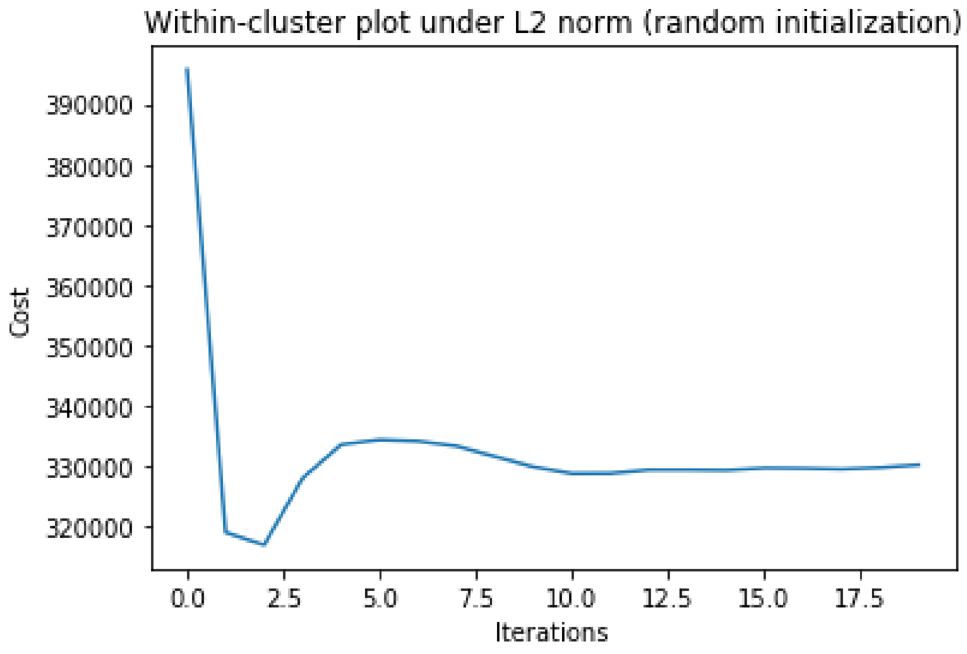

In [12]:

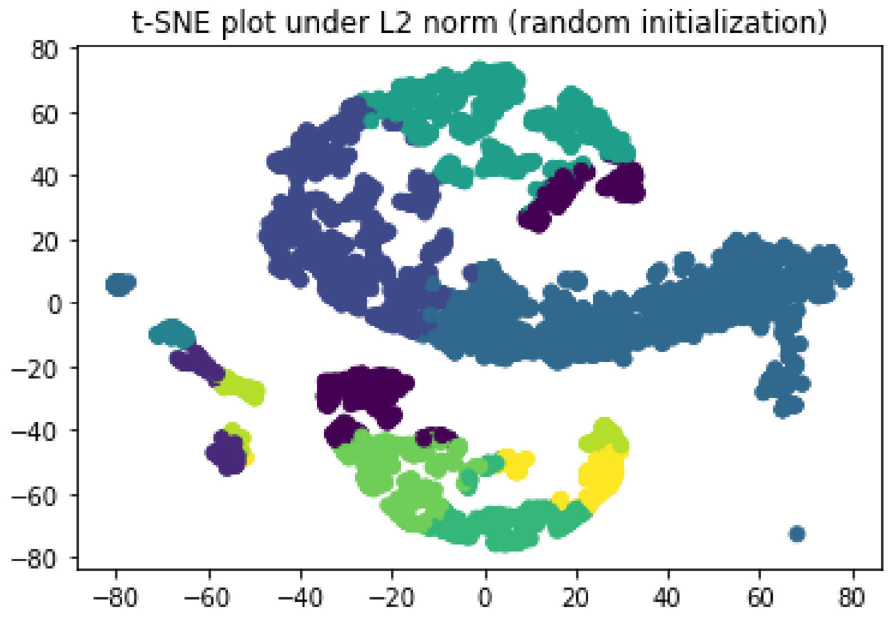
```python
# run kmeans
clustered_2, updated_centroids_2 = kmeans(data, centroids2, L1, 'furthest')
```



In [13]:

```
# run kmeans
clustered_3, updated_centroids_3 = kmeans(data, centroids1, L2, 'random')
# t-SNE visualization
tSNE_vis(clustered_3, L2, 'random')
```




In [14]:

```
# run kmeans
clustered_4, updated_centroids_4 = kmeans(data, centroids2, L2, 'furthest')
# t-SNE visualization
tSNE_vis(clustered_4, L2, 'furthest')
```