Part 1: Line Counting with MapReduce (20 marks)

1. (15 marks) Implement a MapReduce program to count the number of lines in a large text file (shakespeare.txt). Analyze the performance of your implementation, considering factors such as the number of mappers and reducers, input file size, and network communication overhead. Provide a detailed performance analysis, including graphs as needed and a discussion of optimization strategies. (Implement using Hadoop MapReduce)

Implemented in /LineCount/LineCount.java
I used the docker and running command:
javac -classpath `hadoop classpath` -d . LineCount.java
jar -cvf linecount.jar *
hadoop jar linecount.jar LineCount /user/root/input /user/root/output

```
2025-06-01 06:13:24,402 INFO mapreduce.Job: Running job: job_1748755140203_0002
2025-06-01 06:13:29,477 INFO mapreduce.Job: Job job_1748755140203_0002 running in uber mode : false
2025-06-01 06:13:29,480 INFO mapreduce.Job:  map 0% reduce 0%
2025-06-01 06:13:34,528 INFO mapreduce.Job:  map 50% reduce 0%
2025-06-01 06:13:35,542 INFO mapreduce.Job:  map 100% reduce 0%
2025-06-01 06:13:38,557 INFO mapreduce.Job:  map 100% reduce 100%
2025-06-01 06:13:38,571 INFO mapreduce.Job: Job job_1748755140203_0002 completed successfully
2025-06-01 06:13:38,631 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=38
                FILE: Number of bytes written=688218
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2560110
                HDFS: Number of bytes written=11
                HDFS: Number of read operations=11
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Rack-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=14112
                Total time spent by all reduces in occupied slots (ms)=12312
                Total time spent by all map tasks (ms)=3528
                Total time spent by all reduce tasks (ms)=1539
                Total vcore-milliseconds taken by all map tasks=3528
                Total vcore-milliseconds taken by all reduce tasks=1539
                Total megabyte-milliseconds taken by all map tasks=14450688
                Total megabyte-milliseconds taken by all reduce tasks=12607488
        Map-Reduce Framework
                Map input records=58483
                Map output records=58483
                Map output bytes=526347
                Map output materialized bytes=50
                Input split bytes=208
                Combine input records=58483
                Combine output records=2
                Reduce input groups=1
                Reduce shuffle bytes=50
                Reduce input records=2
                Reduce output records=1
                Spilled Records=4
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=102
                CPU time spent (ms)=2210
                Physical memory (bytes) snapshot=823156736
                Virtual memory (bytes) snapshot=18764619776
                Total committed heap usage (bytes)=751304704
                Peak Map Physical memory (bytes)=321527808
                Peak Map Virtual memory (bytes)=5140504576
                Peak Reduce Physical memory (bytes)=188497920
                Peak Reduce Virtual memory (bytes)=8484806656
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=2559902
        File Output Format Counters
                Bytes Written=11
```

File System Counters
FILE: Number of bytes read=38
FILE: Number of bytes written=688218
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=2560110
HDFS: Number of bytes written=11
HDFS: Number of read operations=11
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
HDFS: Number of bytes read erasure-coded=0
Job Counters
Launched map tasks=2
Launched reduce tasks=1
Rack-local map tasks=2
Total time spent by all maps in occupied slots (ms)=14112
Total time spent by all reduces in occupied slots (ms)=12312
Total time spent by all map tasks (ms)=3528
Total time spent by all reduce tasks (ms)=1539
Total vcore-milliseconds taken by all map tasks=3528
Total vcore-milliseconds taken by all reduce tasks=1539
Total megabyte-milliseconds taken by all map tasks=14450688
Total megabyte-milliseconds taken by all reduce tasks=12607488
Map-Reduce Framework
Map input records=58483
Map output records=58483
Map output bytes=526347
Map output materialized bytes=50
Input split bytes=208
Combine input records=58483
Combine output records=2
Reduce input groups=1
Reduce shuffle bytes=50
Reduce input records=2
Reduce output records=1
Spilled Records=4
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=102
CPU time spent (ms)=2210
Physical memory (bytes) snapshot=823156736
Virtual memory (bytes) snapshot=18764619776
Total committed heap usage (bytes)=751304704
Peak Map Physical memory (bytes)=321527808

Peak Map Virtual memory (bytes)=5140504576
Peak Reduce Physical memory (bytes)=188497920
Peak Reduce Virtual memory (bytes)=8484806656
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=2559902
File Output Format Counters
Bytes Written=11

Input file size:

```
root@163d5fa026b0:~/linecount# hdfs dfs -du -h /user/root/input/shakespeare.txt
2.4 M  7.3 M  /user/root/input/shakespeare.txt
```

The success of running the MapReduce program:

```
root@163d5fa026b0:/# hdfs dfs -cat /user/root/output/part-00000
2025-06-01 06:50:22,553 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
line    58483
root@163d5fa026b0:/# hdfs dfs -ls /user/root/output
Found 2 items
-rw-r--r--   3 root supergroup          0 2025-06-01 06:13 /user/root/output/_SUCCESS
-rw-r--r--   3 root supergroup         11 2025-06-01 06:13 /user/root/output/part-00000
```

Result:

```
root@4505d46e64a:~/linecount# hdfs dfs -cat /user/root/output/part-00000
2025-06-02 05:20:26,533 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
line    58483
```

Total Line:58483

See /LineCount/lineCountTerminalOutput.txt
See /LineCount/output/

The MapReduce program succeeded.

2. (5 marks) Propose at least one optimization strategy to improve the efficiency of your line counting MapReduce program. Justify your choice of optimization and quantify its impact on performance. (Describe in words)

My proposal for the strategy to improve efficiency is to add a Combiner

In my current LineCount.java MapReduce program, the mapper would write a key-value pair ("line", 1) for every single line, regardless of how many lines are processed at the mapper. A combiner would perform as a “mini reducer” that runs on each mapper node, aggregating key-value pairs locally before sending the line to the reducer through the network. It acts like gathering the number of lines at each node and sending the number of lines count at each node to reducer through the network and combines. This can reduce the amount of data transferred from the Mapper to the Reducer (Shuffle Size) as well as the amount of records processed by the reducer.

Quantify the impact:

For 100k (100,000) lines processed at each mapper, my MapReduce program would emit 100,000 pairs of ("line", 1), with combiner, only 1 pair of ("line", 100000) would be emitted.

If we have 10 mappers, 10* 100,000 = 1 million key-value pairs would be processed and transferred to reducer and process while with the combiner, only 10 key-value pairs would be processed. This can have (1,000,000-10)/1,000,000 ≈ 99.99% improvement toward the efficiency of the reducer.

Part 2: K-Means Clustering on MapReduce (30 marks)

3. (5 marks) Propose a distributed k-means clustering algorithm using MapReduce. Use the provided dataset (data_points.txt). Your implementation should handle a variable number of clusters. Thoroughly explain your algorithm, including the partitioning strategy, centroid calculation, and convergence criteria. Discuss the choice of distance metric and its rationale. (Describe in words)

Use the skeleton provided in k-means tutorial.

As descried, I will deploy k-means clustering algorithm using MapReduce. The algorithm would be separated as Mapper and Reducer stage and a driver:
Input dataset: data_points.txt, a long list of 2d points
Mapper:

- Read a portion of input points (data_points.txt)
- Loads the current centroids list from HDFS (SequenceFile) and for each point, find the closest centroid based on *Euclidean Distance,*
- Emit key (centroid point) and value (current point) pair

Reducer:

- Each reducer receives a centroid and all the points assigned to it as key (centroid point) and value (current point) pair from the Mapper
- Calculate the new centroid coordinates based on the points.
- Convergence Criteria: Conduct distance-based convergence check based on *Euclidean Distance* between old centroid and new centroid and if the distance is less than the threshold, we will update the CONVERGED counter
- Stores the new centroid for writing back to HDFS (SequenceFile)

Driver:

- Define the number of k and initialize their corresponding coordinates
- The MapReduce runs in a loop for a maximum of a defined number of iterations(20).
- After each iteration, the updated centroids are written to the same path (centroid/cen.seq) and reloaded by mappers in the next round.
- The loop breaks early if all centroids converge, using the Hadoop counter CONVERGED to detect convergence dynamically as the counter is equal to k.

Centroid Calculation:

- Calculate the mean of all points on x and y coordinates respectively for this centroid and assign it as new centroid coordinates.
- $x_{mean} = \frac{1}{n}\sum_i^n x_i \quad y_{mean} = \frac{1}{n}\sum_i^n y_i$

Distance metric and its rationale

- Using Euclidean distance for finding the closest centroid and calculating the distance between old and new centroid
- Euclidean distance is effective for continuous, low-dimensional numerical data as between point and ensure spherical cluster shapes
- $Distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

4. (20 marks) Experiment with different values of k = 5 and 9). For each k, report the cluster centroids, the number of iterations required for convergence (or the maximum iterations reached), the computation time, and a qualitative analysis of the resulting clusters. Visualize your results where possible (e.g., scatter plot of data points with cluster assignments). Analyze the impact of k on the quality of the clustering results and the computational cost. (Implement using Hadoop MapReduce)

Implemented in /Kmeans/KMeans.java
Scatter plot script implemented and visualized in
/Kmeans/kmeans_scatter_plot.ipynb and kmeans_scatter_plot.pdf

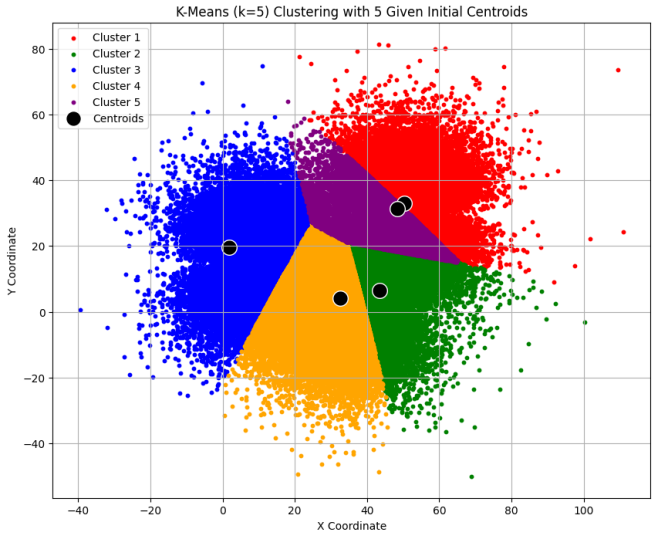
Compile Command:
javac -classpath `hadoop classpath` -d . KMeans.java
jar cf KMeans.jar KMeans*.class
hadoop jar KMeans.jar KMeans

Set iteration to 20, with threshold 0.01 for point distance to check for convergence

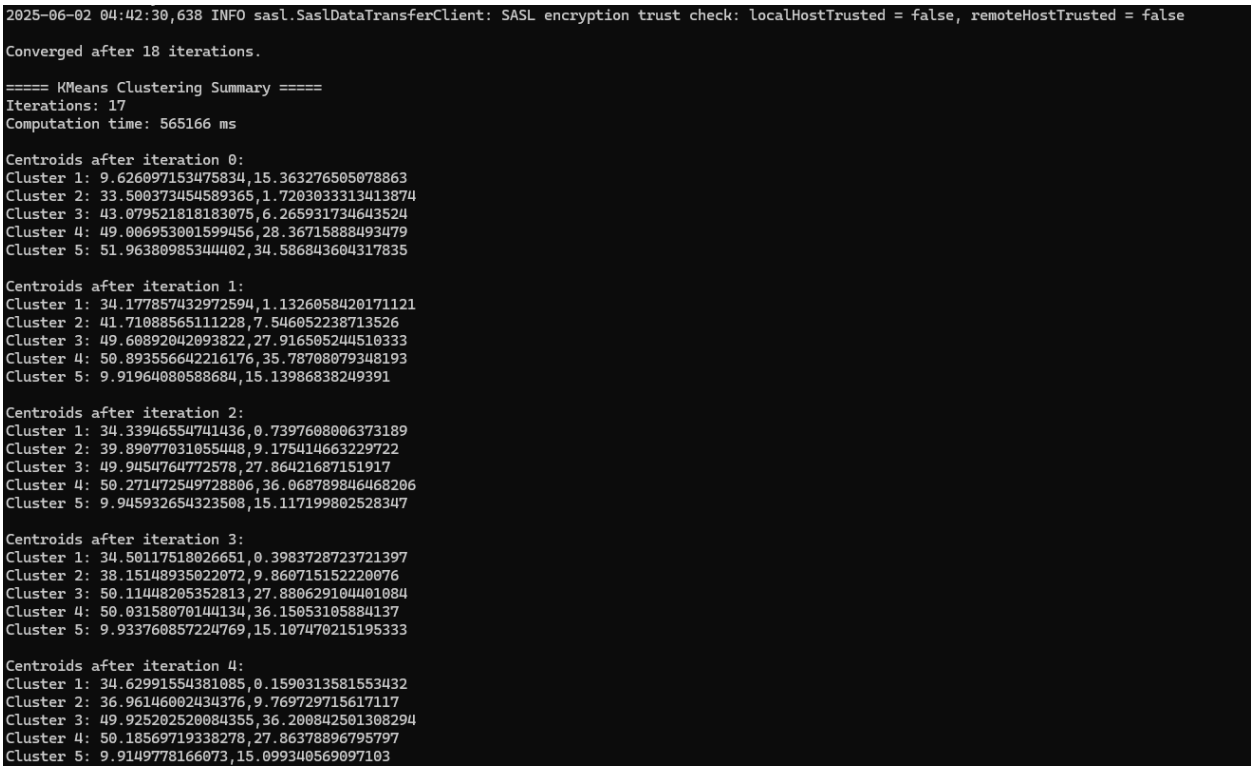
For k = 5

```
        IntWritable.class);
final IntWritable value = new IntWritable(0);
centerWriter.append(new Text("50.197031637442876,32.94048164287042"), value);
centerWriter.append(new Text("43.407412339767056,6.541037020010927"), value);
centerWriter.append(new Text("1.7885358732482017,19.666057053079573"), value);
centerWriter.append(new Text("32.6358540480337,4.03843047564191"), value);
centerWriter.append(new Text("48.41919054694239,31.23767287880673"), value);
```

Initialize centroid
Centroid 1: 50.197031637442876,32.94048164287042
Centroid 2: 43.407412339767056,6.541037020010927
Centroid 3: 1.7885358732482017,19.666057053079573
Centroid 4: 32.6358540480337,4.03843047564191
Centroid 5: 48.41919054694239,31.23767287880673
Scatter plot:



Result
Converged after 18 iterations with Computation time 565166ms

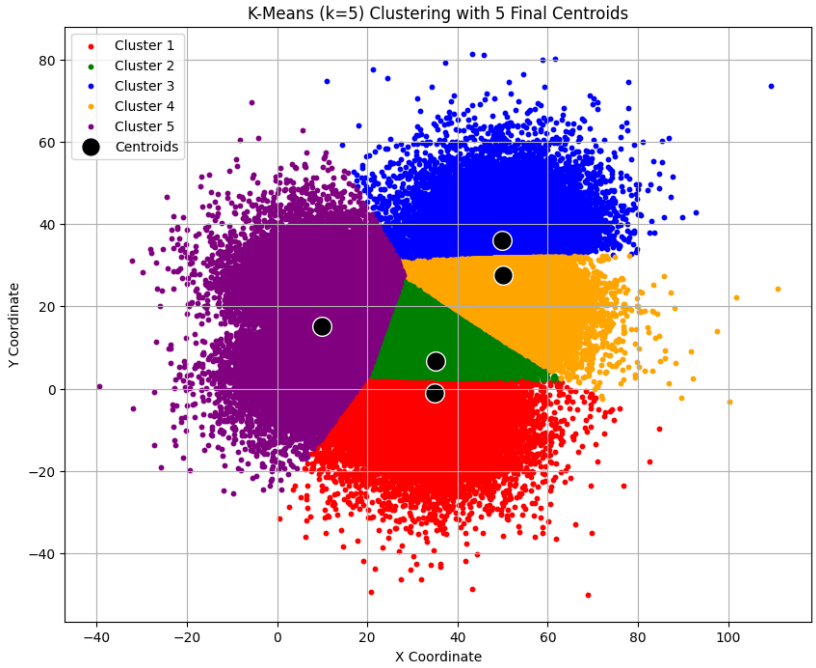


```
Centroids after iteration 13:
Cluster 1: 34.881910518863336,-0.900389791929061
Cluster 2: 35.153601851433464,7.261231243081093
Cluster 3: 49.8847673780295,36.00762518518747
Cluster 4: 50.14239864104818,27.632859238386406
Cluster 5: 9.895582183872,15.104366486735312

Centroids after iteration 14:
Cluster 1: 34.88347735123231,-0.9680814126153188
Cluster 2: 35.136987053378775,7.132026933334772
Cluster 3: 49.88691564611456,35.98463323692881
Cluster 4: 50.13846737119991,27.61843692213444
Cluster 5: 9.89625761114959,15.105199448414604

Centroids after iteration 15:
Cluster 1: 34.88514122873018,-1.032584300780561
Cluster 2: 35.12575455650142,7.013125924677395
Cluster 3: 49.888876864659046,35.96432144908959
Cluster 4: 50.135084635895254,27.606388769275856
Cluster 5: 9.897706168633125,15.105818087380326

Centroids after iteration 16:
Cluster 1: 34.88604694952332,-1.0929509992113928
Cluster 2: 35.116230478263894,6.907532552773523
Cluster 3: 49.89141664622288,35.94866201752815
Cluster 4: 50.13179525198624,27.59695565351712
Cluster 5: 9.89850720974356,15.10647257739537

Centroids after iteration 17:
Cluster 1: 34.88767058806049,-1.1492691361599863
Cluster 2: 35.10524898166031,6.810348684944596
Cluster 3: 49.893531834503435,35.93469249329149
Cluster 4: 50.12782925362032,27.58707497118695
Cluster 5: 9.899168246470962,15.10669211914742

Final centroid: 1: 34.88767058806049,-1.1492691361599863

Final centroid: 2: 35.10524898166031,6.810348684944596

Final centroid: 3: 49.893531834503435,35.93469249329149

Final centroid: 4: 50.12782925362032,27.58707497118695

Final centroid: 5: 9.899168246470962,15.10669211914742
```

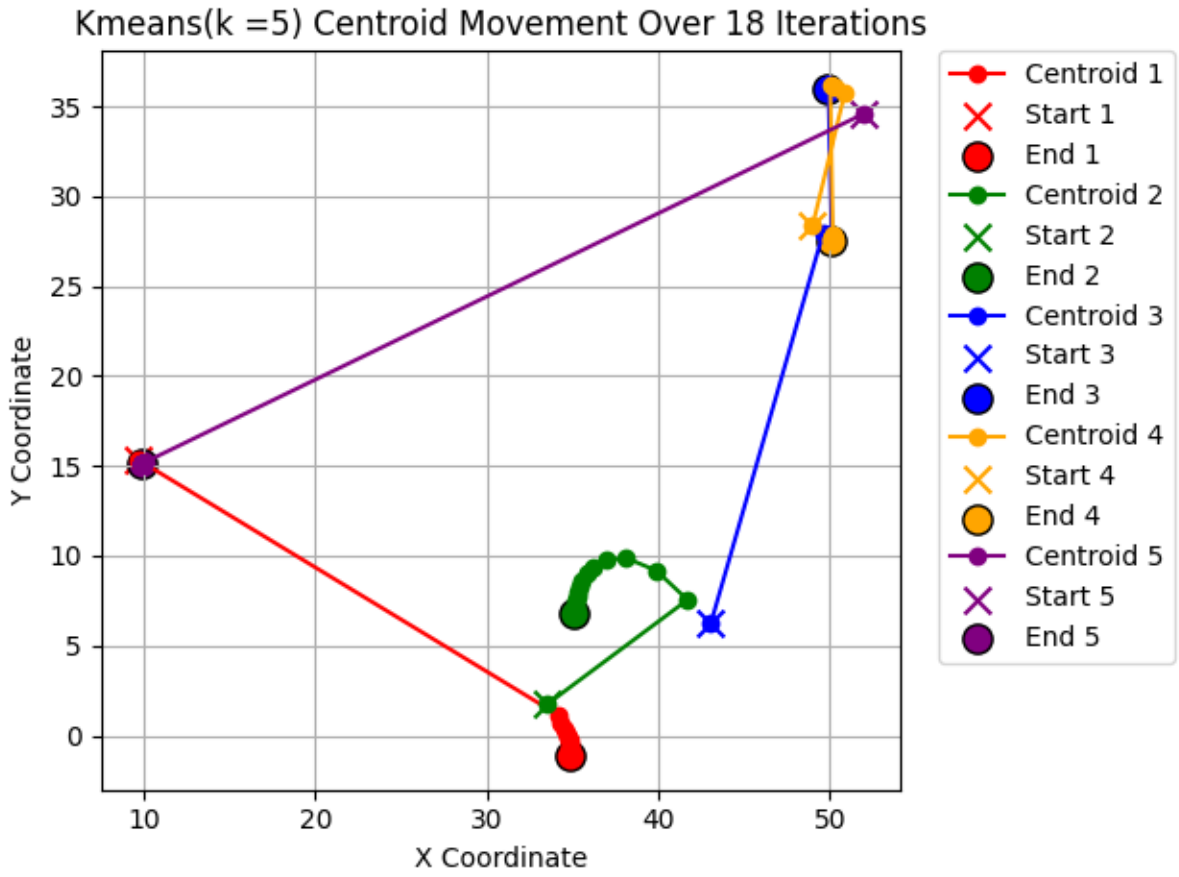
Detail in /Kmean/kmeans_k_5_terminal_report.txt and /Kmeans/output_k_5/

Final centroid: 1: 34.88767058806049, -1.1492691361599863
Final centroid: 2: 35.10524898166031,6.810348684944596
Final centroid: 3: 49.893531834503435,35.93469249329149
Final centroid: 4: 50.12782925362032,27.58707497118695
Final centroid: 5: 9.899168246470962,15.10669211914742

Scatter plot:



Centroid Movement

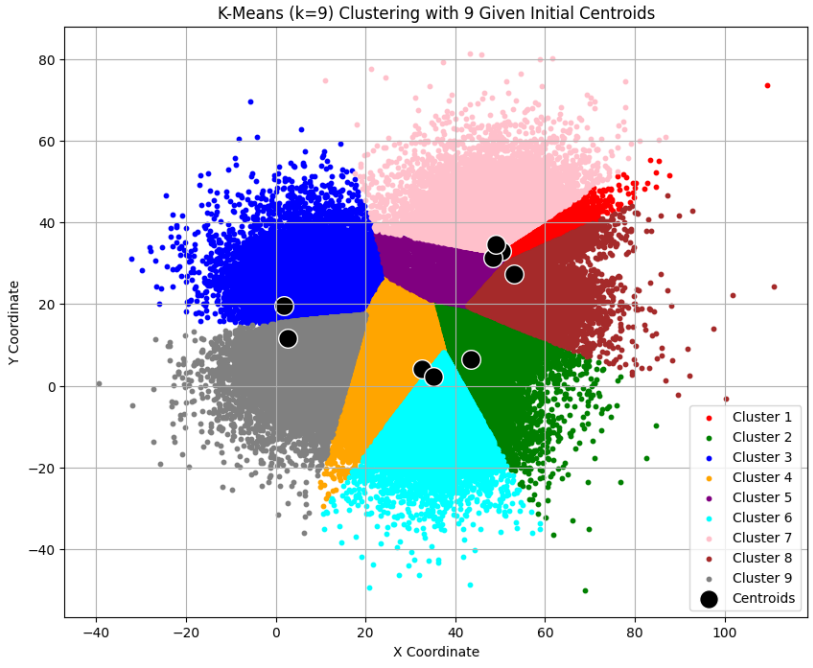


For k = 9
Initialize centroid

```
            IntWritable.class);
        final IntWritable value = new IntWritable(0);
        centerWriter.append(new Text("50.197031637442876,32.94048164287042"), value);
        centerWriter.append(new Text("43.407412339767056,6.541037020010927"), value);
        centerWriter.append(new Text("1.7885358732482017,19.666057053079573"), value);
        centerWriter.append(new Text("32.6358540480337,4.03843047564191"), value);
        centerWriter.append(new Text("48.41919054694239,31.23767287880673"), value);
        centerWriter.append(new Text("35.16296605521855,2.260927131938996"), value);
        centerWriter.append(new Text("49.04816562011978,34.599014498332885"), value);
        centerWriter.append(new Text("53.12345979800841,27.301827115259748"), value);
        centerWriter.append(new Text("2.6203753823708777,11.657150534894178"), value);
        centerWriter.close();
```

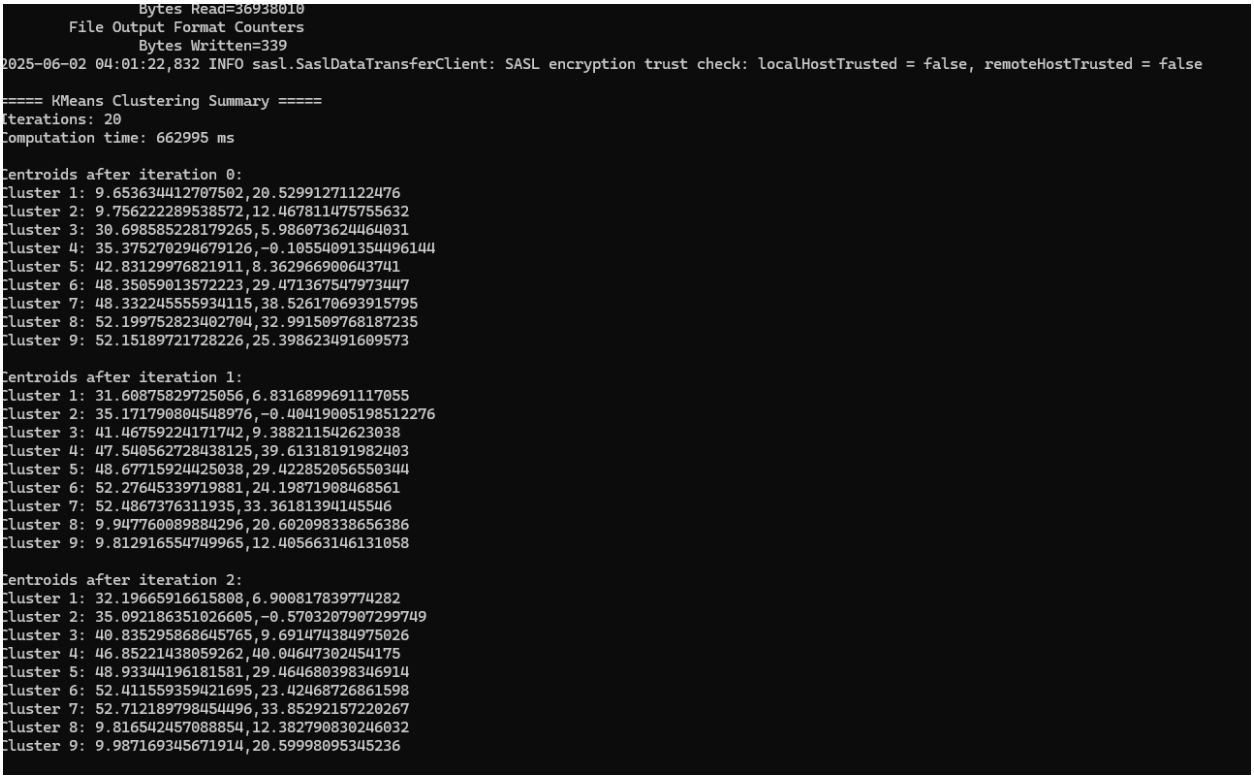
Centroid 1: 50.197031637442876,32.94048164287042
Centroid 2: 43.407412339767056,6.541037020010927
Centroid 3: 1.7885358732482017,19.666057053079573
Centroid 4: 32.6358540480337,4.03843047564191
Centroid 5: 48.41919054694239,31.23767287880673
Centroid 6: 50.197031637442876,32.94048164287042
Centroid 7: 43.407412339767056,6.541037020010927
Centroid 8: 1.7885358732482017,19.666057053079573
Centroid 9: 32.6358540480337,4.03843047564191

Scatter plot:



Running result:

It does not converge, with Computation time 662995ms



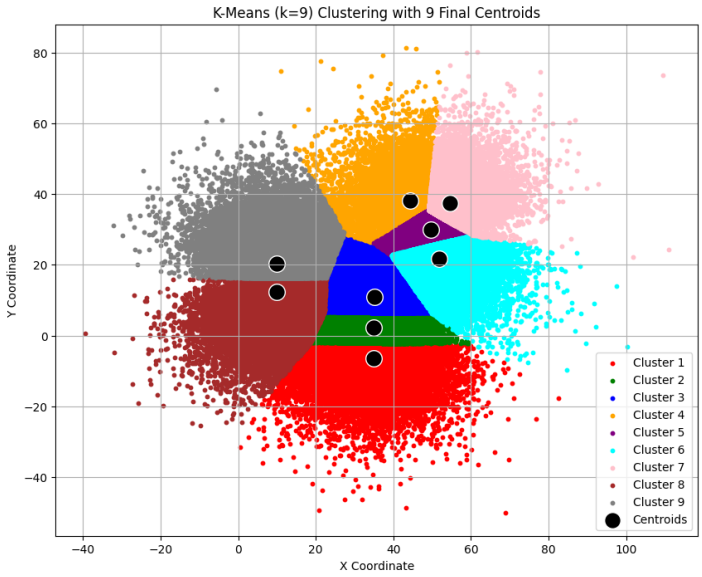
```
                Bytes Read=36938010
        File Output Format Counters
                Bytes Written=339
2025-06-02 04:01:22,832 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false

===== KMeans Clustering Summary =====
Iterations: 20
Computation time: 662995 ms

Centroids after iteration 0:
Cluster 1: 9.653634412707502,20.52991271122476
Cluster 2: 9.756222289538572,12.467811475755632
Cluster 3: 30.698585228179265,5.9860736244664031
Cluster 4: 35.375270294679126,-0.10554091354496144
Cluster 5: 42.83129976821911,8.362966900643741
Cluster 6: 48.35059013572223,29.471367547973447
Cluster 7: 48.332245555934115,38.526170693915795
Cluster 8: 52.199752823402704,32.991509768187235
Cluster 9: 52.15189721728226,25.398623491609573

Centroids after iteration 1:
Cluster 1: 31.60875829725056,6.8316899691117055
Cluster 2: 35.1717908045489 76,-0.40419005198512276
Cluster 3: 41.467592241717 42,9.388211542623038
Cluster 4: 47.540562728438125,39.61318191982403
Cluster 5: 48.67715924425038,29.422852056550344
Cluster 6: 52.27645339719881,24.19871908468561
Cluster 7: 52.4867376311935,33.36181394145546
Cluster 8: 9.947760089884296,20.602098338656386
Cluster 9: 9.8129165547 49965,12.405663146131058

Centroids after iteration 2:
Cluster 1: 32.19665916615808,6.900817839774282
Cluster 2: 35.092186351026605,-0.5703207907299749
Cluster 3: 40.835295868645765,9.691474384975026
Cluster 4: 46.85221438059262,40.046473024541 75
Cluster 5: 48.93344196181581,29.464680398346914
Cluster 6: 52.411559359421695,23.42468726861598
Cluster 7: 52.712189798454496,33.85292157220267
Cluster 8: 9.8165424570888 54,12.382790830246032
Cluster 9: 9.9871693456719 14,20.59998095345236
```

```
Centroids after iteration 17:
Cluster 1: 34.88331688897052,2.302798105858331
Cluster 2: 34.91924888504106,-6.086880423238289
Cluster 3: 35.38410834309288,11.340334063806068
Cluster 4: 44.20988544383486,38.278177999745296
Cluster 5: 49.68311335311762,29.82591390857365
Cluster 6: 51.983542964592324,21.7751041692782 44
Cluster 7: 54.4624926255337,37.320770494347215
Cluster 8: 9.896714686070103,12.250354132468066
Cluster 9: 9.916582725697992,20.356424607481628

Centroids after iteration 18:
Cluster 1: 34.9144156125279,2.2281351481683123
Cluster 2: 34.88627117306026,-6.223799798833676
Cluster 3: 35.198331487595695,11.13223430694099
Cluster 4: 44.21471650683918,38.24555440664027
Cluster 5: 49.693210915939005,29.82921870606926
Cluster 6: 51.9042918476091,21.733270267312914
Cluster 7: 54.486447572044 66,37.349002212417815
Cluster 8: 9.89410691569239,12.248985477241334
Cluster 9: 9.917317599038586,20.352304176779285

Centroids after iteration 19:
Cluster 1: 34.85081483358471,-6.336170885009358
Cluster 2: 34.94276216123986,2.166430378914925
Cluster 3: 35.05506233421857,10.962186968926042
Cluster 4: 44.21759682100193,38.21807344276957
Cluster 5: 49.704128474225236,29.831959466835674
Cluster 6: 51.83387643527099,21.69539513831765
Cluster 7: 54.5045048458008 74,37.372954531930674
Cluster 8: 9.89226253528274,12.248039419094328
Cluster 9: 9.917167634145912,20.34993871969172

Final centroid: 1: 34.85081483358471,-6.336170885009358

Final centroid: 2: 34.94276216123986,2.166430378914925

Final centroid: 3: 35.05506233421857,10.962186968926042

Final centroid: 4: 44.21759682100193,38.21807344276957

Final centroid: 5: 49.704128474225236,29.831959466835674

Final centroid: 6: 51.83387643527099,21.69539513831765

Final centroid: 7: 54.5045048458008 74,37.372954531930674

Final centroid: 8: 9.89226253528274,12.248039419094328

Final centroid: 9: 9.917167634145912,20.34993871969172
```

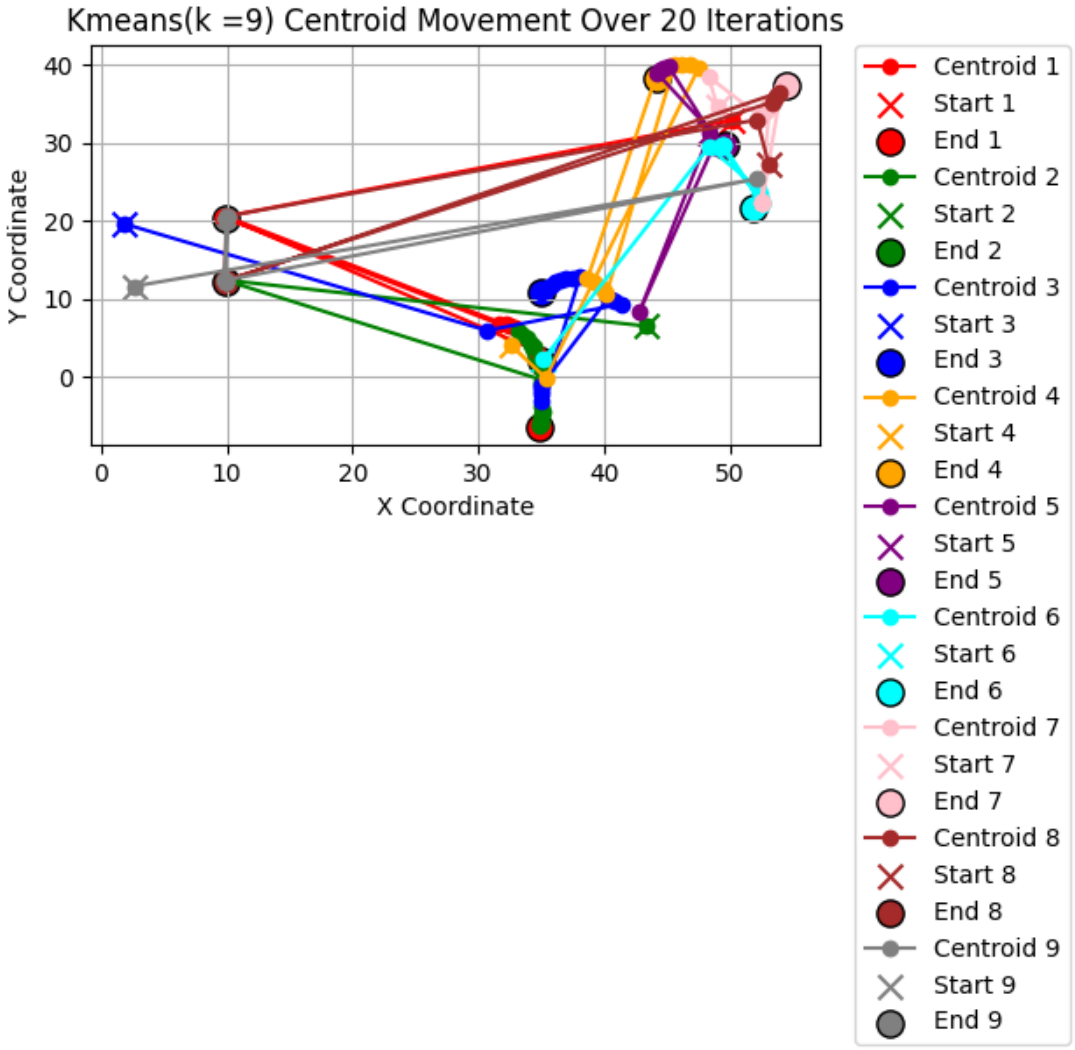
Detail in /Kmean/kmeans_k_9_terminal_report.txt and /Kmeans/output_k_9/

Final centroid: 1: 34.85081483358471,-6.336170885009358
Final centroid: 2: 34.94276216123986,2.166430378914925
Final centroid: 3: 35.05506233421857,10.962186968926042
Final centroid: 4: 44.21759682100193,38.21807344276957
Final centroid: 5: 49.704128474225236,29.831959466835674
Final centroid: 6: 51.83387643527099,21.69539513831765
Final centroid: 7: 54.504504845800874,37.372954531930674
Final centroid: 8: 9.89226253528274,12.248039419094328
Final centroid: 9: 9.917167634145912,20.34993871969172

Scatter plot:



Centroid Movement



Analysis:
With a convergence threshold of 0.1, the algorithm:

- k =5: Converged in 18 iterations taking 565,166 ms.
- k = 9: Did not converge within 20 iterations, taking 662,995 ms.

From the scatter plot for centroid before and after K-Means algorithm, both k=5 and k=9 clustering show that K-Means successfully separates the data into distinct regions after centroid updates. With K=9, the clusters are smaller and more precise, while K=5 results in broader and fewer groupings. This shows that the increase of k can capture finer structure in the data distribution.

With the centroid movement graph, it is revealed that for both k=5 and k=9 cases, centroids converge over iterations, with most centroids stabilizing after significant initial or second shifts. The k=9 clustering exhibits more complex centroid trajectories, which indicates a better partitioning of the data space compared to K=5. Increasing k leads to more localized and specific cluster assignments but also result in the complexity to centroid movement.

5. (5 marks) Critically evaluate the performance of your k-means implementation. Discuss the impact of data distribution and the choice of distance metric (Euclidean, Manhattan, etc.) on the algorithms performance and convergence. Analyze the scalability of your implementation – how does runtime change if you increase the dataset size? (Describe in words)

Performance Convergence Behaviour
With a convergence threshold of 0.1, the algorithm:

- Converged in 18 iterations when k = 5, taking 565,166 ms.
- Did not converge within 20 iterations when k = 9, taking 662,995 ms.

The increased number of clusters k would increase the computation time during each iteration and slow down the convergence due to the increase in the computation complexity of centroid re-assignment.
the choice of distance metric – Euclidean distance

- Works well for dense, spherical clusters with data geometry and suits our way of calculating centroid using mean
- More sensitive to outlier because of the square root of distance
- Manhattan distance would perform better for high dimensional or grid-like data, and need to update rule of updating centroid and other calculations method.

Impact of data distribution:

- If all initial clusters(centroids) are well-separated and spherical, the algorithm converges quickly while if data contains overlapping clusters, it will slow down the convergence and less stability of the centroid

Scalability

- Our implementation scales horizontally due to Hadoop's distributed processing and adding more computation node would help decrease the computation overhead from.
- Runtime would roughly increase linearly with increase number of data points and number of iterations

Part 3: Canopy Clustering and Optimization (15 marks)

Read the provided paper, research as needed and then answer the below questions in words.

6. (5 marks) Explain the advantages and disadvantages of using k-means clustering with MapReduce. Discuss the trade-offs between parallelization, communication overhead, and the inherent limitations of the k-means algorithm itself. (Describe in words)

Advantages of K-Means Clustering with MapReduce

- With the distributed storage, using MapReduce is efficient to handle large scale of data that could be fit in memory of single machine
- MapReduce naturally distributes data and computation across multiple machines parallelly. In the case of K-Means Clustering, each mapper works independently to assign points to centroids, and reducers update centroids in parallel.
- The MapReduce framework would use Hadoop's build-in built-in fault tolerance and automatically retrying failed tasks

Disadvantages of K-Means Clustering with MapReduce

- High communication overhead. At each iteration of assign and updating the centroids must be written to and read from HDFS, it would create huge communication overhead due to data loading, writing and shuffling,
- Each iteration needs to start a new MapReduce job, for smaller datasets, the overhead of using MapReduce jobs would outperform the benefits of parallelization
- Limitation of K-Means Algorithm remains, as the entire efficiency is sensitive to the initiation of centroids If all initial clusters(centroids) are well-separated and spherical, the algorithm converges quickly while if data contains overlapping clusters, it will slow down the convergence and less stability of the centroid

Trade-off

- Parallelization: the parallelization would speed up the processing speed. However, for larger dataset, the benefit of speed up is more important than the overhead caused by the communication. However, for smaller dataset, when deploying parallelization would not bring huge benefits, the communication overhead would become the main issue.
- Using MapReduce is beneficial that does not need the in-memory computation, but it will have an overhead of start job multiple times, once at each iteration.

7. (5 marks) How do you implement Canopy Clustering as a pre-processing step for k-means. Justify your choice of distance metrics for the canopy and k-means stages. Explain how your implementation reduces the number of distance comparisons in the subsequent k-means phase. Clearly explain the parameters used for Canopy Clustering and their impact on the results. (Describe in words)

How to integrate:

- For Canopy Clustering, it is a two-stage clustering technique that include the stage of canopy formation and k-means within canopies, and the second stage is same as our current implementation so only canopy formation step needs to be modified.
- Initialize the with two threshold value T1 as the loose distance threshold and T2 as the tight distance threshold and make sure T1>T2. All input data points are placed into a list called RemainingPoints.
- The algorithm proceeds by repeatedly selecting a point from RemainingPoints as a new canopy center. Once a point is chosen, the algorithm computes the distance(using distance metric) from this center to every other point remaining in the dataset.
- If the distance between the current canopy center and another point is less thanT1, that point is added to the current canopy. If the distance is also less than T2, the point is removed from RemainingPoints to avoid being selected as a canopy center in the future. Repeat this process until all points have either been added to one or more canopies or removed from the list. The resulting canopies are overlapping groups of points that can be used to seed or localize the K-Means clustering process.
- For the second stage of K-means stage, we could continue to use our current design of K-means. Initialize the init centroid from centre point of each canopy and apply K-means only to data point of same canopy and use distance metric to clustering.

Distance Metric:

- Canopy Stage: standard TFIDF cosine-similarity (mentioned in paper) as a cheap distance metric could be deployed as it can significantly decrease the computation as an approximation of distance.
- K-Means Stage: Eculidian distance is used as it can minimize the L2 norm and works well for dense, spherical clusters with 2-D data geometry.

How implementation reduces the number of distances comparison in K-Means stage:

- It can reduce the complexity of computation by filtering the points that are likely to belong together with a relative ‘cheap’ Distance Metric
- In the K-Means stage, each point is only compared to the centroids within its canopy. In contrast, our K-means implementation compares every point to all k centroids in every iteration. This can significantly reduce the computation complexity.

Parameter Impact (T1, T2)

T1 (loose threshold): Determines inclusion in a canopy.

- Larger T1 →fewer, larger canopy → more overlap, higher coverage.
- Smaller T1→ more, smaller canopy → more computation cost, less overlap

T2 (tight threshold): Controls when points stop becoming new centers.

- Larger T2 → few canopy centers → less computation cost but potential poor init centroid
- Smaller T2 → more canopy centers → more computation cost but potential better init centroid

8. (5 marks) How do you integrate Canopy Clustering into your MapReduce-based k-means algorithm. Compare the performance (runtime and cluster quality) of k-means with and without Canopy Clustering as a pre-processing step. (Describe in words)

How to integrate:

- We can use the Canopy Clustering as preprocessing step that runs before launching the K-means jobs, preprocessing with Canopy Clustering with input dataset. Each point in the dataset is compared to existing canopies using a cheap distance metric and grouped by threshold T1 and t2
- For our Mapper part, we are treating each canopy as a separate subset that is processed independently. For each canopy, we would only compare the points within their canopy.
- For reducer, at each node, only data belongs to single canopy will be received and we only update centroid within that canopy.

Compare performance to regular k-means:

- Runtime Performance: Each point is compared to existing canopies using a cheap distance. K-means of Euclidean distance comparison is run on smaller subsets (canopies) Meanwhile, we would initialize centroids with centre point of each canopy as a better initial centroid. This would result in a faster convergence time and a better clustering. They would significantly reduce the computation complexity and computation time. Thus, the runtime performance would improve due to reduced comparisons, and cluster quality would improve because of better initial centroid selection.