Ujjwal Pandit, Hari Prasath KP
Project 2
CS 585: Big Data Management
Advanced Hadoop

Project Documentation

1- Revisiting LinkBook Network with Pig [16 Points]

Task a

Report the frequency of each education level (use HighestEdu as indication) on LinkBook.

Assumptions/ Understading

In this task, we need to report the frequency of each education level (i.e., HighestEdu) on LinkBook. The column representing the highest education level in the data is named degree in the Pig script, and its position is the fifth field (index 4) in the CSV file. The goal is to count the occurrences of each unique education level such as BE, MS, PhD, etc.

In the Hadoop MapReduce implementation, we used a mapper to map each education level to the number 1 and a reducer to sum up these counts. In Apache Pig, however, this process can be simplified as Pig provides higher-level constructs like GROUP BY and COUNT..

PIG Command

(base) ujjwal@orange:~/IdeaProjects/Task A$ pig -x mapreduce taskA.pig

Output

File contents

```
BE,6
MS,6
PhD,7
```

Task b

Find the 10 most popular LinkBook pages, namely, those that got the most accesses based on the AccessLog among all pages. Return Id, NickName, and Occupation.

Assumptions/ Understading

The task is to find the 10 most popular LinkBook pages based on the number of accesses recorded in the AccessLog. We need to count the occurrences of each page being accessed, sort the pages in descending order of access frequency, and retrieve the top 10 most accessed pages. After identifying the top 10 pages, we will join the results with the LinkBookPage data to return the Id, NickName, and Occupation of the corresponding pages.

In Hadoop MapReduce, this was achieved by using two mappers and two reducers to count page accesses and perform the join operation with the LinkBookPage data. In Pig, the process is simplified by leveraging built-in functions like GROUP, COUNT, ORDER, LIMIT, and JOIN.

PIG Command

(base) ujjwal@orange:~/IdeaProjects/Task B$ pig -x mapreduce taskB.pig

Output

Task c

Report all LinkBookPage users (NickName, and Occupation) whose highest education (HighestEdu) is the same as your own highest education (pick one). Note that the highest education in the data file may be random sequence of characters unless you work with meaningful strings like “Undergraduate” or “Graduate”. This is up to you.

Assumptions/ Understading

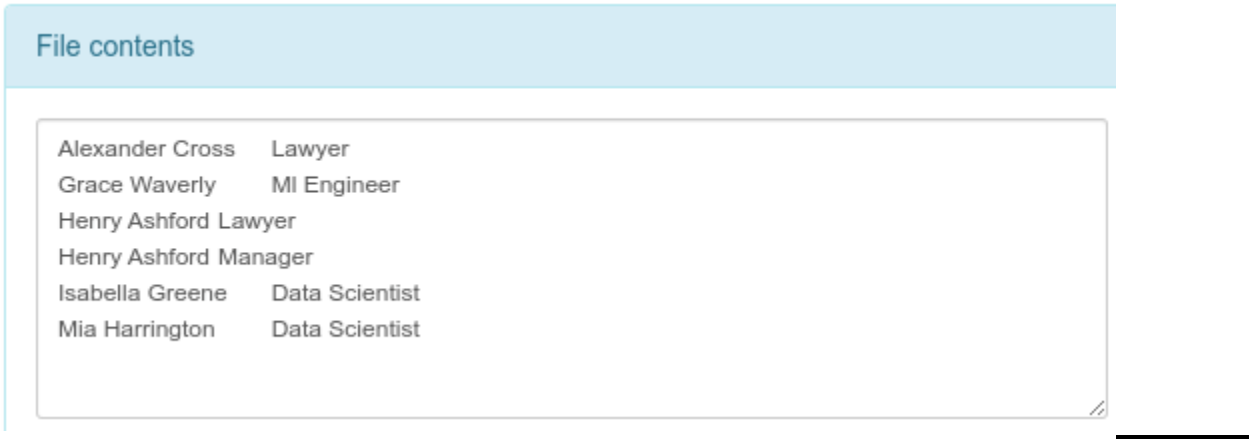
In this task, we need to report all LinkBookPage users whose HighestEdu (highest education) matches a specific degree (in this case, "MS"). The task involves loading the LinkBookPage data, filtering users based on their highest education level, and returning their NickName and Occupation if their degree matches "MS."

In Hadoop MapReduce, this was done by processing the input in the mapper phase to filter records and emit matching user details. In Pig, this can be handled more efficiently using FILTER and FOREACH commands.

PIG Command

(base) ujjwal@orange:~/IdeaProjects/Task C$ pig -x mapreduce taskC.pig

Output



Task d

For each LinkBookPage, compute the “happiness factor” of its owner. That is, for each LinkBookPage, report the owner’s nickname, and the number of relationships they have. For page owners that aren't listed in Associates, return a score of zero. Please note that we maintain a symmetric relationship, take that into account in your calculations.

Assumptions/ Understading

This task involves calculating the "happiness factor" for each LinkBookPage owner. The happiness factor is defined as the number of relationships an owner has, where a relationship is defined based on the Associates file (which contains symmetric relationships between users). If a user is not listed in the Associates file, they get a happiness factor of zero. The result includes the user’s nickname and their corresponding happiness factor.

In the previous Hadoop MapReduce implementation, two mapper phases processed the LinkBookPage and Associates files, and the reducer calculated the happiness factor for each user. In Pig, this can be achieved by loading the two datasets, calculating the relationships for each user, and then joining the data to associate the nickname with the happiness factor.

PIG Command

(base) ujjwal@orange:~/IdeaProjects/Task D$ pig -x mapreduce taskD.pig

Output

File contents

```
Grace Waverly     5
Benjamin Frost    2
Mia Harrington    4
Henry Ashford 1
Olivia Sterling 1
Isabella Greene   3
Noah Whitaker     1
Henry Ashford 2
```

Task e

Determine which people have favorites. That is, for each LinkBookPage owner, determine how many total accesses to LinkBookPage they have made (as reported in the AccessLog) and how many distinct LinkBookPage they have accessed in total. As for the identifier of each LinkBookPage owner, you don't have to report name. IDs are enough.

Assumptions/ Understading

This task focuses on determining how many total pages and how many distinct pages each user (identified by userId) has accessed in the LinkBookPage system. The data for this task comes from the AccessLog, and the results will show the userId, the total number of pages accessed, and the number of distinct pages accessed by each user. The nickname is not required; only the IDs need to be reported.

In the previous Hadoop MapReduce implementation, the mapper phase emitted the user ID and page ID, while the reducer counted the total and distinct page accesses. In Pig, this can be done by grouping the data by userId and then calculating both the total page accesses and distinct page accesses for each user.

PIG Command

(base) ujjwal@orange:~/IdeaProjects/Task E$ pig -x mapreduce taskE.pig

Output

File contents

```
3   1   1
4   1   1
5   2   2
6   2   2
8   4   4
13  2   2
14  1   1
15  1   1
```

Task f

Report all owners of a LinkBookPage who are more popular than an average user,namely, those who have more relationships than the average number of relationships across all owners LinkBookPages.

Assumptions/ Understading

In this task, we aim to identify LinkBookPage owners who are more popular than the average user based on their relationships. To do this, we will load data from both the LinkBookPage and Associates files. We will count the number of relationships each user has and then calculate the average number of relationships across all users. Finally, we will filter and report the names of users whose relationship counts exceed this average.

With Pig, we can efficiently use commands like GROUP, COUNT, and FILTER to get our results. This makes it straightforward to determine the users who are more popular than average and reduces the complexity of the script.

PIG Command

(base) ujjwal@orange:~/IdeaProjects/Task F$ pig -x mapreduce taskF.pig

Output

Output of big data file as the smaller file did not have enough relationships

Task g

Identify "outdated" LinkBookPage. Return IDs and nicknames of persons that have not accessed LinkBook for 90 days (i.e., no entries in the AccessLog in the last 90 days).

Assumptions/ Understading

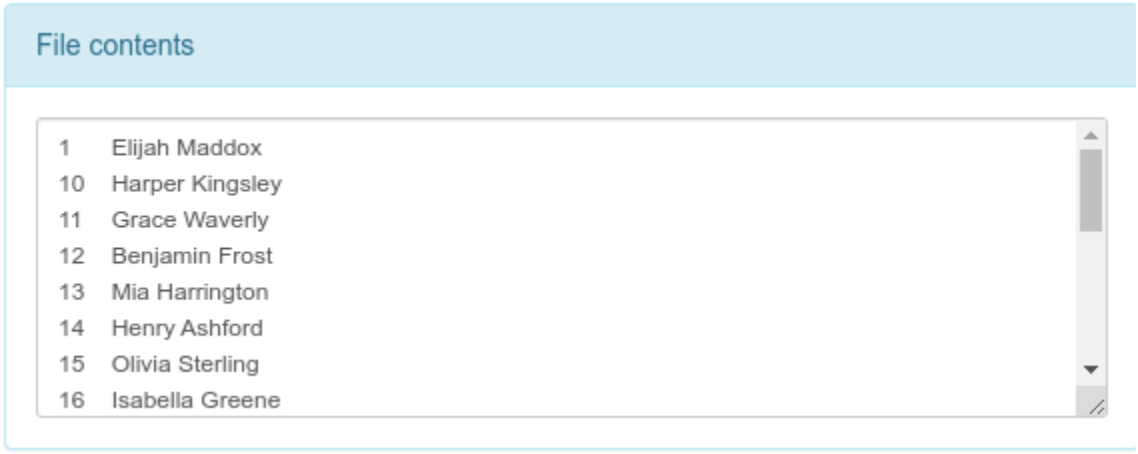
In this task, we aim to identify users who have not accessed the LinkBookPage for 90 days or more by analyzing access logs. We load the AccessLogs and filter out users with access times greater than 129,600 minutes. By grouping these users by their IDs, we extract distinct IDs and then join this data with the LinkBookPage to retrieve their corresponding nicknames. The final output consists of the IDs and nicknames of users who are considered "outdated."

The script uses efficient Pig operations like FILTER, GROUP, and JOIN to process the data.By grouping and getting distinct IDs before joining with LinkBookPage, we reduce the amount of data processed in the join operation.

PIG Command

(base) ujjwal@orange:~/IdeaProjects/Task G$ pig -x mapreduce taskG.pig

Output



Task h

Identify people that have a relationship with someone (Associates); yet never accessed their respective friend's LinkBookPage. Report IDs and nicknames.

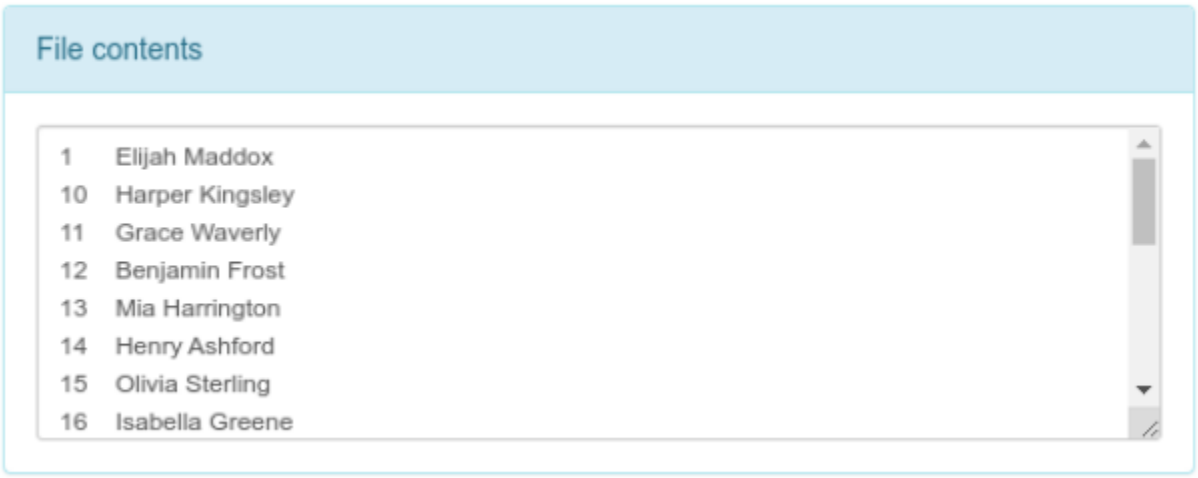
Assumptions/ Understading

This task aims to identify users who have relationships (Associates) but have never accessed their friends' LinkBookPages. We approach this by first joining the Associates data with AccessLogs to determine which users have accessed their friends' pages. Then, we use a LEFT OUTER JOIN to identify users who have associations but are not present in the access logs for their friends' pages. Finally, we join this result with the LinkBookPage data to retrieve the nicknames of these users.
The Pig script efficiently handles the data processing, utilizing joins and filters to isolate the target group of users who have connections but haven't visited their friends' pages. This method simplifies the complex relationship analysis that would be more challenging in a traditional MapReduce approach.

PIG Command

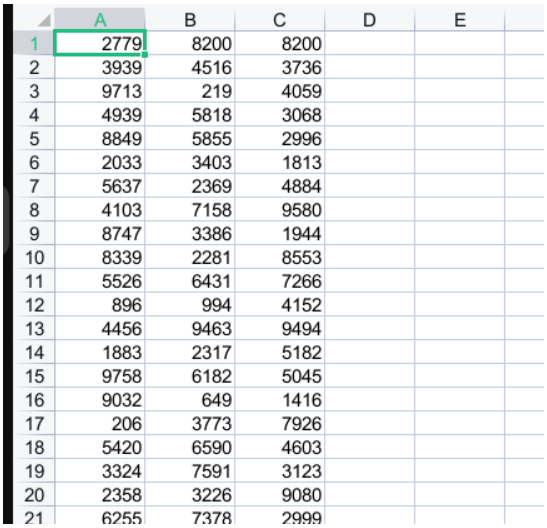
(base) ujjwal@orange:~/IdeaProjects/Task H$ pig -x mapreduce taskH.pig

Output



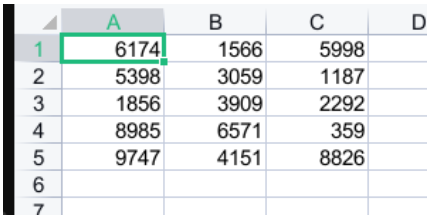
2- K-Means Clustering

We have two files, the main dataset is DataSet.csv and the K seed file is Kseed.csv. Attached is the screenshot of the datasets:



	A	B	C	D	E
1	2779	8200	8200		
2	3939	4516	3736		
3	9713	219	4059		
4	4939	5818	3068		
5	8849	5855	2996		
6	2033	3403	1813		
7	5637	2369	4884		
8	4103	7158	9580		
9	8747	3386	1944		
10	8339	2281	8553		
11	5526	6431	7266		
12	896	994	4152		
13	4456	9463	9494		
14	1883	2317	5182		
15	9758	6182	5045		
16	9032	649	1416		
17	206	3773	7926		
18	5420	6590	4603		
19	3324	7591	3123		
20	2358	3226	9080		
21	6255	7378	2999		

Fig: Dataset.csv



	A	B	C	D
1	6174	1566	5998	
2	5398	3059	1187	
3	1856	3909	2292	
4	8985	6571	359	
5	9747	4151	8826	
6				
7				

Fig: Kseed.csv

Task A: A single-iteration K-means algorithm (R=1) [5 pts]

In single iteration, we just run the K-means algorithm once which might not be our actual centroid because generally it takes multiple iteration for finding the actual centroids.
Clustering_A: This is our main driver class for K-means clustering. It sets up and configures the MapReduce job, specifying input/output paths and mapper/reducer classes.
Clustering_A_Mapper: Mapper reads centroids from Kseed.csv, assigns each input point to the nearest centroid, and emits (centroid_id, point) pairs.
Clustering_A_Reducer: Reducer collects points for each centroid, calculates new centroid positions by averaging, and outputs updated centroids.

JAR Command:
(base) ujjwal@orange:~/IdeaProjects/Clustering_A$ hadoop jar /home/ujjwal/IdeaProjects/Clustering_A/target/Clustering_A-1.0-SNAPSHOT.jar org.ujjwal.Clustering_A /project2/Input_Files/DataSet.csv /project2/Output_Files/Clustering_A_output

Output:

File contents

```
0   4928.090225563909,2596.238345864662,6970.223308270677
1   6121.7829716193655,3069.3856427378964,1825.6560934891486
2   1906.9790476190476,5751.993650793651,4086.3403174603172
3   7707.072625698324,7411.462290502794,2439.722067039106
4   7462.056882821388,6508.41638225256,8046.220705346986
```

Task B: A basic multi-iteration K-means algorithm (the parameter R is set to terminate the process, e.g., R=10). That is no early termination will be done here. [5 pts]

In multiple iterations, we run the K-means algorithm multiple times, which increases the likelihood of finding optimal centroids. With each iteration, we get closer to the true cluster centers. In our case, we set R = 10, but this may not always be sufficient to find the optimal centroids. More iterations might be needed for complex datasets.

Clustering_B: This is our main driver class for K-means clustering. It sets up and configures the MapReduce job, specifying input/output paths and mapper/reducer classes. It runs for a fixed number of iterations (R=10).

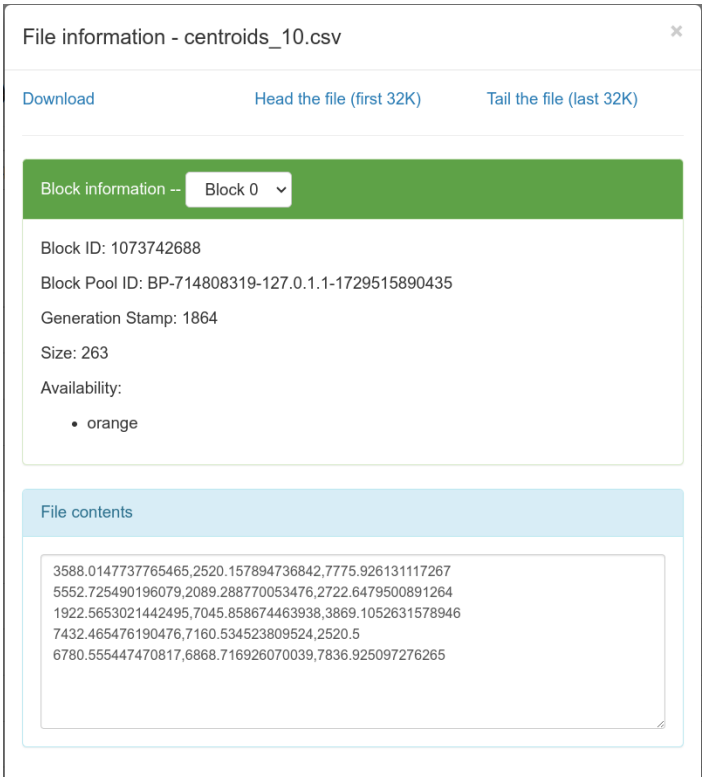
Clustering_B_Mapper: The mapper reads centroids from Kseed.csv, assigns each input point to the nearest centroid, and emits (centroid_id, point) pairs. It calculates distances between points and centroids to determine the nearest centroid for each point.

Clustering_B_Reducer: The reducer collects points for each centroid cluster. It calculates the new centroid positions by averaging all points assigned to that cluster. It then outputs the new centroid for each cluster.

JAR Command:

(base) ujjwal@orange:~/IdeaProjects/Clustering_B$ hadoop jar /home/ujjwal/IdeaProjects/Clustering_B/target/Clustering_B-1.0-SNAPSHOT.jar org.ujjwal.Clustering_B /project2/Input_Files/DataSet.csv /project2/Output_Files/Clustering_B_output 10

Output:



Task C: An additional (advanced) multi-iteration K-means algorithm that terminates potentially earlier if it converges based on some threshold. [6 pts]

In multiple iterations, we run the K-means algorithm multiple times, which increases the likelihood of finding optimal centroids. With each iteration, we get closer to the true cluster centers. In our case, we have set R = 10, but this may not always be sufficient to find the optimal centroids. More iterations might be needed for complex datasets. But here, we have added a termination point so that our algorithm will terminate in case of early finding of centroids. **We got optimal centroids on 13th iteration.**

Clustering_C: This is our main driver class for K-means clustering. It sets up and configures the MapReduce job, specifying input/output paths and mapper/reducer classes. It runs for a fixed number of iterations (R=10).

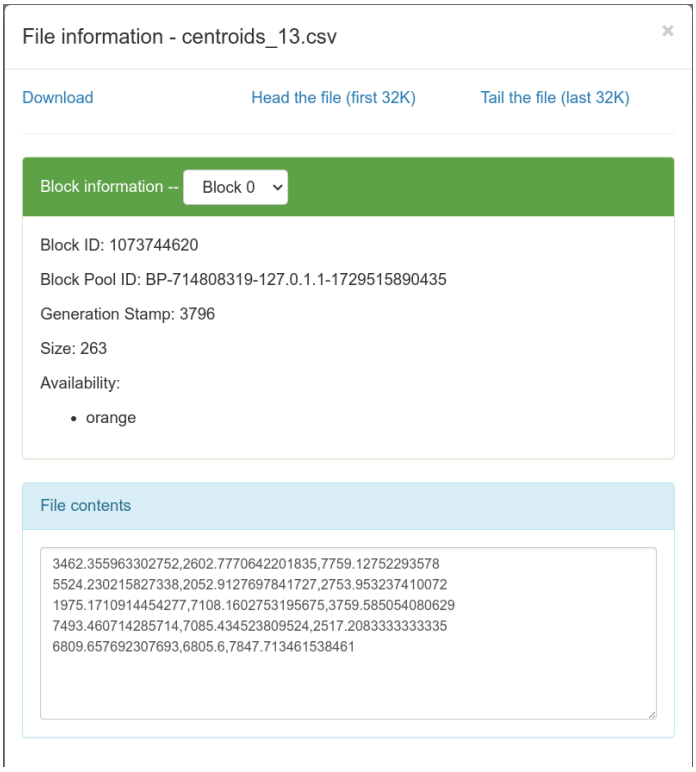
Clustering_C_Mapper: The mapper reads centroids from Kseed.csv, assigns each input point to the nearest centroid, and emits (centroid_id, point) pairs. It calculates distances between points and centroids to determine the nearest centroid for each point.

Clustering_C_Reducer: The reducer collects points for each centroid cluster. It calculates the new centroid positions by averaging all points assigned to that cluster. It then outputs the new centroid for each cluster.

JAR Command:

(base) ujjwal@orange:~/IdeaProjects/Clustering_C$ hadoop jar /home/ujjwal/IdeaProjects/Clustering_C/target/Clustering_C-1.0-SNAPSHOT.jar org.ujjwal.Clustering_C /project2/Input_Files/DataSet.csv /project2/Output_Files/Clustering_C_output 20 0.01

Output:



Task D: An additional (advanced) algorithm that also uses Hadoop Map Reduce optimizations as discussed in class (e.g., a combiner). [6 pts]

Adding a combiner class should enhance the performance of the algorithm, as it reduces the workload for the reducer class. **We got optimal centroids on 13th iteration as before but we were able to do it much faster than without using combiner as in task c.**

Clustering_D: This is our main driver class that manages multiple iterations of K-means clustering, checks for convergence based on a threshold, and handles centroids between iterations, incorporating a Combiner for optimization.

Clustering_D_Mapper: The mapper reads reads current centroids, assigns each input point to the nearest centroid, and emits (centroid_id, point) pairs with an additional count for each iteration.

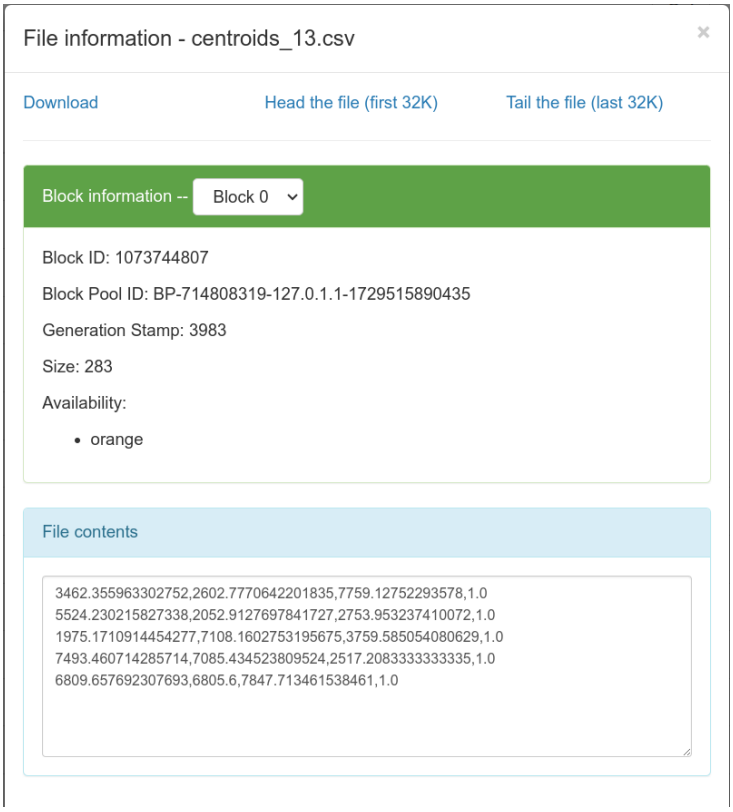
Clustering_D_Combiner: The combiner aggregates points for each centroid within a mapper, partially summing coordinates and counts to reduce data transfer between mappers and reducers.

Clustering_D_Reducer: The reducer collects partially aggregated data for each centroid, calculates final centroid positions by averaging, and outputs updated centroids for the next iteration or final result.

JAR Command:

(base) ujjwal@orange:~/IdeaProjects/Clustering_C$ hadoop jar /home/ujjwal/IdeaProjects/Clustering_D/target/Clustering_D-1.0-SNAPSHOT.jar org.ujjwal.Clustering_D /project2/Input_Files/DataSet.csv /project2/Output_Files/Clustering_D_output 20 0.01

Output:



Task E: For your K-means solution in subproblem (d) above, also extract output.
Consider the two following output variations:
i. return only cluster centers along with an indication if convergence has been reached. [6 pts]
ii. return the final clustered data points along with their cluster centers. [6 pts]

Here we are returning the cluster centers with the message that convergence has been reached after x iterations. We also return another file that has all the clustered data points with their cluster centers. **We got optimal centroids on 13th iteration as before but we were able to do it much faster than without using combiner as in task c.**

Clustering_E: This is our main driver class that manages multiple iterations of K-means clustering, checks for convergence, incorporates a Combiner for optimization, and generates two output variations: cluster centers with convergence indication and final clustered data points with their centers.

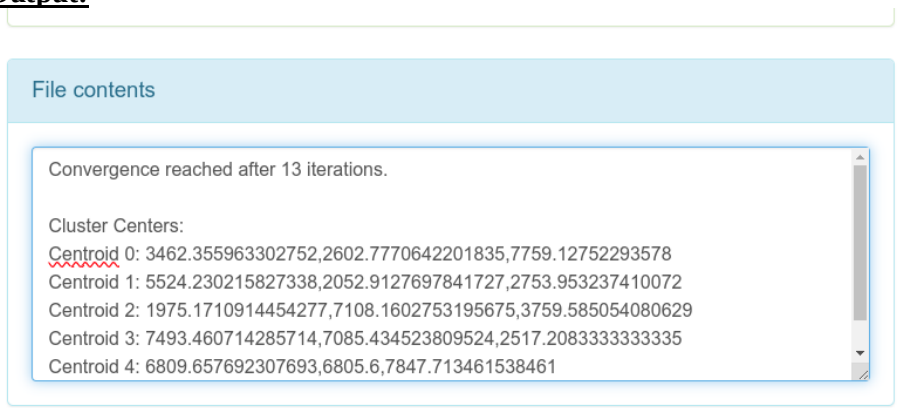
Clustering_E_Mapper: The mapper reads current centroids, assigns each input point to the nearest centroid, and emits (centroid_id, point) pairs. In the final iteration, it emits the original data points instead of partial sums.

Clustering_E_Combiner: The combiner aggregates points for each centroid within a mapper, partially summing coordinates and counts to reduce data transfer between mappers and reducers during intermediate iterations.

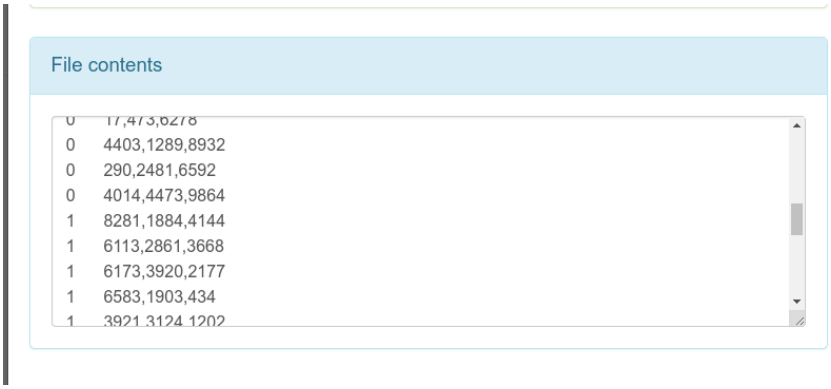
Clustering_E_Reducer: The reducer calculates new centroid positions by averaging in intermediate iterations. In the final iteration, it outputs the clustered data points along with their assigned cluster centers.

JAR Command:
(base) ujjwal@orange:~/IdeaProjects/Clustering_C$ hadoop jar /home/ujjwal/IdeaProjects/Clustering_E/target/Clustering_E-1.0-SNAPSHOT.jar org.ujjwal.Clustering_E /project2/Input_Files/DataSet.csv /project2/Output_Files/Clustering_E_output 20 0.01

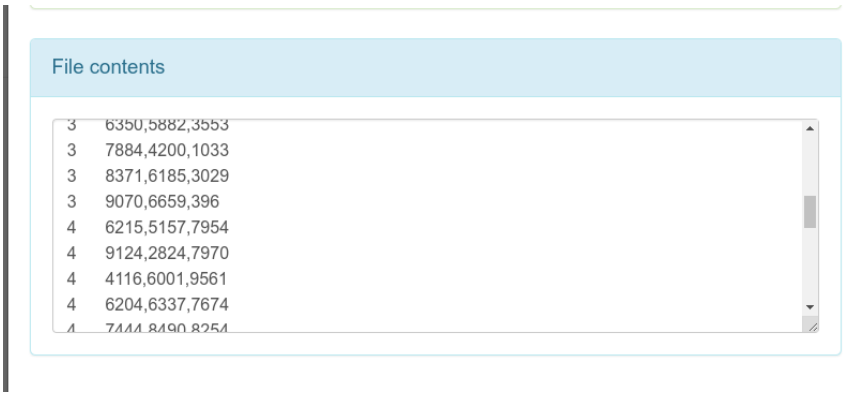
Output:



cluster_centers.txt (returning the cluster centers with the message that convergence has been reached after 13 iterations.)



Head of the final clustering that shows final clustered data points along with its centroids.



Tail of the final clustering that shows final clustered data points along with its centroids.

Task F: Provide a description for each of your above five solutions in your report and conduct experiments over them, for example by choosing different K values (e.g., show how the clusters change for different K values) and different R values (e.g., show how the clusters change over different rounds). describe the relative performance, explain, and analyze your findings. [10 pts]

The choice of k significantly impacts the granularity of the clustering. A larger k (like 10) will create more, smaller clusters, while a smaller k (like 3) will create fewer, larger clusters. The optimal k often depends on the specific dataset and the goals of the analysis. We took k=5 as general. Any value of k will work but it will change the iteration and it also depends upon the threshold.

In **Task A** for single iteration, we just run the K-means algorithm once which might not be our actual centroid because generally it takes multiple iteration for finding the actual centroids.

In **Task B** for multiple iterations, we run the K-means algorithm multiple times, which increases the likelihood of finding optimal centroids. With each iteration, we get closer to the true cluster centers. In our case, we set R = 10, but this may not always be sufficient to find the optimal centroids. More iterations might be needed for complex datasets.

In **Task C** for multiple iterations, we run the K-means algorithm multiple times, which increases the likelihood of finding optimal centroids. With each iteration, we get closer to the true cluster centers. In our case, we have set R = 10, but this may not always be sufficient to find the optimal centroids. More iterations might be needed for complex datasets. But here, we have added a termination point so that our algorithm will terminate in case of early finding of centroids. We got optimal centroids on 13^{th} iteration.

In **Task D**, we added a combiner class to enhance the performance of the algorithm, as it reduces the workload for the reducer class. We got optimal centroids on 13^{th} iteration as before but we were able to do it much faster than without using combiner as in task c.

In **Task E**, we are returning the cluster centers with the message that convergence has been reached after x iterations. We also return another file that has all the clustered data points with their cluster centers. We got optimal centroids on 13^{th} iteration as before but we were able to do it much faster than without using combiner as in task c.

For Task A,
For k=3:

Initializes 3 random data points as means
Assigns all points to the nearest means
Updates means once based on the assignments
Terminates after one iteration
For k=3

File contents

```
0  5539.911549707603,5414.679459064328,7164.1345029239765
1  2404.7147519582245,4804.815274151436,2818.2963446475196
2  7774.95908543923,3854.2888086642597,2046.5920577617328
```

For k=10:
- Initializes 10 random data points as medoids
- Assigns all points to the nearest medoid
- Updates medoids once based on the assignments
- Terminates after one iteration

File contents

```
0  6814.38,4378.142857142857,5334.671428571429
1  1047.9701492537313,3422.0597014925374,8786.824626865671
2  1628.2986425339366,4219.14479638009,1366.0226244343892
3  7501.125786163522,1276.0345911949685,7757.270440251572
4  3467.830188679245,8218.128537735849,2714.1957547169814
5  8484.93265993266,3868.223905723906,2149.6952861952864
6  7520.89039408867,7133.29802955665,7798.4655172413795
7  4107.1353383458645,3300.5131578947367,1972.765037593985
```

For higher K value, eg, K=10, the scenario results in more granular clustering, potentially capturing finer details in the data. For smaller K-value, eg, K=3,4, the scenario resulted into broader and general clusters that captures high level patterns in the data.
As the iteration increases, the convergence rate will decrease and the cluster will become more closer toghether which can be used to find optimal centroids. Compared with single iteration, it does not return optimal centroid, but just returns the centroid, which might or might not be the optimal centroids.

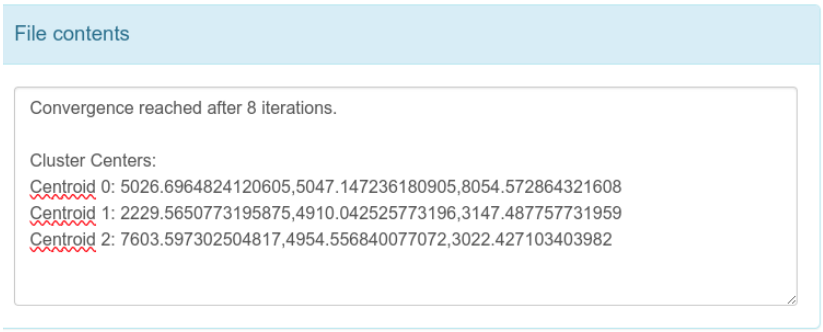
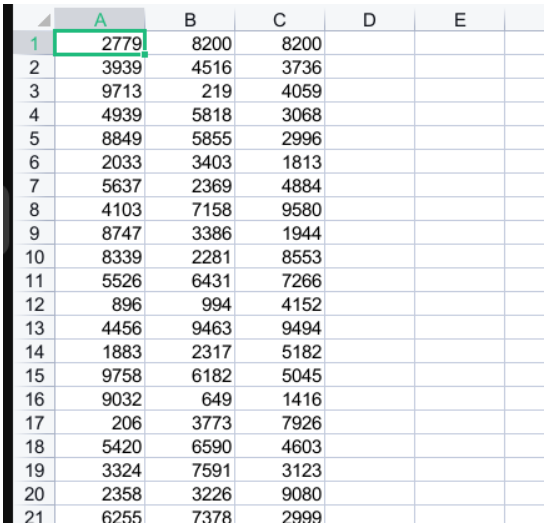


Fig: For K=3, where R=20, and threshold = 0.01

Fig: For K =10, where R =20 and threshold = 0.01

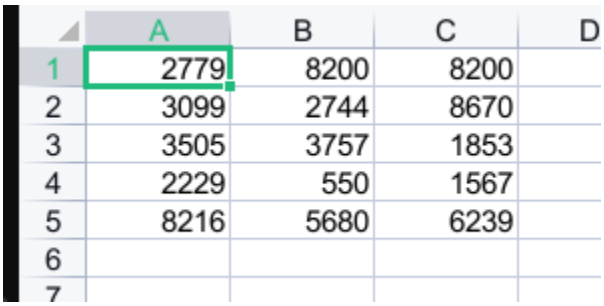
3. **Extension: K-Medoids Implementation**

K-means used random datapoints as initial centroids, but in K-Medoids we use actual datapoints from the dataset as the centroids. We have two files, the main dataset is DataSet.csv and the K seed file is Kmedseed.csv. Attached is the screenshot of the datasets:



	A	B	C	D	E
1	2779	8200	8200		
2	3939	4516	3736		
3	9713	219	4059		
4	4939	5818	3068		
5	8849	5855	2996		
6	2033	3403	1813		
7	5637	2369	4884		
8	4103	7158	9580		
9	8747	3386	1944		
10	8339	2281	8553		
11	5526	6431	7266		
12	896	994	4152		
13	4456	9463	9494		
14	1883	2317	5182		
15	9758	6182	5045		
16	9032	649	1416		
17	206	3773	7926		
18	5420	6590	4603		
19	3324	7591	3123		
20	2358	3226	9080		
21	6255	7378	2999		

Fig: Dataset.csv



	A	B	C	D
1	2779	8200	8200	
2	3099	2744	8670	
3	3505	3757	1853	
4	2229	550	1567	
5	8216	5680	6239	
6				
7				

Fig: Kmedseed.csv (These centroids are the actual datapoints from the Dataset.csv)

Task A: A single-iteration K-medoids algorithm (R=1)

In single iteration, we just run the K-medoids algorithm once which might not be our actual centroid because generally it takes multiple iteration for finding the actual centroids.
Medoids_A: This is our main driver class that sets up and configures the MapReduce job for K-medoids clustering, specifying input/output paths, mapper/reducer classes, and initializing random medoids.

Medoids_A_Mapper: Mapper assigns each input point to the nearest medoid based on a chosen distance metric (in our case Euclidean), emitting (medoid_id, point) pairs for each iteration.

Medoids_A_Reducer: Reducer collects points for each medoid, calculates the total cost for potential medoid swaps, and selects the medoid that minimizes the overall cost within the cluster.

Output:

File contents

```
0    2732.0,7818.0,7283.0
1    3079.0,2512.0,7878.0
2    4257.0,5098.0,2247.0
3    2361.0,1178.0,2321.0
4    7976.0,5970.0,5641.0
```

Task B: A basic multi-iteration K-medoids algorithm (the parameter R is set to terminate the process, e.g., R=10). That is no early termination will be done here.

In multiple iterations, we run the K-medoids algorithm multiple times, which increases the likelihood of finding optimal medoids. With each iteration, we get closer to the true cluster representatives. In our case, we set R = 10, but this may not always be sufficient to find the optimal medoids. More iterations might be needed for complex datasets.

Medoids_B: This is our main driver class for K-medoids clustering. It sets up and configures the MapReduce job, specifying input/output paths and mapper/reducer classes. It runs for a fixed number of iterations (R=10) and manages the selection and updating of medoids between iterations.

Medoids_B_Mapper: The mapper reads current medoids, assigns each input point to the nearest medoid, and emits (medoid_id, point) pairs. It calculates distances between points and medoids to determine the nearest medoid for each point, using a specified distance metric (in our case Eucledian).

Medoids_B_Reducer: The reducer collects points for each medoid cluster. It evaluates potential new medoids within the cluster by calculating the total cost (sum of distances) for each candidate. It then selects the point that minimizes the overall cost within the cluster as the new medoid and outputs it.

Output:

File contents

```
0	3287.0,7392.0,7356.0
1	3496.0,2141.0,7785.0
2	4203.0,7372.0,2277.0
3	5185.0,2141.0,2489.0
4	8076.0,6027.0,6735.0
```

Task C: An additional (advanced) multi-iteration K-medoids algorithm that terminates potentially earlier if it converges based on some threshold. [6 pts]

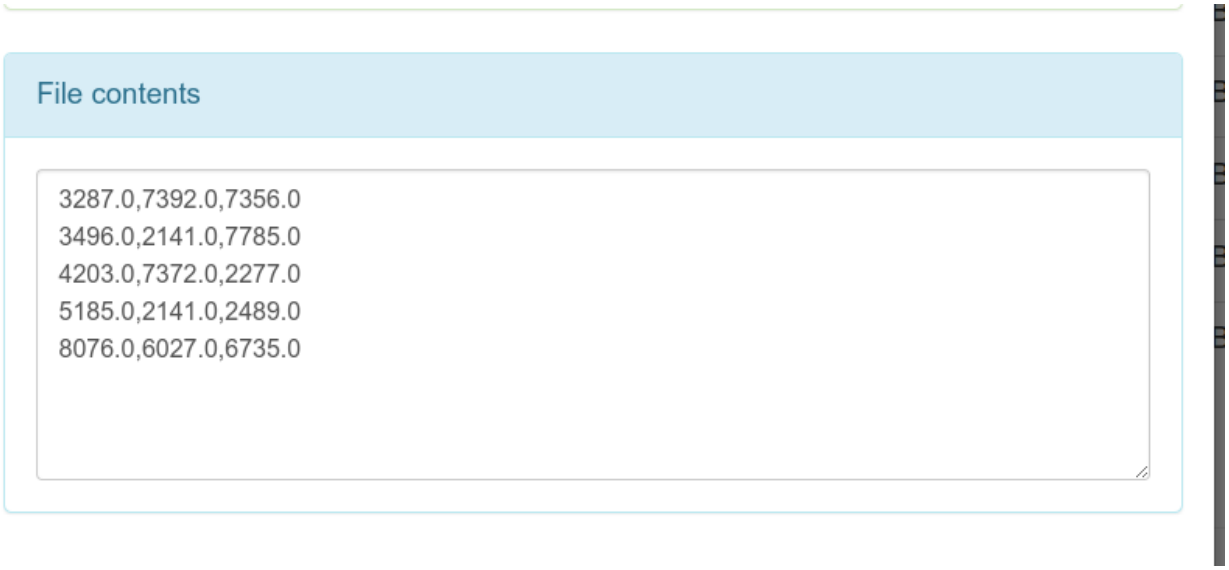
In multiple iterations, we run the K-medoids algorithm multiple times, which increases the likelihood of finding optimal medoids. With each iteration, we get closer to the true cluster representatives. We have set a maximum number of iterations (e.g., R = 10), but we've also added an early termination condition based on a convergence threshold. This allows the algorithm to stop earlier if the medoids stabilize, potentially saving computational resources. **Here convergence reached after 9 iterations.**

Medoids_C: This is our main driver class for K-medoids clustering. It sets up and configures the MapReduce job, specifying input/output paths and mapper/reducer classes. It runs for a maximum number of iterations or until convergence is reached. After each iteration, it checks if the change in medoids is below a specified threshold to determine convergence.

Medoids_C_Mapper: The mapper reads current medoids, assigns each input point to the nearest medoid, and emits (medoid_id, point) pairs. It calculates distances between points and medoids to determine the nearest medoid for each point, using a specified distance metric (in our case Euclidean).

Medoids_C_Reducer: The reducer collects points for each medoid cluster. It evaluates potential new medoids within the cluster by calculating the total cost (sum of distances) for each candidate. It then selects the point that minimizes the overall cost within the cluster as the new medoid and outputs it. The reducer also calculates and outputs the change in medoid position to help determine convergence.

Output:



Task D: An additional (advanced) algorithm that also uses Hadoop Map Reduce optimizations as discussed in class (e.g., a combiner).

Adding a combiner class should enhance the performance of the algorithm, as it reduces the workload for the reducer class. **We achieved optimal medoids on the 9th iteration as before, but we were able to do it much faster than without using a combiner as in task C.**

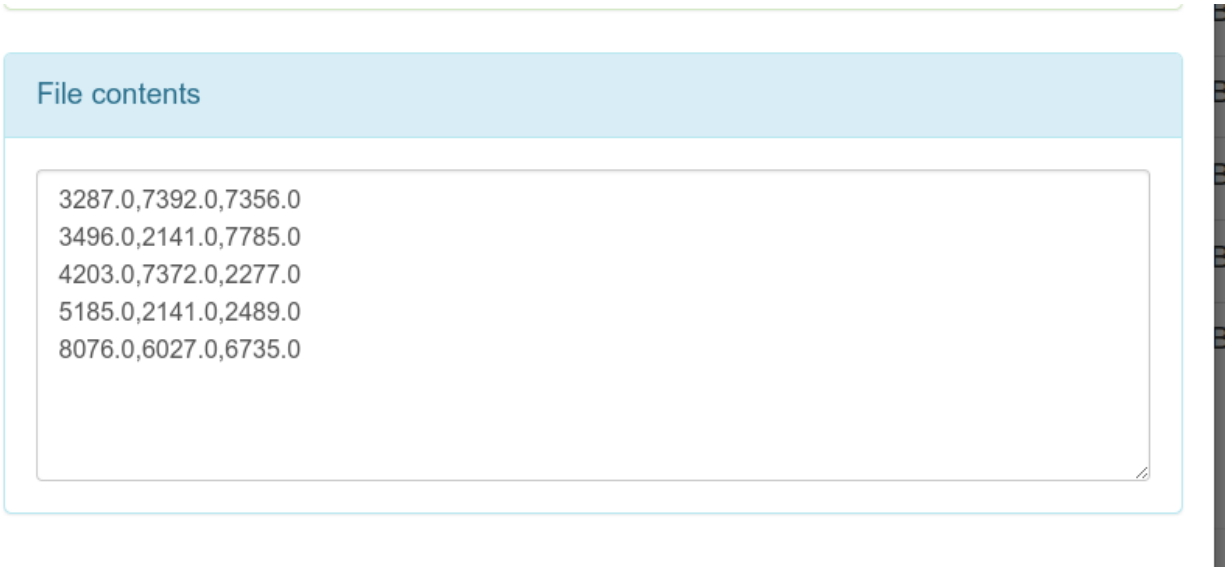
Medoids_D: This is our main driver class that manages multiple iterations of K-medoids clustering, checks for convergence based on a threshold, and handles medoids between iterations, incorporating a Combiner for optimization.

Medoids_D_Mapper: The mapper reads current medoids, assigns each input point to the nearest medoid, and emits (medoid_id, point) pairs. It calculates distances between points and medoids to determine the nearest medoid for each point, using eucledian distance metric.

Medoids_D_Combiner: The combiner partially aggregates points for each medoid within a mapper, calculating partial sums of distances and counts. This reduces data transfer between mappers and reducers, improving overall performance.

Medoids_D_Reducer: The reducer collects partially aggregated data for each medoid cluster, evaluates potential new medoids by calculating the total cost for each candidate, and selects the point that minimizes the overall cost within the cluster as the new medoid. It then outputs the updated medoids for the next iteration or final result.

Output:



Task E: For your K-medoids solution in subproblem (d) above, also extract output.
Consider the two following output variations:
i. return only cluster centers along with an indication if convergence has been reached. [6 pts]
ii. return the final clustered data points along with their cluster centers. [6 pts]

Here we are returning the medoids with the message that convergence has been reached after x iterations. We also return another file that has all the clustered data points with their medoids. **We achieved optimal medoids on the 9th iteration as before.**

Medoids_E: This is our main driver class that manages multiple iterations of K-medoids clustering, checks for convergence, incorporates a Combiner for optimization, and generates two output variations: medoids with convergence indication and final clustered data points with their medoids.
Medoids_E_Mapper: The mapper reads current medoids, assigns each input point to the nearest medoid, and emits (medoid_id, point) pairs. In the final iteration, it emits the original data points instead of partial sums to facilitate the final output.

Medoids_E_Combiner: The combiner partially aggregates points for each medoid within a mapper, calculating partial sums of distances and counts to reduce data transfer between mappers and reducers during intermediate iterations.

Medoids_E_Reducer: The reducer evaluates potential new medoids by calculating the total cost for each candidate in intermediate iterations. In the final iteration, it outputs the clustered data points along with their assigned medoids, as well as the final medoids with convergence information.

Output:

File contents

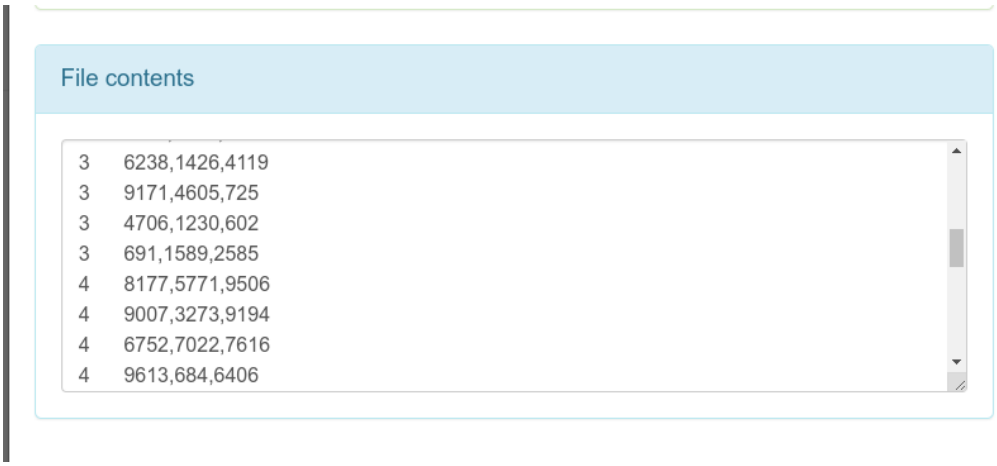
```
Convergence reached after 9 iterations.

Cluster Medoids:
Medoid 0: 3287.0,7392.0,7356.0
Medoid 1: 3496.0,2141.0,7785.0
Medoid 2: 4203.0,7372.0,2277.0
Medoid 3: 5185.0,2141.0,2489.0
Medoid 4: 8076.0,6027.0,6735.0
```

cluster_medoids.txt (returning the cluster centers with the message that convergence has been reached after 9^{th} iterations.)

File contents

```
0	4746,4890,8416
0	5136,6501,6725
0	3516,8731,6399
0	2452,5285,5398
0	365,5092,8025
1	1526,4246,7343
1	5491,3569,5718
1	2699,1402,8502
```

Head of the final clustering that shows final clustered data points along with its centroids.



Tail of the final clustering that shows final clustered data points along with its centroids.

Task F: Provide a description for each of your above five solutions in your report and conduct experiments over them, for example by choosing different K values (e.g., show how the clusters change for different K values) and different R values (e.g., show how the clusters change over different rounds). describe the relative performance, explain, and analyze your findings. [10 pts]

The choice of k significantly impacts the granularity of the clustering in K-medoids. A larger k (like 10) will create more, smaller clusters, while a smaller k (like 3) will create fewer, larger clusters. The optimal k often depends on the specific dataset and the goals of the analysis. We took k=5 as general. Any value of k will work, but it will change the iteration count and also depends upon the threshold.

In **Task A** for single iteration, we just run the K-medoids algorithm once, which might not find our optimal medoids because generally it takes multiple iterations to find the actual representative data points (medoids).

In **Task B** for multiple iterations, we run the K-medoids algorithm multiple times, which increases the likelihood of finding optimal medoids. With each iteration, we get closer to the true cluster representatives. In our case, we set R = 10, but this may not always be sufficient to find the optimal medoids. More iterations might be needed for complex datasets.

In **Task C** for multiple iterations with early termination, we run the K-medoids algorithm multiple times, which increases the likelihood of finding optimal medoids. With each iteration, we get closer to the true cluster representatives. We have added a termination point so that our algorithm will stop in case of early convergence to stable medoids. We found optimal medoids on the 13th iteration.

In **Task D**, we added a combiner class to enhance the performance of the algorithm, as it reduces the workload for the reducer class. We found optimal medoids on the 13th iteration as before, but we were able to do it much faster than without using the combiner as in task C.

In **Task E,** we are returning the cluster medoids with the message that convergence has been reached after x iterations. We also return another file that has all the clustered data points with their cluster medoids. We found optimal medoids on the 13th iteration as before, but we were able to do it much faster than without using the combiner as in task C.

For higher K values, e.g., K=10, the scenario results in more granular clustering, potentially capturing finer details in the data. For smaller K values, e.g., K=3,4, the scenario resulted in broader and more general clusters that capture high-level patterns in the data.
As the iteration count increases, the convergence rate will decrease, and the clusters will become more stable, which can be used to find optimal medoids. Compared with single iteration, it does not return optimal medoids, but just returns the medoids after one round, which might or might not be the optimal representatives.

In K-medoids, unlike K-means, the cluster centers (medoids) are always actual data points, which makes the algorithm more robust to outliers and noise in the dataset.

4. BYOD (Bring Your Own Data):

We got the dataset called College_Data.csv from the Kaggle, and we updated the dataset with only the required columns to use as our data file and named it College_Data_Updated.csv, and we created a College_Kseed.csv as our initial K-seed value.

		Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergr	P.Undergr	Outstate	Room.Boa	Books	Personal	PhD	Terminal	S.F.Ratio	perc.alumr	Expend	Grad.Rate
2	Abilene Ch	Yes	1660	1232	721	23	52	2885	537	7440	3300	450	2200	70	78	18.1	12	7041	60
3	Adelphi Ur	Yes	2186	1924	512	16	29	2683	1227	12280	6450	750	1500	29	30	12.2	16	10527	56
4	Adrian Col	Yes	1428	1097	336	22	50	1036	99	11250	3750	400	1165	53	66	12.9	30	8735	54
5	Agnes Sco	Yes	417	349	137	60	89	510	63	12960	5450	450	875	92	97	7.7	37	19016	59
6	Alaska Pa	Yes	193	146	55	16	44	249	869	7560	4120	800	1500	76	72	11.9	2	10922	15
7	Albertson (	Yes	587	479	158	38	62	678	41	13500	3335	500	675	67	73	9.4	11	9727	55
8	Albertus M	Yes	353	340	103	17	45	416	230	13290	5720	500	1500	90	93	11.5	26	8861	63
9	Albion Coll	Yes	1899	1720	489	37	68	1594	32	13868	4826	450	850	89	100	13.7	37	11487	73
10	Albright Co	Yes	1038	839	227	30	63	973	306	15595	4400	300	500	79	84	11.3	23	11644	80
11	Alderson-B	Yes	582	498	172	21	44	799	78	10468	3380	660	1800	40	41	11.5	15	8991	52
12	Alfred Univ	Yes	1732	1425	472	37	75	1830	110	16548	5406	500	600	82	88	11.3	31	10932	73
13	Allegheny	Yes	2652	1900	484	44	77	1707	44	17080	4440	400	600	73	91	9.9	41	11711	76
14	Allentown	Yes	1179	780	290	38	64	1130	638	9690	4785	600	1000	60	84	13.3	21	7940	74
15	Alma Colle	Yes	1267	1080	385	44	73	1306	28	12572	4552	400	400	79	87	15.3	32	9305	68
16	Alverno Co	Yes	494	313	157	23	46	1317	1235	8352	3640	650	2449	36	69	11.1	26	8127	55
17	American	Yes	1420	1093	220	9	22	1018	287	8700	4780	450	1400	78	84	14.7	19	7355	69

Fig: College_Data.csv

1	1660	1232	721
2	2186	1924	512
3	1428	1097	336
4	417	349	137
5	193	146	55
6	587	479	158
7	353	340	103
8	1899	1720	489
9	1038	839	227
10	582	498	172

Fig: College_Data_Updated.csv (We took Apps, Accept, and Enroll columns from College_Data to create this updated dataset.)

	A	B	C
1	33111.26	20766.49	6084.909
2	35312.91	16630.5	1755.1
3	11579.38	13230.38	4785.914
4	13708.9	17157.53	5155.112
5	30160.91	20126.01	856.6677
6			

Fig: College_Kseed.csv

Task A: A single-iteration K-means algorithm (R=1) [5 pts]

In single iteration, we just run the K-means algorithm once which might not be our actual centroid because generally it takes multiple iteration for finding the actual centroids.
Byod_A: This is our main driver class for K-means clustering. It sets up and configures the MapReduce job, specifying input/output paths and mapper/reducer classes.
Byod_A_Mapper: Mapper reads centroids from Kseed.csv, assigns each input point to the nearest centroid, and emits (centroid_id, point) pairs.
Byod_A_Reducer: Reducer collects points for each centroid, calculates new centroid positions by averaging, and outputs updated centroids.

JAR Command:

(base) ujjwal@orange:~/IdeaProjects/Byod_B$ hadoop jar

/home/ujjwal/IdeaProjects/Byod_A/target/Byod_A-1.0-SNAPSHOT.jar org.ujjwal.Byod_A /project2/Input_Files/College_Data_Updated.csv /project2/Output_Files/Byod_A_output

Output:

File contents

```
0    48094.0,26330.0,4520.0
2    2838.300907911803,1905.6433203631648,745.6407263294423
3    19169.8,14606.0,5326.0
```

Task B: A basic multi-iteration K-means algorithm (the parameter R is set to terminate the process, e.g., R=10). That is no early termination will be done here.

In multiple iterations, we run the K-means algorithm multiple times, which increases the likelihood of finding optimal centroids. With each iteration, we get closer to the true cluster centers. In our case, we set R = 10, but this may not always be sufficient to find the optimal centroids. More iterations might be needed for complex datasets.

Byod_B: This is our main driver class for K-means clustering. It sets up and configures the MapReduce job, specifying input/output paths and mapper/reducer classes. It runs for a fixed number of iterations (R=10).

Byod_B_Mapper: The mapper reads centroids from Kseed.csv, assigns each input point to the nearest centroid, and emits (centroid_id, point) pairs. It calculates distances between points and centroids to determine the nearest centroid for each point.

Byod_B_Reducer: The reducer collects points for each centroid cluster. It calculates the new centroid positions by averaging all points assigned to that cluster. It then outputs the new centroid for each cluster.

JAR Command:
(base) ujjwal@orange:~/IdeaProjects/Byod_B$ hadoop jar /home/ujjwal/IdeaProjects/Byod_B/target/Byod_B-1.0-SNAPSHOT.jar org.ujjwal.Byod_B /project2/Input_Files/College_Data_Updated.csv /project2/Output_Files/Byod_B_output 10

Output:

File contents

```
48094.0,26330.0,4520.0
1730.617469879518,1241.328313253012,487.2620481927711
10134.36607142857,6411.0625,2481.9375
```

Task C: An additional (advanced) multi-iteration K-means algorithm that terminates potentially earlier if it converges based on some threshold. [6 pts]

In multiple iterations, we run the K-means algorithm multiple times, which increases the likelihood of finding optimal centroids. With each iteration, we get closer to the true cluster centers. In our case, we

have set R = 10, but this may not always be sufficient to find the optimal centroids. More iterations might be needed for complex datasets. But here, we have added a termination point so that our algorithm will terminate in case of early finding of centroids. **We got optimal centroids on 8th iteration.**

Byod_C: This is our main driver class for K-means clustering. It sets up and configures the MapReduce job, specifying input/output paths and mapper/reducer classes. It runs for a fixed number of iterations (R=10).

Byod_C_Mapper: The mapper reads centroids from Kseed.csv, assigns each input point to the nearest centroid, and emits (centroid_id, point) pairs. It calculates distances between points and centroids to determine the nearest centroid for each point.

Byod_C_Reducer: The reducer collects points for each centroid cluster. It calculates the new centroid positions by averaging all points assigned to that cluster. It then outputs the new centroid for each cluster.

JAR Command:

(base) ujjwal@orange:~/IdeaProjects/Byod_C$ hadoop jar /home/ujjwal/IdeaProjects/Byod_C/target/Byod_C-1.0-SNAPSHOT.jar org.ujjwal.Byod_C /project2/Input_Files/College_Data_Updated.csv /project2/Output_Files/Byod_C_output 20 0.01

Output:

File contents

```
48094.0,26330.0,4520.0
1736.5157894736842,1245.966917293233,489.12631578947367
10174.738738738739,6429.846846846847,2488.738738738739
```

Task D: An additional (advanced) algorithm that also uses Hadoop Map Reduce optimizations as discussed in class (e.g., a combiner). [6 pts]

Adding a combiner class should enhance the performance of the algorithm, as it reduces the workload for the reducer class. **We got optimal centroids on 13th iteration as before but we were able to do it much faster than without using combiner as in task c.**

Byod_D: This is our main driver class that manages multiple iterations of K-means clustering, checks for convergence based on a threshold, and handles centroids between iterations, incorporating a Combiner for optimization.

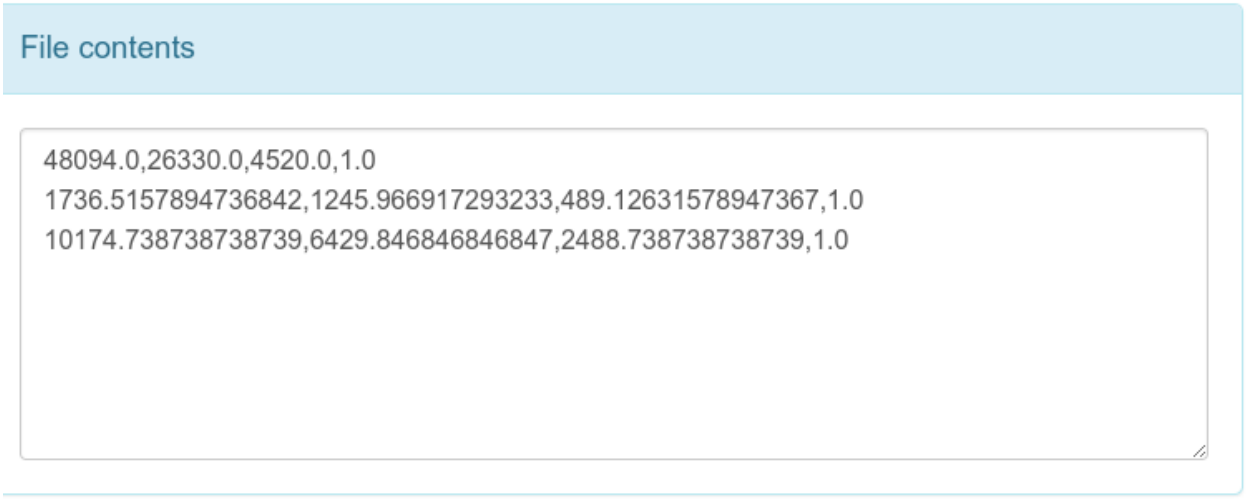
Byod_D_Mapper: The mapper reads reads current centroids, assigns each input point to the nearest centroid, and emits (centroid_id, point) pairs with an additional count for each iteration.

Byod_D_Combiner: The combiner aggregates points for each centroid within a mapper, partially summing coordinates and counts to reduce data transfer between mappers and reducers.

Byod_D_Reducer: The reducer collects partially aggregated data for each centroid, calculates final centroid positions by averaging, and outputs updated centroids for the next iteration or final result.

JAR Command:
(base) ujjwal@orange:~/IdeaProjects/Byod_D$ hadoop jar /home/ujjwal/IdeaProjects/Byod_D/target/Byod_D-1.0-SNAPSHOT.jar org.ujjwal.Byod_D /project2/Input_Files/College_Data_Updated.csv /project2/Output_Files/Byod_D_output 20 0.01

Output:



Task E: For your K-means solution in subproblem (d) above, also extract output.
Consider the two following output variations:
i. return only cluster centers along with an indication if convergence has been reached. [6 pts]
ii. return the final clustered data points along with their cluster centers. [6 pts]

Here we are returning the cluster centers with the message that convergence has been reached after x iterations. We also return another file that has all the clustered data points with their cluster centers.
We got optimal centroids on 8th iteration as before but we were able to do it much faster than without using combiner as in task c.

Byod_E: This is our main driver class that manages multiple iterations of K-means clustering, checks for convergence, incorporates a Combiner for optimization, and generates two output variations: cluster centers with convergence indication and final clustered data points with their centers.

Byod_E_Mapper: The mapper reads current centroids, assigns each input point to the nearest centroid, and emits (centroid_id, point) pairs. In the final iteration, it emits the original data points instead of partial sums.

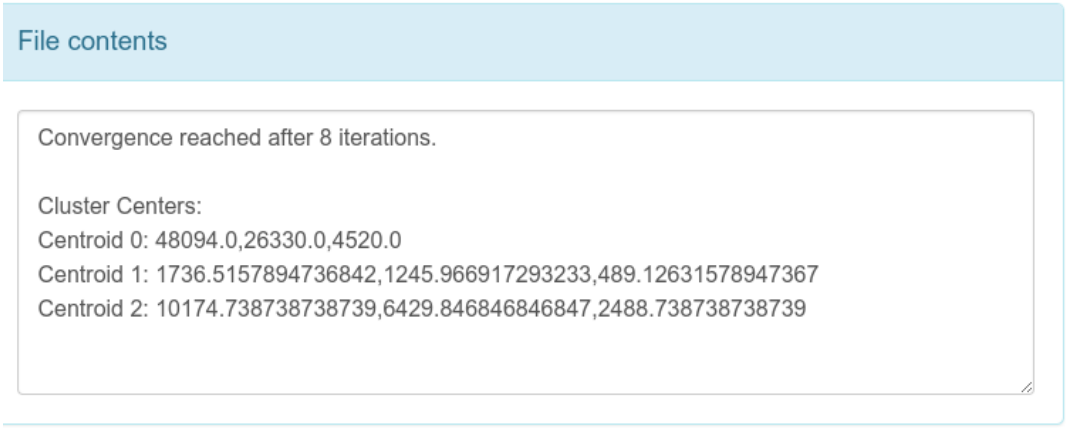
Byod_E_Combiner: The combiner aggregates points for each centroid within a mapper, partially summing coordinates and counts to reduce data transfer between mappers and reducers during intermediate iterations.

Byod_E_Reducer: The reducer calculates new centroid positions by averaging in intermediate iterations. In the final iteration, it outputs the clustered data points along with their assigned cluster centers.

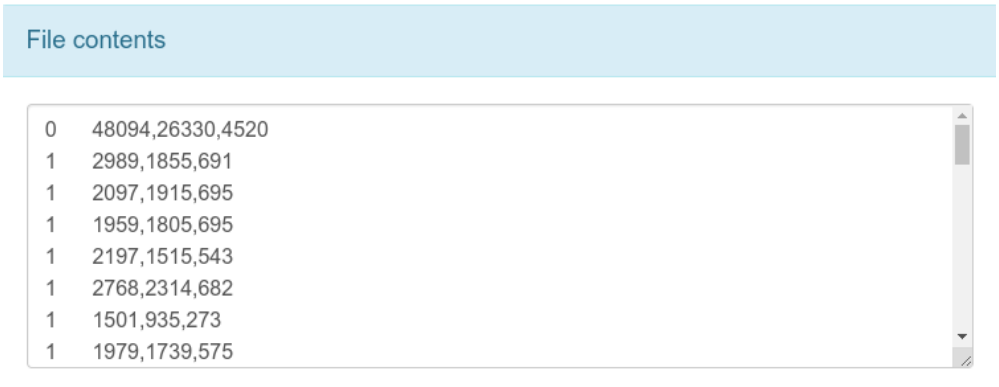
JAR Command:

(base) ujjwal@orange:~/IdeaProjects/Byod_E$ hadoop jar /home/ujjwal/IdeaProjects/Byod_E/target/Byod_E-1.0-SNAPSHOT.jar org.ujjwal.Byod_E /project2/Input_Files/College_Data_Updated.csv /project2/Output_Files/Byod_E_output 20 0.01

Output:



cluster_centers.txt (returning the cluster centers with the message that convergence has been reached after 8 iterations.)



Head of the final clustering that shows final clustered data points along with its centroids.

Task F: Provide a description for each of your above five solutions in your report and conduct experiments over them, for example by choosing different K values (e.g., show how the clusters change for different K values) and different R values (e.g., show how the clusters change over different rounds). describe the relative performance, explain, and analyze your findings. [10 pts]

The choice of k significantly impacts the granularity of the clustering. A larger k (like 10) will create more, smaller clusters, while a smaller k (like 3) will create fewer, larger clusters. The optimal k often depends on the specific dataset and the goals of the analysis. We took k=5 as general. Any value of k will work but it will change the iteration and it also depends upon the threshold.

In **Task A** for single iteration, we just run the K-means algorithm once which might not be our actual centroid because generally it takes multiple iteration for finding the actual centroids.

In **Task B** for multiple iterations, we run the K-means algorithm multiple times, which increases the likelihood of finding optimal centroids. With each iteration, we get closer to the true cluster centers. In our case, we set R = 10, but this may not always be sufficient to find the optimal centroids. More iterations might be needed for complex datasets.

In **Task C** for multiple iterations, we run the K-means algorithm multiple times, which increases the likelihood of finding optimal centroids. With each iteration, we get closer to the true cluster centers. In our case, we have set R = 10, but this may not always be sufficient to find the optimal centroids. More iterations might be needed for complex datasets. But here, we have added a termination point so that our algorithm will terminate in case of early finding of centroids. We got optimal centroids on 13^{th} iteration.

In **Task D**, we added a combiner class to enhance the performance of the algorithm, as it reduces the workload for the reducer class. We got optimal centroids on 13^{th} iteration as before but we were able to do it much faster than without using combiner as in task c.

In **Task E**, we are returning the cluster centers with the message that convergence has been reached after x iterations. We also return another file that has all the clustered data points with their cluster centers. We got optimal centroids on 13^{th} iteration as before but we were able to do it much faster than without using combiner as in task c.

For higher K value, eg, K=10, the scenario results in more granular clustering, potentially capturing finer details in the data. For smaller K-value, eg, K=3,4, the scenario resulted into broader and general clusters that captures high level patterns in the data.
As the iteration increases, the convergence rate will decrease and the cluster will become more closer toghether which can be used to find optimal centroids. Compared with single iteration, it does not return optimal centroid, but just returns the centroid, which might or might not be the optimal centroids.

Teammate Contribution

Team Members : Hari Prasath KP, Ujjwal Pandit
Roles: We did everything together.

We had specified times on when we are going to work in this project. We did each and every step consulting with each other and working on it together.

The toughest part for us was choosing the dataset from Kaggle which meet our requirements for Bring Your Own Data.

We used ChatGPT and other genearative AI to understand the concepts better and it helped us do our work in the efficient way.