**Parallelization of K-means algorithm**

Haider Saeed Khan, Muhammad Usman, Rayyan Masood

SMCS, Institute of Business Administration, Karachi

96003: Parallel and Distributed Computing

Muhammad Zainuddin

14th May, 2024

**Contents**

[**Introduction** 3](#_Toc166614787)

[**Overview** 3](#_Toc166614788)

[**Significance** 3](#_Toc166614789)

[**Objectives** 4](#_Toc166614790)

[**Methodology** 4](#_Toc166614791)

[***Initial Implementation*** 4](#_Toc166614792)

[***Literature Review*** 5](#_Toc166614793)

[***Implementation on Real-life Datasets*** 5](#_Toc166614794)

[***Performance Measurement and Comparison*** 6](#_Toc166614795)

[**Literature Review** 6](#_Toc166614796)

[**References** 10](#_Toc166614797)

[**Code Submission Link** 11](#_Toc166614798)

[**Execution Time Evaluation** 11](#_Toc166614799)

[**Changes For Parallel Approach** 14](#_Toc166614800)

[**Flow Diagram** 16](#_Toc166614801)

[**Future Parameter Considerations** 20](#_Toc166614802)

[**Conclusion** 20](#_Toc166614803)

# **Introduction**

## **Overview**

The goal of the research is to parallelize K-Means clustering to improve its performance on real-world datasets of different sizes. The study claims that the Farm Skeleton parallel pattern may effectively handle larger datasets by reducing the algorithm's execution time by using multiprocessing approaches. Furthermore, multithreading implementation can also be utilized. To test the scalability and efficacy of this technique, two datasets—one large and one small—will be utilized and the relevant metrics such as execution time will be compared.

## **Significance**

In all of data mining, cluster analysis is one of the most extensively studied subjects and is fundamental to machine learning, statistics, bioinformatics, image analytics, and pattern identification.

We propose to parallelize the K-means algorithm in order to accelerate it. K-means can offer significant insights and patterns needed for decision-making in a variety of applications since it can group datapoints into relevant categories. Instead of focusing only on the performance of an algorithm, parallelizing K-means seeks to increase data processing efficiency in general. This has many advantages, for instance, rapid processing speeds up the process of understanding massive volumes of data, which is essential for industry and research.

In practice, we'll experiment with several multithreading/multiprocessing approaches and evaluate the algorithm to find the most effective one. This will allow us to illustrate which tactic works best in certain situations. Our intention in doing this is to offer a helpful manual that will support next initiatives meant to accelerate data analysis and obtain more rapid and efficient insights from massive datasets.

## **Objectives**

The key goals of this project are:

1. Parallelizing K-means algorithm using multithreading and multiprocessing via Farm Skeleton for optimal execution.
2. Assessing the scalability and practicality of the parallelized K-means method on actual datasets of different sizes by recording execution times.

## **Methodology**

Our project begins with implementing an initial sequential K-means implementation. Following this we review the available literature on the algorithm (implementation, parallelization, use cases). This enables us to conduct an inquiry into parallelization technologies. We explore real-world datasets to which these techniques can be applied, and lastly, we analyze the algorithm’s performance on these datasets.

### ***Initial Implementation***

We applied the batch K-means algorithm (like Forgy/Lloyd) in consecutive order on a dataset that was generated at random as part of our project proposal. This stage was essential to confirming that we fully understood the algorithm and had the architecture required to carry it out.

### ***Literature Review***

To provide a baseline knowledge of the state of K-means algorithm applications in parallel computing, we have carried out a thorough literature review. This review investigates the theoretical and practical limits of parallelization by examining vectorization, multithreading, and multiprocessing approaches. While multiprocessing disperses data chunks among many processing units, multithreading splits the dataset into smaller bits or chunks and transmits them to various threads. This document's literature review has comprehensive information about this area.

### ***Implementation on Real-life Datasets***

Our main objective is to evaluate, on two different real-world datasets—one small and one large—the scalability and efficacy of the parallelized K-means method. This stage requires careful dataset selection.

Before applying the algorithm to any dataset, we intend to perform a series of preprocessing steps that are suited to the individual requirements of each dataset. These stages require data cleaning and normalization/scaling to ensure that the datasets are suitable for clustering. Preprocessing a dataset for K-means, which includes transforming categorical data to numerical values and scaling, indicates that all variables contribute equally to the results. This step is crucial because it prevents features with larger magnitudes from dominating the calculations, leading to more balanced and meaningful clustering outcomes.

The parallelized algorithm will then be used on these preprocessed datasets. Two of the parallelized techniques include multiprocessing utilizing the Farm Skeleton framework and multithreading employing either data point or cluster division among threads.

### ***Performance Measurement and Comparison***

We will measure and compare the execution speeds of sequential and parallelized K-means algorithms to try to measure the effect of parallelization efforts. This comparison analyzes scalability by tracking performance gains as variables of dataset size and focuses on the execution time reduction provided by parallelization. We anticipate that these measurements will yield quantitative proof of parallelism.

# **Literature Review**

We want to compare the different K-means algorithm implementations and identify the best ones using two actual datasets. We have carried out an extensive and detailed literature study to improve our comprehension of the method and its application situations.

The algorithm was initially presented as a pulse-code modulation method in 1957 by Lloyd (1982). The input domain is first filled with some number k of point sites by Lloyd's algorithm. Then it repeatedly computes the Voronoi diagram, computes the centroid by integrating each cell of the Voronoi diagram and then each site is then moved to the centroid of its Voronoi cell. In a similar fashion, we will also iteratively assign data points to centroids and recalculate the centroids at the end of each iteration.

As suggested by Morissette and Chartier (2013), there are other popular clustering algorithms in use today. One of these is MacQueen’s algorithm which recomputes centroid location after the introduction of each data point (MacQueen, 1967). This is computationally more expensive than Lloyd’s approach. Research praises Lloyd’s batch technique of iteratively updating clusters by claiming that “conceptually, Lloyd’s algorithm is the simplest” and “for Lloyd’s and other iterative algorithms, improvement of the partitioning and convergence of the error measure E to a local minimum is often quite fast—even when the initial reference points are badly chosen” (Faber, 1994, p. 141). For these reasons we have decided to take the batch approach to update the centroids.

Based on the research papers that we have read, <https://archive.ics.uci.edu/> is a popular source for the datasets. We shall consider choosing datasets from this repository and <https://www.kaggle.com/>.

Before beginning any machine learning task, it is essential to preprocess the data accurately. Preprocessing techniques such as data reduction through methods like Principal Component Analysis (PCA) can be beneficial for K-means clustering (Ding & He, 2004). By reducing the dimensionality of the dataset, PCA can help improve the efficiency and accuracy of the clustering process. Furthermore, Patel and Mehta (2011) suggest that finding and removing unusual data points helps the K-means algorithm work better and faster.

Preprocessing techniques also include scaling or standardization. Adams (2018) points out that we need to carefully consider what distances represent when using K-means. He gives an example from a dataset and shows that if the units of a dataset are changed, the clustering can change dramatically. For this reason, he suggests the programmer to standardize the dataset. More often than not, data can be missing in real-world data sets. He suggests using one of imputation or marginalization. Based on these findings, we aim to incorporate these preprocessing techniques when working on our project.

The optimal number of clusters ensures accuracy and stability in the algorithm. Kordinariya and Makwana (2013) claim several ways of computing initial clusters. These include the elbow method, silhouette method and a general rule of thumb (k = √(n/2), where n is the number of data points). We have decided not to use the elbow method. Research suggests that “we do not have a meaningful measurement of angle, and changing the scaling of the axes (and, e.g., the parameter range of k) may well change the human interpretation of an ‘elbow’” (Schubert, 2023, p. 2). Therefore, we will use only one of the other two.

Lloyd’s algorithm suggests choosing the initial centroids randomly (Morissette & Chartier, 2013). Mayo (2016) reasons that this approach causes the algorithm to be more time consuming in its convergence, therefore, he suggests a deterministic sharding approach whereby a particular composite value (e.g. sum) is derived from each data point, following which the dataset is sorted by this value and split into shards. A centroid is then calculated for each shard (e.g. mean of each corresponding attribute). He concludes that the sharding approach to initializing the centroids is better than the randomization approach. Therefore, we will implement sharding to initialize centroids, following which we will implement batch K-means as suggested in Lloyd’s.

The literature review so far encompasses the key details in the set-up of the K-means algorithm. The remaining part of this section explores the feasibility of parallelization of K-means. Our instructor has suggested not to use vectorized implementation. The research papers we have reviewed corroborate his advice. In particular, it is suggested that “unaligned memory accesses have a large performance penalty” (Shahbahrami et al., 2006, p. 8). Since real life data sets are rarely aligned, it would be naive to use hardware vectorized implementation.

Kittisak Kerdprasop and Nittaya Kerdprasop (2010) suggest a parallelized K-means algorithm as an improvement over sequential execution. The algorithm works by splitting the N data points among P processes equally. They have taken a multiprocessing approach and have measured execution times and speedups for different sized datasets. Their experiment shows that parallelized K-means provides a significant speedup compared to sequential K-means. We may implement the approach of splitting N datapoints among P threads using multithreading. A second approach is to assign each centroid and its corresponding calculations to a unique thread. We plan to use one of these approaches only. In either case, we will compare the execution times between a large and a small dataset as stated in our methodology above. Rao et al. (2009) have shown that multithreading can also be made possible using the OpenMP library. We may or may not follow his approach, since there are various other ways we can implement multithreading (e.g. manually assigning threads).

Poldner and Kuchen (2008) have illustrated that multiprocessing can be implemented using the Farm Skeleton. In particular, the topology of the farmer forwarding the processes to his workers seems to be of utmost importance to us, since our data points are independent of each other.

Gustriansyah et al. (2020) suggest the metric R = (1/k)∑ki=1Rk/(1/k)∑ki,j=1 i≠jRi,j for measuring cluster quality. This is a ratio of the average intra and inter cluster distance. An R value close to 0 indicates that data in the same clusters are more similar. Other metrics such as WSS, BSS, etc. can also be used. Since our primary objective is to parallelize K-means, we may not focus on calculating accuracy.

# **References**

Adams, R. P. (2018). K-Means Clustering and Related Algorithms. *Princeton University*.

Ding, C., & He, X. (2004). *K* -means clustering via principal component analysis. *Twenty-First International Conference on Machine Learning - ICML ’04*, 29. https://doi.org/10.1145/1015330.1015408

Faber, V. (1994). Clustering and the continuous k-means algorithm. *Los Alamos Science*, *22*(138144.21), 67.

Gustriansyah, R., Suhandi, N., & Antony, F. (2020). Clustering optimization in RFM analysis Based on k-Means. *Indonesian Journal of Electrical Engineering and Computer Science*, *18*(1), 470. https://doi.org/10.11591/ijeecs.v18.i1.pp470-477

Kerdprasop, K., & Kerdprasop, N. (2010). Parallelization of K-means clustering on multi-core processors. *International Conference on Applied Computer Science - Proceedings*, *10*, 472–477.

Kodinariya, T., & Makwana, P. (2013). Review on determining number of Cluster in K-Means Clustering. *International Journal*, *1*(6), 90–95.

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, *28*(2), 129–137. https://doi.org/10.1109/TIT.1982.1056489

MacQueen, J. (1967). Some method for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (Vol. 1, pp. 281–297).

Mayo, M. M. (2016). *An Arithmetic-Based Deterministic Centroid Initialization Method for the k-Means Clustering Algorithm* [Columbus State University]. https://csuepress.columbusstate.edu/theses\_dissertations/241

Morissette, L., & Chartier, S. (2013). The k-means clustering technique: General considerations and implementation in Mathematica. *Tutorials in Quantitative Methods for Psychology*, *9*(1), 15–24. https://doi.org/10.20982/tqmp.09.1.p015

Patel, V., & Mehta, R. (2011). Impact of outlier removal and normalization approach in modified k-means clustering algorithm. *International Journal of Computer Science Issues (IJCSI)*, *8*(5), 331.

Poldner, M., & Kuchen, H. (2008). ON IMPLEMENTING THE FARM SKELETON. *Parallel Processing Letters*, *18*(01), 117–131. https://doi.org/10.1142/S0129626408003260

Rao, S. N. T., Prasad, E. V., & Venkateswarlu, N. B. (2009). A scalable k-means clustering algorithm on Multi-Core architecture. *2009 Proceeding of International Conference on Methods and Models in Computer Science (ICM2CS)*, 1–9. https://doi.org/10.1109/ICM2CS.2009.5397976

Schubert, E. (2023). Stop using the elbow criterion for k-means and how to choose the number of clusters instead. *ACM SIGKDD Explorations Newsletter*, *25*(1), 36–42. https://doi.org/10.1145/3606274.3606278

Shahbahrami, A., Juurlink, B., & Vassiliadis, S. (2006). Performance Impact of Misaligned Accesses in SIMD Extensions. *Proceedings of the 17th Annual Workshop on Circuits, Systems and Signal Processing*, 334–342.

# **Code Submission Link**

<https://drive.google.com/drive/folders/1e5GqiXkjJUAeKAJfvrhsPGqZP0ttYNdo?usp=sharing>

# **Execution Time Evaluation**

**Table 1**

*Wine Dataset*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of threads/processes | Serial | Multithread | Multichunk | Multiprocess |
| 2 | 0.1250 | 0.2120 | 48.7424 | 51.2942 |
| 5 | 0.1250 | 0.2659 | 64.7639 | 76.0381 |
| 10 | 0.1250 | 0.2869 | 129.4663 | 135.8537 |
| 20 | 0.1250 | 0.3783 | 234.8775 | 271.1465 |

**Figure 1**

*Wine Dataset Execution Time Comparison*

**Table 2**

*Bank Churners Dataset*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. of threads/processes | Serial | Multithread | Multichunk | Multiprocess |
| 2 | 140.2639 | 120.3963 | 134.3129 | 166.6917 |
| 5 | 140.2639 | 140.3012 | 146.6515 | 173.1303 |
| 10 | 140.2639 | 147.7232 | 233.8635 | 259.6407 |
| 20 | 140.2639 | 153.4281 | 390.6835 | 444.9592 |

**Figure 2**

*Bank Churners Dataset Execution Time Comparison*

# **Changes For Parallel Approach**

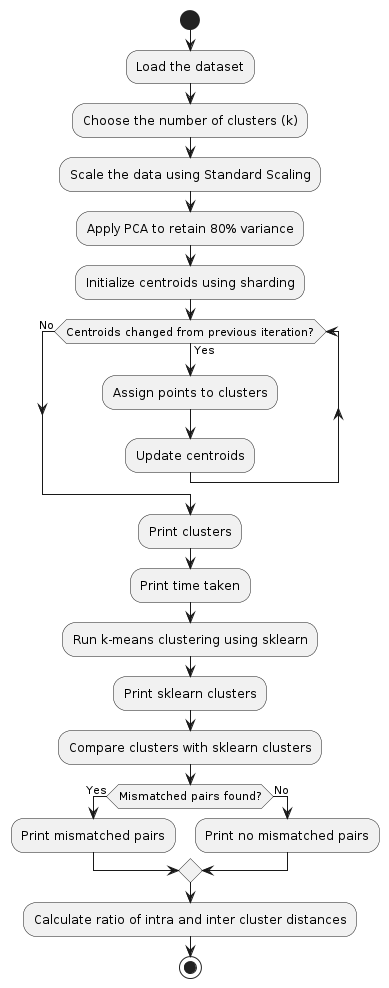
We have implemented parallelization by parallelizing sharding, assigning clusters to data points and updating centroids. We improved on the base paper given in the proposal by using sharding, using threading (as it is more lightweight and has less overhead than multiprocessing in the context of Windows system) and parallelizing the updating of centroids step explicitly.

* Multithread:
* Parallelized centroids initialization where each centroid was initialized by a different thread. The dataset was divided into k chunks and a representative datapoint was extracted from each chunk, where each chunk was assigned to a thread.
* Divided the dataset into n chunks where n is the number of threads.
* For each chunk, the thread assigned the closest cluster to its datapoint in the global clusters dictionary.
* After all points were assigned to their closest cluster, each centroid was updated by a different thread by taking the average of the data points in its cluster.
* Multiprocess:
* All parallelization is done via farm skeleton.
* This follows the farm methodology where the master function assigns tasks to worker functions. There are two stages here: assigning clusters and updating centroids. All data points must be assigned to the cluster before centroids can be updated. This dependency means that this is a farm approach.
* K centroids were initialized from k shards in parallel.
* Each datapoint was assigned the closest cluster in the global clusters dictionary in parallel.
* Each centroid was updated in parallel by taking the average of its assigned data points.
* Multichunking:
* Also implemented via farm skeleton but with a slight modification.
* Centroids were initialized in parallel in the same way as in Multiprocess.
* But the data points were grouped into n chunks prior to the assignment of closest cluster in the global clusters dictionary.
* Then each centroid was updated in parallel in the same way as in Multiprocess.

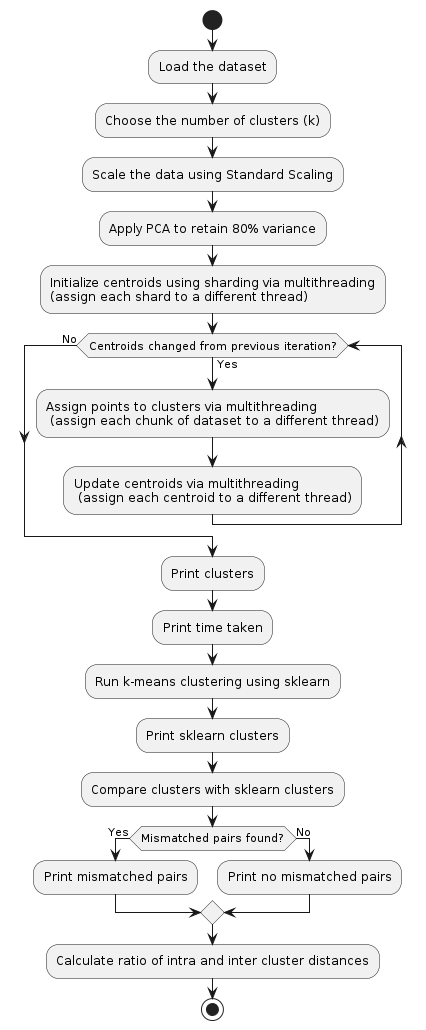
# **Flow Diagram**

**Figure 1**

*Serial Code*

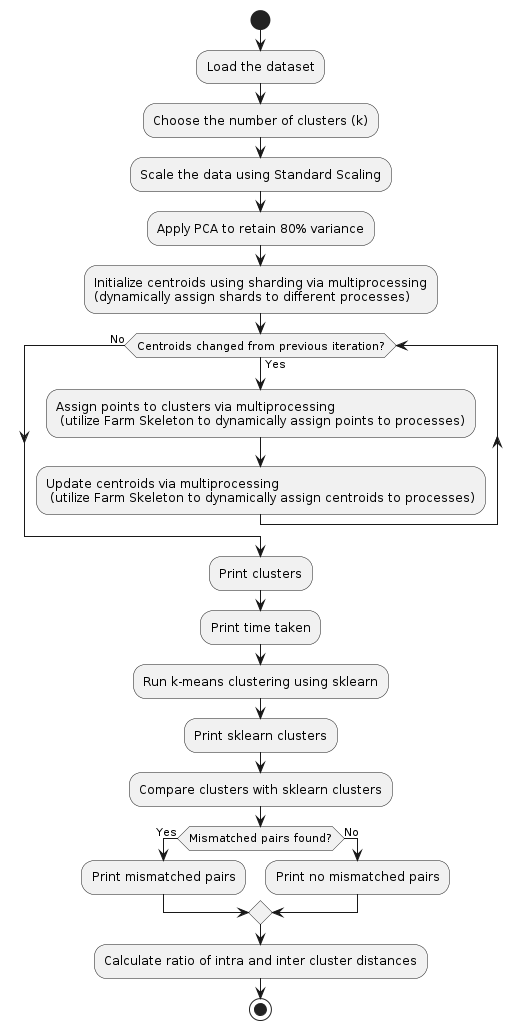


**Figure 2**

*Multithread Code*

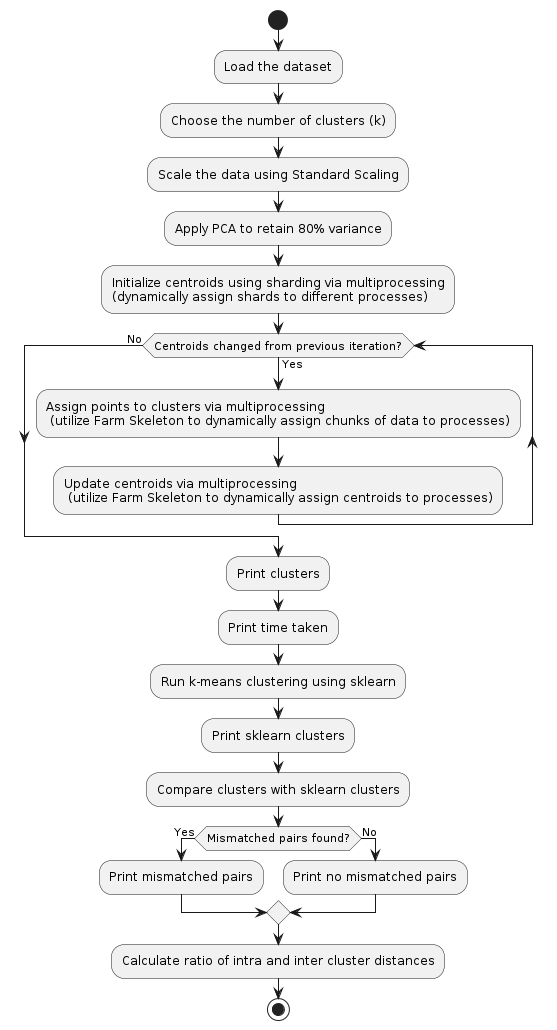
**Figure 3**

*Multiprocess Code*



**Figure 4**

*Multichunk Code*



# **Future Parameter Considerations**

* The number of clusters was chosen as k = sqrt(n/2) based on findings during the literature review process.
* The number of processes for sharding could be experimented with. We have chosen 2 to keep it consistent.
* We have varied the number of datapoints assigned to threads/processes and have reported the findings above.
* The parameter value needs to simply be changed and the code should be executed.
* The data can be gathered in a csv file and relevant graphs can be plotted for execution time against parameter(s).

# **Conclusion**

For the smaller dataset, serial execution without parallelization was fastest. This is no surprise because creation of threads/processes has an added overhead time and resource cost as well as load balancing costs. Implementing the farm skeleton proves to be very costly as shown by the difference in execution time for multithread and multichunk/multiprocess. A reason could be that multichunk/multiprocess dynamically assigns the data to the processes. This means processors do extra work. Overall making more processes/threads is costly because of added overhead time and resource cost as well as load balancing costs. This effect is more pronounced for smaller datasets since the execution time is smaller than larger datasets due to input size.

For the larger dataset, it is observed that for 2 and possibly 3 and 4 threads, multithreading is the fastest means of execution. This is because it is implemented in the blocking fashion where each thread takes up a chunk of data. Multichunking is also faster than serial using two processors as compared to a non-parallelized approach. Multiprocess is the slowest. This means that the data should not be divided too finely.

Another key insight that can be found is that the percentage time difference between farm skeleton approach and multithreading or serial approach becomes lesser as the dataset gets larger. This means that farm skeleton will be useful for very large data, even more so perhaps than multithreading.

This project has been useful as it has served us to explore multiple means of parallelization such as multithreading and multiprocessing. We have discovered adding on to basic approaches for example by using sharding/parallelized sharding. We have applied Machine Learning preprocessing concepts such as Scaling and Principal Component Analysis to enhance our project and quality of data. These findings suggest that parallel implementation in clustering can lead to quick results in the scope of real-world contexts as demonstrated. It can be used to enhance strategic thinking.

Multiple different scenarios can be implemented using these algorithms in the future. Different sharding techniques can be explored. Choosing the number of clusters can be done keeping business scenarios and target classes in mind for example customer segmentation. In addition, K-means clustering can be deployed on a network to classify packets as safe, harmful, etc. and this calculation needs to be done quite quickly.

Note: The cluster quality has been calculated by finding the intra and inter cluster difference. The small dataset yields this ratio to be 0.06 while the large dataset has the ratio 0.01, which indicates good clustering.

Note 2: All codes have been run on Windows intel i5-11th generation machine with intel iRISxe.