Performance Comparison of Centroid Based Clustering Algorithms

Aagyapal Kaur
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
aagyapalkaur@cmail.carleton.ca

December 20, 2021

Abstract

Clustering is a standard method in the data analysis process that is utilised in various domains such as pattern recognition and image segmentation, statistics, bioinformatics, and so on. Because current reality applications generate huge datasets, determining how to effectively deal with this data in a big mining process has proven to be a challenging and enormous challenge. Clustering has a number of algorithms, the most well-known of which is K-means that is centroid-based algorithm, and is well-known for its simplicity, empirical success, and efficient for handling the large datasets. The present research paper is designed to depicts a comparative analysis of the performance of centroid based clustering algorithms that include K-means, K-means++ and Parallel K-means using Message Passing Interface(MPI).

***K*eywords:** Clustering, K-Means, Message passing Interface, Sequential, parallel computing, data mining.

1 Introduction

Enterprises today are dealing with the massive size of data, which have been explosively increasing. The key requirements to address this challenge are to extract, analyse, and process data in a timely manner. Clustering is an essential data mining tool that plays an important role for analysing the big data. However, large scale data clustering has become a challenging task because of the large amount of information that emerges from technological progress in many areas, including finance and business informatics.

Patterns identification groups such as feature vectors ,data items or other observation done by supervised or unsupervised classification method, Clustering is belongs to the method of unsupervised classification method .Objects with similar features are grouped to form a cluster. Clustering has wide intrigue and has been applied to address many field like data compression dimension reduction, feature selection, dimension reduction and vector quantization. Requirement of an application decide the opinion of quality cluster. There are a few strategies for discovering cluster utilizing techniques of splitting-merging, neural networks, and distribution – based clustering. Centroid-based clustering arranges the data into non-hierarchical clusters, in contrast to hierarchical clustering. In centroid –based

clustering there are many types among that K-means is the most widely-used one. Centroid-based algorithms are efficient in grouping but sensitive to outliers and initial conditions[6].

K-means is a well-known clustering algorithm for its simplicity and easy implementation. K-means was ranked second of top 10 algorithms in data mining by the ICDM Conference in October 2006, while C4.5 was ranked first [11]. Compared with other clustering algorithms, K-means algorithm has three major advantages covering simple implementation, efficient when handling a large data sets and a solid theoretical foundation based on the greedy optimization of Voronoi partition[8].

In addition, MPI (Message Passing Interface) is a library specification for message passing, which is proposed as a standard program model. MPI is an application programmer interface of message passing, together with protocol and semantic specifications for how its features must behave in any implementation[5]. MPI aims to maintain the high performance, scalability and portability.The paper is organized as follows: Section 2 is a review of some previously published work relating to the research is presented.The problem description and the method for providing a solution are then covered in Sections 3 and 4, respectively.Section 5 presents the experiment results and performance evaluation.Finally, Section 6 concludes the paper.

2 LITERATURE REVIEW

2.1 K-means Clustering Algorithm

K-means is one of the most famous partitional algorithms in cluster process. K-means has a rich and various history as it was autonomously found in various logical fields by Steinhaus (1956), Lloyd (proposed in 1957, distributed in 1982), Ball and Hall (1965), and MacQueen (1967) . K-means is a method of data analysis. The objective is to partition N data objects into K clusters (K<N) such that the objects in the same cluster are as similar as possible and a dissimilar as possible in different clusters.[6].

Algorithm1: K-Means clustering algorithm

The main steps of K-means algorithm are as follows[7] :

1. Select k points as initial centroids arbitrarily.

2. Calculate the distance (Euclidean Distance) between the rest data and k centroids. Arrange each data into the nearest cluster. According to Euclidean Distance formula, the distance between the two points in the plane with coordinates (x_1, y_1) and (x_2, y_2) is given by:
$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

3. After processing all the points, compute the average of all the records in each same cluster.

4. The average of each cluster is used as the new centroid.

5. Calculate the difference between the new centroid with the original one in the same cluster. If the difference is smaller than the threshold or the number of iterations of the algorithm has been reached for the maximum, the algorithm is over. Otherwise, the new centroid is substituted for the original ones. Return (2) and continue until the convergence is achieved and obtain the final cluster .

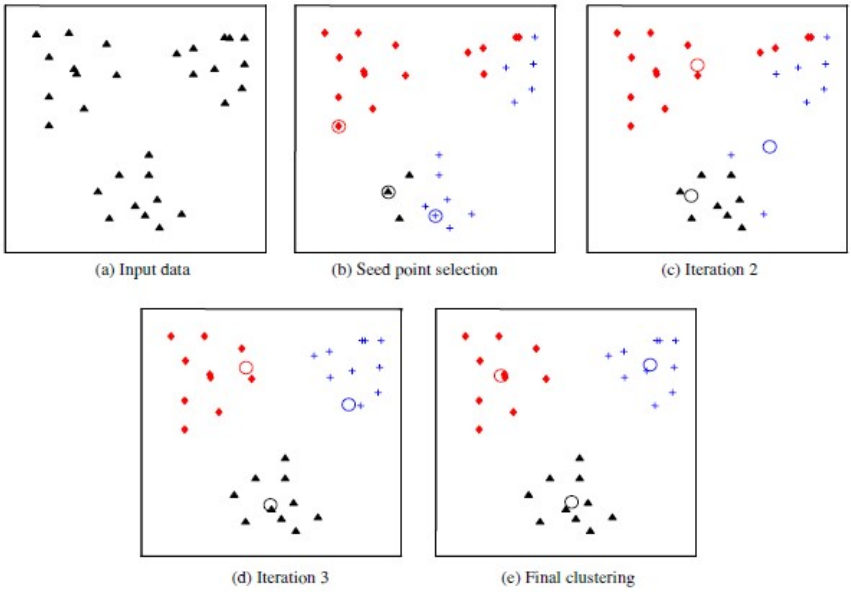

Figure 1: Illustration of K-means algorithm. (a) Two-dimensional input data with three clusters; (b) three seed points selected as cluster centers and initial assignment of the data points to clusters; (c) and (d) intermediate iterations updating cluster labels and their centers; (e) final clustering obtained by K-means algorithm at convergence.[6]

The K-means algorithm requires three user-specified parameters: number of clusters K, cluster initialization, and distance metric. The most critical choice is K. Typically, K-means is run independently for different values of K and the partition that appears the most meaningful to the domain expert is selected. Different initializations can lead to different final clustering because K-means only converges to local minima. One way to overcome the local minima is to run the K-means algorithm, for a given K, with multiple different initial partitions and choose the partition with the smallest squared error. K-means is typically used with the Euclidean metric for computing the distance between points and cluster centers. As a result, K-means finds spherical or ball-shaped clusters in data.

2.2 Limitation of K-means:

The main critical problem of this algorithm is that it might be blocked locally based on the initial random chosen centers. The k-means++ algorithm is developed to solve this problem, spreading out the initial centers with an updating non-uniform distribution[2].

2.3 To overcome the limitation of K-means, we use K-means++ Algorithm:

The K-means++ algorithm is identical to k-means except the initialization of centers. K-means++ tries to choose a set of carefully selected initial centers instead of random initialization .This algorithm ensures a smarter initialization of centroids and ameliorate the quality of clustering. The k-means algorithm begins with an arbitrary set of cluster cen-

ter. At any given time, let D'(m) denote the shortest distance from a data point m to the closest center we have already chosen, the initialization step of k-means++ can be defined as[2],[10]:

1. Choose an initial center c_1 uniformly at random from M where M is a set of datapoints.
2. Choose the next center c_i, selecting $c_i = m \epsilon M$ with probability proportional to

$$\frac{D'(m)^2}{\sum_{m \epsilon M} D(m)^2} \tag{2}$$

3. Repeat Step (2) until we have chosen a total of k centers
4. Proceed as with the standard k-means algorithm

Algorithm 2. k-means++ centers initialization.

1: **procedure** K-MEANS++(M, k)
2: Choose a center $\hat{c}_1$ randomly from dataset.
3: $\hat{C} \leftarrow \{\hat{c}_1\}$.
4: **while** $|\hat{C}| \neq k$ **do**
5: Choosing an item $\vec{m}_i \in M - \hat{C}$ with probability of $\frac{D'(\vec{m}_i)^2}{\sum_{\vec{m}_j \in M} D(\vec{m}_j)^2}$
6: $\hat{C} \leftarrow \hat{C} \cup \{\vec{m}_i\}$.
7: **end while**
8: **return** $\hat{C}$
9: **end procedure**

Figure 2: K-mean++ Algorithm.[10]

2.4 Message Passing Interface

Message Passing Interface (MPI) is a standard developed by the Message Passing Interface Forum (MPIF). MPI is a standard library specification designed to support parallel computing in a distributed memory environment. The first version (version1.0) was published in 1994 and the second version (MPI-2) was published in 1997 [4]. Both point to point and collective communication are supported. MPI is an API library consisting of hundreds of function interfaces called by computers such as computer clusters communicate with one another in the parallel development environment. MPI is a language-independent communications protocol. FORTRAN, C and C++ can directly call the API library. The goals of MPI are high performance, scalability and portability.

The standards of MPI are as follows[1]:

1. Point-to-point communication
2. Collective operations
3. Process groups
4. Communication contexts

5. Process topologies
6. Bindings for Fortran 77 and C
7. Environmental management and inquiry
8. Profiling interface

2.5 Parallel K-means Algorithm

Parallelized in basic have same meaning in partition data so data can be execute in same time. In parallel system, the main idea is partition to each processes so the processes will have same amount of data and can do the processes in same time. Each computer can do the algorithm, and have centroid in each process. After that, the result of each process will be merged in head node[9].

Parallel K-Means Algorithm

Input: Data, K Cluster

Output: K Centroid

1: MPI_INIT// start MPI Procedure

2: Read N object from file

/start parallel proccess by divide same amount of object to each processes/

3: **repeat**

4:Choose K point as intial centroid randomly

5: Initiate each object to the closest centroid by using Euclidean Distance Formula

6: **until** centroid don't change

/merge centroid procedure /

7: Generate cluster id to each object

8: Generate new centroid cluster by centroid result in each processes

9: Generate final centroid

10: MPI_Finalize() // Terminate MPI Process

Figure 3: Parallel K-means Algorithm.[9]

The above Algorithm explain the processing flow of our parallel K means algorithm, which is the key algorithm in our work. This algorithm utilizes K-means and the MPI parallel framework, it is hence simple and portable. All initialization is implemented by the MPI _INIT function, which is the first call of MPI program. It is the first executable statement of all of the MPI program. To start the MPI environment, it means the beginning of the parallel codes. The MPI _FINALIZE function is the last call of MPI program, and it ends the running of MPI program. It is the last executable statement of MPI program, otherwise, results of the procedure is unpredictable. MPI_FINALIZE symbolizes the end of the parallel codes.In our Parallel Kmeans algorithm, the parallelism is implemented by

the data parallelism. The parallel processing in Parallel Kmeans is consistent with that of K means. Data objects are evenly partitioned in all processes and cluster centroids are replicated. The global operation for all cluster centroids is performed at the end of each iteration in order to generate new cluster centroids. Finally, output the clustering results: K centroids, I/O time and clustering time[13],[9].From the above discussion it can be observed at the end of this paper that Parallel configuration of K-means is very good approach in context of dealing with bulkier datasets.

3 Problem Statement

In this day and age, handling the large datasets is a very arduous and challenging task for most of the enterprises. With the advancement in science and technology makes collection of data and storage much easier and very inexpensive than ever before. This led to the formation of enormous datasets in science, MNC's, health care sector, business sector, social networking and so on, which should be processed or sorted to get useful information. For example if we consider the results generated by a search engine for a particular query, user has to shift through the long lists and find the desired solution. But this job can be very difficult for the user if there are millions of web pages displayed as solutions for a given query. Moreover, while performing operations on huge datasets by data scientists, so as to create different machine learning models, sometimes due to lack of parallelism it takes a lot of time to deliver the results. This paper tries to reflect comparative analysis of the different types of centroid based clustering techniques such as K-means, K-means++ and Parallel K-means using MPI in order to tackle with these large datasets not only in effective way but also in timely manner.

4 Proposed Solution

In order to conduct the performance analysis of centroid based clustering algorithm, I used the real-world dataset that is 3D Road Network (North Jutland, Denmark) Data Set[3]. While performing K-means clustering algorithm on 3D Road Network (North Jutland, Denmark) Data Set, the results that I got is impressive, nonetheless, still there is a limitation of K-means that is sensitivity initialization, and owing to this it consumes time while operating on the large datasets. To overcome this limitation, I use K-means++ which is effective way than K-means and the outcomes that I obtained is better than that of the K-means.

But still there is one missing factor that boost the performance of these algorithm significantly and this factor is Parallelism. As I mentioned above in the problem statement that instead of focusing on handling the large datasets we must focus on how to reduce the operating time of these algorithms. So, Parallel K-means using MPI(Message Passing Interface) framework is effective idea and by implementing it, the results that I achieved are outstanding as the operating time reduces sharply and overall performance is improved markedly. In order to implement the Parallel K-means using MPI framework, I utilize the "mpi4py" library of python language and it is a very powerful library that allows the python applications to execute in a parallel environment and improve the runtime performance.

MPI _Init thread() is called at import time by mpi4py, which also installs an exit hook automatically that calls MPI _Finalize() just before the Python process terminates. In

MPI for python, comm is the base class of communicators which allows the communication among the processes.Figure 4 describes the main working of Parallel K-means. First of all, broadcast the start time from rank=0 to other ranks and then scatter the equal chunks of dataset between each processes and then broadcast the centroid matrix to each process. After that realize the sequential algorithm and calculate the distance matrix and determine clustering vectors in each process.In the end, gather the means of centroid from each process by summing them.

```
start_time = comm.bcast(start_time, root=0)
data = comm.scatter(chunks, root=0)
num_clusters = comm.bcast(num_clusters, root=0)
initial = comm.bcast(initial, root=0)
flag = True
while flag == True:
    clusters = []
    cluster = []
    #Calculating dist matrix in each process
    dist = np.zeros((len(data), len(initial)))

    for j in range(len(initial)):
        for i in range(len(data)):
            dist[i][j] = np.linalg.norm(initial[j] - data[i])

    for i in range(len(dist)):
        clusters.append(np.argmin(dist[i]) + 1)

    # Calculating the number of samples in each cluster
    Q_clusts = collections.Counter(clusters)

    #Summing the number of samples for each cluster
    counterSumOp = MPI.Op.Create(addCounter, commute=True)
    totcounter = comm.allreduce(Q_clusts, op=counterSumOp)
    comm.Barrier()

    # From each worker we gather cluster vector and join them
    cluster = comm.gather(clusters, root=0)
    comm.Barrier()

    if rank == 0:
        cluster = [item for sublist in cluster for item in sublist]
    centroids = np.zeros((len(initial), len(initial[0])))
    for k in range(1, num_clusters + 1):
        indices = [i for i, j in enumerate(clusters) if j == k]
        centroids[k - 1] = np.divide((np.sum([data[i] for i in indices], axis=0)).astype(np.float), totcounter[k])

    centroid = comm.allreduce(centroids, MPI.SUM)
    comm.Barrier()

    if np.all(centroid == initial):
        flag = False
        print("Execution time %s seconds" % (time.time() - start_time))

    else:
        initial = centroid
    comm.Barrier()
```

Figure 4: Main part of code for Parallel K-means

5 Experimental Evaluation

In this paper, the computation time of K-means, K-means++ and Parallel K-means is used as a performance metric and regarding the evaluation for quality of clustering, the SSE (Sum of Squared Error) is utilized as a performance measure.

5.1 Cluster Evaluation

Cluster evaluation is a part of clustering analysis. Because the algorithm using Euclidean Distance formula as closeness measurement, so the objective function that can be used for measure the quality of cluster is Sum Squared of Error (SSE). SSE shows the compactness of clusters and a lower SSE means a better clustering .The explanation of the formula is explained below [9]:

$$SSE = \sum_{i=1}^{k} \sum_{x \epsilon c_i} dist(c_i, x)^2 \tag{3}$$

An Elbow Method:

The basic idea of the elbow rule is to use a square of the distance between the sample points in each cluster and the centroid of the cluster to give a series of K values. The sum of squared errors (SSE) is used as a performance indicator. Iterate over the K-value and calculate the SSE is the basic idea. Smaller values indicate that each cluster is more convergent. When the number of clusters is set to approach the number of real clusters, SSE shows a rapid decline. When the number of clusters exceeds the number of real clusters, SSE will continue to decline but it will quickly become slower.[12]

5.2 Experimental Setup

The following results are obtained when comparing the K-means, K-means++ and Parallel K-means in Intel i5-7400 4 core CPU @ 3.0 GHz. The implementation of these algorithms are done by using the python language. As far as machine learning library is concerned, I use "Scikit- learn" library which is specially build for Python programming language and features various classification and clustering algorithms. And for data visualization matplotlib library is used which is an amazing library in python for 2D graphics.I use Anaconda 3 Navigator for the execution environment.

Dataset

3D Road Network (North Jutland, Denmark) Data Set [3]: This dataset was constructed by adding elevation information to a 2D road network in North Jutland, Denmark (covering a region of $185X135km^2$). IT contains 434874 samples and 4 features. Feature information:

1. OSM ID: Open Street Map ID for each road segment or edge in the graph.
2. LONGITUDE: Web Mercaptor (Google format) longitude
3. LATITUDE: Web Mercaptor (Google format) latitude
4. ALTITUDE: Height in meters.

5.3 Result

The table below presents the runtime of each experiment. Experiments are conducted for the number of clusters k=$\{2, 3\}$ and the number of processes np=$\{2, 3, 4, 5, 6, 7, 8\}$.

No.of clusters	np2	np3	np4	np5	np6	np7	np8	K-means	K-means++
K=2	28.080	26.107	23.447	23.562	25.419	25.297	26.100	35.078	26.121
K=3	56.962	50.313	47.342	48.850	48.724	52.765	51.174	67.405	61.001

Table 1:The run-times (in seconds) of parallel, sequential K-means and K-means++ clustering algorithm on 3D Road Network dataset with 10^5 samples.

It is noticed that the most favourable outcome for a particular dataset is achieved on 4 processes while parallelise the K-means using Message Passing Interface. Meanwhile, the computation time of K-means++ is better than K-means.The performance comparison based on computation time of the algorithms is illustrated by the Figure 5 and Figure 6.

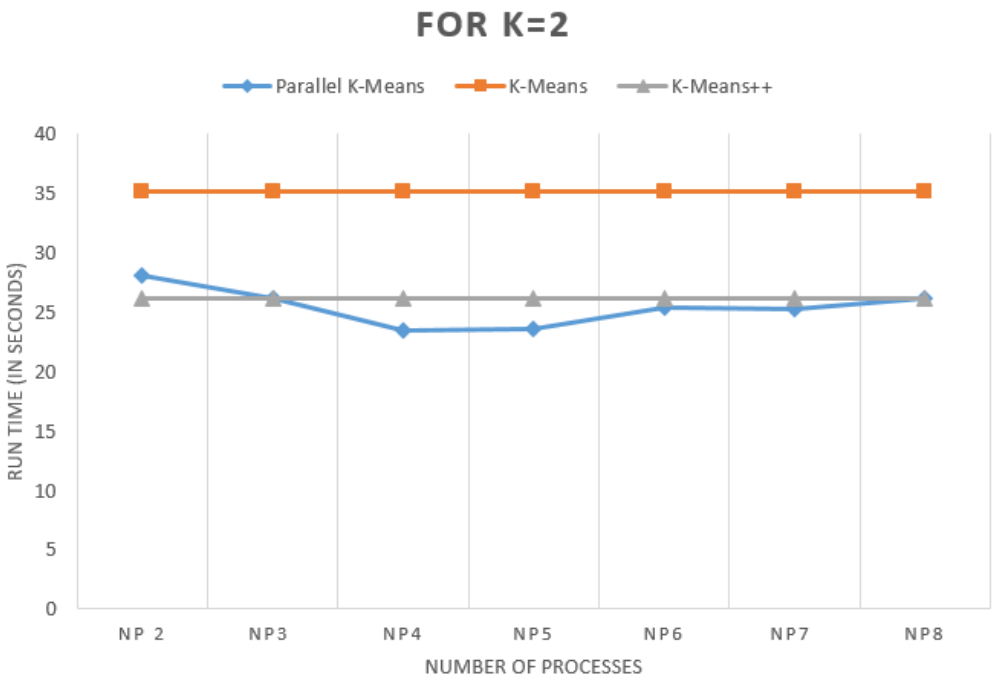

Figure 5: Graphical representation of computation time for two clusters.

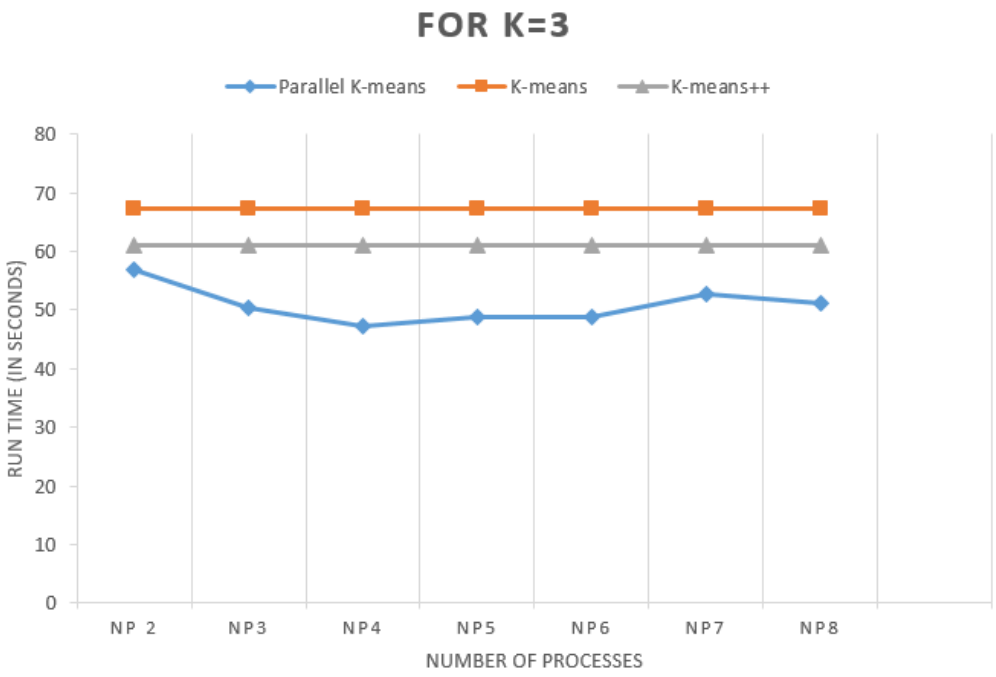

Figure 6: Graphical representation of computation time for three clusters.

Table 2 depicts the sum of squared error by taking the no. of clusters(K-value) from 1 to 10 for a given dataset using the Elbow Method and the figure below shows its graphical representation.It is cleared from Figure 7 that SSE value reduces drastically from K=1 to K=3 and this point is the turning point or elbow point and after this point the SSE value gradually decreases to K=10.As a result, the optimal number of clusters for a given given dataset is K=3.

No. of clusters	**Sum of Squared Error**
1	35270371.038211636
2	11218117.421842316
3	5052998.234094211
4	3008844.9873317108
5	1940707.897814192
6	1412389.355230099
7	1045570.2353036918
8	812220.1138217078
9	656958.9712808387
10	557132.352831361

Table 2: Demonstrates the Sum of Squared Error(SSE) values for K=1 to 10.

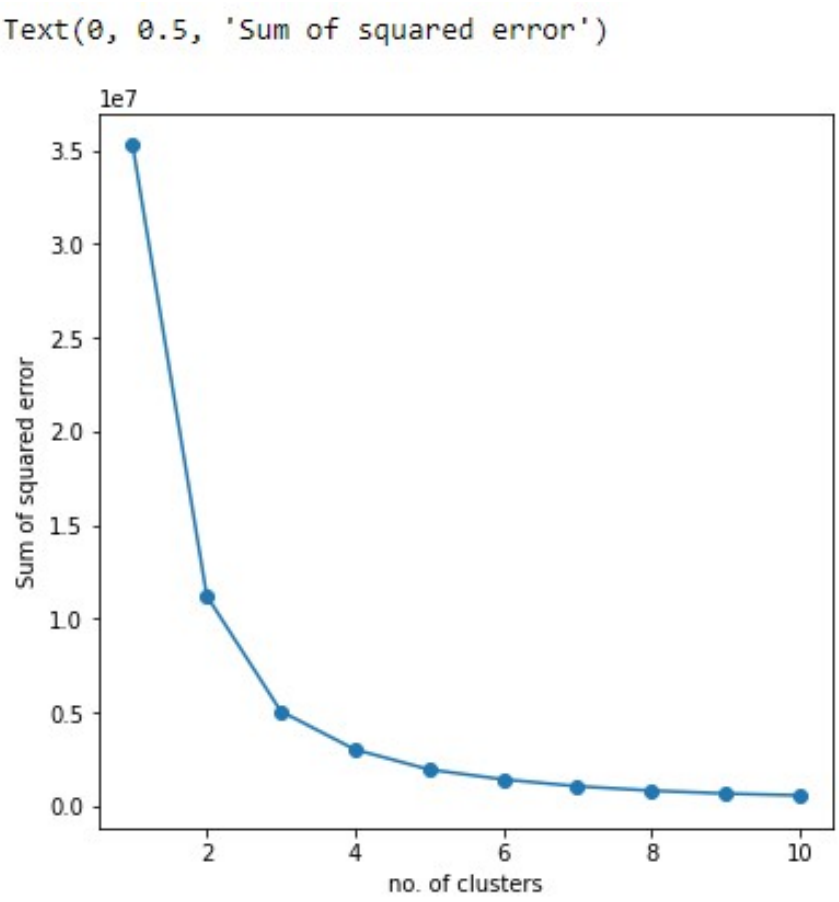

Figure 7: Representation of Elbow Method showing the optimal no. of clusters.

6 Conclusion

The main objective of this paper is to compare the performance of centroid based clustering algorithms while operating on huge datasests.In this paper, different clustering algorithms have been studied and assessed, in order to distinguish between their ability to perform efficiently on a big volume of data.When compared to K-means and K-means++, the experimental findings of two and three cluster formation reveal that K-means with parallel configuration is realtively faster, more stable, portable, and it is more efficient in clustering on big data sets.

References

[1] Yukiya Aoyama, Jun Nakano, et al. *Rs/6000 sp: Practical MPI programming.* IBM Poughkeepsie, New York, 1999.

[2] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding, p 1027–1035. *Society for Industrial and Applied Mathematics*, 2007.

[3] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[4] William Gropp and Ewing Lusk. Implementing mpi: The 1994 mpi implementors' workshop. In *Proceedings Scalable Parallel Libraries Conference*, pages 55–59. IEEE, 1994.

[5] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[6] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[7] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[8] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[9] Fhira Nhita. Comparative study between parallel k-means and parallel k-medoids with message passing interface (mpi). *International Journal on Information and Communication Technology (IJoICT)*, 2(2):27–27, 2016.

[10] Saeed Shahrivari and Saeed Jalili. Single-pass and linear-time k-means clustering based on mapreduce. *Information Systems*, 60:1–12, 2016.

[11] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[12] Chunhui Yuan and Haitao Yang. Research on k-value selection method of k-means clustering algorithm. *J*, 2(2):226–235, 2019.

[13] Jing Zhang, Gongqing Wu, Xuegang Hu, Shiying Li, and Shuilong Hao. A parallel k-means clustering algorithm with mpi. In *2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming*, pages 60–64. IEEE, 2011.