An Efficient MapReduce-based Adaptive K-Means Clustering for Large Dataset

Tapan Chowdhury
Dept. of CSE
Techno India, Salt Lake
Kolkata, India
tapan2005cse@gmail.com

Arijit Mukherjee
Dept. of CSE
IIIT, Hyderabad
Hyderabad, India
mkarijit@gmail.com

Susanta Chakraborty
Dept. of CST
IIEST, Shibpur
Howrah, India
susanta.chak@gmail.com

***Abstract*— MapReduce-based clustering of a large dataset is gaining rapid importance in the fields of data science. Its main task is to make groups of data from the large dataset such that all data points categorized into a single group are similar to one another. K-means clustering method is one of the most extensively used clustering methods but it minimizes clustering criteria by iteratively relocating data points between clusters until a locally optimal partition is attained, so convergence is local and globally optimal solutions cannot be guaranteed in case of a large dataset. Due to the random selection of K-initial seeds, it decreases the quality of clusters. This paper proposes a MapReduce-based adaptive K-means clustering approach. The adaptive nature of our approach, improves the efficiency based on two key concepts. First, it selects the initial seeds that are spread throughout the large dataset via statistical testing. Second, it reduces the impact of the outlier on a large dataset and clustering large dataset using MapReduce framework. Our approach is adaptive for large dataset by adapting the size and nature of the dataset to make the process more robust and efficient. The experimental result shows that our approach improves the performance of clustering compared to earlier works.**

Keywords—Clustering; K-means; MapReduce.

I. INTRODUCTION

A large amount of data is being produced everyday globally. This will expand exponentially over the next few years which require existing analytical methods suitable to handle such huge volume of data. And moreover, the volume of the data is not the only concern; variety is also a big concern. The data being generated is from different sources which are mostly unregulated. The challenge is that extract knowledge from a large amount of data using data mining techniques. In this regard, clustering plays an important role to categorize data into a group of similar data.

K-means [1] is one of the popular methods in data mining because of its simplicity. This method randomly chooses initial centers and repeatedly assigns each input points to its nearest cluster centers by calculating the Euclidian distance between two and then recalculates cluster centers. But in case of a large volume of data K-means is not a suitable clustering method in terms of initial seed selection. In K-means global optimal solutions cannot be achieved due to random initialization.

There are various clustering methods built on K-means. K-means++ [2], which improves the initialization to reach the optimal solutions but it is inherently sequential nature. K-means++ requires K-passes over the data to select the initial set of centers which is not good for clustering large datasets. A parallel version of K-means++ is Scalable K-Means++ [3], which reduce the number of passes for better initialization.

Randomly selecting '*K*' initial seeds is not a good approach as it may compromise with the accuracy. And in cases where the dataset is very large and the number of Clusters needs is few, then selecting the '*K*' initial centroid in random manner would not be a wise choice.

In this paper, we propose an efficient adaptive method to improve the methodology for choosing '*K*' initial centroids. We select a sample size large enough to give an appropriate idea of the entire dataset. Then we select the '*K*' best fit points as the initial seeds. The objective is that the initial seeds would be well spread throughout the dataset and it reduces the number of iterations to select the initial seeds.

Secondly, we propose an approach to reduce the impact of outliers or noisy data points on the efficiency of the method. Centroids are always very sensitive to outliers. The presence of such points in the dataset causes the centroid to shift a great extent towards the outliers in the iterative method. In this paper, certain outlier data points are ignored while updating the centroid. All the computations are done in parallel using MapReduce framework to improve the efficiency of clustering. It demonstrates the effectiveness of parallel computations on large dataset.

The rest of the paper is organized as follows: Section II describes the related works. Section III presents some preliminaries. The proposed method is explained in Section IV. Section V shows experimental results and analysis with three datasets which illustrate the effectiveness of our proposed approach. Finally, the conclusion is drawn in section VI.

II. RELATED WORKS

Several works on clustering technique have been proposed in different kinds of literature. In different surveys several clustering methods have been described [4,5,6], however, we will enter into the deep insight of particular approaches that are

978-1-5386-1356-6/17 $31.00 © 2017 IEEE
DOI 10.1109/iNIS.2017.40

relevant to our work: K-means [1], K-means ++ [2], Parallel K-means [7].

K-means [1] is one of the fundamental methods in data mining and machine learning. In practical it is most popular approach because of its simplicity and effectiveness in different applications. K-means approach belongs to the class of combinatorial optimization also known as iterative relocation technique. K-means approach falls under the class of NP-Hard problem even for *K=2* [8]. This method minimizes a given clustering criterion by iteratively relocating data points between clusters until a locally optimal partition is attained. In this basic iterative approach, convergence is local and the globally optimal solution cannot be guaranteed. In case of large datasets convergence criterion takes more time for clustering and depending on dataset cluster quality can be low. In K-means++ [2] authors introduced different concept of initial seed selection. Their variation proposes to choose the first data point at random and then the following points are chosen with the weighted probability proportional to distance square from the closest point already chosen as a centroid. The problem with K-means++ is that for '*K*' centroids there will be K-initial iterations over the entire dataset which is time consuming. Also, this approach is not suitable for parallel processing of large dataset.

Parallel K-means [7] is the method for the process of a large-scale dataset in parallel. This approach satisfies major three criteria of speedup, scale up and size up. But since it was basically a MapReduce-based version of the K-means method, it didn't deal with the problems exist for large-scale datasets, such as the presence of noisy data points or outliers in the large dataset, which would affect the quality of clustering. Also, there were no mechanisms to ensure selection of good initial centroids.

In our proposed solution we proposed an approach to overcome the problem of selecting suitable initial centroids by statistical testing and dealing with outliers by Chauvenet's criterion [9], in addition, to make the method suitable for Hadoop framework [10,11].

III. PRELIMINARIES

In this section, we introduce the elementary background of our proposed work: sample size selection, Chauvenet's criterion, and MapReduce framework.

A. Sample size selection

Statistically, selection of sample size depends on the level of precision, confidence level and degree of variability.

Definition 1. The level of precision *e* is the range in which the true value of the sample size is estimated. The range is expressed in terms of percentage of points, ±5% [16].

Definition 2. Confidence level is the percentage of all possible samples that can be expected to contain the real dataset parameter [16].

Definition 3. Degree of variability refers to the distribution of attribute value in the dataset. In case of more heterogeneous dataset, to obtain a given level of precision a larger sample size is required [16].

Definition 4. Z-score is the number of standard deviations from the mean of data point. It can be defined as $z=(x-\mu)/\sigma$, where z is the z-score, x is the value of point, μ is the mean and σ is the standard deviation [16].

A sample size n_0 can be defined as [16]

$$n_0 = \frac{z^2 P(1-P)}{e^2} \tag{1}$$

where, z is the z-score, derived from the confidence level. P is the variability and e is the level of precision.

TABLE I. Z-SCORE

Confidence level	*z-score*
90%	1.645
95%	1.96
98%	2.326
99%	2.576

z-score values for corresponding confidence level are shown in Table I.

A given sample size provides proportionately more information for a small dataset than for a large dataset. So the new sample size n' can be defined by Cochran's correction formula [16] as

$$n' = \frac{n_0}{1 + \frac{(n_0 - 1)}{N}} \tag{2}$$

where, N is the size of the dataset.

Definition 5. Compactness is the average distance between every pair of data points in a cluster [5]. Compactness *CP* can be defined as

$$\overline{CP}_k = \frac{1}{|n_k|} \sum_{x_i \in n_k} |x_i - W_k| \tag{3}$$

where, n_k refers to the points present in the k^{th} cluster, W_k is the k^{th} centroid and x_i is the point belongs to n_k.

Compactness in global measures that is the average of all clusters can be defined as

$$\overline{CP} = \frac{1}{K} \sum_{i=1}^{K} \overline{CP_k} \tag{3a}$$

where, $\overline{CP}_k$ is the compactness of cluster k [5].

Definition 6. Fitness is equivalent to the sum of the distance of all points from their corresponding cluster centroid and then divided by the total number of points exist in the dataset. It can be defined as

$$\overline{FT} = \frac{1}{|N|} \sum_{i=1}^{|N|} |x_i - w_{ik}| \tag{4}$$

where, N is the number of points in the dataset, x_i is the i^{th} point in the dataset and w_{ik} is the centroid for the cluster which contains the point x_i.

B. Chauvenet's criterion

Chauvenet's criterion [9] is used to detect the outlier points in a dataset. Outlier points are detected for n number of observations. In this method mean μ and standard deviation σ are calculated. If n*erfc (| point x_i - μ | / σ) < ½ then it rejects the point x_i. A data point within a stipulated cluster would be identified as an outlier based on its Euclidean distance from the centroid.

C. MapReduce framework

To show the effectiveness of parallelism, MapReduce [12, 13, 14] paradigm is used in our proposed method. Google proposed MapReduce framework. It is a parallel programming model to process the large volume of structured, semi-structured and unstructured data. This programming model has three functions: mapper, combiner and reducer. Mapper function generates the intermediary *<key, value>* pair by dividing the dataset into a different number of blocks. Hadoop Distributed file system is used to partition the large dataset into different blocks. All the intermediary *<key, value>* pairs are combined and sorted by Combiner function according to the key. Reduction of the intermediary *<key, value>* pair is done by Reducer function. This function generates unique <key, value> pair as an output.

IV. MapReduce-based Adaptive K-means Clustering

In this paper, we propose MapReduce-based adaptive K-means clustering approach for clustering of a large dataset. Iteration '*K*' times over a huge dataset of might fetch accurate result but will be very time-consuming.

Our solution aims to be more efficient and adaptive while working with large volume of data. Working with a scenario where we need a fixed number of clusters '*K*' and where *K<N*, *N* is the total number of points in the dataset. Selection of '*K*' initial seeds at random leaves a high probability that the initial centroids are not well spread throughout the dataset. This takes a higher number of iterations to reach the desired result and sometimes highly impacts the accuracy of cluster formation. Therefore, we take a larger number of points, say *K*M* which would improve the probability of points being better spread throughout the dataset. Then we would select the best fit points as our initial centroids. Criteria for best fit points would be that the centroids should not lie much closer to one another. Therefore, we would start merging close points until we are left with '*K*' seed points.

Another modification we propose is to make the procedure more adaptive to deal with outliers or noisy data points. The presence of outlier or noisy data points within a cluster can cause the centroid to shift towards the outlier. Our aim is to reduce the impact of outliers on the movement of cluster centers. This would be done by detecting outlier points, present within individual clusters and ignoring them while finding the next centroid of that cluster.

All the computations are done in MapReduce framework for the tasks and data parallelism of our proposed work. Our proposed method is divided into different tasks: M-factor initialization, selection of *K* best-fit points from *K*M* points, clustering with respect to *K* best-fit points and reducing the impact of outliers.

A. M-factor initialization

In our proposed approach instead of selecting K initial points, we select *K*M* initial points uniformly at random. Selection of a large number of points will improve the probability of having well spread out the distribution. This will give a better precision of most suitable points for initial seeds. We select *K*M* points from the entire dataset as our sample. With the increase of sample size, accurate decisions are taken from the sample.

Since the nature of the dataset is unknown, in our proposed method we consider variability *P* value is 0.5, which is the maximum value of proportion. Equation (1) can be redefined as

$$n_0 = \frac{z^2\,(0.5)^2}{e^2} = \frac{z^2\,(0.25)}{e^2} = \frac{1}{4}\left(\frac{z^2}{e^2}\right) \tag{5}$$

By the equation (2) and (3), we can define n' as

$$n' = \frac{z^2 N}{z^2 + 4e^2\,(N-1)} \tag{6}$$

We consider n' to be a good sample size for the selection of initial seeds. In our proposed approach we increase '*K*' number of initial centroids by '*M*' times. Therefore, we select '*K*M*' points which are equal to n', and equation (6) can be redefined as

$$K * M = \frac{z^2 N}{z^2 + 4e^2\,(N-1)} \tag{7}$$

$$M = \frac{1}{K}\left(\frac{z^2 N}{z^2 + 4e^2(N-1)}\right) \tag{8}$$

*Lemma: Inference obtained from the large number of sample points 'K*M' becomes close to the expected result 'K' and try to become closer as many as sample points 'M' are chosen.*

Proof: Let $P_1, P_2, P_3, \ldots, P_n$ are randomly taken value from sample space, where 'n' is the number of sample points. Consider μ is the expected result. According to the l*aw of large numbers*,$\bar{P}_n = \frac{1}{n}(P_1 + P_2 + \cdots + P_n)$ where, $\bar{P}_n$ is the mean or sample average. $\bar{P}_n$converges to the expected result that is $\bar{P}_n \to \mu$ for $n \to \infty$. Therefore, for '*K*' no. of randomly selected centroids, selection of' *K*M'* no. of points will give a well-spread distribution of points. For example consider a large dataset with the value of parameters shown in Table II. By the equation (8), we can compute *M* is 77 for the cluster number *K* is 5.

TABLE II. Dataset Parameter

Parameter	***value***
Size	500000
Confidence level P	95%
z-score	1.96
level of precision e	0.05
Required number K	5

B. Selection of K best-fit points from K*M points

We select only *K* number points to find the K-best fit points from '*K*M*' point. Therefore, we will start with merging the points which are close to one another and assign weight to the merging point. We repeat merging process based on weight, with the closest points until *K* number points are left.

Consider the two points, $P_i(x_{i1}, x_{i2}, ..., x_{ik})$ and $P_i(x_{j1}, x_{j2}, ..., x_{jk})$. Initial weights of these two points are *m* and *n* respectively. After merging these two points by considering their weight, the merging point is P_{ij}. Therefore,

$$P_{ij} = \frac{(m(x_{i1}) + n(x_{j1})}{m+n}, \frac{(m(x_{i2}) + n(x_{j2})}{m+n}, , \frac{(m(x_{ik}) + n(x_{jk})}{m+n} \quad (9)$$

C. Reducing the impact of outlier

The presence of outlier or noisy data points within a cluster can cause the centroid to shift towards the outlier. Our solution would be done by ignoring outlier points present within individual clusters while finding the next centroid of that cluster. In our proposed approach, first we assign each point to its closest cluster center and then the outliers using Chauvenet's criterion. The points would be classified as an outlier based on its Euclidian distance from the present centroid. The points which are identified as an outlier within a cluster would be marked and the value of these points won't be considered while calculating the new cluster center. Therefore, the movement of the centroid would not be affected by points which are very far away from most other points.

D. Proposed Algorithm

We present a parallel implementation of our proposed clustering method using MapReduce framework.Algorithm 1 shows the initial seed selection algorithm. We would take a different approach for initialization. We would select *K*M* initial seeds instead of selecting *K* initial seeds. Then we would keep merging the closer points till we reach *K* points. The intuition behind the algorithm is simply that a higher number of points gives us a better probability of well-dispersed initialization.

Algorithm 1 *K-initial seed selection*

Input: k, m /* k is the number of cluster and m is the multiplication factor */

1: Number of Initial Seeds ← k
2: Multiplication Factor ← m
3: Select k*m points from the Dataset uniformly at random
4: len←k*m
5: d←dimensions
6: store[len][d] /*Store all the selected points
7: update[len] /*Initialize to Zero*/
8: Count ←len
9: while count>k
a: Find the closest Points. Store[x] and store[y]are the closest points.
b: for k← 0 to d
store[x][k]← (store[x][k]*update[x])
+(store[y][k]*update[y])/(update[x]+update[y])
c: end for
d: update[x]++
e: remove store[y]
f: count--
10: end while
11: Remaining K points are the final initialized points.

Algorithm for *Mapper* function is shown in Algorithm 2. The main objective of this algorithm is to compute the distance of data points from the current centroids in a parallel manner. The entire dataset is stored in HDFS in *<key, value>* pair format. Every *Map* task create an array comprising of the presently selected centroids that are considered as global variables. Assign each data points to its closest centroid by calculating the distance between the present centroids and each data points. Output *<key', value'>* pair is the index of the assigned centroid (*key'*) and the data point (*value'*).

Algorithm 2 *Mapper Function*

Input: Centroids[k] /* An array consisting of 'k', selecting centroids from the datasets. <key, value>represents records in a dataset, where key is the offset of the record ID and value is the string comprising of all the attributes of the records.*/

1: Min_Dist. ← ∞
2: index ← -1
3: For i← 0 to n-1
4: Do for j← 0 to k
5: Dis ←Euclid_Dist.[Record[i], Centroid[j])
6: If dis<Min_Dist.{Min_Dist. = dis; index=j}
7: End
8: Output <key', value'>pair.

Algorithm 3 describes the algorithm for *Combiner* function. The *Combiner* function is being used to work with intermediate data from each *Map* Task. It will give a partial sum of all the points which have been assigned to the same cluster and the total number of points allotted to every cluster center by the same *Map* task. Also, we create an array which would store the distance of every point from its assigned centroid.

Algorithm 3 *Combiner Function*

Input: <Key k, Value v> and Centroids []/* Value v stores all attributes of a point assigned to a particular Key k*/

1: array_points[n]; /**n* number of attributes of the records*/
2: count← 0
3: while v.next() <> Null
4: do
5: fori← 0 to x-1
6: Add each attribute of v.next() value to array_points[i]
7: end for
8: count++
9: end while
10: Point_ dist← Euclid(array_points,Centroids[key])
11: key'← key
12: value'← string of array_points andarray_dist
13: output <key', value'>

Algorithm for *Reducer* function is shown in Algorithm 4. The *reducer* function accepts the list of data points and their distance from the centroid. The Chauvenet's function is called by the reducer task. Points which fail to meet the criteria will be considered as outliers. These points will not be considered at the time of new centroid calculation. Once the new centroids have been updated, they will be sent to the all the *Mapper* functions.

Algorithm 4 *Reducer Function*

Input: <*key, List(W)*> /**key* is the Index of the cluster, *List(W)* is the list of nodes and their distance from centroids. */

```
1: Count← 0, j← 0 /* n is the number of dimensions and j is
the number of points satisfy Chauvenet's criterion */
2: while list.next() <> Null
3: do
4: len ← w.length() /* number of points assigned to that
centroid */
5: for i← 0 to n-1
6: Add value of attributes of w.next() to array[Count][i]
7: dist[i] ←w.next()   /* distance from centroid */
8: end while
9: K[]←chauvnets(dist[],len)
10: if k[i] <> 'large value'
11: array_sum[i] += array[i]
12: j++
13: enf if
14: Divide array_sum[1…..n] by j which will give coordinated
from new centroid
15: key'← key
16: value'← string of new centroids' co-ordinates
17: Output <key', value'>.
```

Algorithm 5 shows the algorithm for Chauvenet's criterion function. This function is invoked by the *Reducer* function. It receives the list of data points and their distance from the current centroid. Outliers would be determined based on the distance from the existing centroid. Only the points whose distances are greater than average distance would be checked because our aim is to eliminate the points which are very far from the centroid, not the ones extremely close. The points which are tested positive for outliers would be marked.

Algorithm 5 *Chauvenet's criterion Function*

Input:array_dist [], count

```
1: flag ← 0
2: do
3: m ← mean(array_dist[],count)
4: sd← stddev(array-dist[], count, m)
5: for  i← 0 to count-1
6:    Double p← array_dist[i]
7:     x←Math.abs((p-m)/sd)
8:    Double cal← n*(Erf.erfc(x))
9:    if n.x< 0.5 && p>m
10:      reject array_dist[i]
11:      Index ←i
12:      flag←1
13:   end if
14:   else
15:    Keep array_dist[i]
16: end for
17: remove rejected data and update count value
18: end do
19: while flag <> 0
20: Output <array_dist[], count >
```

V. Experimental Results Analysis

The performance of our proposed approach is evaluated with two machine learning datasets that are collected from UCI Machine Learning repository [15].

Experiments are done in Hadoop clusters using Java version 7 and with PCs of Intel(R) Core(TM) i5- 7600processor @4.10GHZ and 8GB RAM in Ubuntu 16.04.3 platform.

We mainly analyse the performance of the method based on two different criteria: Cluster quality and computation time. Earlier work parallel K-means [7] is scalable and showed a linear speed up. Therefore, in order to measure performance with respect to time, we consider the number of iterations taken to reach the converging condition.

Table III shows that for all the three datasets, our proposed method takes the lower number of iterations to reach the convergence condition than that of the earlier parallel K-means method. For the *Breast Cancer* dataset previous parallel method takes 10 iterations to reach convergence condition whereas our method reaches the convergence condition in 7 iterations. The Diabetes dataset and Heart Disease also follow the same trend, while the previous method takes 16 and 18 iterations respectively to converge, but the proposed method only takes 9 and 10 Iterations respectively to reach a convergence condition. Thus from this result, we can establish that under similar given conditions and same data, the proposed solution will reach convergence condition more quickly.

In order to evaluate the quality of clusters, two measures are taken, Compactness and Fitness. Compactness [5] in global measures is used for the validation of cluster. Clustering method is effective if it creates the clusters with instances that are nearby or analogous to one another. Fitness is equivalent to the sum of the distance of all points from their corresponding cluster centroid and then divided by the total number of points exist in the dataset.

A low compactness symbolizes that clusters are compact in shape, and points are close to the center. Fitness is a measure of the distance of the points from its cluster centroid. The presence of outliers will have a greater impact on compactness than fitness. Table III shows that the compactness and fitness for Diabetes dataset are 50.5 and 59.67 respectively using the earlier parallel K-means method but our proposed approach gives a compactness and fitness of 47.79 and 56.63 respectively. Here we observe that proposed solution gives more compact and well-formed clusters. The almost similar trend can be observed for the Heart disease and Breast cancer datasets. The quality of clustering is better in three cases. Accurate decisions can be taken from the clusters to diagnose particular disease. Therefore, our proposed approach will give better results.

VI. CONCLUSION

Over the years there have been significant researches in the field of data clustering. With the advance of World Wide Web, we are generating a large amount of data every day. Existing clustering methods face problems while working with such large-scale data. In this paper, we proposed the MapReduce-based adaptive parallel approach for clustering of the large dataset. The aim is to make the algorithm more adaptive to deal with two problems which might arise in clustering of large-scale data: initial seed selection and rejection of outliers. The proposed method is evaluated on several datasets and it shows our method performs better than earlier works.

REFERENCES

[1] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*,vol. IF-28, no. 2, pp. 129–137, Mar. 1982.

[2] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding,'' in Proc. 18th Annu. ACM-SIAM Symp. Discrete Algorithms *Soc. Ind. Appl. Math.*, New Orleans, LA, USA, pp. 1027–1035, 2007.

[3] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii,"Scalable k-means++," PVLDB, 5(7):622–633, 2012.

[4] E. Chandra and V.P. Anuradha, "A survey on clustering algorithms for data in spatial database management systems," International Journal of Computer Applications, 24(9):19-26, 2011.

[5] Fahad A, Alshatri N, Tari Z, Alamri A, "A survey of clustering algorithms for Big Data: Taxonomy and empirical analysis," IEEE Transactions on Emerging Topics in Computing, 2(3):267–279, 2014.

[6] Xu R, Wunsch D," Survey of clustering algorithms," IEEE Transactions on Neural Networks. 2005 May; 16(3):645–78.

[7] W. Zhao, H. Ma, Q. He, "Parallel K-Means Clustering Based on MapReduce," Cloud Computing, Lecture Notes in Computer Science,Chapter IV, vol. 5931, pp. 674-679, 2009.

[8] M. Mahajan, P. Nimbhorkar, K. Varadarajan. "The Planar k-Means Problem is NP-Hard," Lecture Notes in Computer Science 5431: 274-285, 2009.

[9] Barnett, Vic and Lewis, Toby, Outliers in Statistical Data, 3rd edition. Chichester: J.Wiley and Sons, 1994.

[10] Alam A, Ahmed J, " Hadoop Architecture and Its Issues," Proceedings of International Conference on Computational Science and Computational Intelligence, pp. 288–291,2014.

[11] Dittrich J, Quiani-Ruiz J, "Efficient Big Data Processing in Hadoop MapReduce," Proceedings of the VLDB Endowment 5(12):2014–2015, 2012.

[12] Dean J, Ghemawat S, "MapReduce: simplified data processing on large clusters". Commun ACM 51(1): 107–113, 2008.

[13] Dawei Jiang,Beng Chin Ooi,Lei Shi,Sai Wu,"The Performance of MapReduce: An In-depth Study", Proceedings of the VLDB Endowment, Vol. 3, No. 1,2010.

[14] A. Srinivasan, T. A. Faruquie, and S. Joshi, "Data and task parallelism in ILP using MapReduce," Machine Learning, vol. 86, pp. 141-168, 2011.

[15] UCI Machine Learning repository: < http://archive.ics.uci.edu/ml/>

[16] G. D. Israel, "Determining sample size," Program Evaluation and Organizational Development, IFAS, University of Florida, 1992. [Online]. Available: http://edis.ifas.ufl.edu/ pd006.

TABLE III. EXPERIMENTAL RESULTS

Dataset	***Parallel K-Means* [7]**			***Proposed Method***		
	Compactness	***Fitness***	***Iterations***	***Compactness***	***Fitness***	***Iterations***
Breast Cancer [15]	3.92	5.69	10	2.78	4.52	7
Diabetes [15]	50.5	59.67	16	47.79	56.63	9
Heart Disease [15]	33.31	33.50	18	31.24	31.43	10