International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 5| ISSN : 2456-3307

Partitional Based Clustering Algorithms on Big Data Using Apache Spark

N. Sukumar[1], Prof. A. Ananda Rao[2], Dr. P. Radhika Raju[3]

*[1]M.Tech Scholar, Department of CSE, JNTUA College of Engineering, Ananthapuramu, Andhra Pradesh, India

[2]Professor, Department of CSE, JNTUA College of Engineering, Ananthapuramu, Andhra Pradesh, India

[3]Ad-hoc Assistant Professor, Department of CSE, JNTUA College of Engineering, Ananthapuramu, Andhra Pradesh, India

ABSTRACT

Apache Spark, a framework similar to the Von Neumann architecture. It has an efficient implementation of in-memory computations and iterative optimization is processed to analyze large volume of data. Data captured at high velocity and from variety of different sources known as Big Data. Such big data can be partitioned and clustered, based on parameters of the data. The parameterized clusters are enhanced under clustering algorithms for better outcomes. In this paper the current approach optimize computation over random sampling algorithms, where empirical evidence exhibit the significant change in computation of partition algorithms. Computations can be carried out in iterative procedure for wide variety of datasets retaining an abstraction known as Resilient Distributed Datasets (RDDs).

Keywords :Clustered Technique, Iterative Computation, Apache Spark, Membership Matrix, PySpark

I. INTRODUCTION

Many end-to-end big data frameworks process voluminous data like Apache Hadoop, Apache Flink, Apache Storm, Apache Samza and Apache Spark, among all these open source frameworks Apache Spark yields better computation results when compared to other frameworks[1]. Even in-memory computations workload can be carried on both batch processing and stream processing. For example, fraudulent banking transaction can be figured out whether the genuine customer is performing the transactions irrespective of the geographical locations.

In conventional approach, Fuzzy C-Means (FCM) clustering technique partition the data is divided into clusters based on the inter related data parameters. Here, the actual inconsistency arises with soft clustering. Overlapping of the cluster center with neighbouring clusters empirically results, the converse technique of the FCM algorithm[2]. A lot of partitional based algorithms like Possibilistic C-Means (PCM), Fuzzy Possibilistic C-Means Algorithm (FPCM) are proposed for computing entire big data, where the results expected are not relevant[1].

This paper presents enhanced Scalable Random Sampling with Iterative Optimization Fuzzy C-Means algorithm (SRSIO-FCM) [1] which is an adopted form of classic clustering model. Vagueness in calculating cluster centers and extending the approach to address very large data results overlapping of clusters. FCM clustering like the Literal Fuzzy C-Means with Alternating Optimization (LFCM/AO) is designed to opt the number of cluster centers randomly by the user, On-line Fuzzy C-Medoids (OFCMD) and History-based Online Fuzzy C-Medoids (HOFCMD) [1, 2].

In current scenario, computations are carried out on big data with respect to data sets where clusters are derived with centroids C by choosing randomly in the initial stage. Such that data points belong to one cluster are more similar than that of other clusters, this process completely enhanced in R analytical studio. Further, iteratively optimized by holding the threshold value on the membership matrix in PySpark, python wrapper for spark.

The rest of the paper organized as follows. In section II overview about various clustering algorithms is presented. Section III, presents about Apache Spark and working of scalable algorithm. Section IV shows the empirical results and performance comparison among the frameworks MapReduce(MR), Spark and Message Passing Interface (MPI). Finally, section V concludes some possible enhancements and future work.

II. RELATED WORK

Many approaches are available for clustering large databases like Clustering Using Representatives (CURE) [1]which is less sensitive towards the outliers, Clustering Large Applications (CLARA) [1] the quality among every object is measured over all the data set is subject as one objective function.

Over years, various types of algorithms have been proposed in multi-dimensional clustering. Formally, minor changes in objective function detect insignificant types of clusters in FCM clustering like Fuzzy C-Elliptotypes (FCE) to obtain linear clusters and Fuzzy C-Shells (FCS) algorithm to derive circles. Fuzzy C-Rectangular Shells (FCRS) algorithm is designed to detect rectangular clusters.

Bezdek proposed fuzzy clustering algorithm has been designed to extract the membership degree from data sets gathered are divided into groups using Eq. (1), membership degree is represented by U and cluster center by V, splitted in to k number of partitions, cluster centers are classified from (v_1, v_2, v_3......, v_n) and the objective function related as

$$J_m(U,V) = \sum_{k=1}^{n}\sum_{i=1}^{c} (U_{ik})^m \left\| x_k - v_i \right\|^2 \qquad (1)$$

$$1 \le m \le \infty$$

Kaufman and Rousseuw developed Partition Around Mediods (PAM) which is robust in context of CURE and K-means. Bezdek proposed the FCM clustering algorithm (FCM) [3,4] to partition the data points and to minimize the dissimilarity measure. In partition clustering algorithm, data points p are splitted into k partitions, centroid c_i, mean of the cluster mi and each partition denotes a cluster, such partitions are derived on an objective function to minimize square-error criterion,

$$E = \sum_{i=1}^{k}\sum_{p \in ci} \left\| p - m_i \right\|^2 \qquad (2)$$

However, in the proposed, to reduce the inconsistency data cleaning is performed on the initial data sets to avoid the dissimilarity measure. Algorithms are implemented in multi-level to re-estimate features of parameters in hard clustering. The constraints like limiting the number of clusters for larger data sets, maximum number of iterations to obtain the final data set, computing the threshold value inter related with the size of data sets enhance to reduce lazy evaluation or long computation time.

III. PROPOSED WORK

Most of the cluster centers are overlapped just because of randomly choosing the data points in the data sets. Here after, the enhanced sampling method

RSIO-FCM [1] forms the cluster centers for various data parameters, over all big data. In fact those data centers are not relatively projecting expected results. Thus, the major problem in choosing the data point over the data sets can be overcome by sampling method, SRSIO-FCM [1, 4] along with Apache Spark by implementing iteratively optimizing technique.

Initially, the big data is classified into data subsets. Figure 1 represents the framework of the work flow, choosing the data point randomly in the data set is as similar to the previous sampling methods. Upon choosing the data point for the first cluster say V_1, the membership matrix $U_{i,j}$ is calculated on the data set parameters for the next iteration of the cluster center. Any deviation in the parameters of the initial cluster V_1 and the secondary cluster V_2 can be stipulated by considering related parameters which are common to both clusters. After clustering the two data sets the relevant change of the both clusters and change in the membership matrix taken into consideration for iteration. Further, if the process is iterated for rest of the data sets the information related to membership matrix leads as a bottleneck for optimize computations as of iterations continues over the sets.

At this point, as the number of iterations carried out on data sets shows a direct impact on membership matrix and its size. The information retained by the membership matrix leads to insignificant results due to its size, in certain cases equally to the partitioned data sets. As of the iterative computations [7] are carried over in-memory as similar to Von Neumann architecture, in fact the data is brought on to the main memory for faster computations. The parameterized information of the data sets increases which leads to further workload of the process. To overcome this problem, threshold value with respective to the data sets is considered as a objective function for evaluation. Further, the cluster centers of remaining data sets will also be affected with threshold value. Here the termination criteria, ϵ a predefined constant taken as $\epsilon = 10^{-3}$ [1].

Cluster centers with respect to termination criteria,

$$\|V' - V\| < \epsilon$$

Objective function to compute cluster membership, the process initiated only for the data sets with certainty to produce and extend the partition and compute the cluster centers V.

i.e., $U_{i,j} = 1$.

$$Ui,j = \frac{\|x_i - V_j\|^{\frac{-2}{m-1}}}{\sum_{k=1}^{c} \|x_i - V_j\|^{\frac{-2}{m-1}}}, \forall\, i,j \quad (3)$$

Data sets that met the termination value with respect to data set or threshold value related to the predefined constant, inter related with the data sets are posted as the final solution sets are cluster centers. The framework tries to resolve the issues like out of memory, time out for iterative optimization using termination criteria and projecting final clusters with Large Data volume (LD). Lazy evaluation in case of stream input data while computations are already in progress.

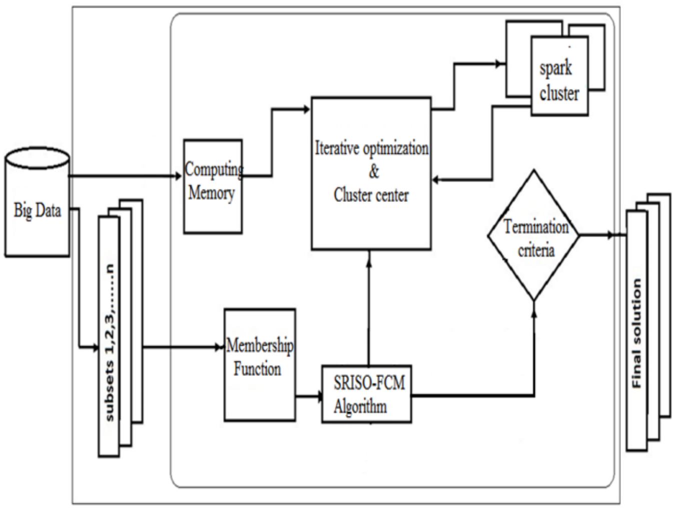


Figure 1: Overview of Scalable Clustering Framework

A step-by-step procedure of the scalable random sampling with iterative optimization [8] for the initial cluster computation is listed below, where data set X, cluster center V_i, centroid C_i and membership matrix $U_{i,j}$. The theme of algorithm focus more on computations of membership matrix and calculating cluster centers.

ALGORITHM: Scalable Random Sampling with Iterative Optimization - FCM

Input: Data sets X
Output: Final Cluster

```
1. Function partition_Dataset(V, X, C)
2. Load data points $X_i$
3. Partition ($x_1, x_2, x_3, x_4, \ldots, x_n$) $\leftarrow$ X
4. Calculate the cluster centers on initial data
   point '$x_1$'
           if(m >1)
   {
           FCM applied on the data sets
           }
5.         else
           goto step:6
6. Calculate membership $U_{i,j}$ of cluster
   Compute the membership for initial
   iterations $U_{i,j}$ // parameters evaluation of
   matrix.
   //not for all values of i,j.
           {
                   if (MI' = $\sum X_i/2$ )
   // MI always less than the number data points
   in  the cluster
   }
7.  Return MI, $U_{i,j}$.
```

IV. RESULTS

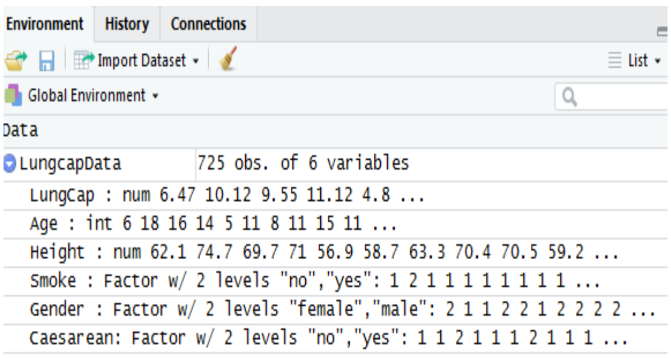
A. Environment Setup:

The implementation of the SRSIO-FCM algorithm is carried on Spark 2.3 along with python for iterative optimization. PyCharm IDE is integrated with the Spark and R analytical data. Experiments carried out on a cluster executed on servers connected with 32 gega bytes of primary memory, 2.40 GHz Intel® Core CPUs with maximum memory bandwidth approximately of 25.6 GB/s with Error-Correcting Code (ECC) memory.

Data servers are occupied with the small and large clusters [8] concurrently for in-memory computations during initial clustering phase. The Spark framework automatically handles the flow of computation with huge memory bandwidth.

B. Evaluation Results:

To inspect the performance of algorithm, initially observations related to the group smokers with six different parameters are summarized on R studio, figure 2 represent the parameterized data before applying the sampling method. The data with the null values is truncated for better results, this process is called data cleaning. Test data for data sets are not taking into considerations, in fact the reduction on initial data set yields inappropriate results by truncation of parameters. Table 1 shows the raw data that is gathered from the distributed cloud points and also from Comprehensive R Archive Network (CARN).



```
LungcapData        725 obs. of 6 variables
  LungCap : num 6.47 10.12 9.55 11.12 4.8 ...
  Age : int 6 18 16 14 5 11 8 11 15 11 ...
  Height : num 62.1 74.7 69.7 71 56.9 58.7 63.3 70.4 70.5 59.2 ...
  Smoke : Factor w/ 2 levels "no","yes": 1 2 1 1 1 1 1 1 1 1 ...
  Gender : Factor w/ 2 levels "female","male": 2 1 1 2 2 1 2 2 2 2 ...
  Caesarean: Factor w/ 2 levels "no","yes": 1 1 2 1 1 1 2 1 1 1 ...
```

Figure 2. Parameterized Data before Clustering

Table 1 represents the information related to the three data sets D1, D2, D3 where the Membership Information (MI), flat file size and binary file size in gigabytes, are widely varied to compare the results with other frameworks like Apache Hadoop with MapReduce (MR) technique and Message Passing Interface (MPI).

Table 1: Input Data Sets

Data Sets	MI	Flat Flie Size	Binary Flie Size
D1	2048	2.3 GB	1.2 GB
D2	3276	4 GB	2.5 GB
D3	4096	3.2 GB	2.2 GB

The clustering with respect to the parameters which enhances scalable results for forming membership matrix. The cluster with parameters enables membership matrix to compute an accurate centroid C_i and cluster centers for further. Figure 3 shows the membership information with respect to parameters.

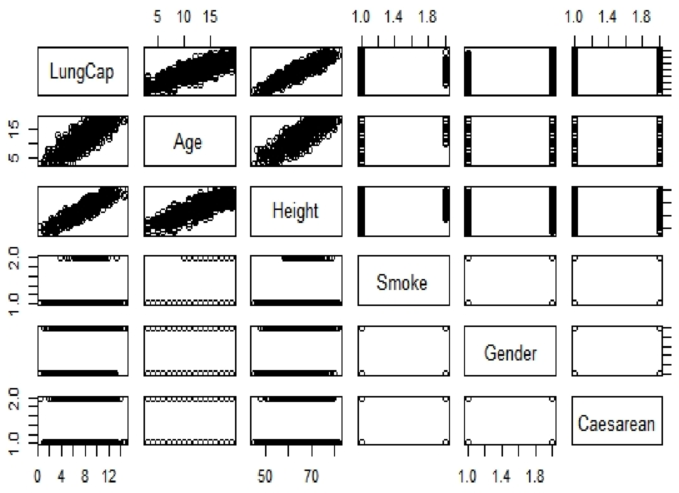


Figure 3. Membership Information with respect to Parameters

To evaluate the scalability, Spark program is initiated along with python in PyCharm to reschedule the server in case of slow response or server crashing cases. The performance is compared with other big data frameworks like Hadoop MapReduce and MPI. MPI generates large amount of intermediate data which cause the abrupt expansion of computation memory and leads to a hassle. Such cases can be easily handled with MI matrix objective function.

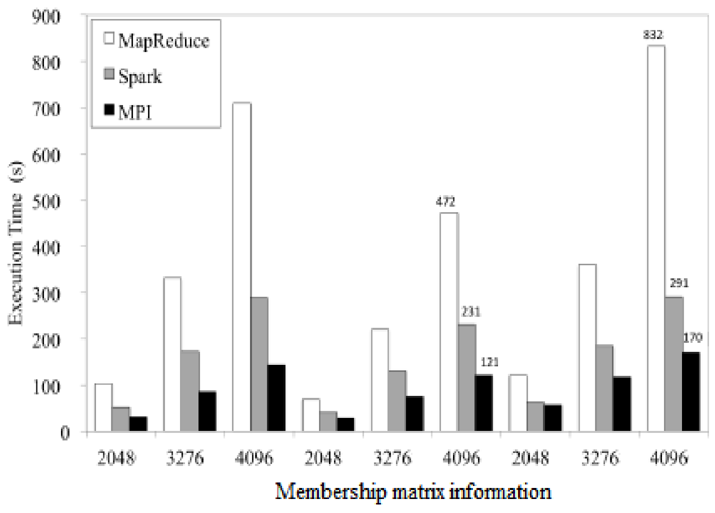


Figure 4. Performance Evaluation

C. Performance Comparison:

To compare the performance, conventionally several numerical analysis software's like R, WEKA, FreeMat an open-source MATLAB are available. The current scenario adopts R studio to compare the number of clusters formed with different data sets D1, D2, D3 varied with different input file size and binary file size as listed in table 1.

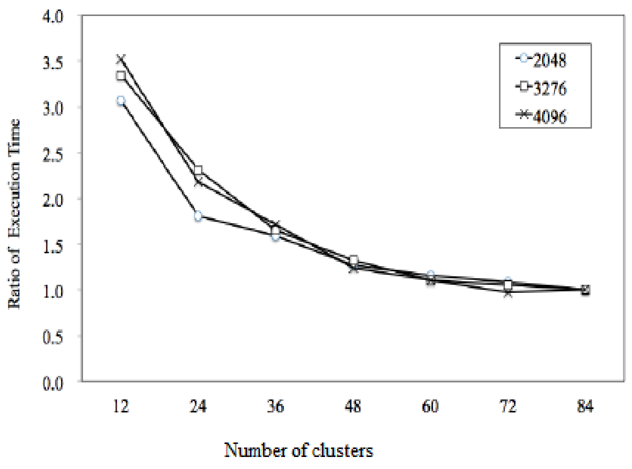


Figure 5. Scalability of Proposed Algorithm SRSIO-FCM

The above graph shows the results, x-axis is the number of clusters and y-axis plotted with the ratio execution of time in seconds. The graph illustrates the decrease in the execution time after iteratively optimizing data sets. However, one more notable aspect is that, data sets with different binary file sizes lead to form a similar number of cluster centers and saturates at one point of time.

V. CONCLUSIONS & FUTURE WORK

This proposed work concludes in such a way that comparison of results with higher data sets without effecting the quality of the cluster. The key role of the eliminating the membership information at certain point is to avoid inappropriate results and workload to the next cluster centers.

In the terms of future work, the two aspects to be considered mainly: Primarily, optimizing the proposed algorithm SRSIO-FCM, reliability and fault-tolerance capabilities over the objective function achieves good scalability on various size. Secondarily, dynamic scheduling of tasks along with RDD feature of Apache Spark at abrupt failure while long duration computations. Just in case, which require ample amount of computation time again and again.

VI. REFERENCES

[1] Neha Bharill, Aruna Tiwari and Aayushi, "*Fuzzy Based Scalable Clustering Algorithms for Handling big data using Apache Spark*". IEEE Transactions on 2016.

[2] R. Krishnapuram, A. Joshi, O. Nasraoui, and L. Yi, "*Low complexity fuzzy relational clustering algorithms for web mining*," IEEE Transactions on Fuzzy Systems, vol. 9, no. 4, pp. 595–607, 2001.

[3] T.C. Havens, J.C. Bezdek, C. Leckie, L.O. Hall and M. Palaniswami, "*Fuzzy c-means algorithms for very large data*," IEEE Transactions on Fuzzy Systems, pp.1130-1146.

[4] J.F. Kolen and T. Hutcheson, "*Reducing the time complexity of the fuzzy c-means algorithm*," IEEE Transactions on Fuzzy Systems, vol. 10, no. 2, pp. 263–267, 2002.

[5] Jain Fu, Junwei Sun, Kaiyuan Wang, "*SPARK—A Big Data Processing Platform for Machine Learning*" 2017. IEEE Conference.

[6] Swarndeep Saket J, Dr. Sharnil Pandya,"*An Overview of Partitioning Algorithms in Clustering Techniques*". IJARCET June, 2016.

[7] Sumin Hong, Woohyuk Choi, Won-Ki Jeong, "*GPU in-memory processing using Spark for iterative computation*", 2017 17th IEEE/ACM.

[8] "HaLoop: Efficient Iterative Data Processing on LargeClusters", Yingyi Bu, Bill Howe, Magdalena Balazinska, Michael D. Ernst, 2010.