ISSN No. 0976-5697

Volume 2, No. 2, Mar-Apr 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Performance based Efficient K-means Algorithm for Data Mining

Rajesh Ahirwar
Department of Information Technology
Samrat Ashok Technological Institute
Vidisha, M.P., India.
rajesh.mtechit2011@gmail.com[1]

Mukesh Goyal
Department of Information Technology
Samrat Ashok Technological Institute
Vidisha, M.P., India.
mukesh.goyal23@gmail.com[2]

Narendra Kumar Kori
Department of Information Technology
Samrat Ashok Technological Institute
Vidisha, M.P., India.
kori_narendra86@rediffmail.com[3]

Abstract-Today, we are witnessing enormous growth in data volume. Often, data is distributed or it can be in the form of streaming data. Efficient clustering in this entire scenario becomes a very challenging problem. Our work is in the context of K-means clustering algorithm. K-means clustering has been one of the popular clustering algorithms. It requires several passes on the entire dataset, which can make it very expensive for large disk-resident datasets and also for streaming data. In view of this, a lot of work has been done on various approximate versions of k-means, which require only one or a small number of passes on the entire dataset. In our work has developed a new algorithm for very large data clustering which typically requires only one or a small number of passes on the entire dataset. The algorithm uses sampling to create initial cluster centers, and then takes one or more passes over the entire dataset to adjust these cluster centers. We have implemented to develop clustering algorithm for distributed data set. The main contribution of this paper is the implementation and evaluation of that algorithm. Our experiments show that this framework can be very effective in clustering evolving streaming data.

Key words- Data Mining; Clustering; Distributed k-means Algorithm; Search Engine Technique.

I. INTRODUCTION

Most of the initial clustering techniques were developed by statistics or pattern recognition communities, where the goal was to cluster a modest number of data instances which often located on a single machine. However today, we are witnessed an enormous growth in the amount of collected data. Additionally, we can also have data distributed in remote machines or we can have streaming type of data which is potentially infinite. Traditional clustering algorithms are inefficient because they were not designed to process large amounts of data or to operate in distributed or streaming environments.

Hence, developing fast and efficient clustering algorithms for massive or distributed or streaming data has been identified as challenging problem in the data mining community. Massive data sets are known as out of core data as they do not fit entirely in main memory. In this case, the run time of the algorithm is dominated by the cost of I/O operation or number of disk scan. Hence, ideally the clustering algorithm for massive data should be optimized for the number of scans over the disk [1].

Distributed data may reside in remote loosely-coupled machines. In this case, communication between the nodes may be a bottleneck. Hence, a distributed algorithm is required. The runtime will be dominated by the slowest communicating nodes. Hence, the goal of distributed algorithm should be to minimize the effort and time expended on communication. Streaming data is potentially infinite and it is not possible to store all the data. Hence, the algorithms for streaming data should preferably be one pass and on-line and they should also require small amount of memory for their execution. The requirement of small memory is targeted towards mobile devices or sensor network nodes. Our work employs the k-means clustering.

The K-means clustering algorithm was developed by Macqueen in 1967 and later improved by Hartigan. Bottou and Bengio proved the convergence properties of the K-means algorithm. It has been shown to be very useful for a body of practical applications. The k-means algorithm is not suitable for massive or distributed or streaming data. A problem with the k-means algorithm is that it makes a complete scan over the entire data for every iteration, and it requires much such iteration before converging to a quality solution. This makes it potentially very expensive algorithm to use, particularly for large disk-resident datasets. Furthermore, in the context of streaming data, it may not always possible to store the complete data. This makes k-means algorithm impractical to use for streaming data. For distributed data, one can use parallel k-means algorithm but this algorithm requires that nodes communicate for every iteration. This makes the parallel k-means very expensive to use in presence of communication delays among the nodes.

The communication problem is especially prominent in loosely connected remote machines. Also, the parallel k-means works on uniformly partitioned data. The nodes are said to posses' data imbalance if the amount of distributed data is not uniform at each node. In presence of such data imbalance, the runtime of parallel k-means algorithm will be dominated by the processing time of the node which contains maximum amount of data. Moreover, other standard practice for clustering distributed data is to download and merge all data in a single machine and then running a sequential clustering algorithm on it. But, downloading huge amount of data and running sequential algorithm is clearly not an efficient solution for clustering distributed data. Hence, a distributed clustering algorithm is required which optimizes the communication and can also operate in presence of data imbalance [2] and [3].

A number of algorithms or approaches focus on reducing the number of passes required over the data sets. These researches have been done in the context of clustering massive data or streaming data. However, these approaches only provide approximate solutions, possibly with deterministic or probabilistic bounds on the quality of the solutions. Also, there is no distributed algorithm available at present for clustering which optimizes communication and can work in presence of data imbalance and can provide exact results. Therefore, my work proposes efficient algorithm which can produce exact results as k-means algorithm for large out of core data, distributed data and streaming data. The paper explain the development and evaluation of a distributed fast and exact k-means algorithm (DFEKM) extending the concepts of fast and exact k-means (FEKM) [1] and [4].

II. BACKGROUND

In background of K-means we discuss only on improvements over k-means, single or a few pass clustering algorithms and parallel or distributed clustering algorithms, clustering algorithms for streaming data.

A. Scalable k-means Using Compression

In recent developed a single pass approximation of multi-pass k-means. This algorithm is initialized as ordinary k-means, after which it is repeatedly made to take as much data as it can, to fit into the main memory. The centers are then updated with points from the main memory. Then, the memory buffer contents are compressed in two steps. The first step, called primary compression, finds points that are close to the cluster they are currently assigned to and discards the point if it is within an estimated radius. Then, in second step, for each point a worst case scenario is set up by perturbing the cluster means within confidence intervals. If the points do not change their cluster membership with the perturbed mean then these points are also considered in the discard set. For rest of the data points, they do a secondary compression using k-means algorithm and store weighted points satisfying some tightness criteria. Some points which do not satisfy the tightness criteria are retained in the buffer. The algorithm updates the model every time it fetches new points with the retained set and weighted points and sufficient statistics from the discard set. The algorithm ends after one scan of the data set.

Further simplified this idea where they store sufficient statistics of all the points in memory buffer and then next time uses the new points and the sufficient statistics of the points stored in previous fetch. They performed extensive experiments with synthetic and real data set and showed that in many occasion Bradley and Fayyad's scalable k-means is slower than the traditional k-means. Furthermore, their simplified scalable k-means has been found faster in all the experiments [a] and [2].

B. BIRCH Using Sampling

BIRCH is another single (or a few) pass hierarchical clustering algorithm, proposed by Zhang and Ramakrishna. It incrementally builds an index tree as it scans the data. They referred this as cluster feature tree (CF Tree). Each node of the tree represents the sufficient statistics triplet (number, sum, sum-squared) of all the points under that node. A node is split if it represents too many points. BIRCH is an efficient one or a few pass hierarchical clustering algorithm which is developed for very large data sets. It produces approximate solution. One bottleneck of BIRCH is the size of the CF Tree. If the size of the CF Tree does not fit into the main memory then BIRCH may become inefficient [5].

C. Approximation Algorithm CURE

In previous developed an efficient hierarchical clustering algorithm "CURE" for very large databases. This algorithm uses random sampling and then it iteratively partitions the data and merges the closest cluster at each pass until it computes k clusters.

D. One Pass Approximation STREAM

One passes clustering algorithm for very large data sets or streaming data. This algorithm was inspired by CURE. They refer to this algorithm as "STREAM". The main premise of their algorithm is to use sampling and approximate facility location algorithm to open best *k* facilities. The k-median or k-center algorithm uses the facility location algorithm iteratively to converge to the best *k* facility locations as cluster centers. They show that their algorithm obtains better quality solution than k-means although k-means is occasionally faster.

E. Using Hoeffding Bound

A faster version (sub-linear) of k-means using sampling based on Hoeffding or similar statistical bound. The algorithm consists of a number of complete runs of k-means algorithm with sample where in every run; sample size is increased to maintain a loss bound with respect to the multi-pass k-means algorithm over complete data set. The goal here is to converge to a solution using a sample from the data set such that this solution is close to the solution of a multi-pass k-means algorithm by a predefined bound with a high probability [6] and [8].

F. Heuristics for Out of Core Computation

More recently, proposed to apply k-means algorithm to cluster massive datasets, scanning the dataset only once. Their algorithm splits the entire dataset into chunks, and each chunk can fit into the main memory. Then, it applies k-means algorithm on each chunk of data, and merge the clustering results by another k-means type algorithm. Good results are shown for a real dataset; however, no theoretical bounds on the results have been established [7] and [9].

G. Parallel Clustering Techniques

In recent proposed a technique called "Recursive Agglomeration of Clustering Hierarchies by Encircling Tactic" (RACHET). This technique is based on sufficient statistics. It collects local dendograms and then merges them to create a global dendogram. However, this needs to iterate until the sufficient statistics converges to the desired quality. Parthasarathy and Ogihara provided an algorithm where the distance metric is formed applying association rules locally.

Kargupta and his group applied PCA to do high dimensional clustering in a distributed fashion. A distributed clustering technique that involves creating local clusters, and then deriving global clusters from them. These algorithms however are not designed to produce exact solution for which it is required to scan over the complete data set [10] and [11].

H. K-Median in Sliding Window over Stream

Most recent developed an algorithm for maintaining k-medians in a sliding window over streaming data. They use the idea of exponential histogram (EH) data structure to enhance the algorithm presented. Here they especially address the problem of merging clusters using the EH data structure.

I. Better Approximation Algorithm for Streaming Data

Provide a constant factor approximation algorithm for k-median problem for streaming data which is one pass and uses poly-logarithmic storage space. This algorithm overcomes the increasing approximation factors in different passes in "CURE" algorithm.

J. Clustering Evolving Streaming Data

Han, Aggarwal and their group proposed a framework for clustering data steams called "CluStream" algorithm. The proposed technique divides the clustering process to two components. The on-line component stores summary statistic about the data streams while the offline component performs clustering on the summarized data according to a number of user preferences such as the time frame and the number of clusters. In their technique, they used a pyramidal time frame and store a number of micro clusters from each snapshot for the offline component. The clustering algorithm then works on the micro clusters. A number of experiments on real datasets have been conducted to prove the accuracy and efficiency of the proposed algorithm [4] and [10].

K. Fast and Exact K-Means (FEKM)

The approximate cluster centers computed using sampling can be corrected and moved to exact cluster centers using only one or a small number of passes on the entire data. By exact cluster centers, refer to the cluster centers that are computed by the original k-means algorithm. Thus, this can use sampling to speed up the computation of exact clusters. There are three key questions to be addressed. First, when approximate cluster centers are computed using sampling, what information needs to be stored? Second, how can this information are used to avoid a large number of passes on the entire dataset. Third, how do we know that we have been able to achieve the same cluster centers as in the original k-means algorithm? Initially run the k-means algorithm on a sample, using the same convergence criteria and same initial points as we would use for the k-means. The following information is stored for future use. After every iteration of k-means on the sampled data, store the centers that have been computed. In addition, it computes and stores another value, referred to as the Confidence Radius of each cluster, whose computation will be described later. This information can be stored in a table with columns, and the number of rows equaling the number of iterations for which k-means was run. Each entry of the table contains a tuple (center, radius) for each cluster. Next, complete one pass through the entire dataset. For every point and each row of the table, determine the cluster to which this point will be assigned at this iteration, assuming that executing the algorithm on the entire dataset produces the same cluster centers as the initial run on sampled data. Next, try to estimate how likely it is that this point will be assigned to a different cluster when the algorithm is executed on the entire dataset. Thus, for a given point and row of the table, determine if this point is a boundary point. If it is, it is stored in a buffer. Otherwise, update the sufficient statistics tuple, which has the number and sum of the data points for the cluster. After the pass through the dataset and storing the boundary point, we do the following processing. Starting from the first row of the table, re-compute centers using the boundary points and sufficient statistics tuple. If any of the new computed centers fall outside the pre-estimated confidence radii which means that computation of boundary points is not valid, need to take another pass through the data. Use the new centers as new initialization points and again repeat all the steps. However, if the new computed centers are within the confidence radius, use these centers for the next iteration and continue. The key observation is that using cluster centers from sampling, boundary points, and sufficient statistics, which are able to compute the same cluster centers that we would have gotten through one pass on the entire dataset. Finally, the algorithm terminates by checking for the same termination condition that one would use in the original algorithm. The paper propose distributed fast and exact k-means algorithm (DFEKM) extending the concepts of fast and exact k-means (FEKM) [1] and [2] and [12].

III. PROPOSED TECHNIQUES

In this paper we propose Distributed Fast and Exact K-means (DFEKM) algorithm. We assume that data to be clustered is available at two or more nodes, which are referred to as the data sources. In addition, we have a node denoted as the central site, where the results of clustering are desired. It is also assumed that additional computation for clustering can be performed at the central site. We only consider horizontal partitioning of the data, i.e., each data source has values along all dimensions of a subset of the points. DFEKM uses ideas from Fast and Exact K-means (FEKM) where pre-computed approximate cluster centers can be corrected and moved to exact cluster centers using only one or a small number of passes on the entire data, and only one or a small number of rounds of communication between data sources and the central site. DFEKM uses ideas from FEKM where pre-computed approximate cluster centers can be corrected and moved to exact cluster centers using only one or a small number of passes on the entire data, and only one or a small number of rounds of communication between data sources and the central site.

There are three key questions to be addressed. First, when approximate cluster centers are computed using sampling, what information needs to be stored? Second, how can this information are used to process data independently on each data source, and to avoid frequent or high volume communication. Third, how can we determine, in a distributed environment, that we have been able to achieve the same cluster centers as in the centralized k-means algorithm. We sample data from each data source, and communicate it to the central node. Then, on the central node, we run the k-means algorithm on this sampled data. The following information is stored for processing on each data source. After every iteration of k-means on the sampled data, we store the centers that have been computed. In addition, we compute and store another value, referred to as the Confidence Radius of each cluster. Then, we send the table to all the data sources. Next, at each data source, we take one pass through the portion of the dataset available at that data source. For every point and each row of the table, we compute the cluster to which this point will be assigned at this iteration. WE also estimate if the point is a stable point as we did it in FEKM. We store the sufficient statistics of the (number, linear sum and square sum) stable points and we store the points which are not stable or the boundary point exactly as discussed in the context of FEKM. After the pass through the dataset and storing the boundary point, all the nodes send their boundary points and sufficient statistics to central node. The central node then does the following processing. Starting from the first row of the table, it recomputed centers using the boundary points and sufficient statistics tuple. If any of the new computed centers fall outside the pre-estimated confidence radius, which means that our computation of boundary points is not valid, we need to send the last corrected centers to all other nodes. Using these centers as the new initialization points, we have to go through iteration and repeat all the steps. However, if the new computed centers are within the confidence radius, we use these centers for the next iteration and continue.

The main data structure in DFEKM is the cluster abstract table or the CAtable and the algorithm starts with building the CAtable from a sample of the original dataset. This is done by the central node. In the pseudo code the process number is denoted by MyId, which is 0 for the central node. Initially, each entry of the CAtable contains the two tuple, the center and the confidence radius of each cluster in that iteration. This is done through the function BuildCATable in the central node. Then the central node broadcasts the CAtable to all other node using the function Broadcast. The argument Comm World denotes a handle for the all communicating processes. After this, at each node, we take one scan over the portion of the dataset available to that node and find out the likely boundary points for each iteration for each row of the table. The function IsBndrPoint checks for each data point if it meets the conditions of being a boundary point. If one point becomes a boundary point for one particular row, it is possible that the same point also be a boundary point for the next rows or next iterations of the CAtable. We also use Buffer and Index lists to store boundary points and their corresponding row numbers as we did it for FEKM. We also store the number and sum of the non-boundary points with each CAtable entry. The function UpdateSufficientStats accomplishes this. Then, the entire central node again gathers the entire boundary points, particularly the Buffer, the Index, and the sufficient statistics from all other nodes.

A. *Proposed Distributed Fast and Exact K-Means Algorithm (DFEKM)-*

The central node re-computes centers for each row of the CAtable from the boundary points corresponding to that row and from the sufficient statistics. It is done through the function *RecomputeCtrs*. We then verify if the new centers are located within the preestimated confidence radius to maintain the correctness. The function *IsCtrsWithinRadii* is responsible for this verification. If we find that the new centers are located within the confidence radius of corresponding clusters, we update the centers of the CAtable in the next row using the function *UpdateCAtableCtrs*. If any of the new centers is found outside the confidence radius of the corresponding cluster, the initial centers are replaced by those new centers and the central node broadcasts the new centers to all other nodes.

```
Input: Di (Data Points), Si (Sample Data Points), InitCtrs
(Initial Centers), € (Stopping criteria in k-means algorithm).
Output: k Cluster Center.
begin
flag← 1
while flag do
        List Buffer ← NULL
        Index [ ] ← NULL
        Table C Atable ← NULL
        Si← collectsample ()
    GatherV (Si, 0, Comm_World)
If MyId = 0  then
 NumRow ← BuildCTable (InitCtrs, Si, CAtable, €);
BroadCast (CAtable, 0, Comm_World);
CollectCATablestat (CATable, Buffer, Index);
GatherV (Buffer, 0, Comm_World);
GatherV (Index [], 0, Comm_World);
GatherV (CAtable, 0, Comm_World);
IF MyId=0
 then
flag ← CorrectCATable (CAtable, Buffer, Index);
Broadcast (flag, 0, Comm_World);
end
outputCATableCtrs (CAtable, row Num Row);
end
//
```

IV. RESULTS

Our experiments compare the DFEKM algorithm to these two approaches. As the main property of our algorithm is that it produces the same results as the exact k-means algorithm applied centrally, we did not compare DFEKM against any of the existing approximate distributed clustering approaches. Another challenge in designing our experiments was to simulate execution on distributed data repositories. As compared to a tightly coupled parallel configuration, executing parallel code on distributed data repositories potentially involves large load imbalance and/or high communication latencies. Our experiments were conducted on a parallel machine, and the above two effects were simulated by introducing delays during each round of communication. We also considered cases in which data has not evenly distributed. For such cases, we compare parallel k-means, sequential k-means, and our DFEKM algorithm. Additionally, we were interested in seeing how many passes over the entire dataset and how many rounds of communication were required by the DFEKM algorithm. In all experiments, the initial center points and the stopping criteria for this algorithm are kept same as those of the k-means algorithm. We used two convergence criteria. The algorithm stops when (1) the new centers are not sufficiently different from those generated in the previous iteration, or (2) it has run for a specified maximum number of iterations. The second criteria are useful with bad initializations, where the algorithm could run for a large number of iterations.

During the sampling phase, we sample 10% of the data from each data source.

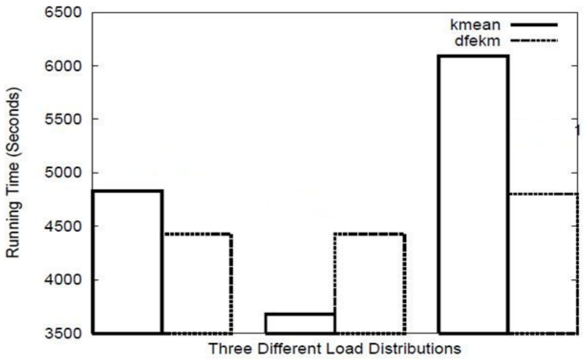

Figure:1 Running Time of DFEKM and Parallel k-means in the Presence of Load Imbalance.

The figure shows that three out puts of running time of DFEKM and Parallel K-means in presence of load imbalance (First two columns define first load imbalance, second two columns define second load imbalance and last two columns define third load imbalance).

V. CONCLUSION AND FUTURE WORKS

We have presented, analyzed, and evaluated a distributed clustering algorithm that provably produces the same cluster centers as the k-means clustering algorithm. We have performed a number of experiments with real and synthetic datasets to evaluate our algorithm. Our results show that DFEKM is clearly better than two other possible options for exact clustering on distributed data, which are down-loading all data and running sequential k-means, or running parallel k-means on a loosely coupled configuration. Moreover, even in a tightly coupled environment, DFEKM can outperform parallel k-means if there is a significant load imbalance.

There can be a few possible future works from this paper. One of them can be to develop the distributed version of the stream clustering algorithms. Another work can be in the direction of improving the algorithm to handle a variable number of clusters instead of a fixed. This will be particularly useful in case the data stream evolves such that number of clusters change. It is also possible to develop a framework of incremental clustering using the concepts from our stream algorithms. Lastly, there is a need of understanding the effects of various stream parameters such as rate of incoming data on the performance of the algorithms, which can also be a potential future work.

VI. REFERENCES

[1] K A Abdul Nazeer, S D Madhu Kumar, M P Sebastian, “Enhancing the k-means clustering algorithm by using a O(n logn) heuristic method for finding better initial centroids”, IEEE 2011 Second International Conference on Emerging Applications of Information Technology, pp- 261-264.

[2] Sun Yuepeng, Liu Min, Wu Cheng, “A Modified k-means Algorithm for Clustering Problem with Balancing Constraints”, IEEE 2011 Third International Conference on Measuring Technology and Mechatronics Automation, pp- 127-130.

[3] Hai-Guang Li, Gong-Qing Wu, Xue-Gang Hu, Jing Zhang, Lian Li1, Xindong Wu, “K-Means Clustering with Bagging and MapReduce”, IEEE Proceedings of the 44th Hawaii International Conference on System Sciences – 2011, pp- 1-8.

[4] Rajendra Pamula, Jatindra Kumar Deka, Sukumar Nandi, “An Outlier DetectionMethod based on Clustering”, IEEE 2011 Second International Conference on Emerging Applications of Information Technology, pp- 535-556.

[5] Hong Zeng and Yiu-ming Cheung, “Semi-supervised Maximum Margin Clustering with Pairwise Constraints”, IEEE 2011.

[6] Amitava Mukhopadhyay, Bipasha Paul Shukla, Diptiprasad Mukherjee and Bhabatosh Chanda, “A Novel Neural Network based Meteorological Image Prediction from a given Sequence of Images”, IEEE 2011 Second International Conference on Emerging Applications of Information Technology, pp-202-205.

[7] Shi Na and Liu Xumin, “Research on k-means Clustering Algorithm”, IEEE Third International Symposium on Intelligent Information Technology and Security Informatics 2010, pp- 63-67.

[8] Mantao Xu and Pasi Franti, “A HEURISTIC K-MEANS CLUSTERING ALGORITHM BY KERNEL PCA”, IEEE 2004 International Conference on Image Processing (ICIP), pp-3503-3506.

[9] Man Sha and Huixian Yang, “Speaker recognition based on APSO-K-means clustering Algorithm”, IEEE 2009 International Conference on Artificial Intelligence and Computational Intelligence, pp- 440-444.

[10] Li Xinwu, “Research on Text Clustering Algorithm Based on Improved K-means”, IEEE 2010 International Conference on Computer Design and Applications (ICCDA 2010), pp- v4- 573-576.

[11] Qin Chen, Jinping Mo, “Optimizing the Ant Clustering Model Based on K-Means Algorithm”, IEEE 2009 World Congress on Computer Science and Information Engineering, pp- 699-702.

[12] Taoying Li and Yan Chen, “An Improved k-means Algorithm for Clustering Using Entropy Weighting Measures”, IEEE Proceedings of the 7th World Congress on Intelligent Control and Automation June 25 - 27, 2008, Chongqing, China, pp- 149-153.

VII. AUTHOR'S PROFILE

Mr. Rajesh Ahirwar presently pursuing M.Tech in Information Technology at Samrat Ashok Technological Institute, Vidisha, M.P., India. The degree of B.E. secured in Information Technology from Indira Gandhi Engineering College, Sagar, M.P., India. Research Interest includes Data Mining, Artificial Intelligence, and Clustering. **E-mail: rajesh.mtechit2011@gmail.com.**

Mr. Mukesh Goyal presently pursuing M.Tech in Information Technology at Samrat Ashok Technological Institute, Vidisha, M.P., India. The degree of B.E. secured in Information Technology from Indira Gandhi Engineering College, Sagar, M.P., India. Research Interest includes Data Mining, Artificial Intelligence. **E-mail: mukesh.goyal23@gmail.com.**

Mr. Narendra Kori presently pursuing M.Tech in Information Technology at Samrat Ashok Technological Institute, Vidisha, M.P., India. The degree of B.E. secured in Information Technology from Truba Institute of Engineering & Information Technology, M.P., India. Research Interest includes Data Mining, Artificial Intelligence. **E-mail: kori_narendra86@rediffmail.com.**