Journal of Engineering and Applied Sciences 9 (10-12): 441-446, 2014
ISSN: 1816-949X

A Study on How to Improve the Performance of k-mean Data Mining Algorithm in a Parallel Environment

[1]R.P.T.H. Gunasekara, [2]M.C. Wijegunasekara and [2]N.G.J. Dias
[1]Department of Computing and Information Systems,
Wayamba University of Sri Lanka, Kuliyapitiya 60200, Sri Lanka
[2]Department of Statistics and Computer Science,
University of Kelaniya, Dalugama, Kelaniya 11600, Sri Lanka

Abstract: The k-mean algorithm is widely used clustering algorithm for large datasets. But, there are limitations when k-mean is used for very large datasets. This study is carried out to enhance the performance of the k-mean data-mining algorithm by using parallel programming methodologies. In this research, mainly two methods of parallelizing k-mean clustering algorithm were compared. They were k-mean clustering on parallel and non-parallel execution in WEKA and k-mean clustering on constructed program using Message Passing Interface (MPI) for parallel k-mean algorithm. Firstly, the cluster building ability of WEKA parallel over non-parallel WEKA for very large datasets was investigated. To identify the performance of parallelizing, the number of machines connected to the WEKA parallel was varied and performances were analyzed for several k values using k-mean algorithm for each setup. The experiment was done on three real electricity consumption data consists of 80,000, 50,000 and 30,000 data entries and with 65 attributes. It was identified that there is a significant improvement in performance of the WEKA parallel. Further WEKA parallel can be applied to very large datasets which were failed to work with WEKA. Secondly, the k-mean algorithm was implemented in C programming language and its performance with non-parallel WEKA was compared. According to that the time taken to build clusters was almost similar for small datasets.

Key words: Clustering alogrithm, clusters, programming language, datasets, attributes

INTRODUCTION

Clustering is the task of partitioning a large set of similar data points into meaningful groups called clusters Clustering is widely used in several areas of computer science such as data mining, pattern recognition, image processing, computer vision and so on. There are lots of clustering algorithms implemented in many perspectives. Among them k-mean algorithm is one of the most popular techniques for clustering.

As the size of the datasets increase exponentially, high performance computing technologies are needed to analyze and to recognize pattern of those data. The applications or the algorithms that are used for these processes have to invoke data records several times iteratively. Therefore, this process is very time consuming and consume more device memory in a very large scale. Some clustering tools such as WEKA are unable to handle large datasets. To find a solution for this, the performance of data mining algorithms should enhance.

During the study of enhancing the performance of data mining algorithms, it was identified that the data mining algorithms that were developed for the parallel processing were based on distributed (cluster or grid) computing environments.

In this research, the serial k-mean algorithm was implemented using C programming language and then it was parallelized for multiple processors using Message Passing Interface (MPI). These two algorithms were compared with existing k-mean clustering algorithm in WEKA and WEKA parallel separately.

Literature review: There are lots of studies curried on clustering algorithms used in data mining. Among them, k-mean algorithm is a very popular algorithm because of its clarity of implementation. In the literature, considerable studies were carried out on k-mean algorithm and parallel k-mean algorithm. Sapna *et al.* (2010) discuss the k-mean algorithm and how to work with WEKA tool for k-mean clustering algorithm. Further, study on the k-mean and its

Corresponding Author: R.P.T.H. Gunasekara, Department of Computing and Information Systems,
Wayamba University of Sri Lanka, Kuliyapitiya 60200, Sri Lanka

performance in comparison with Expectation Maximization (EM) clustering algorithms using WEKA is found by Namita and Deepti (2013). An improvement has been made for WEKA by Sebestian and David. Sebestian and David have tested the parallel WEKA for classification but they have not check for clustering algorithms in WEKA. Using the same hardware setup author was able to run clustering algorithm in WEKA parallel, results were discussed by Gunasekara *et al.* (2013) and its performance was discussed over WEKA for k-mean algorithm.

Wooyoung (2009) has presented is a survey on clustering algorithms and parallel clustering algorithms. This study discusses many types of clustering algorithms, their parallelization and applications. Dhillon and Modha (2000) presented a parallel implementation of k-means on a shared memory using Message Passing Interface (MPI). Their research mainly concentrates on the speed up algorithm over serial algorithm but they did not discussed about other parallel programming implementation. Das *et al.* (2007) and Ene *et al.* (2011) presented two variations of the k-means algorithm for shared memory where in both studies MapReduce framework is used. Geoffrey *et al.* (2011) has addressed Parallel Data Mining from multicore using cloudy grids. In a recent study, Kumar *et al.* (2011) presented a new method for avoiding unnecessary distance calculations using triangle inequality technique. They have addressed the load imbalance problems in their framework when such computations are eliminated. Most recent study Tayfun (2014) provides a theoretical analysis on the performance of k-means algorithm and presenting extensive tests on a shared memory architecture using Graphics Processing Unit (GPU).

MATERIALS AND METHODS

This research is based on the improving the performance of k-mean clustering algorithm using parallel programming methodologies. Here, two serial k-mean implementations methods and two parallel implantation methods were used to compare the performance of each method.

Firstly, the cluster building ability of k-mean algorithm in WEKA parallel over non-parallel WEKA for very large datasets was investigated. To identify the performance of parallelization, the number of machines connected to the WEKA parallel was varied and performances were analyzed for several k values using k-mean algorithm for each setup. The experiment was done on three real electricity consumption data consists of 80,000, 50,000 and 30,000 data entries with 65 attributes.

Secondly, the serial k-mean algorithm was implemented using C programming language and it was parallelized using Message Passing Interface (MPI) programming for distributed computers. To measure the performance of parallel k-mean over non-parallel k-mean, the above datasets were used and above experiment was repeated for the constructed MPI program. In this process, the dataset was partitioned into several segments and those segments are sent to distributed processors then it applies the k-mean algorithm separately. This process is repeatedly executed until the smooth clusters are formed.

To analyze the performance of above algorithms, a large dataset (80,000 entries) was selected and the time consumed to construct the clusters was computed separately. Then the dataset was partitioned in to two datasets (30,000 and 50,000 entries) and the performance analysis was repeated.

Used dataset: The main dataset was collected from the ceylon electricity board which consists of electricity consumption data. This dataset is a registry of each and every consumer who gets the service from Kurunegala area office of North Western Province of Sri Lanka. This dataset is updated when a new consumer is registered to take an electricity connection. There are about 80,000 consumers as at December 2009 in Kurunegala area. This dataset gives details about registered consumers. This dataset consists of 30 attributes which provide details of each consumer. Among 30 attributes there were data on Consumer details (Account number, name, address, etc.), Consumer's Electricity connection details, Electricity consumption data (LAST_RDN, CRNT_RDN, NO_DIGITS, MON_CHARGE), meter reader's details and data entry details/system data.

For this research three datasets were used, one is the full set of electricity consumers' master reading in Kurunegala area, containing 80,000 records. Then another two datasets were formed from the full dataset by filtering domestic consumers and non-domestic consumers. The domestic consumers' dataset consists of 30,000 data and the non-domestic dataset contains remaining 50,000 data.

Data preprocessing: In data preprocessing, electricity consumers master reading data were filtered by omitting some unrelated fields (BILL_CYCLE, USER_ID, LOGIN_DATE, LOGIN_TIME, READER_COD), because they are not directly related to electricity consumption or consumer details. The filtered master reading dataset was merged with 60 data fields using SQL queries. These merged fields are the 60 monthly readings (for 5 years) of monthly consumption data of the consumers. Then it

formed a dataset of 65 data fields with 80,000 records. The initial dataset was stored in DBASE IV format and it was converted to .csv format to work in WEKA.

After forming the full electricity consumption dataset another two datasets were created by splitting into domestic consumers and non-domestic consumers. For easy reference Dataset with all consumers is labeled as Dataset A, domestic consumers dataset is labeled as B and non-domestic dataset is labeled as C.

Cluster analysis of WEKA: For several k values simple k-mean clustering in WEKA machine learning was used to analysis k number of clusters. For each k value, the times taken to build up clusters were recorded for all three datasets.

Setting up the WEKA parallel environment: WEKA Parallel (WEKA 3.7.10) was installed in all server machines that were used in distributed calculations. A WEKA-parallel session was run on a single client machine and distributed severs. For each computer that is to be used as a distributed server, launch the software by entering the following line at command line prompt on that computer:

- Java weka.core.DistributedServer<port number>

Then each machine was configured by creating a configuration file with port number according to the network used.

Experiment on cluster analysis of WEKA parallel: By changing the number of machines connected to the distributed system, the above experiment was repeated. The times taken to build clusters were recorded in parallel for several machine connected. For each setup, simple k-mean algorithm was used by varying k values in WEKA parallel.

The experiment was done for several datasets to find out the time taken build k number of clusters in core i3, 3.3 GHz machine with Linux operating system using WEKA and WEKA parallel on similar distributed machines.

Implementation of serial k-mean: In data mining, k-means clustering is a method of cluster analysis which aims to partition n observations into k (where k is the number of selected groups) clusters in which each observation belongs to the cluster with the nearest mean. The grouping is done by minimizing the sum of squared distances (Euclidean distances) between items and the corresponding centroid (Center of mass of the cluster).

The k-mean is one of the simplest unsupervised learning algorithms used in clustering in data mining. The k-mean algorithm follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed. For each cluster, the mean value of each is assigned. Therefore, k number of centroids (means) have to be assigned first. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early grouping is formed.

Here, the nearest points to the centroid are found by calculating the distance between the centroid and each point. In k-mean algorithm, the distance is calculated by using euclidian distance or the manhatten distance.

After all elements are gathered to initial k centroids, k clusters were formed. At this point, we need to calculate the meanof each cluster as center of mass of the clusters. Then check the new means are similar to the initially assigned centroids.

After we have these k new centroids, a new binding has to be done between the same data set points and re-calculates nearest new centroid. A loop for several iterations has been generated. As a result of this loop, it may notice that the k centroids (means) change their location step by step until no more changes is to occur. In other words centroids do not move any more (Jiawei and Micheline, 2009; Bharat and Manan, 2012).

In summary, the k-mean algorithm consists of the following steps:

- Place k points into the space represented by the objects that are being clustered. These points represent initial group centroids
- Assign each object to the group that has the closest centroid
- When all objects have been assigned, recalculate the positions of the k centroids
- Repeat steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated

The k-mean algorithm was implemented using C programming language. Here, the dataset was stored in a two dimensional array and the k points are selected randomly. By minimizing the sum of squared distances of each point to selected meanit creates k number of arrays. Then, the mean of each array element is calculated and again gather the array elements to each array. This is continued until two consecutive two means are similar.

Parallelizing k-mean algorithm with Message Passing Interface: To increase the performance the process of parallelization mainly includes the data partitioning and parallel processing. Here, the data set is partitioned according to number of processers. The dataset is stored in a two dimensional array and this array is equally partitioned into q number of blocks, where:

$$q = \frac{\text{All elements in the array}}{\text{No.of processes}}$$

These blocks consists of rows and are assigned to each processor. In the first node (processor) the k number of cluster centers are randomly assigned and broadcast to each node.

In each processor, it will compute the distances from centroids (randomly assigned k number of centroids) to each element of each block which are locally stored on each processor. By considering these distances, most nearest elements are gathered to form new clusters in each processor. A series of assignments are generated mapping elements to clusters. Each process then gathers the sum of all arrays allocated to a given cluster and computes the mean of the block elements assigned to a particular cluster. This is repeated for every cluster and a new set of centroids is available on every process and the block of elements can be re-assigned with respect to the newly calculated centroids. This assignment is repeated until two consecutive assigned means are similar. Finally, all locally produced means are broadcast globally.

The procedure used in the parallelization process is summarized below where N represents the all data points in the two dimensional array and P denotes the number of processors.

- Assign N/P data points to each processor
- Node 0 randomly chooses k points and assigns them as cluster means and broadcast
- In each processor for each data point find membership (nearest value) using the cluster mean
- Recalculate local means for each cluster in each processor
- Go to step 3 and repeat until two consecutive assignments of means are similar
- Globally broadcast, all local means for each processor find the global mean

The serial k-mean algorithm which was implemented in C programming language was changed to parallel by including MPI coding. For each task of parallelization, the used MPI routines are summarized in Table 1.

Table 1: MPI routines

MPI routine	Description
MPI_Init()	Initialize the MPI execution environment
MPI_Comm_size()	Returns the number of processes currently running the program
MPI_Comm-rank()	Returns the process identifier for the calling process
MPI_Bcast()	Broadcast "message" form a process with identifier "root" to all the processes
MPI_Allreduce (A, B, MPI_SUM)	Sums up all the local copies of "A" in all the processes (reduction operation) and place the result in "B" on all of the processes (broadcast operation)
MPI_Wtime()	Returns the number of seconds since some fixed, arbitrary point of time in the past. Between the two calls to MPI_Wtime() to get the execution time of the parallel program, this past reference is kept the same
MPI_Finalize()	Terminate the MPI execution environment

RESULTS AND DISCUSSION

In this study, the times taken to build clusters were recorded for two methods (WEKA and C code) for serial k-mean by changing k (number of clusters) for all three datasets. Then cluster building times were recorded for two parallel methods (WEKA parallel and MPI program) by changing the number of processors for given k for all three datasets. The execution time for all data set when using k-mean clustering in WEKA serial interface is shown in Table 2 for k = 3, 4, 5. The same dataset for same k values, the time taken to build clusters for implemented k-mean algorithm using C language was recorded in Table 3. By considering both tables, the implemented k-mean using C programming language was able to find clusters on the dataset A for k = 5 which was unable in WEKA and the cluster building time was lesser than the WEKA.

In parallelization, the times taken to build clusters in WEKA parallel and Parallel k-mean implemented using MPI is recorded in Table 4-6. By considering the Euclidean distances of each cluster it was identified that most suitable number of clusters is k = 5. Therefore, the number of clusters is taken as 5. Here, in the parallel environment dataset (block of rows) were broadcast to 2, 3 and 4 processors. By changing the number of processors, the two parallel programs were run. The cluster building time of dataset "A" using WEKA parallel and MPI programs when k = 5 is represented in Table 4. Similarly, the experiment was repeated to datasets "B" and "C" and results are shown in Table 5 and 6, respectively.

In parallel implementation the time taken to build clusters for deferent datasets are shown in Table 3-5. The dataset with 30,000 data (Table 5) gives lesser time than the dataset with 50000 data (Table 4). The cluster building time is depends on the size of the dataset, number of clusters created and the number of machines connected.

Table 2: Cluster building time using serial k-mean in weka for all datasets

No. of clusters (k)	Execution time (Dataset A)	Execution time (Dataset B) (sec)	Execution time (Dataset C) (sec)
3	4.08 sec	1.18	3.02
4	4.89 sec	1.46	3.36
5	Machine fails to finish the task	3.11	4.08

Table 3: Cluster building time using serial k-mean (C program) for all datasets

Number of clusters (k)	Execution time (Dataset A) (sec)	Execution time (Dataset B) (sec)	Execution time (Dataset C) (sec)
3	4.02	0.95	2.68
4	4.5	1.65	3.02
5	5.2	2.56	3.58

Table 4: Result on dataset "A" using weka parallel and mpi progam (when K = 5)

No. of processors	Time taken in mpi (sec)	Time taken in WEKA (sec)
2	1.84	2.95
3	1.43	2.03
4	0.78	1.46
5	0.24	1.15

Table 5: Result on dataset "B" using weka parallel and mpi progam (when K = 5)

No. of processors	Time taken in mpi (sec)	Time taken in WEKA (sec)
2	0.54	0.92
3	0.36	0.69
4	0.26	0.54
5	0.18	0.28

Table 6: Result on dataset "C" using weka parallel and mpi progam (when K = 5)

No. of processors	Time taken in mpi (sec)	Time taken in WEKA (sec)
2	1.56	1.93
3	1.06	1.69
4	0.81	1.45
5	0.51	1.23

According to the study, the performance of WEKA parallel is higher than the WEKA. According to the result in Table 1, it takes comparatively long time to build clusters and when it requires higher number of clusters WEKA does not support such a requirement. When k is 3 or 4 it build clusters but when k is 5 WEKA is not support for the full dataset (80,000 entries). Therefore, WEKA is not support for large datasets when it request to building higher number of clusters. However, k-mean implemented in C programming was able to clusters for large values of k.

Further, this study was extended to find the faster parallel programing methodology for k-mean algorithm from WEKA parallel and implemented MPI program. According to Fig. 1, the implemented MPI program for distributed processors gives lesser time than WEKA parallel to build clusters. It can be concluded that the fastest algorithm from the above four algorithms is the

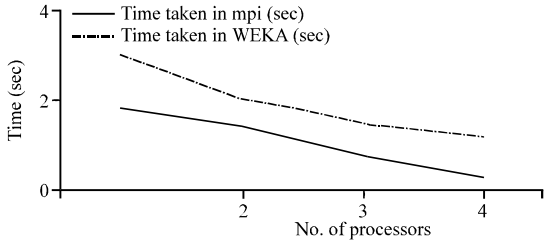

Fig. 1: Cluster building time when k = 5 for 80000 dataset

MPI program implemented for distributed processors. This experiment can be extended to check cluster boundaries and increase the quality of the clusters.

CONCLUSION

Then the above C program for k-mean algorithm was improved to parallel k-mean using Message Passing Interface (MPI) programming for distributed computers. To measure the performance of parallel k-mean over non-parallel k-mean the above datasets were used and above experiment was repeated for the constructed MPI program. Here, the dataset was partitioned into several segments and those segments are sent to distributed processors then it applies the k-mean algorithm separately. This process is repeatedly executed until the smooth clusters are formed. Further we compared the performance of the constructed parallel k-mean algorithm and the k-mean in WEKA parallel for very large datasets.

ACKNOWLEDGEMENT

Researcher should thank the Area Engineer of the Ceylon Electricity Board and his staff of Kurunegala Area office, North Western Province of Sri Lanka for helping me to collect electricity consumption data and giving me permission to use above data to my research.

REFERENCES

Bharat, C. and P. Manan, 2012. A Comparative Study of clustering algorithms Using weka tools, International Journal of Application or Innovation in Engineering and Management (IJAIEM), 1 (2): 2319-4847.

Das, A., M. Datar, A. Garg and S. Rajaram, 2007. Google News Personalization: Scalable Online Collaborative Fil-tering. In WWW, pp: 271-280.

Dhillon, S. and D.S. Modha, 2000. A Data Clustering Algorithm on Distributed Memory Multiprocessors. In: Know. Data Discovery (KDD), Lecture Notes in Computer Science, 1759: 245-260.

Ene, A., S. Im and B. Moseley, 2011. Fast Clustering using MapReduce. In: Know. Data Discovery (KDD), pp: 681-689.

Geoffrey, F., B. Seung-Hee, E. Jaliya, Q. Xiaohong and Y. Huapeng, 2011. Parallel Data Mining from Multicore to Cloudy Grids Informatics Department, Indiana University, USA, Computer Science Department and Community Grids Laboratory, Indiana University USA cUITS Research Technologies, Indiana University.

Gunasekara, R.P.T.H., N.G.J. Dias and M.C. Wijegunasekara, 2013. Performance of k-mean data mining algorithm with the use of WEKA-parallel, 14th Annual Research Symposium 2013, Faculty of Graduate Studies, University of Kelaniya, pp: 90-91.

Jiawei, H. and K. Micheline, 2009. Data Mining Concepts and Techniques. 2nd Edn. Morgan Kaufmann Publishers.

Kumar, J., R.T. Mills, F.M. Hoffman and W.W. Hargrove, 2011. Parallel k-means Clustering for Quantitative Eco-region Delineation Using Large Data Sets. Proceedings of the International Conference on Computational Science, ICCS, 4: 1602-1611.

Namita, B. and M. Deepti, 2013. Comparative Study of EM and k-means Clustering Techniques in Weka Inter-face. International Journal of Advanced Technology and Engineering Research (IJATER).

Sapna, J., M.A. Afshar, M.N. Doja, 2010. K-means Clustering Using Weka Interface. Proceedings of the 4th National Conference; INDIACom-2010 Computing For Nation Development, Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi.

Tayfun, K., 2014. Parallel k-means Algorithm for Shared Memory Multiprocessors. J. Computer and Communicat., 2: 15-23.

Wooyoung, K., 2009. Parallel Clustering Algorithms: Survey.