A Parallel Cop-Kmeans Clustering Algorithm Based on MapReduce Framework

Chao Lin, Yan Yang, and Tonny Rutayisire

School of Information Science & Technology, Southwest Jiaotong University, Chengdu, 610031, P.R China

Abstract. Clustering with background information is highly desirable in many business applications recently due to its potential to capture important semantics of the business/dataset. Must-Link and Cannot-Link constraints between a given pair of instances in the dataset are common prior knowledge incorporated in many clustering algorithms today. Cop-Kmeans incorporates these constraints in its clustering mechanism. However, due to rapidly increasing scale of data today, it is becoming overwhelmingly difficult for it to handle massive dataset. In this paper, we propose a parallel Cop-Kmeans algorithm based on MapReduce- a technique which basically distributes the clustering load over a given number of processors. Experimental results show that this approach can scale well to massive dataset while maintaining all crucial characteristics of the serial Cop-Kmeans algorithm.

Keywords: Parallel Clustering, Cop-Kmeans Algorithm, MapReduce.

1 Introduction

Clustering algorithms are often useful in applications in various fields such as data mining, machine learning and pattern recognition. They conduct a search through a space of a dataset, grouping together similar objects while keeping dissimilar objects apart, as much as possible. Normally this search proceeds in an entirely unsupervised manner. For some domains, however, constraints on which instances must (ML) or cannot (CL) reside together in the same cluster either are known or are computable automatically from background knowledge [1]. Cop-Kmeans is one popular clustering algorithm which has incorporated these instance based constraints in its clustering mechanism [2]. The approach in this algorithm has been shown to be successful in guiding the clustering process toward more accurate results. However this algorithm only works satisfactorily with relatively small dataset, when the size of the dataset is very large it becomes terribly slow.

With the rapid development of information technology, data volumes processed by typical business applications are very high today which in turn pushes for high computational requirements. To process such massive data, a highly efficient, parallel approach to clustering needs to be adopted. Recently, several attempts have been made to improve the applicability of K-Means algorithm for massive applications

Y. Wang and T. Li (Eds.): Knowledge Engineering and Management, AISC 123, pp. 93–102.
springerlink.com

through parallelization [3-5]. To our knowledge though, widely used semi-supervised clustering algorithms are serial and can only run on a single computer. This greatly hinders their capability to handle very large dataset.

Cop-Kmeans is probably the most popular semi-supervised clustering algorithm that has been used in a variety of applications [2]. Due to its popularity, we believe that proposing a parallel version based on MapReduce can be of significant use to the clustering community. MapReduce is a parallel programming model for processing huge dataset using large numbers of distributed computers (nodes), collectively known as a cluster [6]. The major contributions of this work are two folds. First is to address the issue of constraint-violation in Cop-Kmeans by emphasizing a sequenced assignment of cannot-link instances after conducting a Depth-First Search of the cannot-link set. Second is to reduce the computational complexities of Cop-Kmeans by adopting a MapReduce Framework.

The rest of the paper is organized as follows; Section 2 introduces the underlying mechanism of Depth-First Search and further illustrates its role in solving constraint-violation in Cop-Kmeans. Section 3 presents our approach and the proposed parallel Cop-Kmeans algorithm based on MapReduce framework. Section 4 presents our experimental results as well as evaluation for the proposed algorithm. Finally we draw our conclusions and future work in Section 5.

2 Parallel Cop-Kmeans Algorithm Based on MapReduce

2.1 Depth-First Search

Depth-First Search (DFS) is a general technique for traversing a graph whose principal is "going forward (in depth) while there is such possibility, otherwise backtrack". The mechanism is about choosing a starting vertex and then explore as far as possible along each branch before backtracking. Fig. 1 illustrates the process of Depth-First Search.

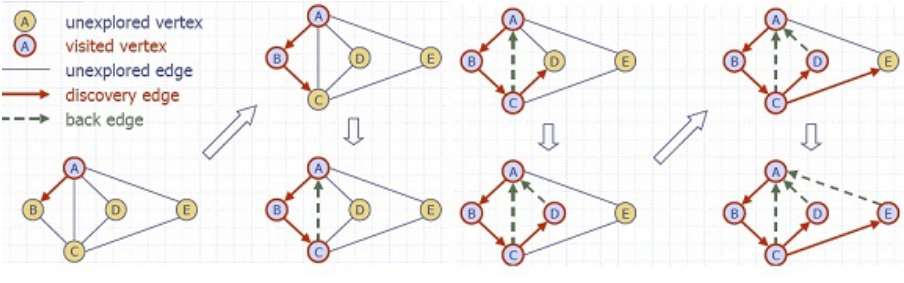

Fig. 1. Depth-First Search

The overall depth first search algorithm will simply initializes a set of markers so we can tell which vertices are visited, chooses a starting vertex A, initializes a tree T to A, and calls DFS(A). Then, we traverse the graph by considering an arbitrary edge (A, B) from the current vertex A. If the edge (A, B) takes us to a visited vertex B, then

we back down to the vertex A. On the other hand, if edge (A, B) takes us to an unvisited vertex B, then we paint the vertex B and make it our current vertex, and repeat the above computation. When we get to a point where all the edges from our current vertex take us to visited vertices, we then backtrack along the edge that brought us to that point. We take the immediate visited vertex that we find on the way back, make it our current vertex and start the computations for any edge that we missed earlier. When the depth-first search has backtracked all the way back to the original source vertex, A, it has built a DFS tree of all vertices reachable from that source.

2.2 Solving Constraint-Violation in Cop-Kmeans Algorithm

Whereas clustering with pairwise constraints is generally proven to enhance the conventional K-Means, it is often associated with a problem of constraint-violation; this brings about failure in hard-constrained clustering algorithms like Cop-Kmeans [2] when an instance has got at least a single cannot-link in every cluster. For this to occur, it can be caused by two situations: either there is no feasible solution for that particular clustering or a wrong decision about the assignment order of instances to clusters was made.

Definition 1. Feasibility Problem [7]**:** Given a dataset X, a collection of constraints C, a lower bound Kℓ and an upper bound Ku on the number of clusters, does there exists a partition of X into k groups such that K$\ell \leq k \leq$ Ku and all the constraints in C are satisfied?

Definition 2 [7]**:** A feasibility problem instance is β-easy if a feasible solution can be found by an algorithm β given the ordering of instances in the training set which determines the order they will be assigned to clusters.

From definitions 1 and 2, we can see that much as there could be a feasible solution for a particular clustering, it may not be necessarily easy to find one. As it is interpreted in Brook's theorem [7], that it is only when the number of CL constraints involving on instance is less than K (number of clusters), that one is assured of a feasible solution regardless of the order in which instances are assigned to clusters. Since this condition is not always the case, previous work [8, 9] have studied the problem of constraint-violation and used different approaches to solve it in Cop-Kmeans Algorithm.

The solution in this work capitalizes on re-arranging the CL-set (before assignment) in a sequence such that any CL instance will have at least one cluster for assignment. This kind of sequence is produced by the Depth-First Search function in Algorithm 1 below. In the Depth-Search mechanism, clustering under CL constraints is looked at as traversing a graph. The vertices represent the CL-Instances, edges representing the CL between any two instances while the path of traversing it represents the order of assigning the CL-Instances to clusters.

For simplicity let's illustrate this process using Fig. 1 above: where {C $\neq$ (A, B), C $\neq$ (A, C), C $\neq$ (A, D), C $\neq$ (A, E), C $\neq$ (B, C), C $\neq$ (C, D), C $\neq$ (C, E)}. From this CL-Set we can observe that the maximum number of CL involving one instance, Δ is 4. Assuming K is 4 and taking Brook's theorem into consideration, a conventional Cop-Kmeans could easily fail due to constraint-violation. Since it has no clear

mechanism for sequencing CL instances before assignment, it is possible to assign {A, D, B, E, C} in that order, supposing that A, D, B and E are each assigned to a different cluster, then C will have no feasible cluster for assignment. On the other hand, using the depth-first search to traverse this CL-Set (graph), the CL instances will be sequenced in a stack produced by a specific path depending on the first CL instance encountered. For example if we consider the path in Fig. 1 the following stack will be produced {A, B, C, A, D, A, E, A} in which case all red-colored instances represent back edged vertices. Note that since the principal of a stack is "Last-in-First-out", these CL-instances will be assigned in the order {A, E, D, C, B} in which case even if (worst scenarios) A, E, D and C are each assigned to a different cluster, B could still be assigned in either D or E and there could not be any constraint-violation and hence no failure of the algorithm. Although back edged vertices (instances) may appear more than once in the stack, they can only be assigned once in the cluster as shown in the order above. Note that the DFS mechanism ensures that while the first CL instance encountered may be randomly selected, instances with the highest degree (highest number of CL involving a single instance) are always among the first to be assigned. This means that most of those instances involved in CL with these higher degree instances could have a high chance of being assigned together in the same cluster since most of them may not necessarily have CL ties among themselves.

3 Parallel Cop-Kmeans Algorithm Based on MapReduce

In this section, we present the proposed parallel Cop-Kmeans algorithm, but prior to that, we introduce the underlying mechanism of the core approach used in this algorithm namely: MapReduce.

3.1 The MapReduce Framework

MapReduce is a programming model introduced by Google to support distributed computing of large dataset on clusters of computers. The name "MapReduce" was inspired by the "map" and "reduce" functions in the functional programming. Users specify the computations in terms of "map" and "reduce" functions and the underlying runtime system automatically parallelizes the computation across a large cluster of computers, handles machine failures and schedules inter-computer communication to make efficient use of the network and the disks. This enables programmers with no experience with distributed systems to easily utilize the resources of a large distributed system.

In MapReduce [3], the Map function processes the input in the form of key/value pairs to generate intermediate key/value pairs, and the Reduce function processes all intermediate values associated with the same intermediate key generated by the Map function. Fig. 2 below illustrates the different phases of MapReduce model.

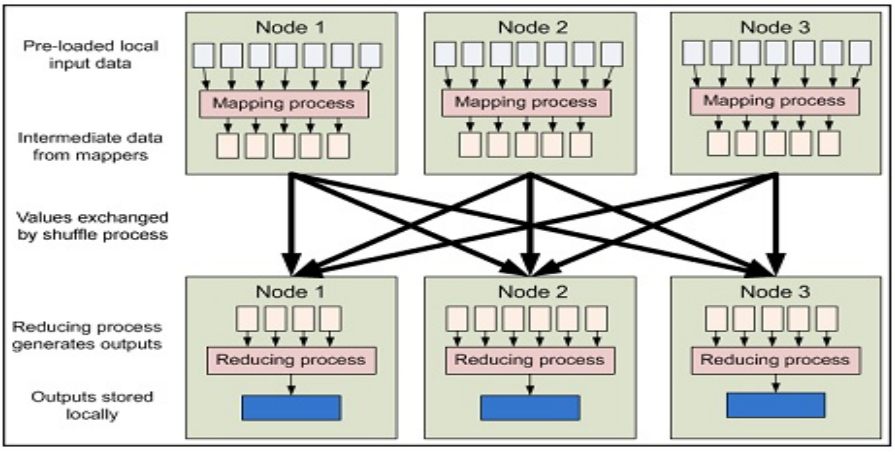

Fig. 2. The MapReduce model

3.2 The Proposed Cop-Kmeans Algorithm Based on MapReduce

The underlying idea behind the original Cop-Kmeans algorithm is to assign each instance in the dataset to the nearest feasible cluster center. We can see that the algorithm involves typically three steps: calculating the shortest distance between each instance and the cluster centers, assigning instance while avoiding constraint-violation and updating cluster centers. From the serial point of view, all these steps are handled by a uniprocessor and all the concerned data is kept in its local memory. However, in order to incorporate these tasks in MapReduce model, some crucial modifications must be made.

Map-Function. Ideally, since the task of distance computations is so bulky and would be independent of each other, it is reasonable to execute it in parallel by the Map function. The challenge about this is that the next task of assigning the instance to the cluster center with which it has the shortest distance takes into consideration constraint-violation. This would mean dependence on assignment of instances in other mappers which could have a ML or CL with the instance in question. To overcome this issue and simplify, we generate constraints from the partial dataset allocated to each mapper. This would ensure that distance computations and considerations of ML and CL are purely independent of other mappers. This map function outputs intermediate data to be used in the Reduce function.

Algorithm 1 shows the map function of our proposed parallel Cop-Kmeans.

Algorithm 1: Map Function

Input: Partial dataset *X* (Key, Value), K Global initial centers
Output: (Key, Value) pairs where the value is the instance information, while the key represents its closest center
(1) Generate ML and CL constraints from the partial dataset X
(2) For every randomly selected instance xi in X;
(a) if xi is a CL-instance, create an empty stack S, then Depth- First Search (xi, CL) is implemented.
(b) else assign it to the nearest cluster C_j such that Violate-constraints (x_i, C_j, *ML*, *CL*) returns false. If no such cluster is found, return { }.
(3) Take index of closest cluster as Key
(4) Take the instance information as value
(5) Return (Key, Value) pair

Depth-First Search (x_i, CL)
(1) Visit x_i;
(2) Insert x_i into *S*.
(3) For each child w_i of x_i
if w_i is unvisited (not in S)
{
Depth-First Search (w_i, CL);
Add edge (x_i,w_i) to tree T;
}
(4) For every instance S_i in *S* (Last-in First-out), assign it to the nearest cluster C_j such that Violate-constraints (L_i, C_j,*ML*, *CL*) returns false. If no such cluster is found, return { }.

Violate-constraints (object x_i, cluster *C*, must-link constraints *ML*, cannot-link constraints *CL*)
(1) If x_c is already in cluster *C* and (x_i, x_c) ∈ *CL*, return true.
(2) If x_m is already assigned to another cluster other than *C* and (x, x_m) ∈ *ML*, return true.

Reduce Function. The Reduce Function gets its input from the Map Function of each mapper (host). As shown in algorithm 1 above, in every mapper, the map function outputs a list of instances each labeled with its closest center. Therefore it follows that, in each mapper, all instances labeled with same current cluster center are sent to a single reducer. In the reduce function, the new cluster center can be easily computed by averaging all the instances in the reducer. Algorithm 2 details the Reduce Function of the parallel Cop-Kmeans.

Algorithm 2: Reduce Function (Key, L)

Input: Key is the index of a particular cluster, L is the list of instances assigned to the cluster from different mappers

Output: (Key, Value) where, key is the index of the cluster, Value is the information representing the new centre

(1) Compute the sum of values for each dimension of the instances assigned to the particular cluster.

(2) Initialize the counter to record the number of instances assigned to the cluster

(3) Compute the average of values of the instances assigned to the cluster to get coordinates of the new centre.

(4) Take coordinates of new cluster centre as value.

(5) Return (Key, Value) pair

As shown in the algorithm 2, the output of each reducer is consequently the cluster index (key) and its new coordinates (value). The new centers are then fed back to the mappers for the next iteration and this goes on until convergence.

4 Experiments

4.1 Experimental Methodology

To evaluate the performance of the improved Cop-Kmeans, we have compared it with original Cop-Kmeans with respect to their proportions of failure (constraint-violation) and F-measures. For this purpose, we used four UCI numerical dataset namely: Iris (150, 4, 3), Wine (178, 13, 3), Zoo (101, 16, 7) and Sonar (208, 60, 2).The figures in the brackets indicate the number of instances, number of attributes and number of classes for each dataset respectively. In this set of experiments, for a given number of constraints, each of the two algorithms was run 100 times on a dataset. Note that in both algorithms, initial cluster centers and all pairwise constraints are randomly generated from the dataset. Also note that both algorithms are designed to return an empty partition (fail) whenever constrain-violation arises. With increasing number of constraints as inputs, we generated corresponding average proportions of failure and F-measure for each algorithm running on a given dataset (see Fig. 3 and Fig. 4).

In the second set of experiments, we evaluate the efficiency of the proposed parallel Cop-Kmeans with respect to Speedup and Sizeup characteristics. Note that the idea behind this algorithm is to intelligently distribute the computational workload across a cluster of computer nodes. In this set of experiment therefore we don't evaluate accuracy but rather efficiency of the parallel Cop-Kmeans in processing massive dataset. The experiments were run on Hadoop MapReduce platform of four nodes; each having 2.13 GHz of processing power and 2GB of memory. And in this part, we use USCensus1990 dataset (823MB) which owns 68 categorical attributes. Many of the less useful attributes in the original dataset have been dropped, the few continuous variables have been discretized and the few discrete variables that have a

large number of possible values have been collapsed to have fewer possible values. For comparison purposes, we divided the dataset into 4 groups of files, approximately having 200MB, 400MB, 600MB and 800MB in that order.

Speedup of a parallel system is defined by the following formula:

$$\text{Speedup } (p) = T_1/T_P \tag{1}$$

Where p is the number of nodes, T1 is the execution time on one node and TP is the execution time on p nodes. To measure the Speedup of a parallel system, we keep the dataset constant while increasing the number of nodes in the system. In the experiment, we compared the Speedup performances produced when dataset of varying sizes are given as inputs.

On the other hand, the Sizeup metric is defined by formula (2):

$$\text{Sizeup } (D, p) = T_{SP}/T_{S1} \tag{2}$$

where D is the size of the dataset, p is the multiplying factor of the dataset, T_{SP} is the execution time for p * D, T_{S1} is the execution time for D. To measure the Sizeup of a system, we keep the number of nodes in the system constant while growing the size of the dataset by p. For our experiment, we compared the Sizeup performances produced by fixing the number of nodes in the system to 1, 2, 3, and 4.

4.2 Experimental Results

Fig. 3 below depicts the average proportion of failures out of the 100 runs for both the original Cop-Kmeans and the improved Cop-Kmeans at a given number of constraints as input. It can be observed from the Fig. 3 (left) that in Cop-Kmeans, the proportion of failure worsens as more constraints are given. On the other hand, when the Depth-First Search mechanism is used to sequence CL-instances before assignment in the improved Cop-Kmeans, there is no single case of failure as shown in the Fig. 3 (right). The improvement also extends in terms of accuracy where Fig. 4 below shows relatively higher instances of average F-measures in our improved Cop-Kmeans compared to the original Cop-Kmeans on two dataset. The results at each point on the curve are obtained by averaging the F-measures over the 100 runs for a given number of constraints. Note that in both algorithms, initial cluster centers and all pairwise constraints are generated randomly from the dataset.

Fig. 5 reports the Speedup and Sizeup evaluations for our proposed parallel Cop-Kmeans Algorithm. As shown in the Fig. 5 (left), for each dataset, we increase the number of nodes in the system while reporting corresponding Speedup for that dataset. It can be noted that the results show a good Speedup performance for our proposed parallel Cop-Kmeans. Further more, it can be noted that as the size of the dataset increases, also Speedup performance increases; an indication that indeed our parallel algorithm can efficiently handle massive dataset. Sizeup evaluation also shows good performance of the parallel Cop-Kmeans. From the results (see Fig. 5 (right)), we can observe that as we increase the number of nodes in the system, we get better results for Sizeup.

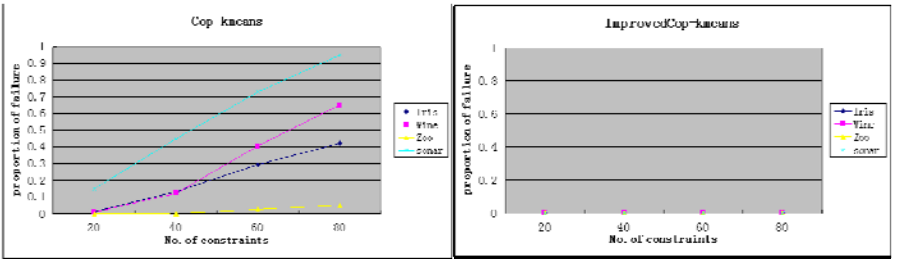

Fig. 3. Proportion of Failure for Cop-Kmeans and Improved Cop-Kmeans

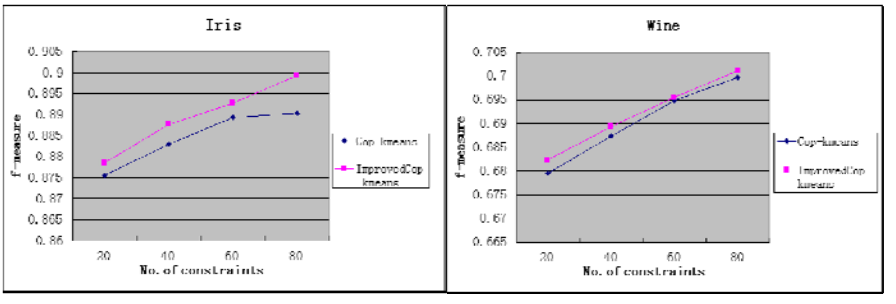

Fig. 4. The Average F-measures for Cop-Kmeans and Improved Cop-Kmeans

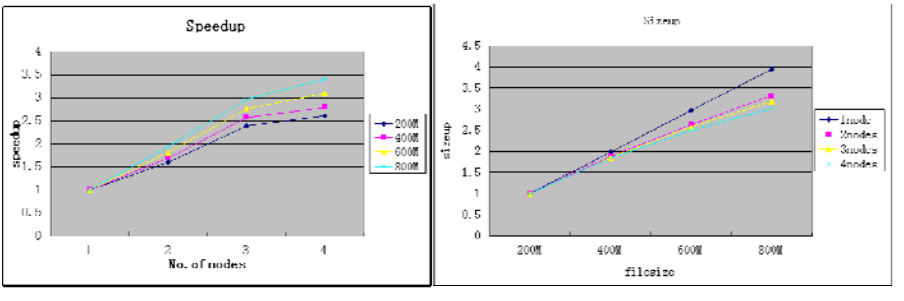

Fig. 5. Speedup and Sizeup evaluations of parallel Cop-Kmeans algorithm

5 Conclusion

In this paper, we proposed a parallel Cop-Kmeans Algorithm based on MapReduce Framework. This Algorithm adopts two crucial mechanisms: Depth-First Search and MapReduce. Inspired by the sensitivity to assignment order of instances, DFS capitalizes on sequencing CL-instances in a way that will not allow constraint-violation to happen. Results from our experiments confirm that this improvement enhances the accuracy of Cop-Kmeans without a single case of failure. Parallelizing the algorithm on a MapReduce Framework also gave positive results in term of speeding up and sizing up the computational processes and hence our proposed

algorithm confirmed efficient applicability to massive dataset typical to today's business applications.

Acknowledgements. This work is partially supported by the National Science Foundation of China (Nos. 61170111 and 61003142) and the Fundamental Research Funds for the Central Universities (No. SWJTU11ZT08).

References

1. Wagstaff, K., Cardie, C.: Clustering with instance level constraints. In: Proceedings of the International Conference on Machine Learning, pp. 1103–1110 (2000)
2. Wagstaff, K., Cardie, C., Rogers, S., Schroedl, S.: Constrained K-means clustering with background knowledge. In: Proceedings of International Conference on Machine Learning, pp. 577–584 (2001)
3. Malay, K.: Clustering Large Databases in Distributed Environment. In: Proceedings of the International Advanced Computing Conference, pp. 351–358 (2009)
4. Zhang, Y., Xiong, Z., Mao, J.: The Study of Parallel K-Means Algorithm. In: Proceedings of the World Congress on Intelligent Control and Automation, pp. 5868–5871 (2006)
5. Zhao, W., Ma, H., He, Q.: Parallel *K*-Means Clustering Based on MapReduce. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) Cloud Computing. LNCS, vol. 5931, pp. 674–679. Springer, Heidelberg (2009)
6. Xuan, W.: Clustering in the Cloud: Clustering Algorithm Adoption to Hadoop Map/Reduce Framework. Technical Reports-Computer Science, paper 19 (2010)
7. Davidson, I., Ravi, S.S.: Identifying and Generating easy sets of constraints for clustering. In: Proceedings of American Association for Artificial Intelligence, pp. 336–341 (2006)
8. Wagstaff, K.: Intelligent clustering with instance-level constraints. Cornell University (2002)
9. Tan, W., Yang, Y., Li, T.: An improved COP-KMeans algorithm for solving constraint violation. In: Proceedings of the International FLINS Conference on Foundations and Applications of Computational intelligence, pp. 690–696 (2010)