Parallel K-means Clustering Algorithm on NOWs

Sanpawat Kantabutra and Alva L. Couch
Department of Computer Science
Tufts University, Medford, Massachusetts, 02155, USA
http://www.cs.tufts.edu/~{sanpawat, couch}

ABSTRACT – Despite its simplicity and its linear time, a serial K-means algorithm's time complexity remains expensive when it is applied to a problem of large size of multidimensional vectors. In this paper we show an improvement by a factor of O(K/2), where K is the number of desired clusters, by applying theories of parallel computing to the algorithm. In addition to time improvement, the parallel version of K-means algorithm also enables the algorithm to run on larger collective memory of multiple machines when the memory of a single machine is insufficient to solve a problem. We show that a problem size can be scaled up to O(K) times a problem size on a single machine.

KEY WORDS -- Clustering algorithms, K-means algorithms, Parallel Algorithms, Computational Geometry, Data Mining

บทคัดย่อ -- ถึงแม้ว่าอัลกอริทึมเคมีนจะง่ายและทำงานในเวลาเชิงเส้น แต่เมื่อใช้ในการแก้ปัญหาเวกเตอร์แบบหลายมิติขนาดใหญ่ก็จะใช้เวลาในการทำงานที่มากและซับซ้อน ในบทความนี้ได้มีการนำเสนอวิธีการปรับปรุงอัลกอริธึมเคมีนโดยการนำการคำนวณแบบขนานเข้ามาร่วมด้วย ผลที่ได้แสดงให้เห็นว่าได้ผลดีขึ้นด้วยปัจจัย O (K/2) โดยที่ K คือจำนวนกลุ่มที่ต้องการ นอกจากนี้อัลกอริธึมเคมีนรุ่นที่ใช้การคำนวณแบบขนานยังสามารถทำงานในเครื่องหลายๆ เครื่องที่มีหน่วยความจำแบบสะสมได้ขนาดใหญ่ เมื่อหน่วยความจำของเครื่องใดเครื่องหนึ่งไม่เพียงพอต่อการแก้ปัญหาอีกด้วย ผลการทดลองแสดงให้เห็นว่าขนาดของปัญหาสามารถเพิ่มขึ้นในขนาด O (K) เท่าของขนาดปัญหาบนเครื่องเดี่ยว

คำสำคัญ -- อัลกอริทึมแบบกลุ่ม อัลกอริทึมเคมีน อัลกอริทึมแบบขนาน เรขาคณิตเชิงคำนวณ เหมืองข้อมูล

1. Introduction

Clustering is the grouping of similar objects and a clustering of a set is a partition of its elements that is chosen to minimize some measure of dissimilarity [1]. Clustering algorithms are often useful in applications in various fields such as visualization, pattern recognition, learning theory, and computer graphics. A classical vector quantization problem is usually solved as a gradient-descent problem. However, in practice a more convenient computing scheme is batch computation, usually named the K-means algorithm [2].

Given a set S of N D-dimension vectors without any prior knowledge about this set, the serial K-means clustering algorithm forms K disjoint nonempty subsets $\{C_1,C_2,C_3,\ldots,C_K\}$ of vectors such that each vector v_{ij}, $1\le i \le K$, $1\le j \le |C_i|$ has the closest distance (i.e., Euclidean distance) to means $\bar{X}_i$, $1\le i \le K$ [1] (See figure 1 for an example of K-means categorization of 25 points for K = 4). The algorithm achieves this result by minimizing a square-error function E such that

$$E = \Sigma_{i=1,K} \Sigma_{v\in Ci} \| \bar{X}_i - v \|^2$$

A K-means algorithm is measured by two criteria: intra-cluster criterion and inter-cluster criterion. An intra-cluster criterion (i.e., the inner summation) represents how good the cluster C_i is. Typical intra-cluster criteria are the diameter, radius, variance, variance multiplied by $|C_i|$, variance multiplied by $|C_i|^2$ of point set $|C_i|$, a Euclidean distance, and a square-error criterion. In this paper we use a square-error function as our criterion because 1) our purpose is to parallelize K-means algorithm and 2) the square-error criterion, defined above, is

the most commonly used and a good measure of the within-cluster variation across all the partitions [5]. For more information about other criteria, we suggest the paper [1] by Mary Inaba, Naoki Katoh, and Hiroshi Imai. The inter-cluster criterion (i.e., the outter summation) is defined as the total cost of the K clusters. Generally, the inter-cluster criteria are $max\{intra(C_1),intra(C_2),\ldots,intra(C_K)\}$ and $\Sigma_{i=1,K} intra(C_i)$ where $intra(C_i)$ is an intra-cluster criterion of C_i. In this paper we use the latter criterion. By minimizing each intra-cluster criterion locally (i.e., moving only vectors that reduce the error function to a new appropriate cluster), we expect that the algorithm will globally yield an optimal inter-cluster criterion. The K-means clustering algorithm is then to find a K clustering that minimizes the inter-cluster criterion or the error function.

Serial K-means Algorithm
1. Randomly select |S|/K members of the set S to form K-subsets
2. While Error E is not stable:
3. Compute a means X_i, $1 \le i \le K$ for each of the K subsets.
4. Compute distance d(i,j), $1 \le i \le K$, $1 \le j \le N$ of each vector such that $d(i,j) = \| \bar{X}_i - v_j \|$
5. Choose vector members of the new K subsets according to their closest distance to $\bar{X}_i$, $1 \le i \le K$.
6. End

The serial K-means algorithm has time complexity $O(R_s KN)$ where K is the number of desired clusters and R_s is the number of iterations [3].

2. Proposed Algorithm

In this section we present some related works, motivations, materials, our parallel K-means algorithm, and complexity analysis.

2.1 Related Works and Motivations

In [4] authors present a clustering using a coarse-grained parallel genetic algorithm to obtain an optimal minimum squared-error partitions and use a distributed algorithm to improve the total execution time. Their algorithm is based on a SIMD model. In [7] authors present an optimal adaptive K-means algorithm with dynamic adjustment of learning rate to induce a near-optimal clustering solution in a situation where the pattern ensemble is not available. In [8] authors define the notion of a well-separated pair decomposition of points in d-dimensional space and develop efficient sequential and parallel algorithms for computing such a decomposition. The authors then present a decomposition of multidimensional point sets and its applications to k-Nearest-Neighbors and n-body potential fields. Their parallel algorithm is based on a CREW PRAM model. These are some papers related to parallel clustering and K-means algorithms that are known to us.

Many applications for clustering algorithms, particularly applications in data mining, usually require the algorithms to work on massive data sets with an acceptable speed. For instance, in [6] NASA launches satellites for studying the earth's ecosystems. The Earth Observing System (EOS) is capable of generating about a terabyte of data per day. These terabytes of data will then be used to identify anomalies on earth by a visualization program. A grouping of such data sets could be done by clustering algorithms. However, when a data set is large, processing time and space requirement by the serial algorithms have become a serious concern. To our knowledge, no parallel non-heuristic K-means clustering algorithm has been developed on a message-passing model of a network of workstations (NOWs). We are interested in developing K-means algorithm because it is simple and widely used in practice. In addition, we are also motivated by an advantage of wide availability and relatively inexpensive costs of parallel computing on a network of workstations. Our contributions in this paper are then 1) to significantly reduce time complexity of the serial K-means algorithm by data parallelism and 2) to eliminate a limitation of memory requirement on a single machine when data sets are massively large.

2.2 Materials and Parallel K-means Algorithm

We use a network of homogeneous workstations with Ethernet network and use message-passing for communication between processors. In an Ethernet network, all communications consist of packets transmitted on a shared serial bus available to all processors [9]. Message-Passing Interface (MPI) is used as a library routine in C programming language for communication between processes. The following is a description of the parallel K-means clustering algorithm. A master-slave single program multiple data approach (SPMD) is used.

Let $T_{startup}$ be a constant time needed in sending a blank message and let T_{data} be a constant time needed to send one data element (i.e., 4 bytes of integer). Note that in practice these two time constants may vary from one system to another.

Master Process	Complexity
1.Randomly form K equal subsets of set S	
2.Send each subset to each of the K slaves	$K(T_{startup}+ N/KT_{data})$
3.Receive K resulting subsets from K slaves.	

Slave Process	Complexity
1.Receive a vector subset P from master process	
2.While Error E is not stable:	
3. Compute a mean $\bar{X}_{myrank}$ of the subset P	$\|P\|$
4. Broadcast the mean $\bar{X}_{myrank}$ to every other slaves	$T_{startup} + T_{data}$
5. Compute distance d(i,j), $1\le i\le K, 1\le j\le \|P\|$ of each vector in P such that $d(i,j) = \| \bar{X}_i - v_j \|$	$K\|P\|$
6. Choose vector members of the new K subsets according to their closest distance to $\bar{X}_i$, $1\le i \le K$	$K\|P\|$
7. Broadcast K subsets computed in step 6 to every other slaves	$T_{startup} + \|P\|T_{data}$
8. Form the new subset P by collecting vectors that belong to $\bar{X}_{myrank}$ that were sent from other slaves in step 7	$\|P\|$
9.End	
10.Send the subset P to master process	$T_{startup} + \|P\|T_{data}$

It is worth noting that broadcasting in this algorithm does not undermine the overall performance. In fact, broadcasting is intentionally used in order to improve the performance of the algorithm because broadcasting only requires one setup time for each broadcast while a pair of send() and receive() requires one setup time each time the message is sent from one process to another. Since the setup time for each message passing in MPI is large, broadcasting helps alleviate this problem significantly while achieving the same effect as send() and receive().

2.3 Time/Space Complexity Analysis

We analyze the algorithm into communication steps and computation steps. Let T_{comm} be the time complexity of communication and T_{comp} be the time complexity of computation. There are 4 communication phases and 3 computation phases in the algorithm. Each phase can be described as follows.

Phase 1: Master process sends K equal subsets to K slaves. Thus,

$$T_{comm1} = K(T_{startup} + N/KT_{data})$$

Phase 2: After each slave process receives a subset from master process, it computes its mean $\bar{X}_{myrank}$ from subset P. This step takes

$$T_{comp1} = |P|$$

Phase 3: In this phase each slave broadcasts its own mean to every other slave. Thus,

$$T_{comm2} = T_{startup} + T_{data}$$

Phase 4: Each slave computes Euclidean distance of each vector in subset P and computes vector members of the new K subsets. This phase takes

$$T_{comp2} = 2K|P|$$

Phase 5: In this phase each slave broadcasts K subsets to every other slave. Thus,

$$T_{comm3} = T_{startup} + |P|T_{data}$$

Phase 6: Each slave forms the new subset P. Hence,

$$T_{comp3} = |P|$$

Phase 7: Each slave sends its subset P to master. Hence,

$$T_{comm4} = T_{startup} + |P|T_{data}$$

Let R_p be the number of iterations of the while loop at step 2, TCM be total communication time, and TCP be total computation time. The total time complexity can then be computed.

Total time = TCM + TCP

$$\begin{aligned} TCM &= T_{comm1} + T_{comm2} + T_{comm3} + T_{comm4} \\ &= K(T_{startup}+ N/KT_{data}) + \\ &\quad R_p(2T_{startup}+(|P|+1)T_{data}) + T_{startup} + |P|T_{data} \\ &= (2R_p+K+1)T_{startup} + (N+R_p(|P|+1)+|P|)T_{data} \\ &= O(N+R_p|P|) \end{aligned}$$

$$\begin{aligned} TCP &= T_{comp1} + T_{comp2} + T_{comp3} \\ &= R_p(2|P|+2K|P|) = O(2R_pK|P|) \end{aligned}$$

$$\begin{aligned} \text{Total time} &= O(N+R_p|P|) + O(2R_pK|P|) \\ &= O(2R_pK|P|) \end{aligned}$$

Given a uniformly distributed data set of N vectors, each slave process on one machine requires space of O(N/K). Hence, the parallel K-means algorithm has total space complexity O(N).

3. Experimental Results

Both versions of K-means algorithms are run with K = 4 and D = 2. All experimental input data are uniformly distributed random vectors. Execution times, the number of iterations, and speedup are measured as follows:

N D=2	Iter. Serial	Iter. Para	Exec. Sec. Serial	Exec. Sec. Para	Speedup
100K	35	34	127	268	-
200K	22	22	174	328	-
300K	32	31	351	648	-
400K	31	31	472	859	-
500K	27	28	543	1044	-
600K	26	26	946	1749	-
700K	29	31	3683	3322	1.11
800K	36	36	6871	5923	1.16
900K	28	28	13146	6248	2.10
1000K	-	25	-	8216	-

From the table, we observe that there is no speedup from N = 100,000 to N = 600,000 because communication time is much greater than computation time. (The MPI library seems to require a lot of time for initialization and cleanup stages. In addition, a MPI broadcast is a simulation on a single bus.) However, from N = 700,000 to N = 900,000, the computation time starts to dominate the communication time, resulting in substantial speedup. The parallel version also allows one to use larger problem sizes because data are distributed to several machines. In the experiment the serial version could not continue when N = 1,000,000 because a single machine did not have sufficient memory to meet computational requirement. The parallel version is expected to gain more speedup as N increases and computation time starts to dominate communication time. However, due to our limited memory space, we can show experimental results of N as large as 1,000,000 vectors (See figure 2).

4. Conclusion

Time complexity of the serial K-means algorithm is $O(R_sKN)$ whereas time complexity of the parallel K-means algorithm is $O(2R_pK|P|)$. Since R_s and R_p are inherently equivalent and $N >> |P|$, $O(2R_pK|P|)$ is asymtotically better than $O(R_sKN)$.

Suppose N is sufficiently large and data is uniformly distributed. |P| would be close to N/K. We can measure performance of the parallel K-means algorithm against the performance of the serial algorithm by using speedup S(K), where K is the number of processors:

$$S(K) = \frac{\text{Execution time of single processor}}{\text{Execution time of K processors}}$$

$$= \frac{O(RsKN)}{O(2RpK|P|)} = \frac{O(RsK2|P|)}{O(2RpK|P|)}$$

$$= O(K/2)$$

The experiment confirms that when N is sufficiently large, speedup gain is O(K/2) as predicted.

In a system with P processors a possible maximum speedup is P. In our parallel algorithm the number of processors P is essentially equal to K. Let T be the efficiency defined as

T = (speedup gain/maximum speedup)100%

Thus, efficiency of parallel K-means algorithm

T = (K/2)/K x 100% = 50%

We can conclude that our parallel K-means algorithm achieves 50% efficiency of time complexity. 50% is relatively efficient and cost effective if we consider that the system used in this paper is an Ethernet-based message-passing system and that the K-means clustering algorithm operates globally by nature. In terms of space complexity, the parallel K-means algorithm has the same total complexity O(N) as the complexity of the serial version. However, the parallel version allows one to use larger problem sizes because of its distributive nature. The parallel algorithm can scale a problem size up to O(K) times the size of the problem on a single machine.

5. Future Work

The speedup of the proposed algorithm may be improved even more significantly if we reduce communication time by grouping two or more subsets together on one machine. However, this will likely affect scalability of the algorithm. In addition, our algorithm needs to use exactly K machines to operate. Future work could be making the algorithm more flexible by allowing it to adapt itself into any number of machines. In [9] the authors work on overlapping connectivity Ethernets. Their configuration can be expected to reduce even more communication time of our algorithm significantly without having impacts on scalability. Besides, because most clustering applications tend to apply to a massive size of data

sets, we are interested in finding a method to partition a data set by K-means algorithm in such a way that we can temporarily ignore some resulting partitions in order to work on a few chosen specific partitions without sacrificing clustering quality. A motivation is to save some significant working space and we believe, by means of doing so, we can find a way to add a new portion of a data set into the existing partitions without having to rerun the K-means algorithm from the start. Some partially-related work has already been studied in [8]. A by-product of this study is a domain decomposition for possibly applying Divide-and-Conquer strategies to parallelize the algorithm for better speedup.

6. Acknowledgements

We would like to thank the Department of Electrical Engineering and Computer Science at Tufts University for allowing us to use computer facilities in our experiment.

7. References

[1] M. Inaba, N. Katoh, and H.Imai, "Application of Weighted Voronoi Diagrams and Randomization to Variance-Based k-Clustering", *Proceedings of the 10th Annual Symposium on Computational Geometry*, 1994. pp. 332

[2] T. Kohonen, "Self-Organizing Maps", Springer Series in Information Sciences, 1995.

[3] O. Zamir and O. Etzioni, "Web Document Clustering: A Feasibility Demonstration", *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1998, pp. 46-54

[4] N. K. Ratha, A. K. Jain, and M. J. Chung, "Clustering using a coarse-grained parallel Genetic Algorithm: A Preliminary Study", *Proceedings of the 1995 Computer Architectures for Machine Perception*, 1995, pp. 331-338

[5] S. Guha, R. Rastogi, and K. Shim, "CURE: An Efficient Clustering Algorithm for Large Databases", *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1998, pp. 73-84

[6] K. Assiter, K. P. Lentz, A. Couch, and C. Currey, "Locating Anomalies in Large Data Sets", *Society for Computer Simulation Military, Government, and Aerospace Simulation*, April 5, 1998, pp. 218-223

[7] C. Chinrungrueng and C. H. Sequin, "Optimal Adaptive K-Means Algorithm with Dynamic Adjustment of Learning Rate", *IEEE Transaction on Neural Networks*, January 1995, pp. 157-169

[8] P. B. Callahan and S. R. Kosaraju, "A Decomposition of Multidimensional Point Sets with Applications to k-Nearest-Neighbors and n-Body Potential Fields", *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 1992, pp. 546

[9] B. Wilkinson and M. Allen, "Parallel Programming 1st Edition", Printice-Hall Inc., 1999

Sanpawat Kantabutra received an B.A. degree in accountancy from Chiang Mai University, Chiangmai, Thailand, and an M.S. degree in computer engineering from Syracuse University, Syracuse, New York, USA, and is currently a Ph.D. student in computer science at Tufts University, Medford, Massachusetts, USA. He is also a scholarship recipient of the Ministry of University Affairs of Thailand and will be on the faculty of the Computer Science Department at Chiang Mai University after graduation. His research interests include parallel computing, visualization, and theory of computation.

Alva L. Couch received a S.B. degree in architecture from the Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, and an M.S. and Ph.D. in mathematics from Tufts University, Medford, Massachusetts, USA. He is currently an Associate Professor of computer science at Tufts University. His research interests include parallel computing, visualization, and human interface design.

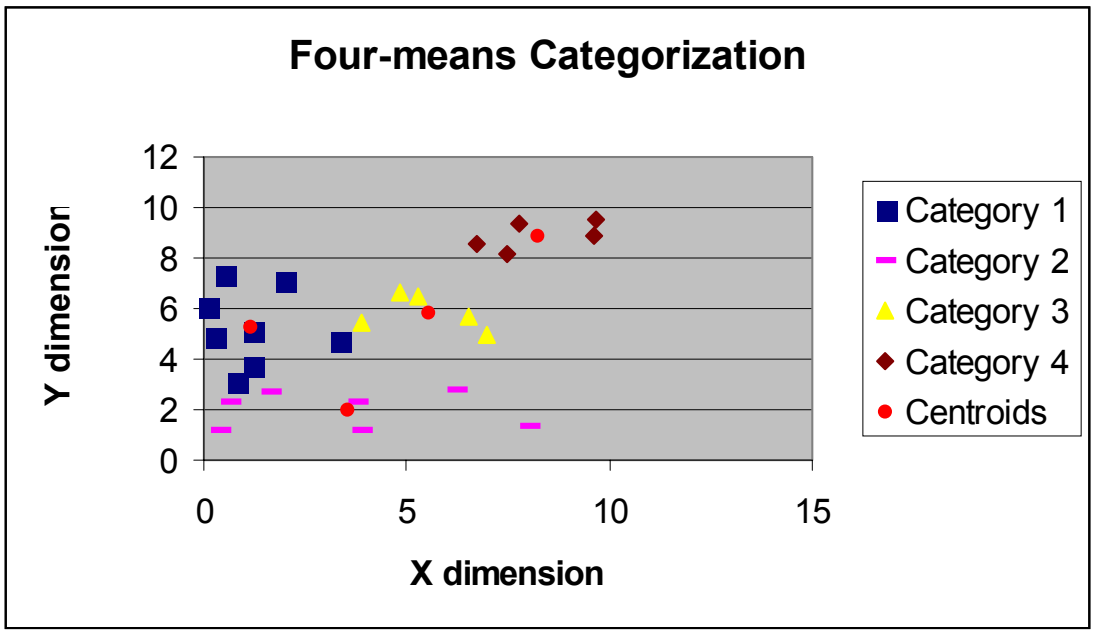

Figure 1. Two-dimensional data are clustered into 4 categories by running K-means algorithm on the data. Each color represents each category. The centroids are the representatives of each category.

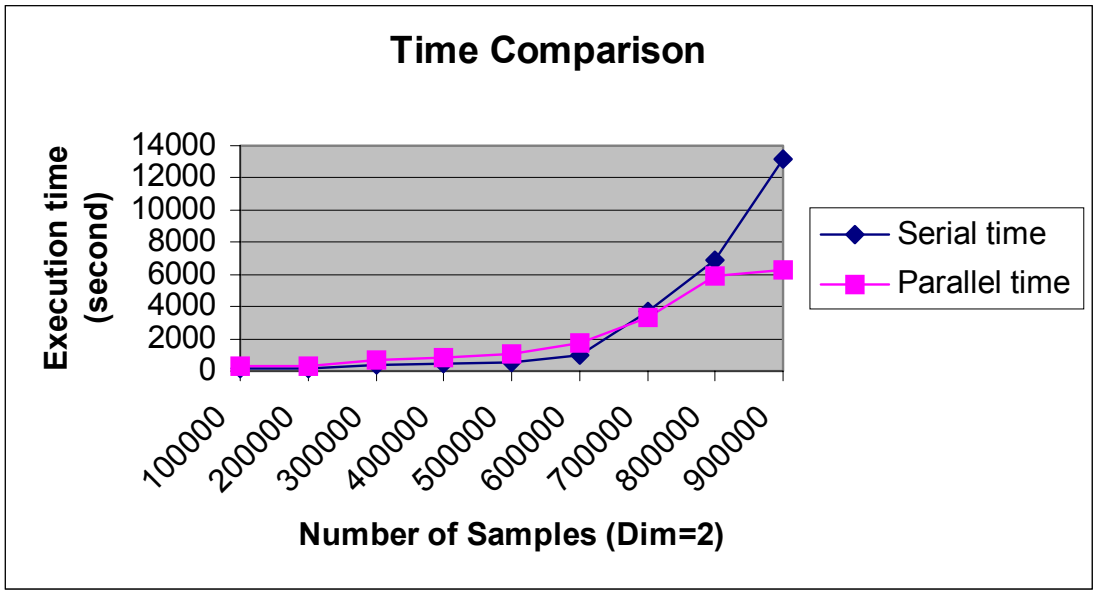

Figure 2. The running time of parallel and serial K-means algorithms are shown in the graph. It can be seen that the parallel time is less than the serial time when the number of samples is greater than 700,000 and it is about twice less than the serial time when the number of samples is 900,000.