Linear Speed-Up for a Parallel Non-Approximate Recasting of Center-Based Clustering Algorithms, including K-Means, K-Harmonic Means, and EM

George Forman, Bin Zhang
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2000-93
July 13th, 2000*

E-mail: gforman,bzhang@hplhp.com

Expectation-Maximization, multidimensional data clustering, data mining, very large databases, parallel algorithms, scale-up

Data clustering is one of the fundamental techniques in scientific data analysis and data mining. It partitions a data set into groups of similar items, as measured by some distance metric. Over the years, data set sizes have grown rapidly with the exponential growth of computer storage and increasingly automated business and manufacturing processes. To cluster such large data sets, efficient parallel algorithms are called for, both to reduce the computation time, and to bring the resources of multiple machines to bear on a given large problem in order to scale up the largest problem size one can handle.

We describe a technique for parallelizing center-based data clustering algorithms. The central idea is to communicate only sufficient statistics, yielding linear speed-up with excellent efficiency. The technique does not involve approximation and may be used orthogonally in conjunction with sampling or aggregation-based methods, such as BIRCH, to lessen the quality degradation of their approximation or to handle larger data sets.

We have demonstrated elsewhere the decomposition and theoretical speed-up behaviors for three clustering algorithms: K-Means, K-Harmonic Means and Expectation Maximization (EM) [Zhang, Hsu, Forman '00]. Here we present experimental measurements of their parallel behaviors, e.g., 93% speed-up efficiency for EM on a million data points spread over 128 computers connected via 100baseT Ethernet.

* Internal Accession Date Only
Approved for External Publication

Linear Speed-Up for a Parallel Non-Approximate Recasting of Center-Based Clustering Algorithms, including K-Means, K-Harmonic Means, and EM [1]

George Forman and Bin Zhang
Hewlett-Packard Labs, Data Mining Solutions Dept.
1501 Page Mill Rd. MS 1U-4; Palo Alto, CA 94304
gforman,bzhang@hpl.hp.com

July 12, 2000

ABSTRACT

Data clustering is one of the fundamental techniques in scientific data analysis and data mining. It partitions a data set into groups of similar items, as measured by some distance metric. Over the years, data set sizes have grown rapidly with the exponential growth of computer storage and increasingly automated business and manufacturing processes. To cluster such large data sets, efficient parallel algorithms are called for, both to reduce the computation time, and to bring the resources of multiple machines to bear on a given large problem in order to scale up the largest problem size one can handle.

We describe a technique for parallelizing center-based data clustering algorithms. The central idea is to communicate only sufficient statistics, yielding linear speed-up with excellent efficiency. The technique does not involve approximation and may be used orthogonally in conjunction with sampling or aggregation-based methods, such as BIRCH, to lessen the quality degradation of their approximation or to handle larger data sets.

We have demonstrated elsewhere the decomposition and theoretical speed-up behaviors for three clustering algorithms: K-Means, K-Harmonic Means and Expectation Maximization (EM) [Zhang, Hsu, Forman '00]. Here we present experimental measurements of their parallel behaviors, e.g., 93% speed-up efficiency for EM on a million data points spread over 128 computers connected via 100baseT Ethernet.

Keywords

Expectation-Maximization, multidimensional data clustering, data mining, very large databases, parallel algorithms, scale-up.

1. INTRODUCTION

Multidimensional data clustering is one of the principal tools data mining, scientific data analysis, and visualization. Its purpose is to organize a dataset into a set of groups, or clusters, which contain "similar" data items, as measured by some distance function. *Center-based* clustering algorithms iteratively position abstract 'centers,' which define a Voronoi partition on the data space (each data item belongs to the center that it is closest to, as in K-Means), or into fuzzy clusters given by the local density functions (as in K-Harmonic Means or EM). Example applications include document categorization, customer/market segmentation and exploratory analysis of the US Census data [e.g. 14,12]. The latter represents an example of a very large data set—an increasing trend brought about by advances in computer technology, Internet connectivity, and pervasive automation of science, business, and manufacturing processes.

The many algorithms for data clustering developed in recent decades all face a major challenge in scaling up to very large database sizes. Clustering algorithms with quadratic (or higher order) computational complexity, such as agglomerative algorithms [7], do not scale up. Even for more efficient algorithms, such as K-Means and Expectation-Maximization (EM), which have linear cost per iteration, research is needed to improve their scalability for ever growing data sets.

In a companion paper [17], we developed a class of iterative parallel parameter estimation algorithms—covering K-Means [10] [6], K-Harmonic Means [16], and EM algorithms [4]0—that is both efficient and operates without approximation of the original algorithms. For a detailed decomposition and theoretical analysis, refer to that paper. In this paper, we briefly lay out the parallel algorithms in Section 2 and discuss our experiments to determine their empirical speed-up and scale-up behaviors in Section 3. But first, we provide some background on parallel data-clustering algorithms.

1.1 Background

There have been several recent publications in scaling up K-Means and EM by approximation. For example, in BIRCH [18] and in the technical reports [1][3], a single scan of the data and subsequent aggregation of each local cluster into a single representative "point" containing sufficient statistics for the points it represents enables a data set to be pared down to fit the available memory. Such algorithms provide an approximation to the original algorithm and have been successfully applied to very large datasets. However, the higher the aggregation ratio, the less accurate the results are in general. It is also reported in the BIRCH paper that the quality of the clustering depends on the original scanning order.

[1] ACM SIGKDD Workshop on Distributed and Parallel Knowledge Discovery, KDD-2000, Boston, MA, August 20, 2000.

There are many papers on cleverly organizing the data set or the centers to optimize sequential K-Means [e.g. 6,13]. Some of these optimizations may be combined with the parallel K-Means presented here to reduce the runtime cost of K-Means on each processor independently. These types of optimizations typically demand additional memory, but this can be provided by leveraging the resources of multiple computers.

There is also recent work on non-approximated, parallel versions of K-Means. The Kantabutra and Couch algorithm [8] re-broadcasts the data set to all computers each iteration, which leads to heavy network loading and significant communication protocol processing overhead. Their analytical and empirical analysis estimates 50% utilization of the processors. The technology trend is for processors to improve faster than networks are improving, making the network a greater bottleneck in the future. Also, their algorithm limits the number of computing units to the number of clusters to be found. The parallel algorithm by Dhillon and Modha [5] was discovered independently and is an example of the class of parallel algorithms we described in [17]. This paper extends their result to 8 times as many processors and covers two other algorithms.

In our previous paper [17], we described a parallel decomposition of a class of iterative parameter relaxation methods, which includes center-based clustering algorithms, that limits inter-processor communication to sufficient statistics only, reducing the network bottleneck. The dataset is partitioned randomly across the memory of the processors and does not need to be transferred between iterations. Load balancing can be achieved trivially by migrating data points selected arbitrarily. The number of computing units is not limited in any way by the number of clusters sought. There is no approximation introduced by the method; the results are exactly as if the original algorithm were run on a single computer. The method can be used in conjunction with sampling or aggregation techniques (as in BIRCH); by combining with our approach, even larger data sets can be handled or better accuracy can be achieved by less aggregation.

We emphasize running the parallel clustering algorithm on existing networking structures (LAN) because of practical and economic considerations. The total computing resources in a collection of 'small' computers, modern PCs or desktop workstations, easily exceeds the total computing resources available in a supercomputer. Small computers, which are already everywhere, are much more accessible and can even be considered a free resource if they can be used when they would otherwise be idle.

2. PARALLEL SUFFICIENT STATISTICS

To find K centers, a center-based data clustering problem is formulated as an optimization (minimization) of a performance function, $Perf(S, M)$, depending on both the N D-dimensional points in the data set S and the K center location estimates M. A popular performance function for measuring the goodness of a clustering is the total within-cluster variance, or the sum of the mean-square error (MSE) of each data point to its center. The K-Means algorithm attempts to find a local optimum for this performance function, and is one of the most popular, used widely across many disciplines for its easily interpretable result. The K-Harmonic Means (KHM) algorithm optimizes the harmonic average of these distances. The advantages of KHM are that its convergence rate can be adjusted with a parameter, and it is highly insensitive to the initialization of the center locations, a major problem for K-Means that many authors have tried to address with clever initializations [2]. The Expectation-Maximization (EM) algorithm is also widely used and, in addition to the centers, optimizes a covariance matrix and a set of mixing probabilities.

These three algorithms fit a class of iterative center-based clustering algorithms that parallelizes as follows:

1. Arbitrarily distribute the N elements of the data set S to the local memories of a set of P computers.
2. Pick the K initial center location estimates M by any scheme, such as a random sample. A coherent copy is kept on each computer throughout the computation.
3. Iterate until the performance function converges, or after a fixed number of iterations:
   - 3.1. Each node independently computes its contribution to a set of global sufficient statistics SS, which includes information for computing the performance function.
   - 3.2. Global reduction (summation across processors) of the sufficient statistics, followed by broadcasting the global results back to all nodes.
   - 3.3. Independent local computation to adjust the center location estimates. The results are identical on each computer and are exactly the same as the uniprocessor sequential algorithm would produce.
4. Output all the centers M.

Regarding the distribution of the data set: the initial partitioning is random and has nothing to do with the clustering structure in the data. It has no effect on the computed results and is static, unless one wishes to migrate some data points for load balancing to enhance speedup efficiency. The sizes of the partitions, besides being constrained by the storage of the individual units, are ideally set to be proportional to the speed of the computing units. Partitioned thus, it will take about the same amount of time for each unit to finish its computation in each iteration, improving overall efficiency.

What varies among the individual algorithms are the sufficient statistics, how they are used to update the centers, and the performance function:

2.1 K-Means

The K-Means algorithm is a two step iteration corresponding to steps 3.1 and 3.3 above:

1. For each data item, assign it to the closest center, resolving ties arbitrarily. A proof can be found in [6] that this phase gives the optimal partition for the given centers.
2. Recalculate all the centers. Each center is moved to the geometric centroid of the points assigned to it. A proof can be found in [6] that this phase gives the optimal center locations for the given partition of data.

The (local contribution to the) sufficient statistics for this algorithm consist of (1) for each center, the count and vector sum of the (local) data points assigned to it, and (2) the sum of the squared distance from each (local) point to its center—the performance function. Once the sufficient statistics are globally totaled, the vector sums divided by the counts determine the revised center locations.

2.2 K-Harmonic-Means

The K-Harmonic Means iteration step adjusts the new center locations to be a weighted average over the entire data set, where the contribution weight for point $s \hat{I} S$ to center $m \hat{I} M$ is given by

$$\frac{1}{\| s-m \|^{c} \left(\sum_{x \in M} \frac{1}{\| s-x \|^{2}} \right)^{2}}$$

where $\|s\text{-}m\|$ is the distance between the points, and c is a parameter in the range (2,4] that controls the convergence speed. (For comparison, in K-Means the weights are membership functions selecting the nearest center.)

The sufficient statistics for this algorithm are, for each center, the sum of the weights and the weighted vector sum of the (local) data points. As before, the quotient of these determines the revised center locations.

2.3 Expectation-Maximization (EM)

Unlike K-Means and K-Harmonic Means in which only the centers are to be estimated, the EM algorithm also estimates the co-variance matrices and the mixing probabilities. It is an iterative algorithm with the following two steps:

E-Step: Estimate the percentage of $s \hat{I} S$ belonging to each cluster m, $p(m|s)$. This is done entirely locally for each point.

M-Step: With the fuzzy membership function from the E-Step, find the new center locations, co-variance matrices and mixing probabilities that maximize the performance function.

The revised center locations are determined by an average over the entire data set weighted by $p(m|s)$. The sufficient statistics for this algorithm are, for each center, the sum of the weights, the vector sum of the weighted (local) data points, and the weighted matrix sum of $s^T s$ for each data point (KD^2 doubles); plus a scalar for the performance function. Due to space limitations, we omit the details of the mathematics here; please refer to [17].

2.4 Analysis

Table 1 shows the computation and communication workload per node per iteration for each of the three algorithms. Since computation scales with N but communication does not, speed-up efficiency improves as N scales up. Increases in K or D, however, affect both computation and communication equally; EM scales both at D^2. As a practical matter for clustering, $N >> K$ and $N >> D$. Since N is dominant, the bottleneck is computation, and so increasing K or D will improve speed-up efficiency until the sufficient statistics run up against the network bandwidth. Typical message overhead has a large latency cost, but then handles additional bytes more efficiently. Hence, significant increases can be withstood before communication slows down significantly.

Table 1. Comparison of algorithms

Algorithm	Computation	Communication
K-Means	$O(N K D / P)$	$1+K+KD$
K-Harmonic Means	$O(N K D / P)$	$K+KD$
EM	$O(N K D^2 / P)$	$1+K+KD+KD^2$

3. EXPERIMENTS

The ideal speed-up experiment compares a parallel implementation against the best sequential implementation available, though *relative speed-up* experiments compare only against the parallel code running on a single processor, where it wastes some overhead checking whether to send and receive from its potential neighbors, etc. As groundwork, we compared the performance of hand-written sequential C code, our parallel code running on a single processor, and MATLAB generated sequential C code. MATLAB is a high-level interpreted language and has the capability to generate C code for compilation. For many, MATLAB is the tool of choice, and only more so if its compiled performance is good. We found the parallel code gave the best uniprocessor performance, so although our comparisons are in fact relative, the results are true speed-up figures.

We wrote our implementation in the parallel language ZPL [15], which is especially adept at expressing data-parallel computations. For example, our ZPL subroutine for K-Means is significantly shorter and more readable than our *sequential* C version, which would inflate and obfuscate substantially were we to add inter-processor communication code. The ZPL compiler outputs optimized C code in the single-program multiple-data (SPMD) paradigm, which we compile and link with the MPI communication library, providing efficient reduction and broadcast primitives. Elapsed real-time measurements are taken by performing a barrier synchronization across all processors before and after many clustering iterations. We monitor page faults to make sure we are not exceeding memory capacity.

We ran our experiments on a network of workstations (NOW) so they would be generalizable to the computing infrastructure commonly available in business and academic settings. Results on a specialized multiprocessor with a low-latency, high bandwidth interconnect would only improve speed-up. We obtained measurements on a collection of 128 computers with 500 MHz Pentium III processors connected via 100baseT.

Because of the nature of our parallel algorithms, the actual data set values have absolutely no bearing on the performance measurements, unlike BIRCH. Hence, we used random data.

3.1 Speed-Up

Speed-up is computed as the time on a uniprocessor divided by the time on multiprocessor. Speed-up efficiency is speed-up divided by the number of processors P. The three figures on the following page show the speed-up and speed-up efficiency for each of the three algorithms as we vary the number of processors from 1 to 128 and data set sizes from 1 to 10 million (less a factor of ten for the slower algorithm EM), where K=100 and D=2. The graphs have uniform scales for visual comparison.

Overall, all three algorithms exhibit good linear speed-up. On the smaller problem size, K-Means loses efficiency at high numbers of processors. For P=128, there are less than 8000 points per processor, and each iteration takes about $1/20^{th}$ of a second. Communication begins to dominate at this point, but it still obtains ~70% speedup efficiency. K-Means falls off in efficiency faster than the other two algorithms because it has significantly less computation per point that can benefit from parallelism. KHM was approximately six times slower and EM was over 200 times slower per point/iteration in these measurements.

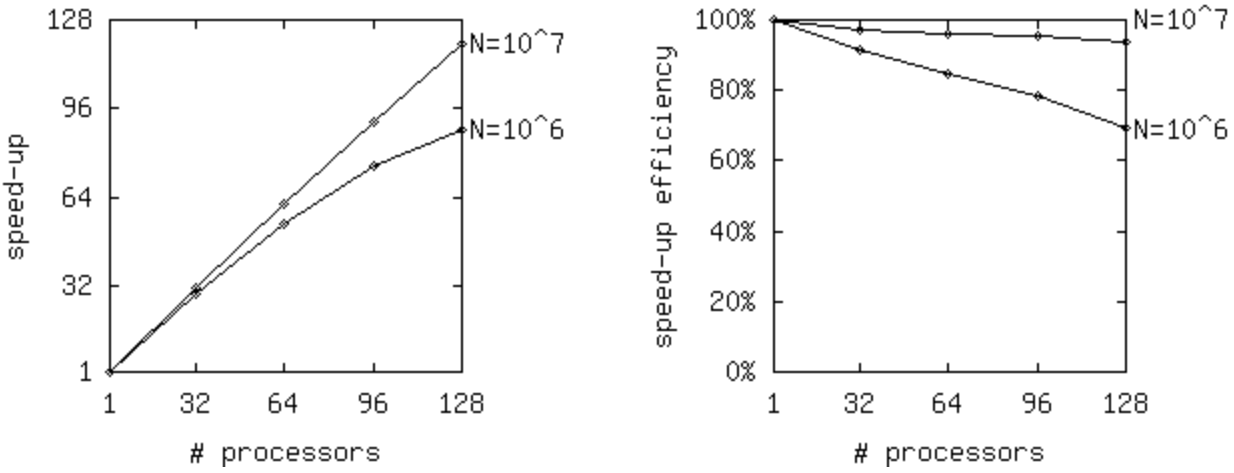

Figure 1. Speed-up and efficiency curves for K-Means

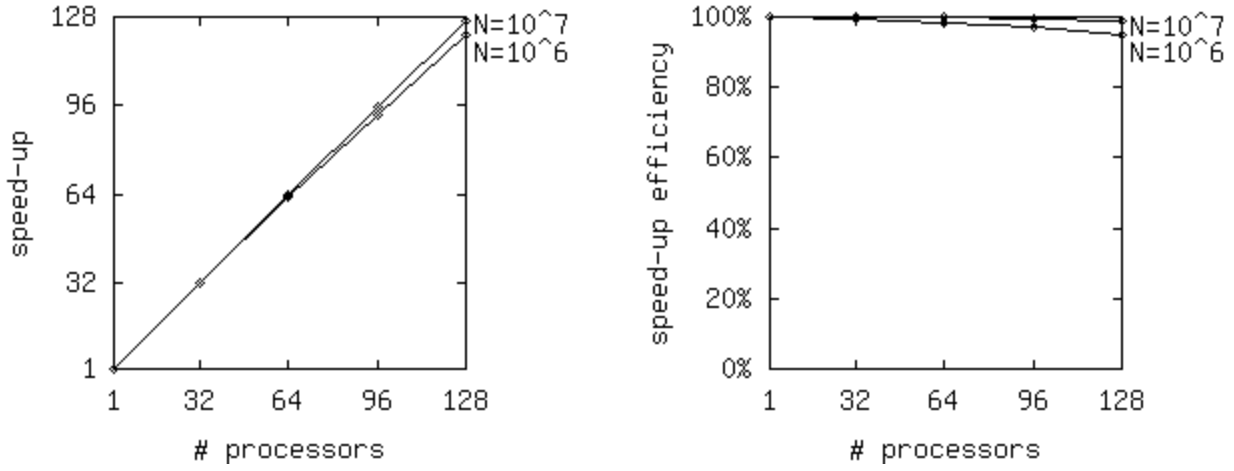

Figure 2. Speed-up and efficiency curves for K-Harmonic Means

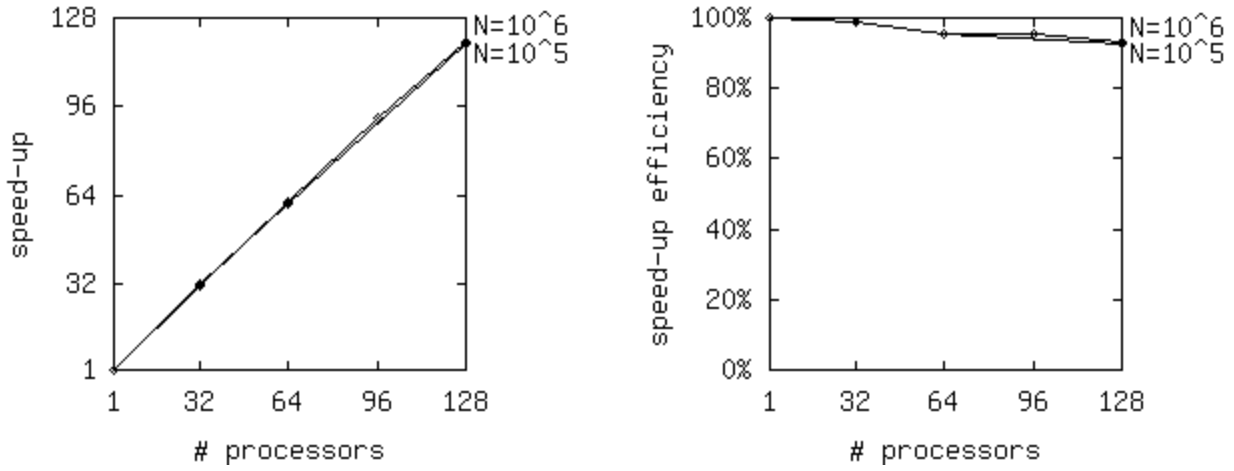

Figure 3. Speed-up and efficiency curves for Expectation-Maximization (EM)

Varying K & D: These results are for K=100 centers and D=2-dimensional data. Per the discussion in section 2.4, we varied K=10..1000 and D=2..200, measuring their effect on speedup efficiency for K-Means on 8 processors networked with a slow 10 Mbps Ethernet. Even with K=1000 and D=200 dimensions simultaneously, the computation bottleneck prevailed over the communication, and the speed-up efficiency only improved. At the low end, with K=10 and D=2, we saw a 7% drop in efficiency because of the lack of computation to parallelize, not because the sufficient statistics are large.

3.2 Scale-Up

In data mining with very large databases, the emphasis is on what the largest manageable data set size is. The question becomes: given that a single processor can handle data sets up to a given size, how efficiently can the we scale up the data set if a proportional number of processors are added simultaneously? This leads to separate questions for space and time:

Space: We consider only K-Means here—the other algorithms are similar. The major memory requirements for K-Means expressed on a per-processor basis are O(ND/P+2KD+2K) 8-byte doubles. Note that N/P dominates K. So, we expect linear scale-up. To validate this, we experimentally determined the largest data size N_1 (K=100, D=2) that fits on a workstation with 208 MB RAM without substantial paging. N_1=13 million. We repeated this for eight workstations, finding N_8 to be 104 million points (=8*N_1), yielding 100% efficiency, as expected. Similar results apply when varying K and D.

Time: Given that a data set can be clustered on a uniprocessor in time T_1, how does this scale as we increase the dataset size and processor pool size simultaneously? This is the question of *relative scale-up*, defined as the ratio of time for N points on 1 processor vs. P*N points on P processors.

In the analysis in Table 1, the time per iteration is proportional to N/P. Hence, multiplying N and P by the same factor should not change the iteration time. To validate this, we took measurements of K-Means and EM as we scaled the number of processors P=1..128 and the data set size N=P*N_1. (K=100, D=2) For K-Means, each node has 100,000 points, but since EM runs so much slower, we gave it just 1000 points per node. The results are presented in Figure 4. As we expect, the time per iteration is flat. (Considering the difference in problem sizes, EM is over 200 times slower per data point than K-Means.)

If we scale up the dataset by increasing K or D instead of N (with N still being dominant), all three algorithms scale similarly, except that EM is quadratic in D. For EM, a linear increase in the number of processors cannot hope to keep up with large increases in D. Fortunately, realistic data sets grow most naturally in N, occasionally in the number of clusters in the distribution, but rarely increase dramatically in dimensionality.

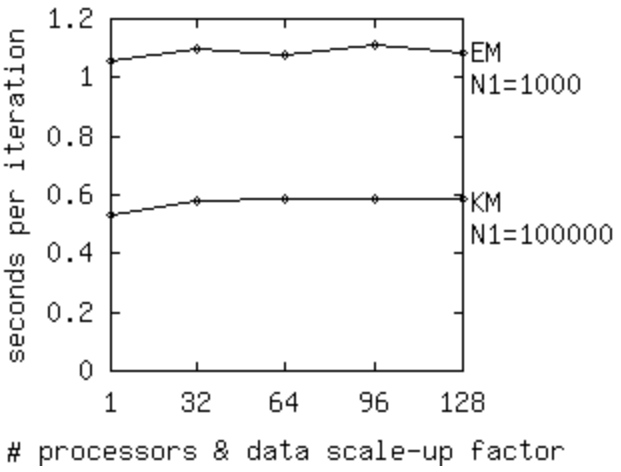

Figure 4. Scale-up curves for K-Means and EM.

4. CONCLUSIONS

By restructuring the mathematics of a class of iterative parameter estimation algorithms, we produce a straightforward parallel implementation based on communicating only a small amount of data—sufficient statistics—yielding highly efficient speed-up and scale-up for very large data sets. Here we have demonstrated the parallel transformation for three algorithms, and experimentally evaluated their speed-up efficiency.

Our experiments show that data sets can be effectively scaled up linearly in computing resources. A practical result is that large scale data clustering can be efficiently carried out on ordinary workstations connected via Ethernet, as science and business organizations commonly have available. Because such machines are typically heterogeneous in computing power and memory capacity, load balancing may be added to maximize efficiency. Fortunately, these algorithms allow fine grain balancing of arbitrary data points. Practical load balancing on heterogeneous machines is a topic for future work.

ACKNOWLEDGMENTS

We would like to thank professor Larry Snyder, Brad Chamberlain and the students of the ZPL Parallel Language and Optimizing Compiler research project at the University of Washington for ZPL and their support. Our deepest thanks to Sung-Eun Choi and the Advanced Computing Lab, Los Alamos National Labs, for running measurements on their large 'rockhopper' Linux cluster. Thanks also to the anonymous reviewers who helped improve this paper. We appreciate Douglas Low for cheerfully obtaining the compiled MATLAB measurements. MATLAB is a trademark of Mathworks Inc.

REFERENCES

[1] Bradley, P., Fayyad, U. M., and Reina, C.A., "Scaling EM Clustering to Large Databases," Microsoft Technical Report, 1998.

[2] Bradley, P., Fayyad, U. M., and Reina, C.A., "Refining Initial Points for KM Clustering", Microsoft Technical Report MSR-TR-98-36, May 1998.

[3] Bradley, P., Fayyad, U. M., and Reina, C.A., "Scaling Clustering to Large Databases", KDD98, 1998.

[4] Dempster, A. P., Laird, N.M., and Rubin, D.B., "Miximum Likelihood from Incomplete Data via the EM Algorithm", Journal of the Royal Statistical Society, Series B, 39(1):1-38, 1977.

[5] Dhillon, I.S. and Modha, D.S. "A data clustering algorithm on distributed memory machines," ACM SIGKDD Workshop on Large-Scale Parallel KDD Systems (with KDD99), August, 1999.

[6] Gersho & Gray, "Vector Quantization and Signal Compression", KAP, 1992

[7] Anil K. Jain, Richard C. Dubes, "Algorithms for Clustering Data (Prentice Hall Advanced Reference Series : Computer Science)", Prentice Hall, 1977.

[8] Kantabutra, S. and Couch, A.L., "Parallel K-Means Clustering Algorithm on NOWs", NECTEC Technical Journal, Vol. 1, No. 1, March 1999.

[9] Kaufman, L. and Rousseeuw, P. J., "Finding Groups in Data: An Introduction to Cluster Analysis", John Wiley & Sons, 1990.

[10] MacQueen, J. Some methods for classification and analysis of multivariate observations. Pp. 281-297 in: L. M. Le Cam & J. Neyman [eds.] Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, v.1. University of California Press, Berkeley. xvii + 666 p. 1967.

[11] McLachlan, G. J. and Krishnan, T., "The EM Algorithm and Extensions.", John Wiley & Sons, Inc., 1997.

[12] A commercial recommender system, http://www.netperceptions.com

[13] Pelleg, D. and Moore, A, "Accelerating Exact K-Means Algorithms with Geometric Reasoning", KDD-99, Proc. of the Fifth ACM SIGKDD Intern. Conf. On Knowledge Discovery and Data Mining, page 277-281.

[14] Ruocco A. and Frieder O., "Clustering and Classification of Large Document Bases in a Parallel Environment," Journal of the American Society of Information Science, 48(10), October 1997.

[15] Snyder, L., "A Programmer's Guide to ZPL (Scientific and Engineering Computation Series)", MIT Press; ISBN: 0262692171, 1999. See also: http://www.cs.washington.edu/research/zpl

[16] Zhang, B. and Hsu, M., "K-Harmonic Means – A Data Clustering Algorithm", Hewlett-Packard Labs Technical Report HPL-1999-124.

[17] Zhang, B., Hsu, M., and Forman, G. "Accurate Recasting of Parameter Estimation Algorithms using Sufficient Statistics for Efficient Parallel Speed-up: Demonstrated for Center-Based Data Clustering Algorithms," Hewlett-Packard Labs Technical Report HPL-2000-6. To appear at the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), September 13-16, 2000.

[18] Zhang, T., Ramakrishnan, R., and Livny, M., "BIRCH: an efficient data clustering method for very large databases", ACM SIGMOD Record, Vol. 25, No. 2 (June 1996), Pages 103-114 in: SIGMOD '96.