Distributed Data Clustering can be Efficient and Exact

George Forman
Hewlett-Packard Research Labs.
1501 Page Mill
Palo Alto, CA 94304
gforman@hpl.hp.com

Bin Zhang
Hewlett-Packard Research Labs.
1501 Page Mill
Palo Alto, CA 94304
bzhang@hpl.hp.com

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. Many of these datasets are geographically distributed across multiple sites, e.g. different sales or warehouse locations. To cluster such large and distributed data sets, efficient distributed algorithms are called for to reduce the communication overhead, central storage requirements, and computation time, as well as to bring the resources of multiple machines to bear on a given problem as the data set sizes scale-up. We describe a technique for parallelizing a family of 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 demonstrate in this paper that even for relatively small problem sizes, it can be more cost effective to cluster the data in-place using an exact distributed algorithm than to collect the data in one central location for clustering.

Keywords

multidimensional data clustering, data mining, very large databases, parallel algorithms, distributed computing.

1. INTRODUCTION

Multidimensional data clustering is one of the principal tools of 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 and EM). Example applications include document categorization, customer/market segmentation and exploratory analysis of the US Census data [e.g. RF97, NetPerception]. 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 [JD77], 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 ability to handle ever-growing data sets.

In a companion paper [ZHF00], we developed a class of parallel iterative parameter estimation algorithms, covering the center-based clustering algorithms K-Means [M67] [GG92], K-Harmonic Means [ZHD00a][Z00b], and EM [DLR77][MK97]. The parallelization is resource efficient and operates without approximation to the original sequential algorithms. For a detailed decomposition and analysis, refer to that paper.

Here, we extend the applicability of this class of parallel clustering algorithms to show that it works very well for clustering data that is inherently distributed, for example, click-stream log files at Web sites mirrored across the world. By applying the parallel version of the clustering algorithms, the data can be clustered in-place with the exact same computational result as if the data set had been assembled at a central site for clustering, but without:

1. the communication costs and delays for transmitting large volumes of data to the central site,
2. the extra storage space and computing resources that would be needed at the central site, and
3. the administration complexity needed to manage copies of the remote portions of the data set, replacing them when the distributed data change.

In Section 2 we briefly lay out the foundation of the parallel algorithm, and then in Section 3 we discuss our experiments to determine the algorithm's efficacy in a high-latency, geographically distributed situation vs. transferring the data to a centralized location for clustering. For the remainder of this section, we provide some background on data-clustering algorithms for large data sets.

1.1 Background

It is fundamentally challenging to perform data clustering on very large databases. There have been several recent publications in scaling up K-Means and EM by approximation. For example, in BIRCH [ZRL96] and in [BFR98] [BFR98a], 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 scale up 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.

There is also recent work on non-approximated, parallel versions of K-Means. The Kantabutra and Couch algorithm [KC99] 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; such an algorithm becomes completely impractical in a distributed wide-area networking (WAN) environment. Even in a local-area networking (LAN) environment, the technology trend is for processors to improve faster than networks are improving, making the network a greater bottleneck in the future. Finally, their algorithm limits the number of computing units to the number of clusters to be found. The parallel algorithm by Dhillon & Modha [DM99] was discovered independently and is an instance of the class of parallel algorithms we described in [ZHF00].

In our previous paper [ZHF00], we described a parallel decomposition for center-based clustering algorithms that limits inter-processor communication to sufficient statistics only, reducing the network bottleneck. The data set 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. The results are exactly as if the original algorithm were run on a single computer, i.e. no approximation. 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. Because the amount of communication is small, the parallel decomposition achieves excellent speed-up efficiency on networks of workstations without any special low-latency or high-bandwidth interconnect—note that the computing resources in a collection of PCs or desktop workstations can easily exceed the the total computing resources available in a supercomputer. Further, they can be considered a free resource if they can be utilized when they would otherwise be idle.

Because of the small amount of communication per iteration, in this paper commend this algorithm for the geographically distributed case where the interconnect is a high-latency, low-bandwidth WAN, instead of a common LAN.

2. PARALLEL SUFFICIENT STATISTICS

To find K cluster 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 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. 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:
   - 3.1. Each computer 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 computers.
   - 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. Stop when the performance function converges, or after a fixed number of iterations.
5. Output the K centers M.

Regarding the distribution of the data set: the partitioning may be arbitrary 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, optimizing overall efficiency.

What varies among the individual algorithms are the sufficient statistics, how they are used to update the centers, and the performance function. We describe these differences in the following three subsections.

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 [GG92] 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 [GG92] that this phase gives the optimal center locations for the given partition of data.

The (local contribution at each processor 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 $s \in S$ to center $m \in M$ is given by

$$\frac{1}{\| s-m \|^{p+2} \left(\sum_{m' \in M} \frac{1}{\| s-m' \|^{p}} \right)^{2}}$$

where $\|s\text{-}m\|$ is the distance between the data point s and the center m, and $p>2$ is a parameter. For details, see [Z00b]. 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 $p(m|s)$ of each data point s belonging to each cluster m. 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; plus a scalar for the performance function. Due to space limitations, we omit the details of the mathematics here; please refer to [ZHF00].

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	$O(N K D^2 / P)$	$1+K+KD+KD^2$

2.4 Theoretical Analysis

Table 1 shows the computation and communication workload per iteration per processor for each of the three algorithms: N is the number of points in the data set, D is the dimensionality of the problem, K is the number of centers to find, and P is the number of processors. Since for each algorithm, the amount of computation scales with N but communication volume 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, $N >> K$, $N >> D$, $N >> P$, and N is the most natural parameter to scale up,. Since N is dominant, the bottleneck is computation, and so increasing K or D will improve speed-up efficiency until the size of the sufficient statistics severely runs up against the network bandwidth limitations. Typical message overhead has a large start-up latency cost, but then additional bytes are transferred more efficiently, hence, significant increases can be withstood before communication slows significantly. For geographically distributed applications as we are considering in this paper, latency is significantly more than a LAN and bandwidth can be significantly less. If the connection is poor enough, little speed-up efficiency can be had. But this must be weighed against the alternative of first transferring the remote portions of the dataset over the poor connection to a central location for (parallel) clustering on a local-area network of workstations. We performed empirical measurements and analysis to determine the appropriate trade-off point.

3. EXPERIMENTS

In prior work, we conducted many experiments showing that the performance of the parallelized algorithms against the best available sequential implementations have great speed-up efficiency on a local-area network of workstations. In this paper, we show that for clustering problems of any substantial size, even for a distributed WAN environment, transferring the data to a central site for clustering is slower than clustering the distributed data in-place with the parallel algorithm paying a hefty communication delay per iteration. Therefore, in most real distributed applications, parallel clustering is preferred, especially since no approximation is required.

We ran our experiments on workstations separated by the Atlantic Ocean—Bristol, England, and Palo Alto, USA. Both are Hewlett-Packard C160's running HP-UX. Using the *ping* command, we measured the roundtrip communication latency between the two sites to average 140 ms. Using *FTP*, we measured the steady transfer rate between the two computers to average 235 KB/second. We repeated the set of experiments on several different days at different times of the day to get an average performance. We found that the communication latency varies little. The steady transfer rate varies more. Since the amount of sufficient statistics that have to be transferred over the network is typically small (e.g. a few kilobytes), the communication delay is dominated by latency. Hence, the variation on the steady transfer rate of the network has little impact on the performance of the distributed clustering algorithm.

We wrote our implementation in the parallel language ZPL [S99], which is especially adept at expressing data-parallel computations. 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 reduction and broadcast primitives. Elapsed real-time measurements are taken by performing a barrier synchronization across all processors before and after multiple clustering iterations to achieve sufficiently reliable timing granularity. We verified there were few page faults to make sure we were not mistakenly measuring the disk paging speed. 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. Our earlier papers showed the speedup of the distributed clustering algorithms in practice are insensitive to the

dimensionality D and the number of clusters K and is consistent for each of the three algorithms. In the interest of space and pith, we present here only the results for K-Means run with $D=2$ and $K=100$. We vary the data set size in N, supposing that half of the data is located at each site.

We measured the amount of time to bring all the data to one central place by carrying out the actual transfer via FTP multiple times and calculating the average. It took on average 68 seconds to transfer 16 MB of data (N = 1 million 2-D points) from England to the computer in Palo Alto, which is about 235,000 bytes/second. The communication of the sufficient statistics for 100 centers and 2 dimensional data comes to about 2400 bytes, which measured via *ping* takes 165 ms per iteration on the WAN, vs. 6 ms on a LAN. In order for it to be beneficial to first transfer the data from England to Palo Alto and then run parallel clustering on a pair of local workstations connected on a fast Ethernet, the following inequality should hold:

$$\textit{Data transfer time} < \textit{WAN communication overhead for parallel clustering}$$

which is (supposing the data representation is 8-bytes per floating point double and the data is split evenly between the two sites)

$$\frac{8*\frac{N}{2}*D}{transfer\ \ rate} < \#\ iterations\ *(WAN\ delay - LAN\ delay).$$

and solving for N yields approximately

$$N < 9400*\frac{number\ \ of\ \ iterations}{D}$$

Considering that 10 to 100 iterations may be needed for convergence, we determine that even for problem sizes of about 50,000 to 500,000 two-dimensional points, respectively, it should take less time to compute the clustering in-place across the WAN rather than transfer the data and cluster it in parallel on a LAN. Taking another view of this inequality, it says that for N=500,000 two-dimensional points, it is more efficient to perform distributed clustering in-place if convergence happens in under ~100 iterations.

To validate this result empirically, we measured the parallel clustering time for 50 iterations over our WAN vs. LAN as we varied N. The equilibrium point for our particular parallel implementation, which has not been particularly optimized for round-trip latency, came to N=80,000. This number will vary depending on the computer speed, data transfer rate over the network, latency of the network, the number of iterations of the algorithm to run, the dimensionality of the data, and the details of the communication in the implementation.

As an additional validation, Figure 1 shows the elapsed time per iteration for both LAN and WAN connections as we vary N from 10,000 to 2,000,000. We see that, except for very small (fast) problem sizes, the roughly constant latency overhead of the WAN is dwarfed compared to the computation time. Even if the ping time in another situation were twice as bad, it would affect the results little for significant problem sizes. We also see that as the problem size doubles so does the computation time. Hence, if only a uniprocessor were available at the central site, it would nearly always be more efficient to cluster in-place across the WAN.

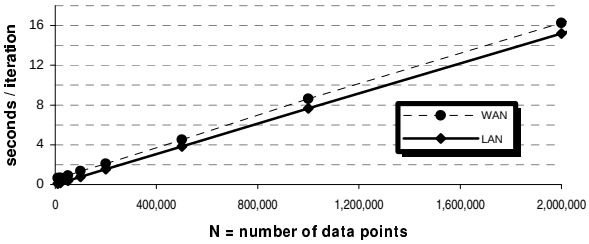

Figure 1: WAN vs. LAN Iteration times

For the purposes of analysis and exposition, we have kept the problem formulation simple. In actual practice, there may easily be more than two sites involved, the speeds of the machines at the various sites are likely to vary, as well as the size of the data partition at each site. Even so, considering the proportion of communication to computation shown in Figure 1 as data set sizes grow, in-place clustering across the wide-area network is likely to be more efficient than moving much data. If there were a large data imbalance, the algorithm allows for excess data points at one site to be moved (copied) to a site with spare computational resources to improve the time per iteration. Alternately, multiple computers could be employed at sites holding a disproportionately large fraction of the data set.

4. CONCLUSIONS

By restructuring the mathematics of a class of iterative parameter estimation algorithms, we produced 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. In this paper we have promoted its use for clustering problems whose data are naturally distributed, showing that even for small datasets, it can be worthwhile to accept the additional latency of a wide-area network, compared to the time needed to copy the remote data to central site for parallel clustering. In addition, other drawbacks to copying the data include the excess storage and computation resources that need to be available at the central site to process the duplicated data, as well as the administrative burden of copying remote portions of the dataset to the central site & keeping the copies coherent with the remote data as it grows.

5. ACKNOWLEDGMENTS

We would like to thank Prof. Larry Snyder, Douglas Low and the other students of the ZPL Parallel Language and Optimizing Compiler research project at the University of Washington for ZPL and their support.

6. REFERENCES

[BF98] Bradley, P., and Fayyad, U.M., "Refining Initial Points for KM Clustering," Microsoft Technical Report 98-36, May 1998.

[BFR98] Bradley, P., Fayyad, U. M., and Reina, C.A., "Scaling EM Clustering to Large Databases," Microsoft Technical Report, 1998.

[BFR98a] Bradley, P., Fayyad, U. M., and Reina, C.A., "Scaling Clustering to Large Databases," KDD98, 1998.

[DLR77] 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.

[DM99] 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.

[GG92] Gersho & Gray, "Vector Quantization and Signal Compression," KAP, 1992

[JD77] Anil K. Jain, Richard C. Dubes, "Algorithms for Clustering Data (Prentice Hall Advanced Reference Series : Computer Science)," Prentice Hall, 1977.

[KC99] Kantabutra, S. and Couch, A.L., "Parallel K-Means Clustering Algorithm on NOWs," NECTEC Technical Journal, Vol. 1, No. 1, March 1999.

[KR90] Kaufman, L. and Rousseeuw, P. J., "Finding Groups in Data : An Introduction to Cluster Analysis," John Wiley & Sons, 1990.

[M67] 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, Vol. 1. University of California Press, Berkeley. xvii + 666 p. 1967.

[MK97] McLachlan, G. J. and Krishnan, T., "The EM Algorithm and Extensions," John Wiley & Sons, Inc., 1997.

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

[RF97] 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.

[S99] 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

[ZHD00a] Zhang, B., Hsu, M. and Dayal, U. (2000). "*K-Harmonic Means: A Spatial Clustering Algorithm with Boosting*." In Proc. International Workshop on Temporal, Spatial and Spatio-Temporal Data Mining, TSDM2000, Lyon, France, Lecture Notes in Artificial Intelligence, 2007. Roddick, J. F. and Hornsby, K., Eds., Springer.

[Z00b] Zhang, B. "*Generalized K-Harmonic Means -- Boosting in Unsupervised Learning*", Hewllet-Packard Laboratories Technical Report: http://www.hpl.hp.com/techreports/2000/HPL-2000-137.html.

[ZHF00] 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," 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), September 13-16, 2000. Also available as Hewlett-Packard Labs Technical Report HPL-2000-6.

[ZRL96] Zhang, T., Ramakrishnan, R., and Livny, M., "BIRCH: an efficient data clustering method for very large databases," ACM SIGMOD Record, Vol. 25, No. 2, pages 103-114, June 1996.