Accurate Recasting of Parameter Estimation Algorithms Using Sufficient Statistics for Efficient Parallel Speed-up—Demonstrated for Center-Based Data Clustering Algorithms

Bin Zhang, Meichun Hsu, George Forman
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2000-94
July 17, 2000*

E-mail: bzhang, mhsu, gforman@hpl.hp.com

parallel algorithms, data mining, data clustering, K-Means, K-Harmonic Means, Expectation-Maximization, speed-up, scale-up, sufficient statistics

Fueled by advances in computer technology and online business, data collection is rapidly accelerating, as well as the importance of its analysis—data mining. Increasing database sizes strain the scalability of many data mining algorithms. Data clustering is one of the fundamental techniques in data mining solutions. The many clustering algorithms developed face new challenges with growing data sets. Algorithms with quadratic or higher computational complexity, such as agglomerative algorithms, drop out quickly. More efficient algorithms, such as K-Means EM with linear cost per iteration, still need work to scale up to large data sets. This paper shows that many parameter estimation algorithms, including K-Means, K-Harmonic Means and EM, can be recast without approximation in terms of Sufficient Statistics, yielding a superior speed-up efficiency. Estimates using today's workstations and local area network technology suggest efficient speed-up to several hundred computers leading to effective scale-up for clustering hundreds of gigabytes of data. Implementation of parallel clustering has been done in a parallel programming language, ZPL. Experimental results show above 90% utilization.

* Internal Accession Date Only Approved for External Publication

Accurate Recasting of Parameter Estimation Algorithms using Sufficient Statistics for Efficient Parallel Speed-up

Demonstrated for Center-Based Data Clustering Algorithms

Bin Zhang, Meichun Hsu and George Forman
Data Mining Solutions Department, Hewlett-Packard Laboratories
bzhang@hpl.hp.com, mhsu@hpl.hp.com, gforman@hpl.hp.com

Abstract Fueled by advances in computer technology and online business, data collection is rapidly accelerating, as well as the importance of its analysis—data mining. Increasing database sizes strain the scalability of many data mining algorithms. Data clustering is one of the fundamental techniques in data mining solutions. The many clustering algorithms developed face new challenges with growing data sets. Algorithms with quadratic or higher computational complexity, such as agglomerative algorithms, drop out quickly. More efficient algorithms, such as K-Means EM with linear cost per iteration, still need work to scale up to large data sets. This paper shows that many parameter estimation algorithms, including K-Means, K-Harmonic Means and EM, can be recast without approximation in terms of Sufficient Statistics, yielding an superior speed-up efficiency. Estimates using today's workstations and local area network technology suggest efficient speed-up to several hundred computers, leading to effective scale-up for clustering hundreds of gigabytes of data. Implementation of parallel clustering has been done in a parallel programming language, ZPL. Experimental results show above 90% utilization.

Keywords: Parallel Algorithms, Data Mining, Data Clustering, K-Means, K-Harmonic Means, Expectation-Maximization, Speed-up, Scale-up, Sufficient Statistics.

1. Introduction

Clustering is one of the principle workhorse techniques in the field of data mining. 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. Example applications of clustering include document categorization, scientific data analysis, and customer/market segmentation.

Many algorithms for clustering data have been developed in recent decades, however, they all face a major challenge in scaling up to very large database sizes, an accelerating trend brought on by advances in computer technology, the Internet, and electronic commerce. Clustering algorithms with quadratic (or higher order) computational complexity, such as agglomerative algorithms, scale poorly. 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 very large and ever increasing data sets. In this paper, we develop a class of iterative parallel parameter estimation algorithms—covering K-Means, K-Harmonic Means, and EM algorithms—that are both efficient and accurate.

There have been several recent publications in scaling up K-Means and EM by approximation. For example, in BIRCH [19] and Microsoft-TR by Bradley et. al. [4], a single scan of the data and subsequent aggregation of each local cluster into Sufficient Statistics (SS) 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 result is in general. As 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 [10]. The Kantabutra and Couch algorithm requires re-broadcasting the data set to all computers each iteration, which leads to heavy communication 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. Finally, the number of slave computing units in their algorithm is limited to the number of clusters to be found.

In this paper, we produce a parallel K-Means algorithm that limits inter-processor communication to SS only, reducing the network bottleneck. The dataset is partitioned across the memory of the processors and does not need to be communicated between iterations. The number of computing units is not limited in any way by the number of clusters sought. The method applies not only to K-Means, but also to other iterative clustering methods, such as K-Harmonic Means and EM.

There is no approximation introduced by the method; the results are exactly as if the original algorithm were run on a single computer. Further, it is also complementary to aggregation techniques (as in BIRCH), and by combining the two approaches in a hybrid, even larger data sets can be handled or better accuracy can be achieved with less aggregation.

There has also been work done parallel clustering using special hardware [1] [2]. The parallel algorithms we present require no special hardware or special networking, even though special networking may help further scale up the number of computers that can be deployed with high utilization. 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 utilized when they would otherwise be idle.

An example application of this idea is in on-line commercial product recommendations where clustering is involved in calculating the recommendations (see collaborative filtering and recommender systems [14][15]). Large numbers of PCs that are used during business hours for processing orders can be used at night for updating the clustering of customers and products based on the daily revised sales information (customer buying patterns).

The parallel algorithm presented in this paper is not limited to data clustering algorithms. We use the class of center-based clustering algorithms, which includes K-Means [11], K-Harmonic Means [18] and EM [5][13], to illustrate the parallel algorithm for iterative parameter estimations. By finding $\boldsymbol{K}$ centers (local high densities of data), $\boldsymbol{M} = \{m_i \mid i=1,\ldots,K\}$, the data set, $\boldsymbol{S} = \{x_i \mid i=1,\ldots,N\}$, can be partitioned either into discrete clusters using the Voronoi partition (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).

The problem is formulated as an optimization of a performance function, *Perf(S, M)*, depending on both the data items and the center locations. A popular performance function for measuring the goodness of a clustering is the sum of the mean-square error (MSE) of each data point to its center. The popular K-Means algorithm attempts

to find a local optimum for this performance function. The K-Harmonic Means (KHM) algorithm optimizes the harmonic average of these distances. KHM is very insensitive to the initialization of the centers, a major problem for K-Means. The Expectation-Maximization (EM) algorithm, in addition to the centers, optimizes a covariance matrix and a set of mixing probabilities.

The next section presents an abstraction of a class of center-based algorithms and the following section presents the parallel algorithm. Section 4 applies this to three examples: K-Means, K-Harmonic Means and EM. Section 5 gives an analysis of the utilization of the computing units, and we conclude in section 6.

2. A Class of Center-Based Algorithms

Let R^{dim} be the Euclidean space of dimension dim; $S \subseteq R^{dim}$ be a finite subset of data of size $N = |S|$; and $M = \{m_k \mid k=1,\ldots,K\}$, the set of parameters to be optimized. (The parameter set M consists of K centroids for K-Means, K centers for K-Harmonic Means, and K centers with co-variance matrices and mixing probabilities for EM.) We write the performance function and the parameter optimization step for this class of algorithms in terms of SS. The performance function is decomposed as follows:

Performance Function: $$F(S,M) = f_0(\sum_{x\in S} f_1(x,M), \sum_{x\in S} f_2(x,M), \ldots\ldots, \sum_{x\in S} f_R(x,M)). \quad (1)$$

What is essential here is that f_0 depends only on the SS, represented by the sums, whereas the remaining f_i functions can be computed independently for each data point. The detailed form of f_i, $i=1,\ldots,R$, depend on the particular performance function considered. It will become clear when examples of K-Means, K-Harmonic Means and EM are given in later sections.

We write the center-based algorithm, which minimizes the value of the performance function over M, as an iterative algorithm in the form of Q SS ($I()$ stands for the iterative algorithm, and $\sum_{x\in S} g_j$, $j=1,\ldots,Q$, stands for SS.):

$$M^{(u+1)} = I(\sum_{x\in S} g_1(x,M^{(u)}), \sum_{x\in S} g_2(x,M^{(u)}), \ldots\ldots, \sum_{x\in S} g_Q(x,M^{(u)})). \quad (2)$$

$M^{(u)}$ is the parameter vector after the uth iteration. We are only interested in algorithms that converge: $M^{(u)} \rightarrow M$. The values of the parameters for the 0th iteration, $M^{(0)}$, are by initialization. One method often used is to randomly initialize the parameters (centers, covariance matrices and/or mixing probabilities). There are many different ways of initializing the parameters for particular types of center-based algorithms in the literature [3]. The computation carried out here will be identical to the traditional, sequential equivalent. The set of quantities

$$Suff = \{\sum_{x\in S} f_r(x,M) \mid r = 1,\ldots\ldots,R\} \bigcup \{\sum_{x\in S} g_q(x,M) \mid q = 1,\ldots\ldots,Q\}. \quad (3)$$

is called the global SS of the problem (1)+(2). As long as these quantities are available, the performance function and the new parameter values can be calculated and the algorithm can be carried out to the next iteration. We will show in Section 4 that K-Means, K-Harmonic Means and Expectation-Maximization clustering algorithms all belong to this class defined in (1)+(2).

3. Parallelism of Center-Based Algorithms

This decomposition of center-based algorithms (and many other iterative parameter estimation algorithms) leads to a natural parallel structure with minimal need for

communication. Let L be the number of computing units (with a CPU and local memory – PCs, workstations or multi-processor computers, shared memory is not required). To utilize all L units for the calculation of (1)+(2), the data set is partitioned into L subsets, $S = D_1 \cup D_2 \cup ..... \cup D_L$, and the l^{th} subset, D_l, resides on the l^{th} unit. It is important not to confuse this partition with the clustering:

a) This partition is arbitrary and has nothing to do with the clustering in the data.

b) This partition is static. Data points in D_l, after being loaded into the memory of the l^{th} computing unit, need not be moved from one computer to another. (Except for the purpose of load balancing among units, whose only effect is on the execution time of the processors and does not affect the algorithm. See Section 5.)

We need not assume homogeneous processing units. The sizes of the partitions, $|D_i|$, 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 on each iteration, improving the utilization of all the units. A scaled-down test could be carried out on each computing unit in advance to measure the actual speed (do not include the time of loading data because it is only loaded once at the beginning) or a load balancing over all units could be done at the end of first iteration.

The calculation is carried out on all L units in parallel. Each subset, D_l, contributes to the refinement of the parameters in M in exactly the same way as the algorithm would have been run on a single computer. Each unit independently computes its partial sum of the SS over its data partition. The SS of the l^{th} partition are

$$Suff_l = \{\sum_{x \in D_l} f_r(x, M) \mid r = 1,....., R\} \bigcup \{\sum_{x \in D_l} g_q(x, M) \mid q = 1,....., Q\} \cdot \qquad (4)$$

One of the computing units is chosen to be the Integrator. It is responsible for sums up the SS from all partitions (4), to get the global SS (3); calculates the new parameter values, M, from the global SS; evaluates the performance function on the new parameter values, (2); checks the stopping conditions; and informs all units to stop or sends the new parameters to all computing units to start the next iteration. The duties of the Integrator may be assigned as a part time job to one of the regular units. There may also be more than one computer used as Integrators, possibly organized in a hierarchy if the degree of parallelism is sufficiently high. Special networking support is also an option. If broadcast is supported efficiently, it may be effective to have every node be an Integrator, eliminating one direction of communication. Studies of this sort can be found in the parallel computing literature [6][9].

The Parallel Clustering Algorithm:

Step 0: Initialization: Partition the data set and load the l^{th} partition to the memory of the l^{th} computing unit. Use any preferred algorithm to initialize the parameters, $\{m_k\}$, on the Integrator.

Step 1: Broadcast the integrated parameter values to all computing units.

Step 2: Compute at each unit independently the SS of the local data based on (4).

Step 3: Send SS from all units to the Integrator

Step 4: Sum up the SS from each unit to get the global SS, calculate the new parameter values based on the global SS, and evaluate the performance function. If the Stopping condition[1] is not met, goto Step 1 for the next iteration, else inform all computing units to stop.

[1] Typically test for sufficient convergence or the number of iterations.

4. Examples

This section demonstrates the decomposition for three examples: K-Means, K-Harmonic Means, and EM.

4.1 K-Means Clustering Algorithm

K-Means is one of the most popular clustering algorithms ([11][17] and other references therein). The algorithm partitions the data set into K clusters, $S = (S_1,\ldots,S_K)$, by putting each data point into the cluster represented by the center nearest to the data point. K-Means algorithm finds a local optimal set of centers that minimizes the total within cluster variance, which is K-Means performance function:

$$Perf_{KM}(X,M) = \sum_{k=1}^{K}\sum_{x\in S_l} \| x - m_k \|^2, \tag{5}$$

where the k^{th} center, m_k, is the centroid of the k^{th} partition. The double summation in (5) can instead be expressed as a single summation over all data points, adding only the distance to the nearest center expressed by the *MIN* function below:

$$Perf_{KM}(X,M) = \sum_{i=1}^{N} MIN\{\| x_i - m_l \|^2 \mid l = 1,\ldots,K\}. \tag{6}$$

The K-Means algorithm starts with an initial set of centers and then iterates through the following steps: For each data item, find the closest m_k and assign the data item to the k^{th} cluster. The current m_k's are not updated until the next phase (Step 2). A proof of optimality can be found in [8].

1. Recalculate all the centers. The k^{th} center becomes the centroid of the k^{th} cluster. A proof can be found in [8] that this phase gives the optimal center locations for the given partition of data.
2. Iterate through 1 & 2 until the clusters no longer change significantly.

After each phase, the performance value never increases and the algorithm converges to a local optimum. More precisely, the algorithm will reach a stable partition in a finite number of steps for finite datasets. The cost per iteration is $O(K \cdot dim \cdot N)$.

The functions for calculating both global and local SS for K-Means are the 0^{th}, 1^{st} and 2^{nd} moments of the data on the unit belonging to each cluster as shown in (7). Both the K-Means performance function and the new center locations can be calculated from these three moments.

$$\begin{cases} g_1(x,M) = f_1(x,M) = (\delta_1(x), \delta_2(x), \ldots\ldots, \delta_K(x)), \\ g_2(x,M) = f_2(x,M) = f_1(x,M)\cdot x, \\ g_3(x,M) = f_3(x,M) = f_1(x,M)\cdot x^2. \end{cases} \tag{7}$$

$\delta_k(x) = 1$ if x is closest to m_k, otherwise $\delta_k(x) = 0$ (resolve ties arbitrarily). The summation of these functions over a data set (see (3) and (4)) residing on the l^{th} unit gives the count, $n_{k,l}$, first moment, $\Sigma_{k,l}$, and the second moment, $s_{k,l}$, of the clusters (this is called the CF vector in the BIRCH paper [19]). The vector $\{n_{k,l}, \Sigma_{k,l}, s_{k,l}, \mid k=1,\ldots\ldots,K\}$, has dimensionality $2\cdot K + K\cdot dim$, which is the size of the SS that have to be communicated between the Integrator and each computing unit.

The set of SS presented here is more than sufficient for the simple version of K-Means algorithm. The aggregated quantity, $\Sigma_k\, s_{k,l}$, could be sent instead of the individual $s_{k,l}$. But there are other variations of K-Means performance functions that require individual $s_{k,l}$, for evaluating the performance functions. Besides, the quantities that dominate the communication cost are $\Sigma_{k,l}$.

The l^{th} computing unit collects the SS, $\{ n_{k,l}, \Sigma_{k,l}, s_{k,l}, \mid k=1,\ldots\ldots,K \}$, on the data in its own memory, and then sends them to the Integrator. The Integrator simply adds up the SS from each unit to get the global SS,

$$n_k = \sum_{l=1}^{L} n_{k,l},\ \Sigma_k = \sum_{l=1}^{L} \Sigma_{k,l},\ s_k = \sum_{l=1}^{L} s_{k,l}\ .$$

The leading cost of integration is $O(K \cdot dim \cdot L)$, where L is the number of computing units. The new location of the k^{th} center is given by $m_k = \Sigma_k / n_k$ from the global SS (this is the $I()$ function in (2)), which is the only information all the computing units need to start the next iteration. The performance function is calculated by (proof by direct verification), $Perf_{KM} = \sum_{l=1}^{L} s_k$.

The parallel version of the K-Means algorithm gives exactly the same result as the original centralized K-Means because both the parallel version and the sequential version are based on the same global SS except on how the global SS are collected.

4.2 K-Harmonic Means Clustering Algorithm

K-Harmonic Means is a clustering algorithm designed by the author [18]. A major strength is its insensitivity to the initialization of the centers (*cf.* K-Means, where the dependence on the initialization has been the major problem and many authors have tried to address the problem by finding good initializations).

The iteration step of the K-Harmonic Means algorithm adjusts the new center locations to be a weighted average of all x, where the weights are given by

$$1 \Big/ d_{x,k}{}^{s} \Big(\sum_{x \in S} \frac{1}{d_{x,l}{}^{2}} \Big)^{2} . \tag{8}$$

(K-Means is similar, except its weights are the nearest-center membership functions, making its centers centroids of the cluster.) Overall then, the recursion equation is given by

$$m_k = \sum_{x \in S} \frac{1}{d_{x,k}{}^{s} \Big(\sum_{l=1}^{K} \frac{1}{d_{x,l}{}^{2}} \Big)^{2}} \quad x \Big/ \sum_{x \in S} \frac{1}{d_{x,k}{}^{s} \Big(\sum_{l=1}^{K} \frac{1}{d_{x,l}{}^{2}} \Big)^{2}} . \tag{9}$$

where $d_{x,k} = ||x - m_k||$ and s is a constant ≤ 4. The decomposed functions for calculating SS (see (3) and (4)) are then

$$\begin{cases} g_1(x,M) = 1 \Big/ \sum_{k=1}^{K} \frac{1}{d_{x,k}^{2}} \\ g_2(x,M) = g_1^{2}(x,M) \cdot \left(\frac{1}{d_{x,1}^{s}}, \frac{1}{d_{x,2}^{s}}, \ldots\ldots, \frac{1}{d_{x,K}^{s}} \right) . \\ g_3(x,M) = g_1^{2}(x,M) \cdot \left(\frac{1}{d_{x,1}^{s}}, \frac{1}{d_{x,2}^{s}}, \ldots\ldots, \frac{1}{d_{x,K}^{s}} \right) x \end{cases} \tag{10}$$

Each computing unit collects the SS,

$$Suff_l = \{ \sum_{x \in D_l} g_2(x,M), \sum_{x \in D_l} g_3(x,M) \} \tag{11}$$

on the data in its own memory, and then sends it to the Integrator. The size of the SS vector is $K + K \cdot dim$ (g_3 is a matrix). The Integrator simply adds up the SS from each unit to get the global SS. The new centers are given by the *component-wise* quotient:

$$(m_1, m_2, \ldots\ldots, m_K) = \sum_{x \in S} g_3(x,M) \Big/ \sum_{x \in S} g_2(x,M) = \sum_{l=1}^{L} \sum_{x \in D_l} g_3(x,M) \Big/ \sum_{l=1}^{L} \sum_{x \in D_l} g_2(x,M) , \tag{12}$$

which is the only information the units need to start the next iteration. This calculation costs $O(K \cdot dim \cdot L)$. The updated global centers are sent to each unit for the next iteration. If broadcasting is an option, this is the total cost in time. If the Integrator finds the centers stop moving significantly, the clustering is considered to have converged to an optimum and the units are stopped.

4.3 Expectation-Maximization (EM) Clustering Algorithm

We limit ourselves to the *EM* algorithm with linear mixing of K bell-shape (Gaussian) functions. Unlike K-Means and K-Harmonic Means in which only the centers are to be estimated, the EM algorithm estimates the centers, the co-variance matrices, Σ_k, and the mixing probabilities, $p(m_k)$. The performance function of the EM algorithm is

$$Perf_{EM}(X, M, \Sigma, p) = -\log\left\{\prod_{x \in S}\left[\sum_{k=1}^{K} p_k \cdot \frac{1}{\sqrt{(2\pi)^D \det(\Sigma_k)}} \cdot EXP(-(x-m_k)\Sigma_k^{-1}(x-m_k)^T)\right]\right\}. \quad (13)$$

where the vector $p = (p_1, p_2, \ldots\ldots, p_K)$ is the mixing probability. *EM* algorithm is a recursive algorithm with the following two steps:

E-Step: Estimating "the percentage of x belonging to the k^{th} cluster",

$$p(m_k \mid x) = p(x \mid m_k) \cdot p(m_k) \Big/ \sum_{x \in S} p(x \mid m_k) \cdot p(m_k), \quad (14)$$

where $p(x|m)$ is the prior probability with Gaussian distribution, and $p(m_k)$ is the mixing probability.

$$p(x \mid m_k) = \frac{1}{\sqrt{(2\pi)^D \det(\Sigma_k)}} \cdot EXP(-(x-m_k)\Sigma_k^{-1}(x-m_k)^T) \quad (15)$$

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

$$m_k = \frac{\sum_{x \in S} p(m_k \mid x) \cdot x}{\sum_{x \in S} p(m_k \mid x)}, \quad \Sigma_k = \frac{\sum_{x \in S} p(m_k \mid x) \cdot (x-m_k)^T (x-m_k)}{\sum_{x \in S} p(m_k \mid x)}, \quad p(m_k) = \frac{1}{|S|}\sum_{x \in S} p(m_k \mid x). \quad (16\text{-}18)$$

For details see [5][13]. The functions for calculating the SS are:

$$f_1(x, M, \Sigma, p) = -\log\left[\sum_{l=1}^{K} p(x \mid m_k) p(m_k)\right]$$

$$g_1(x, M, \Sigma, p) = (p(m_1 \mid x), p(m_2 \mid x), \ldots\ldots, p(m_K \mid x))$$

$$g_2(x, M, \Sigma, p) = (p(m_1 \mid x)x, p(m_2 \mid x)x, \ldots\ldots, p(m_K \mid x)x)$$

$$g_3(x, M, \Sigma, p) = (p(m_1 \mid x)x^T x, p(m_2 \mid x)x^T x, \ldots\ldots, p(m_K \mid x)x^T x)$$

The vector length (in number of scalars) of the SS is $1+K+K \cdot dim + K \cdot dim^2$. The global SS is also the sum of the SS from all units. The performance function value is given by the first global sufficient statistic. The global centers are from the component-wise "ratio" of the third and the second global SS (see (16)), the co-variance matrices from (17) and the mixing probability from (18). All these quantities, $\{m_k, \Sigma_k, p(m_k) \mid k = 1, \ldots\ldots, K\}$, have to be propagated to all the units at the beginning of each iteration. The vector length is $K+K \cdot dim + K \cdot dim^2$.

5. Time/Space Complexities

The storage required at each node is $O(dim \cdot (N/L + K))$—the data set S is partitioned across the processors and the list of centers M is replicated at each processor. In

contrast, the algorithm in [10] partitions the centers across the processors. Utilization is determined by the percentage of time each unit works on its own data to adjust the centers and collect SS, vs. the time waiting between sending out the local SS to the Integrator and receiving the new global parameters (see Fig. 1).

When the data size on each unit is far greater than the number of computing units ($N>>L$, which is true in general), the amount of work each unit has to do, $O(K \cdot dim \cdot N/L)$, is far greater than the amount of work the Integrator has to do, $O(K \cdot dim \cdot L)$. Therefore, the integration time is marginal compared to the time for collecting SS. The main source of waiting comes from network communication of the SS and global parameters. The ratio between the speed of network communication and the speed of the computing units is an important factor for determining the utilization (see (19) later). Since the size of the SS is very small compared with the original data size on each unit (see the following table), we will give an example to show that the speed ratio between the network transmission and CPU processing of current technology is sufficient to support high utilization. If the network is slower (relative to the unit's CPU speed), the number of units, L, has to be smaller to maintain high utilization. The faster the network, the higher degree of parallelism can be achieved for a given utilization target. Fig. 2 shows the breakdown of four periods in each iteration. Since the parameters sent from the Integrator to all units are the same, broadcasting, if supported by the network, may shorten the waiting time of the units. Let t_d be the unit of time to process one datum, t_n be the unit of time to transmit one datum, and t_c the communication latency for sending an empty message. The utilization U of the (slave) units can be estimated by (using K-Means as an example),

$$Utilization = \frac{C \cdot K \cdot \dim \cdot (N/L) \cdot t_d}{C \cdot K \cdot \dim \cdot (N/L) \cdot t_d + C' \cdot K \cdot \dim \cdot L \cdot t_d + L \cdot (C'' \cdot K \cdot \dim \cdot t_n + t_c)} \qquad (19)$$

$$\approx \frac{1}{1 + (C'/C) \cdot (L^2/N) + (C''/C) \cdot (L^2/N) \cdot (t_n/t_d)}.$$

where C, C', and C'' are constants. The communication latency is shared by all $O(K \cdot dim)$ data items that are transmitted together. When $K \cdot dim$ is so large that the latency t_c becomes negligible compared with the transmission time $C'' \cdot K \cdot \dim \cdot t_n$, the term $t_c / K \cdot \dim$ on the second line of (19) may be ignored. Both the computational cost at each unit and the communication costs grow linearly in *dim* and *K*. Therefore, the ratio of the two costs, which is close to the utilization of the units if the starting cost of communication is ignored, does not depend on *dim* and *K*.

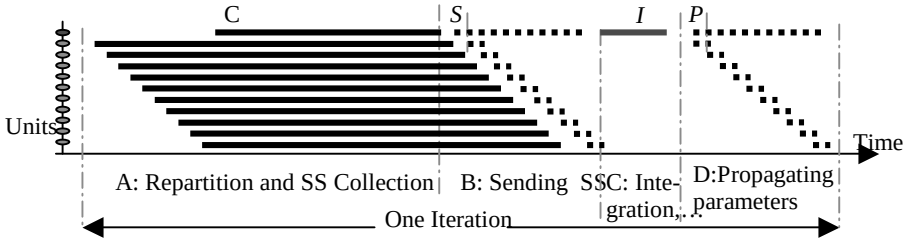

Fig 1. The timeline of all the units in one iteration. Network has no broadcasting feature. The top unit is the Integrator given a lighter load of data. The slope in period B and D are not same in general. Communication time from the Integrator to the units is drawn as sequential supposing of no support for broadcasting. Period A and Period D overlap.

The length of each period is determined by the following factors:

Table 1. The Analysis of Computational Costs (*N*>>*L*).

	The size of SS from each unit (in # floating points).	The size of data on each unit.	The cost of collecting the SS on each unit/iteration.	The cost of integration per iteration.
KM	2·K+K·dim	N/L	O(K·dim·N/L)	O(K·dim·L)
KHM	K+K·dim	N/L	O(K·dim·N/L)	O(K·dim·L)
EM	1+K+K·dim+K·dim2	N/L	O(K·dim2 ·N/L)	O(K·dim2 ·L)

6. Experimental Results – Speedup Curves

The parallel clustering algorithms presented in this paper have been implemented in the parallel programming language ZPL [16]. The plots in Fig. 2 show the speed-ups as the number of processors change from 1 to 8. Four different data sizes are used, *N = 50k, 100k, 500k,* and *10M.* At *N=50,000* points, it takes only 0.24 seconds per iteration to cluster on 8 processors. The dotted line represents 90% speed-up efficiency. The data used in all experiments are randomly generated with uniform distribution, which could be any other distribution and has no effect on speed-up for the clustering algorithms. The number of clusters *K=100*. The dimensionality *dim=2*. We have shown analytically in Section 5 that the speed-ups are very insensitive to *K*, and *dim*. The measurements are taken on 8 identical HP-UX 9000/735's each with 208 MB memory sharing a normal 10Mbps Ethernet. The HP 9000/735 has roughly the same floating point SPECmarks as the 233 MHz PentiumMMX, for comparison.

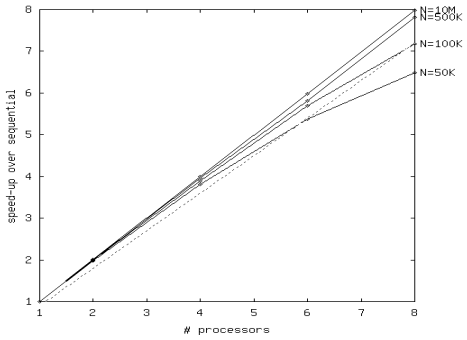

Fig 2. The speedup curves under various *N*, and *L*.

7. Dynamic Load Balancing

Data items can be moved from one computing unit to another during the period (B+C+D) (see Fig. 1) without affecting the correctness of the algorithm. Moving data around during this period to increase the utilization of the units in the future is called dynamic load balancing. Dynamic load balancing corrects the initial setup of the load, which may not give the best utilization of the computing units, especially if the data set consists of sparse vectors with variable numbers of non-zero elements in them. For dense vectors, the prediction of calculation time based on the number of data vectors is straightforward; a linear scale-up based on the test result from a small number of vectors will work well. But for sparse vectors, the amount of data to be loaded on to each machine becomes more difficult to estimate, due to the variation of

the actual vector lengths even though the cost of partitioning and collecting SS on each subset of data is still deterministic. Only the first or maybe a second re-balancing would be needed. The necessity of a load re-balancing can be determined based on the utilization of the units calculated from the previous iteration.

8. Conclusion

We restructured the mathematics of a class of clustering algorithms to reveal a straightforward parallel implementation based on communicating only a small amount of data—SS—yielding highly efficient speed-up and scale-up for very large data sets. The ideas presented in this paper apply to iterative parameter estimation algorithms where the size of the SS is small relative to the data size. By the time this paper is accepted, we have conducted many experiments. One of them used 128 loosely connected processors. The experimental results are published in KDD 2000 workshop on distributed and parallel computing in Boston [7].

References

[1] Apostolakis, J., Coddington, P., and Marinari, E. ‘‘New SIMD algorithms for cluster labeling on parallel computers,” *Intl J. of Modern Physics C*, 4(4):749-763, Aug. 1993.

[2] Baillie, C.F.,Coddington, P.D.‘‘Cluster identification algorithms for spin models,” *Concurrency:Practice and Experience*,3(2):129-144, April 1991. Caltech Report C3P-855.

[3] Bradley, P., Fayyad, U. M., and Reina, C.A., “Scaling EM Clustering to Large Databases,” MS Technical Report, 1998.

[4] Bradley, P., Fayyad, U. M., C.A., “Refining Initial Points for KM Clustering”, MS Technical Report MSR-TR-98-36, May 1998.

[5] Dempster, A. P., Laird, N.M., and Rubin, D.B., “Miximum Likelyhood from Incomplete Data via the EM Algorithm”, J. of the Royal Stat. Soc., Series B, 39(1):1-38, 1977.

[6] T. J. Fountain , “Parallel Computing : Principles and Practice,” Cambridge Univ Pr 1994.

[7] Forman, G., Zhang, B.,“Linear speedup for a parallel non-approximate recasting of center-based clustering algorithms, including K-Means, K-Harmonic Means, and EM”, To appear in DPKD 2000, Boston.

[8] Gersho & Gray, “Vector Quantization and Signal Compression”, KAP, 1992

[9] Vipin Kumar, “Introduction to Parallel Computing,” Addison-Wesley, 1994.

[10] Sanpawat Kantabutra and Alva L. Couch, “Parallel K-means Clustering Algorithm on NOWs,” NECTEC Technical journal (www.nectec.or.th)

[11] MacQueen, J., “Some methods for classification and analysis of multivariate observations,” Pp. 281-297, Proceedings of the 5th Berkeley symposium on mathematical statistics and probability, Vol. 1. University of California Press, Berkeley. xvii + 666 p, 1967.

[12] McKenzie, P. and Alder, M., “Initializing the EM Algorithm for Use in Gaussian Mixture Modeling”, The Univ. of W. Australia, Center for Information Processing Systems, Manuscript.

[13] McLachlan, G., Krishnan, T., “The EM algorithm and extensions”,John Wiley &Sons.

[14] A Commercial Recommender System http://www.netperceptions.com

[15] P. Resnick and H. Varian, “Recommender Systems,” in CACM March 1997.

[16] Snyder, L., “A Programmer's Guide to ZPL (Scientific and Engineering Computation Series)”, MIT Press; ISBN: 0262692171, 1999.

[17] Selim, S.Z. and Ismail, M.A., “*K*-Means-Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality”, IEEE Trans. On PAMI-6, #1, 1984.

[18] Zhang, B., Hsu, M., Dayal, U., “K-Harmonic Means – A Data Clustering Algorithm”, Hewllet-Packard Research Laboratory Technical Report HPL-1999-124.

[19] 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.