Scale Up Center-Based Data Clustering Algorithms by Parallelism

Bin Zhang, Meichun Hsu
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2000-6
January, 2000

E-mail:bzhang@hpl.hp.com

parallel algorithms, data mining, data clustering, K-Means, K-Harmonic Means, Expectation Maximization, parallel, E.M

As data collection increases at an accelerating rate with the advances of computers and networking technology, analyzing the data (data mining) becomes very important. Data clustering is one of the basic tools widely used as a component in many data mining solutions. Even though many data clustering algorithms have been developed in the last few decades, they face new challenges in front of huge data sets. Algorithms with quadratic (or higher order) computational complexity, like agglomerative algorithms, drop out very quickly. More efficient algorithms like K-Means and EM, which have linear cost per iteration, also need scale-up before they can be applied to very large data sets. This paper shows that many parameter estimation algorithms, including the clustering algorithms like K-Means, K-Harmonic Means and EM, have intrinsic parallel structure in them. Many workstations over a LAN or a multiple-processor computer can be efficiently used to run this class of algorithms in parallel. With 60 workstations running in parallel (on a fast LAN), clustering 28.8 GBytes of 40 dimensional data into 100 clusters, the utilization of the computing units is above 80%.

Internal Accession Date Only

1. Introduction

Data clustering is one of the common techniques used in data mining. An ideal case is to group related data (measured by a distance function) into the same cluster and unrelated data into different clusters. Examples of applications of clustering include customer segmentation, document categorization, and scientific data analysis.

Even though many data clustering algorithms have been developed in the last few decades, they face new challenges in front of huge data sets. Algorithms with quadratic (or higher order) computational complexity, like agglomerative algorithms, drop out very quickly. More efficient algorithms like K-Means and EM, which have linear cost per iteration, also need scale-up before they can be applied to very large data sets.

There have been several recent publications to approximately scale-up *K*-Means and/or EM data clustering algorithms to very large data sets. In BIRCH [ZLR96] and the MS Technical Report [BFR98] [BFR98a], a single scan of the data and subsequent aggregation of local clusters into a single "point" by just maintaining the sufficient statistics so that the aggregated data fits in the memory available. These algorithms provide an approximation to the original algorithm and have been successfully applied to very large datasets. However, in general, the higher ratio of the aggregation the less accurate the results. It is also reported in the BIRCH paper that the quality of clustering depends on the scanning order.

There is also a recent work on parallel K-Means by Kantabutra and Couch [KC99]. Their algorithm requires broadcasting the data set over the Internet in each iteration, which causes a lot of network traffic and overhead. Reported utilization of the computers by their algorithm is 50%. The number of slave computing units in their algorithm is the same as the number of clusters to be found. The parallel *K*-Means developed in this paper sends only the sufficient statistics, which

is significantly smaller than the data itself, between the master and the slave computing units. The data loaded to each computing unit stays there in all iterations. The number of computing units is not related to the number of clusters. Our parallel algorithm not only apply to K-Means, but also to many iterative parameter estimation problems including *K*-Harmonic Means and EM.

We use the intrinsic parallelism in the center-based clustering algorithms to run them in parallel on multiple computers. These is no approximation introduced. The results are exactly the same as the algorithms would have been run on a single computer. The parallel algorithm we present is also complementary with the aggregation-based scale-up, by combining the two, even larger data sets can be handled or better accuracy can be achieved with less amount of aggregation.

Parallel algorithm has been a well-known topic and its benefits well justified. We try to be brief on this. The total amount of computing resources in "small" computers, like "PCs" or desktop workstations, is far greater than the total amount of computing resources available in super computers. Small computers or multi-processors computers (without sharing memory) also offer a lower computing cost than super computers or multi-processors with shared memory. Small computers, which is already everywhere, are much more accessible and can even be considered as a free resource if they can be utilized during non-business hours.

The parallel data clustering algorithm we present in this paper requires 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 economical considerations.

We see potential applications of this idea in on-line commercial product recommendations where clustering is one of the techniques for calculating recommendations (see collaborative filtering and Recommender Systems [RV97][NetPerception]). Large number of PCs that are used during business hours for processing orders can be used during the night for updating the clustering of customers and products based on the updated sales information (customer buying patterns).

We use the class of center-based clustering algorithms, which includes *K*-Means [M67] [GG92]., *K*-Harmonic Means [ZH99] and EM [DLR77] [MK97] to illustrate the parallel algorithm for iterative parameter estimations. By finding *K* centers (local high densities of data), $M=\{m_i | i=1,\ldots,K\}$, the data set, $\boldsymbol{S}=\{x_i | i=1,\ldots,N\}$, can either be partitioned into clusters using the voronoi partition (*K*-Means -- every data item goes with the center that it is closest to) or fuzzy clusters given by the local density functions (*K*-Harmonic Means or EM). To find *K* centers, the problem is formulated as an optimization (minimization) of a performance function, *Perf(X, M),* defined on both the data items and the center locations.

A popular performance function for measuring the goodness of the clustering is the total within-cluster variance, or the total mean-square error (MSE). A popular algorithm for finding a local optimal of this function is called *K*-Means. Another performance/algorithm of center-based clustering is *K*-Harmonic Means. The harmonic average of distances is used in the performance function and the algorithm for optimizing the performance function is very insensitive to the initialization of the centers, which has been a major problem with *K*-Means. Expectation-Maximization (EM) is yet another "center-based" algorithm for clustering. In addition to the centers, a covariance matrix and a set of mixing probabilities are also to be optimized (estimated). We use all three of them as examples to show how our parallel parameter estimation algorithm works.

The rest of the paper is organized as follows: the abstraction of a class of center-based algorithms and its parallel version; examples – *K*-Means, *K*-Harmonic Means and EM; analysis of the utilization of the computing units; conclusion.

2. A Class of Center-based Algorithms

Let R^{dim} be the Euclidean space of dimension *dim;*

$\boldsymbol{S} \subseteq R^{dim}$ be a finite subset of data and $|S| = N$; and

$M = \{m_k \mid k=1,\ldots,K\}$, the set of parameters to be optimized

(*K* centroids for *K*-Means, *K*-centers for *K*-Harmonic Means, and

K-<centers, co-variance matrices and mixing probabilities> for EM).

We write the performance and the algorithm that finds an optimal of the function in terms of the sufficient statistics. A "center-based" performance function is a function of the data set and the set of parameters:

Performance Function:

$$F(X,M) = f_0(\sum_{x\in S} f_1(x,M), \sum_{x\in S} f_2(x,M), \ldots\ldots, \sum_{x\in S} f_R(x,M)). \qquad (1)$$

A center-based algorithm, that minimize the value of the performance function over *M*, is an iterative algorithm in the following form:

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

$M^{(u)}$ is the parameter vector after the (u)-th iteration. We are only interested in the algorithms that converge: $M^{(u)} \rightarrow M$. The set of quantities

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

is called the global sufficient statistics of the problem (1)+(2) because 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. (Note: f_0 is not included in the sufficient statistics.)

We will show that *K*-Means, *K*-Harmonic Means and Expectation Maximization clustering algorithms all belong to this class defined in (1)+(2).

3. Parallelism of The Center-based Algorithms

The class of center-based algorithms (and many other iterative parameter estimation algorithms) have natural parallel structure in them. Let L be the number of computing units (with a CPU and some local memory -- PCs or 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 structure in data.

b) This partition is static. Data points in D_l, after loaded into the memory of the lth computing unit, will not be moved from one computer to another. (Except for the purpose of load balancing among all units, which has nothing to do with the clustering algorithm. See Section 5.)

The size of the partitions, $|D_i|$, are set to be proportional to the speed of the computing units (we do not assume homogeneous processing units) so that it will take about the same amount of time for each unit to finish its task in each iteration. Doing so improves 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 very beginning) or a load balancing on 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, contribute to the refinement of the parameters in M in exactly the same way as the algorithm would have been run on a single computer. This is done by introducing the sufficient statistics of the l-th partition:

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

The values of the parameters, M, for the 0-th iteration is from initialization. Randomly generating the parameters (centers, covariance matrices and mixing probabilities) is one initialization method often used. There are many different ways of initializing the parameters for particular type of center-based algorithms in the literature [BF98]. They will do exactly the same for the parallel algorithm here as they did before.

One of the computing unit is chosen to be the Integrator (can be done as part time job by one of the units), which

- sums up the sufficient statistics from all partitions (4), to get the global sufficient statistics (3);
- calculates the new parameter values, M, from the global sufficient statistics;
- evaluates the performance function on the new parameter values, (2);
- checks stopping rules; and
- informs all units to stop or sends the new parameter values to all computing units to start the next iteration.

There could be more than one computer used as the Integrators or even organized in a hierarchical fashion if the degree of parallelism is very high and special networking structure is

also an option. This kind of studies can be found in the parallel computing literature [F94][KGGK94].

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.

Using any preferred algorithm to initialize the parameters, $\{m_k\}$, on the Integrator.

Step 1: Sending the same set of parameter values to all computing units.

Step 2: Computation at each computing unit:

Calculating the sufficient statistics of the local data set based on (4).

Step 3: Sending sufficient statistics to the Integrator

Step 4: Summing up the sufficient statistics from each unit to get the global sufficient statistics, calculating new parameter values based on the global sufficient statistics, evaluate the performance function and checking Stopping Rules

If checking stopping rules returns STOP, inform all computing units to stop;

Else goto Step 1 for the next iteration.

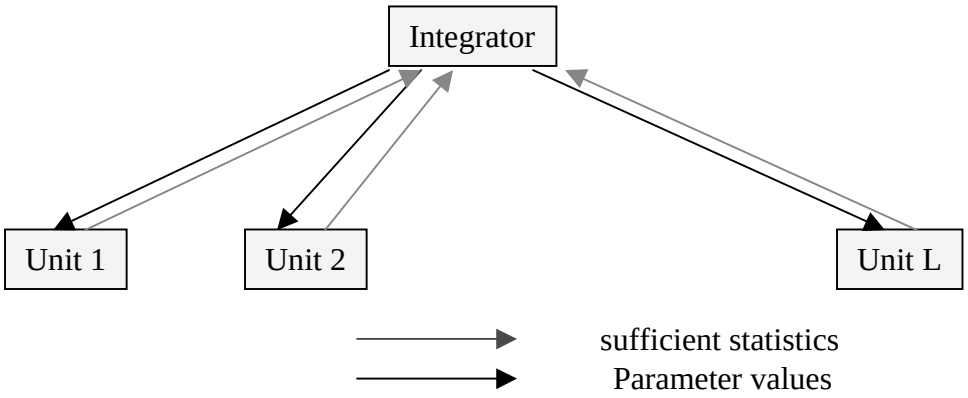

Figure 1. The communications between the Integrator and other units.

Three examples, *K*-Means, *K*-Harmonic Means, and EM, are given in the next section. Utilization of the computing units is analyzed in a later section.

4. Examples

4.1 K-Means Clustering Algorithm

K-Means is one of the most popular clustering algorithms ([GG92][M67][SI84] and many other references therein). *K*-Means algorithm partitions the data set into *K* clusters, $S = (S_1,\ldots\ldots,S_K)$, by putting each data point into the cluster represented by the center the data point is closest to (Voronoi partition). *K*-Means algorithm finds a local optimal set of centers that minimizes the total with-in cluster variance, which is *K*-Means performance function:

$$Perf_{KM}(\{x_i\}_{i=1}^{N},\{m_k\}_{k=1}^{K}) = \sum_{k=1}^{K}\sum_{x\in S_l}\| x - m_k \|^2, \qquad (5)$$

where m_k, the *k*-th center, is the centroid of the *k*-th partition; and S_k is the *k*-th partition. The double summation in (5) can be considered as a single summation over all *x* (data points) and the squared distance under the summations can be expressed by *MIN()*:

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

MacQueen [M67] is considered by many people to be the first one defined K-Means (also see [L84] by Lloyd, which was first published in 1957 as a technical note.)

The *K*-Means algorithm starts with an initial set of centers, $\{ m_k \mid k=1, \ldots,K\}$, and then iterates through the following steps:

1. 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 can be found in [GG92] that this phase gives the optimal partition for the given centers.
2. Recalculate all the centers. The *k*-th center is the centroid of the k^{th} cluster. A proof can be found in [GG92] that this phase gives the optimal center locations for the given partition of data.

3. Loop through 1 & 2 until the clusters no longer change.

After each phase, the total within cluster variance never increases (this is called the monotone convergence property of the algorithm) and the algorithm converges to a local optimum of the performance function. More precisely, the algorithm will reach a stable partition in finite number of steps because there is only a finite number of partitions of a finite data set. The cost per cycle (phase 1+ phase 2) is *O(K*dim*N)*, multi-linear in the number of clusters, the dimensionality of data and the number of data points. More information on *K*-Means can be found in [GG92] or [SI84] and the references there.

The functions for calculating both global and local sufficient statistics for *K*-Means are

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

$\delta_k(x) = 1$ 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 lth unit gives the count, $n_{k,l}$, first moment, $m_{k,l}$, and the 2nd moment, $s_{k,l}$, of the clusters (this is called the CF vector in the BIRCH paper [ZRL96]). The vector $\{n_{k,l}, \Sigma_{k,l}, s_{k,l}, \mid k=1,\ldots\ldots,K\}$, has dimensionality $2*K+K*dim$, where *dim* is the dimensionality of the data (and centers). This is the size of the sufficient statistics that have to be communicated between the Integrator and one of the other computing units.

The set of sufficient statistics 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 dominates the communication cost are $\Sigma_{k,l}$,.

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

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

The leading cost of integration is *O(K*dim*L)*, where *L* is the number of computing units. The new location of the *k*-th center is given by Σ_k/n_k from the global sufficient statistics (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 summing up the 2nd moments in the global sufficient statistics (the function f_0 in (1)).

We want to emphasize that the parallel version of the *K*-Means algorithm gives exactly the same result as the original centralized *K*-Means. It is exactly the same algorithm except that the computation and data are split into *L* parallel tracks.

4.2 K-Harmonic Means Clustering Algorithm

K-Harmonic Means is a data clustering algorithm designed by this author [ZHD99]. *K*-Harmonic Means is very insensitive to the initialization of the centers (comparing with *K*-Means, the dependency of convergence results on the initialization has been the major problem of *K*-Means and many authors have tried to address the problem by finding good initializations).

The performance function of the *K*-Harmonic Means clustering algorithm is

$$Perf_{KHM}(S,M) = \sum_{x \in S} HA\{ \| x - m_l \|^2 \mid l = 1,\ldots,K \} = \sum_{x \in S} \frac{K}{\sum_{l=1}^{K} \frac{1}{\| x - m_l \|^2}}. \qquad (8)$$

where *HA()* is the harmonic average function. Comparing with the *K*-Means performance function the *MIN()* used in *K*-Means is replaced by a softer "MIN()" – the Harmonic Average. The *K*-Harmonic Means algorithm that converges to a local optimal of the performance function is derived by taking all partial derivatives of the performance and set them to zero. "Solving" the set of equations, a recursive formula is derived (see [ZHD99] for details on the algorithm, experimental results and comparisons with *K*-Means and EM). The recursion is

$$m_k = \frac{\sum_{x \in S} \frac{1}{d_{x,k}^{3} (\sum_{l=1}^{K} \frac{1}{d_{x,l}^{2}})^2} x}{\sum_{x \in S} \frac{1}{d_{x,k}^{3} (\sum_{l=1}^{K} \frac{1}{d_{x,l}^{2}})^2}}. \qquad (9)$$

where $d_{x,k} = ||x - m_k||$. This formula shows that the new center locations are a simple weighted average of all the *x*. This is the same as *K*-Means except that the weights in *K*-Means are the membership functions and the weights in *K*-Harmonic Means are

$$\frac{1}{d_{x,k}^{3} (\sum_{x \in S} \frac{1}{d_{x,l}^{2}})^2}.$$

Therefore the centers are not centroids any more.

The functions for calculating sufficient statistics (see (3) and/or (4)) are

$$\begin{cases} f_1(x, M) = \dfrac{1}{\sum_{k=1}^{K} \frac{1}{d_{x,k}^2}} \\ g_1(x, M) = f_1(x, M) * (\frac{1}{d_{x,1}^3}, \frac{1}{d_{x,2}^3}, ..., \frac{1}{d_{x,K}^3}). \\ g_2(x, M) = f_1(x, M) * (\frac{x}{d_{x,1}^3}, \frac{x}{d_{x,2}^3}, ..., \frac{x}{d_{x,K}^3}) \end{cases} \qquad (11)$$

The relation between the global sufficient statistics and the local ones, which the Integrator has to deploy, are simple summations.

Each computing unit collects the sufficient statistics,

$$Suff_l = \{\sum_{x \in D_l} f_1(x, M), \sum_{x \in D_l} g_1(x, M), \sum_{x \in D_l} g_2(x, M)\} \qquad (12)$$

on the data in its own memory, and then send it to the Integrator. The size of sufficient statistics vector is *1+K+K*dim* (g_2 is a matrix). The Integrator simply adds up the sufficient statistics from each unit to get the global sufficient statistics. The new locations of the *k*-th centers are given by the **component-wise** quotient:

$$(m_1, m_2, ..., m_K) = \frac{\sum_{x \in S} g_2(x, M)}{\sum_{x \in S} g_1(x, M)}$$

from the global sufficient statistics (the *I()* function in (2)), which is the only information all the computing units need to start the next iteration. The cost of this calculation is *O(K*dim*L)*.

The first global sufficient statistics, f_1, is already the performance function (see (8) and (11)).

The updated global centers are sent to each unit for the next iteration. The cost per unit is *O(K*dim)*. If broadcasting is an option, this is the total cost in time.

4.3 Expectation-Maximization Clustering Algorithm

We limit ourselves to the *EM* algorithm with linear mixing of *K* bell-shape (Gaussian distribution) functions. Definitions are given in this section but for more details, see [DLR77][MK97].

Unlike *K*-Means and *K*-Harmonic Means in which only the centers are to be estimated, in the EM algorithm, the co-variance matricies, Σ_k, and the mixing probabilities, $p(m_k)$, in addition to the centers, are also to be estimated.

The performance function of the EM algorithm is (the vector $p = (p_1, p_2, \ldots\ldots, p_K)$ is the mixing probability)

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

(Note: -log(Π()) in the formula above is essentially a summation.)

EM algorithm is a recursive algorithm with the following two steps:

E-Step: Estimating "the percentage of x belonging to the k-th cluster" (fuzzy membership function),

$$p(m_k \mid x) = \frac{p(x \mid m_k) * p(m_k)}{\sum_{x \in S} p(x \mid m_k) * p(m_k)}, \quad (14)$$

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

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

and $p(m_k)$ is the mixing probability.

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

$$m_k = \frac{\sum_{x \in S} p(m_k \mid x) * x}{\sum_{x \in S} p(m_k \mid x)}, \qquad (16)$$

$$\Sigma_k = \frac{\sum_{x \in S} p(m_k \mid x) * (x-m_k)^T (x-m_k)}{\sum_{x \in S} p(m_k \mid x)}, \qquad (17)$$

$$p(m_k) = \frac{1}{|S|} \sum_{x \in S} p(m_k \mid x). \qquad (18)$$

For more details, see [MK97] and the references there.

The functions for calculating the sufficient statistics are:

$$f_1(x, M, \Sigma, p) = -\log\left[\sum_{l=1}^{K} p(x \mid m_k) p(m_k)\right]$$
$$g_1(x, M, \Sigma, p) = (p(m_1 \mid x), p(m_2 \mid x), ..., p(m_K \mid x))$$
$$g_2(x, M, \Sigma, p) = (p(m_1 \mid x)x, p(m_2 \mid x)x, ..., p(m_K \mid x)x)$$
$$g_3(x, M, \Sigma, p) = (p(m_1 \mid x)x^T x, p(m_2 \mid x)x^T x, ..., p(m_K \mid x)x^T x)$$

The vector length (in number of scalars) of the sufficient statistics is $1+K+K*dim+K*dim^2$. The global sufficient statistics is also the sum of the sufficient statistics from all units. Performance

function value is given by the first global sufficient statistics. The global centers are from the component-wise "ratio" of the third and the second global sufficient statistics (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 $K+K*dim+K*dim^2$.

4. Time/Space Complexities

Before going into detailed analysis, we explain briefly the support for achieving high utilization. High utilization is determined by the amount of time each unit work on its own data to repartition the data and to collect sufficient statistics, and the time waiting between sending out the local sufficient statistics to the Integrator and receiving the new global parameters (see Figure 2). The leading costs of each period for *K*-Means is shown as follows (in units of time):

On the *l*th unit:

Partitioning the data: $C_1*K*dim*|D_l|$,

Collecting Statistics: $C_2*K*dim*|D_l|$,

Integrator's calculation: $C'*K*dim*L$,

Sending and receiving statistics: $C''*K*dim$.

The unit of time for network transmission is different from the unit of time for the previous three items. *C*'s are constants.

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, $O(K*dim*N/L)$, is far greater than the amount of work the Integrator has to do, $O(K*dim*L)$. Therefore the integration time is marginal comparing with the time for collecting sufficient statistics. The major cause for waiting comes from network communication of the sufficient statistics and global parameters. The ratio between the speed of network communication and the speed of the computing units is a very important

factor for determining the utilization (See (19) later). Since the size of the sufficient statistics is very small comparing with the original data size on each unit (See the next 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 very high utilization. If the network is slower, the number of units, *L*, has to be smaller to avoid low utilization. The faster the network (relative to the unit's CPU speed), the higher degree of parallelism can be achieved under a given utilization.

Figure 2 and Figure 3 show the breakdown of four periods in each iteration. Since the parameters send from the Integrator to all units are the same, broadcasting, if supported by the network, can be used to shorten the waiting time of the units.

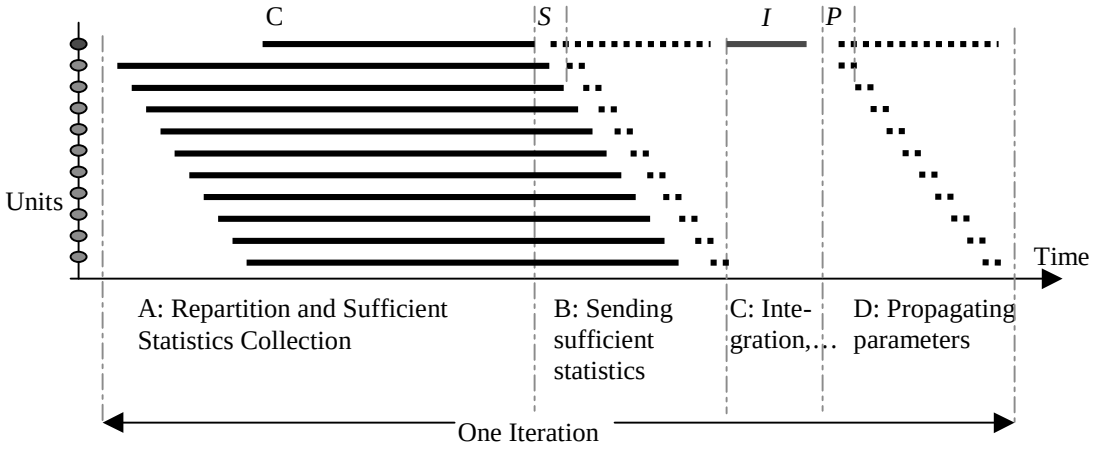

Figure 2. The timeline of all the units in one iteration. Assuming that the network has no broadcasting feature. The unit in red is the Integrator which has a lower load on collecting statistics so that it has more of its time for communication and integration. The slop in period B and period D are not same in general because the amount of data send between the Integrator and other units are not exactly the same. Communication time from the Integrator to the units are drawn as sequential because of no broadcasting. Period A and Period D overlap.

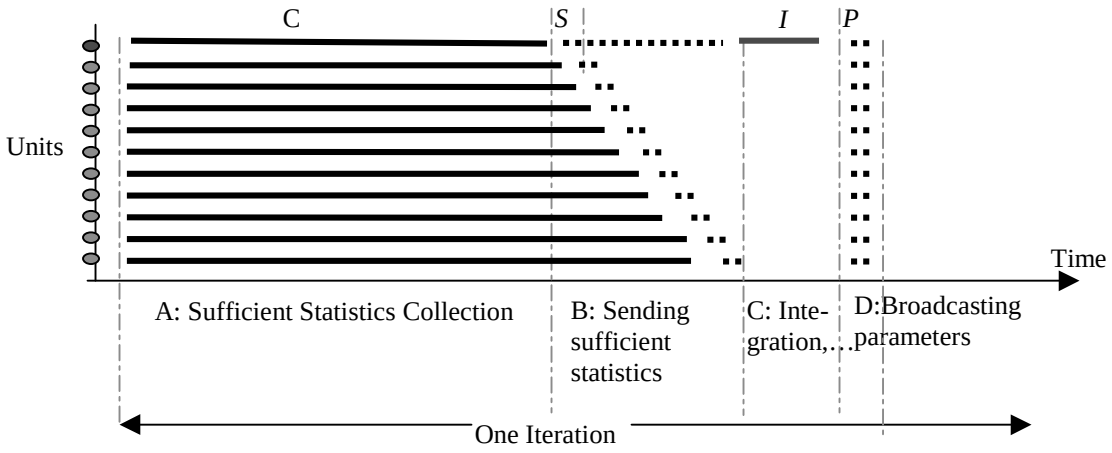

Figure 3. The timeline of all the units in one iteration. Assuming that the network has broadcasting feature. The load is adjusted so that waiting is minimized.

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

	The size of sufficient statistics from each unit in number of floating points.	The size of data on each computing unit.	The computational cost for collecting the sufficient statistics on each unit in each iteration.	The computational cost of integration of the sufficient statistics in each iteration.
K-Means	*2*K+K*dim*	*N/L*	*O(K*dim*N/L)*	*O(K*dim*L)*
K-Harmonic Means	*1+K+K*dim*	*N/L*	*O(K*dim*N/L)*	*O(K*dim*L)*
EM	$1+K+K*dim+K*dim^2$	*N/L*	$O(K*dim^2*N/L)$	$O(K*dim^2*L)$

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

Let t_d be the unit of time to process one data, t_n be the unit of time to transmit one data, and t_c the time to establish a connection, the utilization of the (slave) units can be estimated by (using *K*-Means as an example),

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

$$= \frac{1}{1+(C'/C)*(L^2/N)+(L^2/N)*(C''*t_n + t_c/K*\dim)/(C*t_d)} \qquad (19)$$

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

where *C, C',* and *C"* are constants. The connection time is shared by all data items that are transmitted together, which is *O(K*dim).* When *K*dim* is so large that the connection time t_c becomes negligible comparing with the transmission time $C''*K*\dim*t_n$, the term $t_c/K*\dim$ on the second line of (19) can be ignored and the utilization of the units is almost independent of the dimensionality, *dim*, and the number of clusters, *K* (the third line of (19)).

Test Result:

Using *K*-Means as an example, let N/L = 1,500,000 and dim = 40. The data size is 480 Mbytes per computing unit (8bytes per floating point number). The number of clusters K=100. It took 100 seconds to repartition and collect sufficient statistics on each unit, which is a HP/PC with 450 MHz Intel processor and enough memory. The size of the sufficient statistics is (2*100+100*40)*8 = 33600Bytes. It took less than 0.2 seconds (a very safe bound, the actual is much smaller) to transmit the data over the network (LAN, most of it is to start the connection. The actual transfer rate is about 800Kbytes/second). Assuming that 20 computing units are running in parallel (*N=30,000,000*), the total waiting time for 21 transmissions is 4.2 seconds (only 21 transmissions need to be counted. See Figure 2). The time for the Integrator to calculate new parameters (centroids) is negligible. The utilization of the units is 100/(100+4.2) = 96%. Running 60 units in parallel (*N=90* million and total data size is 28.8 GBytes), assuming the same

network performance (the transfer rate was tested on a real LAN with hundreds of computers during normal work hours), the utilization of the units is still greater than 89%. We assumed perfect alignment of all the periods in the estimation above. In reality, due to the randomness of the network behavior, there could be more delays. But it is very save to claim a 80% utilization because 100/(100+25) = 80%. We reserved at least another 12 seconds for imperfect alignment.

5. Dynamic Load Balancing

Data items can be moved from one computing unit to another during the period (B+C+D) (See Figure 2 or 3) 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 when the data set is consist of sparse vectors with variable number of non-zero elements in them. It is very difficult to pre-calculate the amount of time needed to collect the statistics on a given number of data vectors.

For dense vectors, the prediction of calculation time based on the number of data vectors is very accurate; a linear scale up based on the test result from a small number of vectors will work pretty well. But this is not true for sparse vectors. For sparse vectors, the amount of data to be loaded on to each machine becomes more difficult to estimate, even though the cost of partitioning and collecting sufficient statistics on each subset of data is still deterministic. A scale-down pre-test by sampling data provides an estimate. But if the number of samples is too small or the data size has a very skewed distribution, the estimate could be far off. To solve this problem, we propose to re-balance the load based on the real time measurements from the first iteration. Since the amount of data to be moved from one unit to another is still based on estimates, re-balancing could be applied again after the second iteration and so on. But the need for re-balancing after the each iteration diminishes very quickly. Only the first or maybe the

second re-balancing is needed. The necessity of a load re-balancing can be determined based on the utilization of the units calculated from the previous iteration.

Another possibility for the need of dynamic load balancing is following the slow trend of network performance change when the communication period (B+D) is significant.

Synchronization of the clocks on all units is not required because only the elapsed time on each unit is needed.

6. Conclusion

A class of clustering algorithms is scaled up for very large data sets by parallelism. Dynamic load balancing is used to maximize the utilization of all computing units. The idea presented in this paper applies to all iterative parameter estimation algorithms as long as the size of the parameters and the size of the sufficient statistics are small relative to the data size. We see potential applications of this idea in on-line commercial product recommendations where clustering is one of the techniques for calculating recommendations. Large number of PCs that are used during business hours for processing orders can be used during the night for updating the clustering of customers and products based on the updated sales information (customer buying patterns).

References

[A73] Anderberg, M. R. 1973. Cluster analysis for applications. Academic Press, New York. xiii + 35p.

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

[BF98] Bradley, P., Fayyad, U. M., C.A., "Refining Initial Points for KM Clustering", MS Technical Report MSR-TR-98-36, May 1998.

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

[DH72] Duda, R., Hart, P., "Pattern Classification and Scene Analysis", John Wiley & Sons, 1972.

[DVJCP99] DuMouchel, W., Volinsky, C., Johnson, T., Cortes, C. and Pregibon, D., "Squashing Flat Files Flatter," Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August, 1999.

[DLR77] Dempster, A. P., Laird, N.M., and Rubin, D.B., "Miximum Likelyhood from Incomplete Data via the EM Algorithm", Journal of the Royal Statistical Society, Series B, 39(1):1-38, 1977.

[FPU96] Fayyad, U. M., Piatetsky-Shapiro, G. Smyth, P. and Uthurusamy, R., "Advances in Knowledge Discovery and Data Mining", AAAI Press 1996.

[F94] T. J. Fountain , "Parallel Computing : Principles and Practice," Cambridge Univ Pr 1994.

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

[KGGK94] Vipin Kumar, Ananth Grama, Anshul Gupta, George Karypis , "Introduction to Parallel Computing : Design and Analysis of Parallel Algorithms," Addison-Wesley Pub Co, 1994.

[KC99] Sanpawat Kantabutra and Alva L. Couch, "Parallel K-means Clustering Algorithm on NOWs," NECTEC Technical journal (www.nectec.or.th)

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

[L84] S. P. Lloyd, "Least Squares Quantization in PCM," Technical Note, Bell Lab, 1957 and IEEE Trans. on Information Theory 1982.

[M67] MacQueen, J. 1967. 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.

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

[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

[PM99] 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.

[RV97] P. Resnick and H. Varian, "Recommender Systems," in CACM March 1997.

[RW84] Rendner, R.A. and Walker, H.F., "Mixture Densities, Maximum Likelihood and The EM Algorithm", SIAM Review, vol. 26 #2, 1984.

[SI84] 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.

[SK99] Sanpawat Kantabutra, "Parallel K-Means Clustering Algorithm on NOW", Semptember, 1999.

[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 (June 1996), Pages 103-114 in: SIGMOD '96.