A Collaborative Divide-and-Conquer K-Means Clustering Algorithm for Processing Large Data

Huimin Cui
SKL of Computer Architecture, Institute of Computing Technology, CAS, China
cuihm@ict.ac.cn

Gong Ruan
University of Chinese Academy of Sciences, China
ruangong@ict.ac.cn

Jingling Xue
School of Computer Science and Engineering, University of New South Wales, Australia
jingling@cse.unsw.edu.au

Rui Xie
SKL of Computer Architecture, Institute of Computing Technology, CAS, China
xierui@ict.ac.cn

Lei Wang
SKL of Computer Architecture, Institute of Computing Technology, CAS, China
wlei@ict.ac.cn

Xiaobing Feng
SKL of Computer Architecture, Institute of Computing Technology, CAS, China
fxb@ict.ac.cn

ABSTRACT

K-means clustering plays a vital role in data mining. As an iterative computation, its performance will suffer when applied to tremendous amounts of data, due to poor temporal locality across its iterations. The state-of-the-art streaming algorithm, which streams the data from disk into memory and operates on the partitioned streams, improves temporal locality but can misplace objects in clusters since different partitions are processed locally. This paper presents a collaborative divide-and-conquer algorithm to significantly improve the state-of-the-art, based on two key insights. First, we introduce a break-and-recluster procedure to identify the clusters with misplaced objects. Second, we introduce collaborative seeding between different partitions to accelerate the convergence inside each partition. Compared with the streaming algorithm using a number of wikipedia webpages as our datasets, our collaborative algorithm improves its clustering quality by up to 35.3% with an average of 8.8% while decreasing its execution times from 0.3% to 80.1% with an average of 48.6%.

1. INTRODUCTION

K-means clustering is a commonly used algorithm for data mining, which was proposed by S. P. Lloyd in 1957 [26]. It partitions n objects into k clusters such that similar objects belong to the same cluster, according to some similarity function. In spite of the fact that K-means was proposed over 50 years ago and thousands of clustering algorithms have been published since then, K-means is still widely used in a variety of areas, ranging from market segmentation, computer vision, geostatistics, astronomy to agriculture. As a representative scenario, billions of Web pages create terabytes of new data every day, with many of these data streams being unstructured, adding to the difficulty in analyzing them. The K-means clustering algorithm can be used to discover the natural groups of the data, allowing us to understand, process and summarize the data.

Specifically, the K-means clustering algorithm [26] assigns each object to the cluster whose center (also called centroid) is the nearest. The algorithm starts with an initial set of cluster centers, chosen at random or according to some heuristic procedure. Then the algorithm iteratively assigns each object to one of the clusters. In each iteration, each object is assigned to its nearest cluster center according to the Euclidean distance between the two. Then the cluster centers are re-calculated [31]. The pseudo code of the K-means clustering algorithm is shown in Algorithm 1.

Algorithm 1 K-means clustering algorithm.

procedure KMeans(S, k)
1: Initialize k empty clusters C_1, C_2, ..., C_k.
2: Initialize cluster centers for C_1, C_2, ..., C_k randomly or heuristically.
3: **while** The convergence criterion is not met **do**
4: **for** each object s in S **do**
5: Compute the distance from s to all centers.
6: Assign s to its nearest cluster.
7: **end for**
8: **for** each cluster C_i in $\{C_1, C_2, ..., C_k\}$ **do**
9: Update the center of C_i according to all objects belonging to C_i.
10: **end for**
11: **end while**

As the data volume is getting more tremendous, the original K-means algorithm would face a big challenge of reusing

CF '14, May 20 - 22 2014, Cagliari, Italy

data. When the data volume is large, every object would be fetched from disk into memory for each iteration, which means the data in memory cannot be reused, causing poor temporal locality. To alleviate this problem, Guha et al presented a one-pass divide-and-conquer k-means algorithm in [15], as shown in Algorithm 2, which divides the data into partitions, clusters each of these partitions in order to find $a * k$ centers (called a *Local_Cluster*), and then again clusters the obtained centers (with each center weighted by the number of points assigned to it, called a *Merge_Cluster*) .

Algorithm 2 Divide-and-Conquer streaming K-means clustering algorithm [15].

procedure Streaming-KMeans(S, k)
1: Divide S into P partitions $S^1, S^2, ..., S^P$.
2: **for** each partition S^i ($i \in [1, P]$) **do**
3: *Local_Cluster:* Using K-means to cluster S^i into $a * k$ clusters.
4: **end for**
5: Let S' be the $a * P * k$ centers obtained, with each center being weighted by the number of points assigned to it.
6: *Merge_Cluster:* Using K-means to Cluster S' into k clusters.

The state-of-the-art divide-and-conquer algorithm assumes that all input data are used only once. The *Merge_Cluster* phase uses each center to represent its corresponding cluster, unaware of the distribution of individual objects inside the cluster. In particular, if two objects are misplaced into one cluster in the *Local_Cluster* phase, the *Merge_Cluster* phase would have no opportunity to eliminate the error which would be eventually embodied in the final results, as the motivating example will show in Section 2. To reduce such errors, Guha et al introduced the constant a into the algorithm [15], thus causing the computations to be magnified by a times. As a result, users are required to determine an appropriate a to make a tradeoff between clustering quality and efficiency. Furthermore, how to determine a is still an open question.

In this paper, we propose a collaborative divide-and-conquer K-means algorithm to avoid introducing the above constant a while retaining good temporal locality exhibited by Algorithm 2, and additionally, to achieve good clustering quality. Our key insights are:

- *Collaborative Merging.* We introduce a break-and-recluster procedure into the *Merge_Cluster* phase, which allows us to identify a cluster with misplaced objects, break the cluster and re-assign its objects to their appropriate destination clusters.
- *Collaborative Seeding.* As more and more partitions are processed, the centers obtained from completed partitions would be gradually approaching the real ones. Therefore, we present a collaborative seeding mechanism, which enables the early-finished partitions to deliver achieved cluster centers to subsequent partitions as their initial points, enabling the clustering process to converge in a small number of iterations.
- We implement the collaborative divide-and-conquer algorithm on an 8-core Intel platform. Validation using a number of wikipedia webpages as the datasets shows that our collaborative approach improves the clustering quality by 8.8% in average, up to 35.3%. Meanwhile, Our approach decreases the execution time from 0.3% to 80.1%, with an average of 48.6%.

The rest of the paper is organized as follows. Section 2 introduces the background and motivation. Section 3 presents our collaborative divide-and-conquer algorithm. Section 4 presents the theoretical discussion. Section 5 discusses the parallel implementation issues. Section 6 describes our experimental validation. Section 7 discusses the related work. Section 8 concludes.

2. BACKGROUND AND MOTIVATION

2.1 Background - K-means Clustering

We introduce briefly the K-means clustering algorithm, which is typically as shown in Algorithm 1.

Cluster Center $\mu(C_i)$.
In algorithm 1, for each iteration, we need to update the centers for all clusters according to the objects belonging to that cluster. For a given cluster C_i, its center is defined as the mean or centroid of all objects in the cluster, as

$$\mu(C_i) = \frac{1}{|C_i|} \sum_{x \in C_i} x \tag{1}$$

Objective Function.
A typical measure of how well the centroids represent the members of their clusters is the *residual sum of squares*, which is a typical objective function for K-means and the goal of the algorithm is to minimize it. In this paper, we use the *residual sum of squares* as our objective function, denoted as RSS, which is the squared distance of each vector from its centroid summed over all vectors, as

$$\begin{aligned} &RSS = \sum_{i=1}^{k} RSS_k \\ &where, \\ &RSS_i = \sum_{x \in C_i} |x - \mu(C_i)|^2 \end{aligned} \tag{2}$$

where N is the number of objects.

Termination Condition.
In general, the K-means clustering algorithm has three termination conditions: (1) a fixed number of iterations has been executed; (2) the centroids of clusters do not change between iterations, or the RSS difference between two iterations is smaller than a threshold; (3) assignment of objects to clusters does not change between iterations. Typically, (2) or (3) ensures that the clustering is of a desired quality, and (1) limits the execution time of the clustering algorithm. In this paper, we use (2) as our termination condition, but our approach works if one of the other two termination conditions is used.

2.2 Problem of Existing Divide-and-Conquer Approach

As mentioned in Section 1, the state-of-the-art divide-and-conquer K-means algorithm is able to exploit temporal locality for large data. However, due to the unawareness of cluster internal organizations during the *Merge_Cluster* phase, it has to introduce a constant a and requires users to make a tradeoff between clustering quality and efficiency. In this section, we use a motivating example to discuss this issue in detail.

Figure 1 shows a motivating example, for $k = 3$ and $a = 1$. First, the data S in Figure 1(a) are divided into two asymmetric partitions, S^1 and S^2. Then, the *Local_Cluster* phase clusters each partition into three clusters, denoted as C^{11}, C^{12}, C^{13} and C^{21}, C^{22}, C^{23}, respectively, as shown in Figures 1(b) and (c). For convenience, we use C^{ij} to represent a cluster and c^{ij} for the corresponding cluster center. Finally, the *Merge_Cluster* phase would further group the achieved 6 centers into 3 clusters (Figure 1(d)), and the final clustering results are shown in Figure 1(e).

From Figure 1, we can observe that when $a = 1$, the asymmetric partition led to a "bad" cluster (C^{23}), as shown in Figure 1(c), which inaccurately contains objects that should have belonged to two different clusters. Afterwards, the *Merge_Cluster* phase represents the objects located in one cluster as one concentrated point, thus having no opportunity to correct the inaccuracy. Therefore, the unexpected results are kept till the final results, as shown in Figure 1(e). To tackle this problem, a can be enlarged to get more clusters in the *Local_Cluster* phase and reduce the possibility of generating "bad" clusters. However, this approach increases the computation by a times, furthermore, there is no good model to determine the value of a.

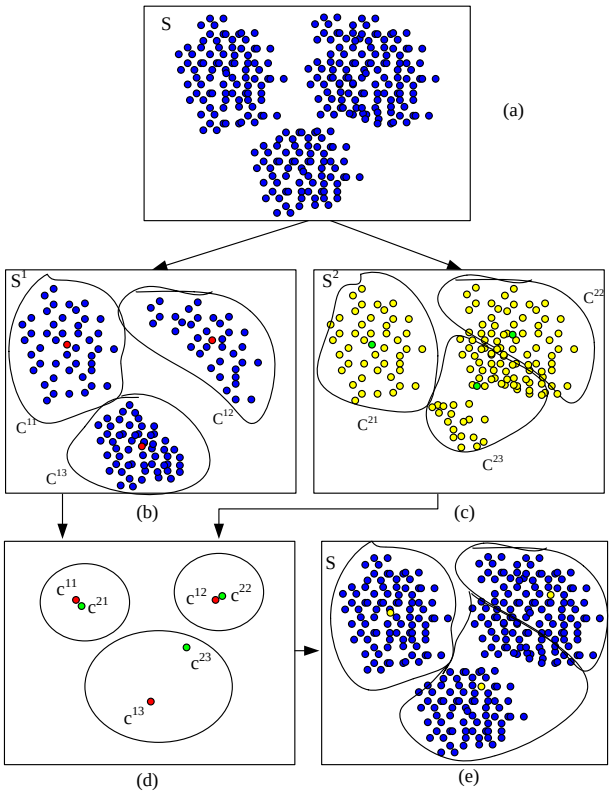

Figure 1: Unexpected results of the divide-and-conquer algorithm [15].

2.3 Our Motivation

Examining Figure 1(d), we can observe that c^{11} and c^{21} are very close to each other, which means they can be folded into one cluster, and similarly for c^{12} and c^{22}. However, c^{23} is located almost at the midpoint of c^{13} and $c^{12}(c^{22})$. This motivates us to split C^{23} into two parts, with one assigned to C^{13} and the other to $C^{12}(C^{22})$.

Figure 2 shows our key idea, where Figure 2(a) is the cluster centers obtained after the *Local_Cluster* phase, which is identical with Figure 1(d). Figure 2(b) shows the operation which splits C^{23} into two parts, which are successively assigned to C^{13} and $C^{12}(C^{22})$ respectively. In particular, we break C^{23} and re-assign each object inside it into C^{13} or $C^{12}(C^{22})$ according to the shortest distance. Now we can obtain three new cluster centers, as c^1, c^2, c^3 in Figure 2(c), which would be closer to the real centers compared with Figure 1(e).

3. COLLABORATIVE DIVIDE-AND-CONQUER ALGORITHM

3.1 Overview

Algorithm 3 gives our collaborative divide-and-conquer k-

Algorithm 3 Collaborative divide-and-conquer K-means clustering algorithm.

procedure Co-DCKMeans(S, k, P)

1: Divide S into P partitions $S^1, S^2, ..., S^P$.
2: Initialize k centers $c = (c_1, c_2, ..., c_k)$ randomly or heuristically.
3: **for** each partition S^i ($i \in [1, P]$) **do**
4: Set initial points as c.
5: *Local_Cluster:* Using K-means to cluster S^i into k clusters, as C^i.
6: Let S' be the $i*k$ centers of $C^1, ..., C^i$, with each center being weighted by the number of points assigned to it.
7: *Co_Seeding:* Using collaborative seeding to update $c = Co\_Seeding(S')$.
8: **end for**
9: Let C be the $P * k$ local clusters obtained.
10: *Co_Merging:* Using collaborative merging to Cluster C into k clusters.

means clustering algorithm, which takes three parameters, S for the input data to be clustered, k for the number of clusters, and P for the number of partitions.

First, the input data S is evenly divided into P partitions, as $S^1, S^2, ..., S^P$ (line 1), and k initial cluster centers are randomly or heuristically generated as the seed c, which would be taken as the initial points for the first partition (line 2). Second, the P partitions are sequentially processed. Each partition S^i takes c as its initial points, and uses the K-means algorithm to determine its own local cluster centers C^i (lines 4-5). After one partition is processed, we use *Co_Seeding* to update the seed c (lines 6-7), which will be used as the initial points of next partition. Third, the achieved $P * k$ local clusters are merged together into k new global clusters using collaborative merging *Co_Merging* (lines 9-10). Details of collaborative merging and seeding will be discussed in Section 3.2 and 3.3.

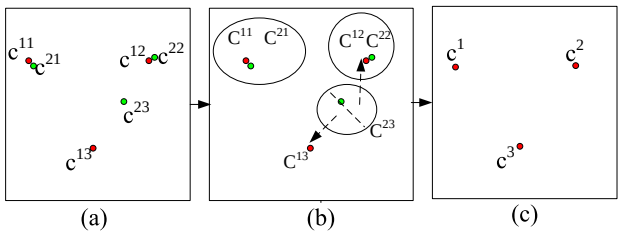

Figure 2: Example of the *break-and-recluster* operation.

3.2 Collaborative Merging

In Algorithm 2, *Merge_Cluster* uses K-means to cluster the weighted local centers S' into k clusters, unaware of the objects inside each local cluster. Instead of representing the objects located in one local cluster as one concentrated point, our collaborative merging identifies "bad" local clusters and traverses the objects inside them. We call the $P * k$ clusters after *Local_Cluster* as "local clusters", while the final k clusters as "global clusters". We extend the K-means-based approach in Algorithm 2 and propose a collaborative merging algorithm.

First, we use the original *Merge_Cluster* phase to cluster the $P * k$ centers to obtain k global clusters, with each center weighted by the number of points assigned to it. Substantially, we traverse the $P * k$ local clusters seeking for candidates for break-and-recluster. If a local cluster LC_i is closer to *more than one* global cluster centers, $c_{j1}, ..., c_{jm}$, we remove LC_i from its original global cluster, break LC_i and re-assign the objects in LC_i into $C_{j1}, ..., C_{jm}$.

In the collaborative merging, we must address two issues: 1) how to select the local cluster candidates for break-and-recluster, i.e., "bad" clusters, and 2) for a selected local cluster, how to determine its destination global clusters? We do so by introducing a so-called the *ε-Nearest Set* for a given local cluster, which includes all global clusters "near" to the given local clusters. If the set contains more than one element, the corresponding local cluster should be taken as a candidate for break-and-recluster, and the elements in the *ε-Nearest Set* are the destination global clusters.

Below in this section, we first introduce our weighted distance definition (Section 3.2.1) and then present our merging algorithm (Section 3.2.2).

3.2.1 Weighted Distance Definition

For two objects, we use Euclidean distance to measure the distance between them. While in the collaborative merging, we need to measure the distance between two clusters, and the distance between a cluster and a point.

Def 1 (Weighted Distance Between Cluster-Point). *Given a cluster (C_i) and a point (p), the weighted distance $\Omega(C_i, p)$ between C_i and p is defined as:*

$$\Omega(C_i, p) = |C_i| * |\mu(C_i) - p| \tag{3}$$

Lemma 1. *When two clusters C_i and C_j are folded into one $C_{i,j}$, the new cluster would contain all objects in C_i and C_j, and its center is the centroid of all of these objects, as:*

$$\begin{aligned} |C_{i,j}| &= |C_i| + |C_j| \\ \mu(C_{i,j}) &= \tfrac{\mu(C_i)*|C_i| + \mu(C_j)*|C_j|}{|C_i| + |C_j|} \end{aligned} \tag{4}$$

Def 2 (Weighted Distance Between Clusters). *Given two clusters (C_i and C_j), their weighted distance $\Omega(C_i, C_j)$ is defined as the RSS increment obtained after C_i and C_j are folded together, specifically as:*

$$\Omega(C_i, C_j) = RSS_{i,j} - (RSS_i + RSS_j) \tag{5}$$

Using Lemma 1, Equation 5 can be rewritten as:

$$\begin{aligned} \Omega(C_i, C_j) &= RSS_{i,j} - (RSS_i + RSS_j) \\ &= (\sum_{x \in C_i} |x - \mu(C_{i,j})|^2 + \sum_{x \in C_j} |x - \mu(C_{i,j})|^2) - \\ &\quad (\sum_{x \in C_i} |x - \mu(C_i)|^2 + \sum_{x \in C_j} |x - \mu(C_j)|^2) \\ &= \tfrac{|C_i| * |C_j|}{|C_i| + |C_j|} \quad |\mu(C_i) - \mu(C_j)|^2 \end{aligned} \tag{6}$$

Def 3 (ε-Nearest Set). *For a given local cluster LC_i and a set of global clusters C, let $\eta(LC_i, C)$ be the closest global cluster to LC_i in C. Thus LC_i's ε-nearest set $\varepsilon NS(LC_i, C)$ is defined as a set containing all the clusters whose weighted distance to LC_i is not larger than $(1 + \varepsilon) * \Omega(LC_i, \eta(LC_i, C))$:*

$$\begin{aligned} &\varepsilon NS(LC_i, C) = \\ &\quad \{C_j | \Omega(LC_i, C_j) < (1 + \varepsilon) * \Omega(LC_i, \eta(LC_i, C))\} \end{aligned} \tag{7}$$

In our definition of ε-nearest set, ε is a user-specified variable, for balance between clustering quality and efficiency. When ε is set to 0, break-and-recluster would be disabled, and our collaborative merging will behave as the *Merge_Cluster* in Algorithm 2. When ε is set to infinity, all input objects would be traversed, which would be inefficient. We will discuss and evaluate the influence of ε in Section 6.

3.2.2 Merging Algorithm

Algorithm 4 shows the collaborative merging algorithm, with the input C as the $P * k$ local clusters obtained at the *Local_Cluster* phase. First, we use K-means to cluster the local cluster centers into k global clusters, as in Algorithm 2 (lines 1-2). Then we traverse the $P*k$ local clusters (lines 3-8). If the ε-nearest set of one local cluster includes more than one global cluster, we break the local cluster and re-cluster its objects into its ε-nearest set (lines 4-7).

Algorithm 4 Collaborative merging algorithm.

procedure Co_Merging(C)
1: Let C' be the $P * k$ centers of C, $C'_i = \mu(C_i)$ and the weight of C'_i is $|C_i|$.
2: Use K-means to cluster the C' into k global clusters $G = (G_1, ..., G_k)$.
3: **for** each C_i from C **do**
4: Compute C_i's ε-nearest set $E = \varepsilon NS(C_i, G)$.
5: **if** $|E| > 1$ **then**
6: break and re-cluster $G = break(C_i, E, G)$.
7: **end if**
8: **end for**
9: Output G.

In particular, the break-and-recluster algorithm is shown in Algorithm 5. For each individual object s in the cluster to be broken (C^*), we compute the Euclidean distance from s to all centers of the global clusters in the corresponding ε-nearest set (E), and assign s to its nearest cluster. We will prove that the break-and-recluster operation would always improve the clustering results, in Section 4.

Algorithm 5 The break-and-recluster algorithm.

procedure break(C^*, E, G)
1: Let G^* be the global cluster that C^* belongs to.
2: **for** each object s in C^* **do**
3: Compute the Euclidean distance from s to all global centers in E.
4: Assign s to its nearest global cluster in E.
5: **end for**
6: **for** each cluster G_i in E **do**
7: Update the center of G_i.
8: **end for**
9: Output G.

3.3 Collaborative Seeding

The key insight of our collaborative seeding is that, intuitively the centers we get from partial partitions are good candidates as initial points for the remaining partitions, for large data. As shown in Figure 1, if we use the local centers obtained from S^1 as the initial points for processing S^2, only a small number of iterations would be required. Therefore we introduce this as a heuristic of "collaborative seeding", which would not be efficient for small or ordered data, but works for large and randomly distributed data.

As shown in Algorithm 3 (line 7), the collaborative seeding takes the weighted local cluster centers of completed partitions as the input S', which includes $i * k$ elements. Our objective is to find the k centers of $S^1, ..., S^i$ as the initial points for clustering S^{i+1}. Therefore, we leverage the K-means algorithm, and use the obtained k centers as the new seed.

4. THEORETICAL DISCUSSION

Compared with the state-of-the-art divide-and-conquer algorithm in [15], our approach introduces an extra pass for accessing the input data. Therefore, the multiple partitions can be clustered in a collaborative way. In particular, we have proposed collaborative merging to improve clustering quality without introducing the cost of constant a in Algorithm 2. As discussed above, the key point is that we have introduced the break-and-recluster operations into the merging step. Different from representing the objects located in one local cluster as one concentrated point in [15], our algorithm is aware of the distribution of individual objects inside one local cluster. We will prove that the break-and-recluster operations are beneficial.

In our theoretical analysis, *cost* is used to represent the objective function value for a clustering result, which is the RSS in this paper.

Theorem 1. *When clustering n local clusters ($LC_1, ..., LC_n$, with the weights as $w_1, ..., w_n$) into k clusters ($G_1, ..., G_k$), let $\mathcal{C}$ be the solution achieved using Algorithm 2. If we break M_i into more than one cluster using Algorithm 5 and let the solution be $\mathcal{C}'$, the cost will be optimized, i.e., $\Phi_{\mathcal{C}'} \leqslant \Phi_{\mathcal{C}}$.*

Proof. Assume one local cluster LC_x is assigned into G^*_a in $\mathcal{C}$, and we have decided to break it and re-cluster it into G^*_a and G_b in $\mathcal{C}'$.

We introduce an intermediate state $\mathcal{C}_0$, which describes the state before LC_x is assigned to any global clusters. In particular, $\mathcal{C}_0$ contains $k+1$ clusters, k global clusters and one local cluster (LC_x). Let G_a be the global cluster which is closest to LC_x. Therefore Algorithm 2 would assign LC_x into G_a, turn G_a into G^*_a and obtain $\mathcal{C}$.

We use RSS_0 to represent the RSS for $\mathcal{C}_0$. Let $x = |LC_x|$, $a = |G_a|$, $a^* = |G^*_a|$, $X = \mu(LC_x)$, $A = \mu(G_a)$, and $A^* = \mu(G^*_a)$.

In the solution of $\mathcal{C}$, LC_x is completely assigned to G_a, and the RSS increment over RSS_0 can be computed using Equation 6, specifically as:

$$\begin{aligned} \Delta RSS_{\mathcal{C}} &= RSS_{\mathcal{C}} - RSS_0 \\ &= \tfrac{x*a}{x+a} \quad |X - A|^2 \end{aligned} \tag{8}$$

On the other hand, in the solution of $\mathcal{C}'$, LC_x is broken and re-clustered. Assume LC_x is split into LC_a and LC_b, with LC_a re-assigned into G_a and LC_b into G_b.

Let $x_a = |LC_a|$, $x_b = |LC_b|$, $X_a = \mu(LC_a)$, $X_b = \mu(LC_b)$, $b = |G_b$, and $B = \mu(G_b)$. The RSS increment over RSS_0 is

$$\begin{aligned} \Delta RSS_{\mathcal{C}'} &= RSS_{\mathcal{C}'} - RSS_0 \\ &= \tfrac{x_a*a}{x_a+a} \quad |X_a - A|^2 + \tfrac{b*x_b}{b+x_b} \quad |X_b - B|^2 \end{aligned} \tag{9}$$

With Algorithm 5,

$$\forall x \in X_b, |x-A| \geqslant |X-A|, so$$
$$m|X-A| = \Sigma_{x\in X_a}|x-A| + \Sigma_{x\in X_b}|x-A|$$
$$\geqslant x_a|X_a-A| + x_b|X_b-A|$$

Therefore, we can get $|X_a - A| \leqslant |X - A|$.
Meanwhile, Algorithm 5 gives $|X_b - B| \leqslant |X - A^*|$. So

$$\begin{aligned}
&\Delta RSS_{\mathcal{C}'} - \Delta RSS_{\mathcal{C}} \\
&= \tfrac{x_a * a}{x_a + a} \quad |X_a - A|^2 + \tfrac{b * x_b}{b + x_b} \quad |X_b - B|^2 - \tfrac{x * a}{x + a} \quad |X - A|^2 \\
&\leqslant \tfrac{x_a * a}{x_a + a} \quad |X - A|^2 + \tfrac{b * x_b}{b + x_b} \quad |X - A^*|^2 - \tfrac{x * a}{x + a} \quad |X - A|^2 \\
&= \tfrac{x_a * a}{x_a + a}|X - A|^2 + \tfrac{b * x_b}{b + x_b}(\tfrac{a}{a+x})^2|X - A^*|^2 - \tfrac{x * a}{x+a}|X - A|^2 \\
&= -\tfrac{a+b+x}{(a+x_a)(b+x_b)(a+x)^2}(a^2 {x_b}^2)|X - A^*|^2 \\
&\leqslant 0
\end{aligned}$$

Therefore, the break-and-recluster operation reduces the cost, and the proof can be extended for breaking one local cluster into more than two other global clusters. Thus the theorem has been proved. □

5. PARALLEL IMPLEMENTATION

In this paper, we focus on machines with shared memory and consider fine-grained parallelization using OpenMP. Liao parallelized the basic K-means algorithm (Algorithm 1) in [3], which annotated the two *for* loops inside one iteration as *omp parallel*. In particular, all input objects are traversed and assigned to the destination cluster in parallel, and all cluster centers are updated also in parallel. We leverage this approach in our implementation, to parallelize *Local_Cluster* phase. Meanwhile, the collaborative seeding is also a K-means algorithm, which can be parallelized in the same way.

Furthermore, we have also parallelized the collaborative merging algorithm (Algorithm 4). Since the set of global clusters G is globally maintained, we keep the traversal of C as sequential and parallelize the ε-nearest set computation and break-and-recluster operation. For the ε-nearest set computation, the weighted distances to all global clusters are computed in parallel. And for the break-and-recluster operation, both of the two *for* loops in Algorithm 5 are parallelized.

However, our algorithm can be parallelized with coarse granularity in clusters using MPI. In particular, the partitions can be processed in parallel, with one or more partitions assigned to one node in a cluster. The important thing to note is that, for collaborative seeding, there exists dependences between different partitions. Thus, each node maintains its own seed and the collaborative seeding would not cross different nodes. Furthermore, we can also exploit two levels of parallelism implemented using MPI+OpenMP. More details about the coarse-grained implementation are beyond the scope of this paper.

6. EVALUATION

We demonstrate using a number of real-world datasets containing wikipedia webpages for document clustering using the K-means algorithm. The target platform used is an 8-core Intel platform with two 2.40GHz quad-core Xeon E5620 processors. Each processor has 32KB L1 data cache per core, 32KB L1 instruction cache per core, 256KB L2 cache per core, and 12MB L3 cache shared by all cores. The physical memory is 4G Bytes.

6.1 Methodology and Dataset

We use the document clustering as our target application. Document clustering, also called as text clustering, automatically organizes documents into clusters, and is widely used for topic extraction, information retrieval and filtering. Each document is turned into a vector using TF-IDF, which leverages "term frequency - inverse document frequency" to reflect how important a word is to a document in a collection of corpus. In particular, each dimension corresponds to a separate term, e.g., word, and the dimensionality of the vector is the number of words in the vocabulary (the number of distinct words occurring in the corpus) [33, 32].

We download 14 datasets from Wikipedia database [4], with each dataset containing a large number of wikipedia webpages for clustering. We then extract each document into a TF-IDF vector using Lucene [1]. The number of non-zero element in the datasets ranges from 30M to 2G. For each dataset, we set the number of clusters k as 6, 8, 12 and 20 respectively. Therefore, we have $14 * 4 = 56$ workloads in total. In our evaluation, we compare the clustering quality and efficiency with the divide-and-conquer streaming algorithm in [15] (Algorithm 2, for which a is set to 1). We implement both algorithms using OpenMP, and the partition size is set as $20M$. For both algorithms, the iteration ends when the difference of RSS between two iterations is below 0.001. For our collaborative divide-and-conquer algorithm, the parameter of ε is set to 0.5.

6.2 Clustering Quality

We use the RSS as the objective function for measuring the clustering quality, so the smaller, the better. Figure 3 depicts the normalized RSS for the divide-and-conquer streaming (as the "Streaming" bars) algorithm and our collaborative algorithm (as the "Collaborative" bars).

For the 56 workloads, our collaborative divide-and-conquer approach decreases the RSS by 8.8% in average, up to 35.3% for workload 20. The benefit of our clustering quality varies with workloads. If the streaming algorithm generates few "bad" local clusters, such as workloads 1, 2, 3, 41, our collaborative algorithm would get similar qualities with the streaming algorithm. Otherwise, our collaborative algorithm would exhibit better clustering quality.

6.3 Efficiency

Figure 4 compares the execution times of the streaming algorithm and our collaborative algorithm, normalized by the execution time of the streaming algorithm. Our approach decreases the execution time from 0.3% to 80.1%, with an average of 48.6%.

For both algorithms, we use the same partition size and the convergence criterion. The performance benefit of our collaborative approach comes from our collaborative seeding. Due to the good initial points selection, our approach

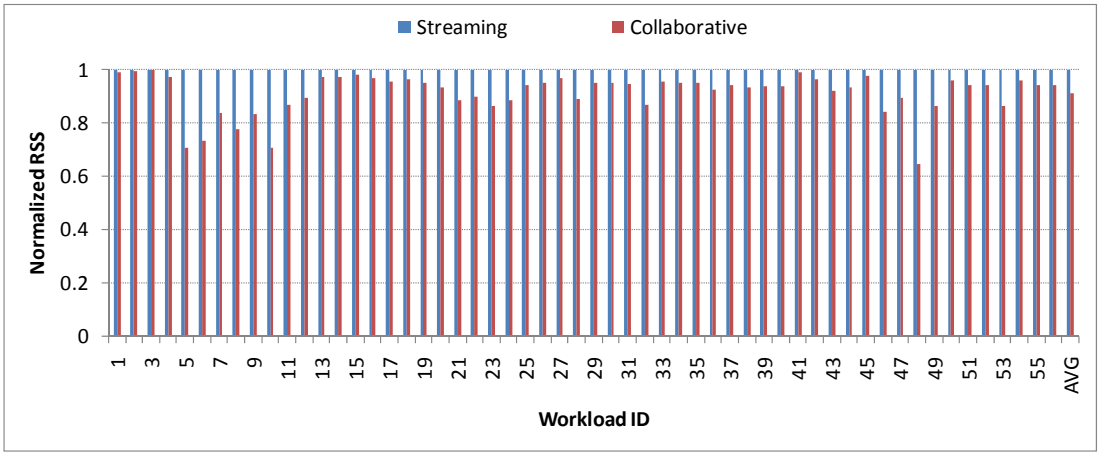

Figure 3: Normalized RSS for the 56 workloads of document clustering.

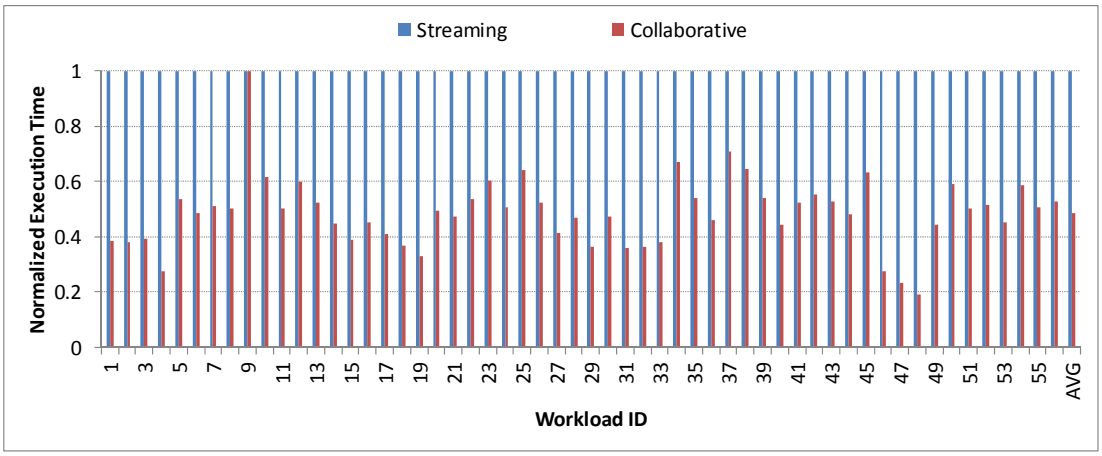

Figure 4: Normalized execution time for the 56 workloads of document clustering.

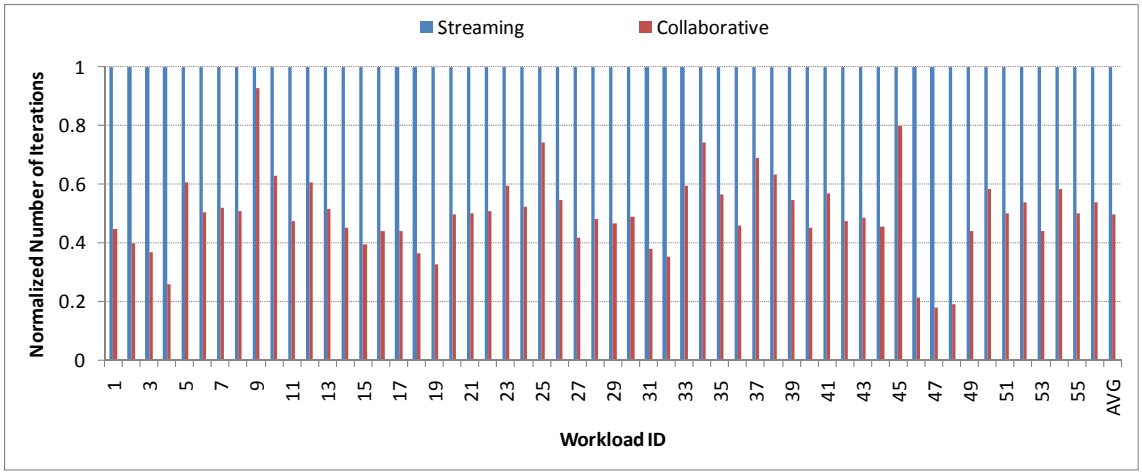

Figure 5: Normalized number of iterations for the 56 workloads of document clustering.

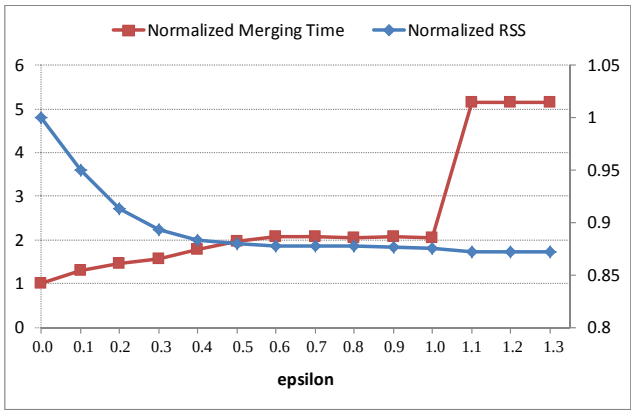

Figure 6: Normalized RSS / merging time varying with ε.

significantly reduces the number of iterations required to meet the convergence criterion. Figure 5 shows the number of iterations for the two algorithms, normalized by the streaming algorithm. The good correlation between Figure 4 and 5 illustrates that decreasing the number of iterations leads to the performance improvement.

Note that in Figure 4 and 5, our approach does not significantly decrease the number of iterations for workload 9, thus yielding negligible performance improvement. The reason is that this workload takes only a small number of iterations until the convergence criterion is met, i.e., 7. Our collaborative algorithm reduces the number of iteration from 7 to 6, therefore the execution time is only slightly decreased.

6.4 Effects of ε

To evaluate the effects of the parameter ε when calculating the ε-nearest sets in the collaborative merging, we take workload 11 for further analysis. Figure 6 shows the normalized RSS and merging time varying with ε. The blue line represents the normalized RSS when ε varies from 0 to 1.3 (against the right y-axis), and the red line represents the normalized merging time (against the left y-axis).

As ε increases, our collaborative merging would identify more "bad" local clusters for the break-and-recluster procedure, leading to better clustering results, i.e., smaller RSS. Meanwhile, as more local clusters are identified for break-and-recluster, more computations are introduced, causing the merging time increased. As Figure 6 shows, when ε increases from 0 to 0.5, the RSS decreases rapidly while the merging time increases slightly. We observed similar results for a number of workloads. Therefore, we set ε to 0.5 by default.

In Figure 6, we can observe that when ε exceeds 1.0, the merging time increases dramatically while the clustering result exhibits negligible improvement. The reason is that when ε is too large, some local clusters would be misidentified as "bad", causing a number of unnecessary break-and-recluster operations.

6.5 Comparing with Optimal Results

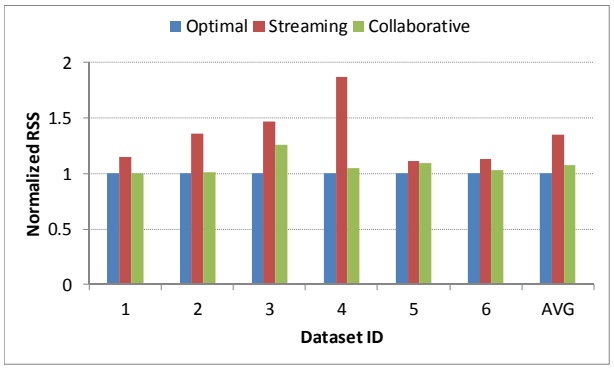

Figure 7: Normalized RSS for the 6 synthetic datasets.

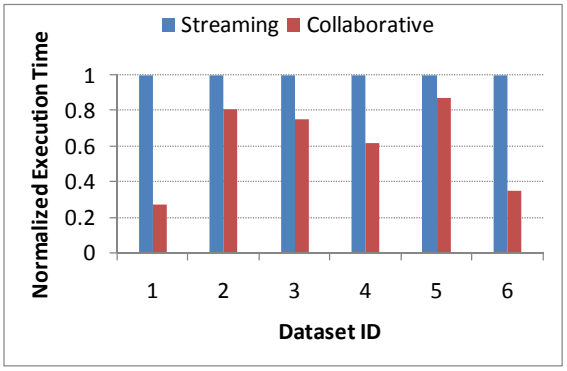

Figure 8: Normalized execution time for the 6 synthetic datasets.

The K-means clustering algorithm does not guarantee to find the global optimal solution. Therefore, for the 54 workloads of document clustering, we cannot use the optimal clustering results for comparison.

To compare our results with the optimal values, we generate 6 additional synthetic datasets using the approach in [5, 6]. In particular, we choose k random points from the d-dimensional space as the cluster centers, and then add n/k gaussian random points (with variance 1) around each center. The configurations of the 6 datasets are listed in Table 1.

Table 1: Configurations of 6 additional synthetic datasets.

Dataset ID	Number of Objs (n)	Dimension(d)	k
1	10000000	8	10
2	1600000	10	10
3	1600000	12	12
4	160000	20	12
5	1000000	8	40
6	10000000	4	20

Figure 7 shows the clustering results for the 6 synthetic datasets in Table 1, with the RSS normalized to the optimal value. For the streaming algorithm, the achieved RSS is 34.8% larger than the optimal value in average, ranging from 11.5% to 86.4%. In contrast, our collaborative algorithm

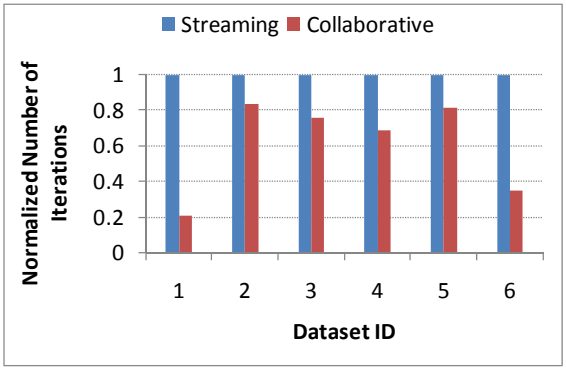

Figure 9: Normalized number of iterations for the 6 synthetic datasets.

achieves more closer clustering results to the optimal value, with the RSS only 7.4% larger than the optimal value in average, ranging from 0% to 25.7%.

Figure 8 and 9 show the normalized execution time and number of iterations for the 6 synthetic datasets. Our collaborative approach reduces the number of iterations by 18.9%-79.1%, causing the execution times to be shortened by 13.3%-73.2%.

7. RELATED WORK

There has been a lot of work on clustering algorithms, especially the K-means clustering which is one of the fundamental algorithms in machine learning [14, 15, 34, 23, 7, 6, 20]. Lloyd's approach is the most popular approach in practice, due to its simplicity [26], but its clustering quality is limited. In the theory community, a number of researchers made great efforts to improve the clustering quality. Inaba et al [19] presented an exact algorithm for the K-means problem, with the running time of $O(n^{kd})$, and Vega et al [10], Har-Peled et al [17], Kumar et al [24] presented some polynomial time approximation schemes. However, these algorithms are highly exponential in k, and are impractical even for small n, k, and d, where n is the number of objects, k is the number of clusters, and d is the number of dimensions. In recent years, Ostrovsky et al [28] and Arthur et al [6] presented K-Means++, a probability-based seeding technique, which can be leveraged together with Lloyd's algorithm to obtain good clustering quality and performance.

As the data volume is getting more tremendous, researchers proposed streaming models for data clustering. In particular, datasets are far too large to fit in main memory and are typically stored in secondary storage devices. Therefore, the datasets are constrained to be accessed in a streaming way for efficiency [15]. Munro and Paterson first studied the space requirement of selection and sorting as a function of the number of passes over the data [27], and Henzinger et al formalized the model in [18]. Charikar et al presented a constant-factor approximation algorithm for the K-center problem, which is very close to the K-means problem [9]. Guha et al proposed a divide-and-conquer approach for the streaming K-means clustering [16, 15], as we have discussed in Section I. Comparing with existing streaming divide-and-conquer algorithm [15], our approach introduced an extra pass to access the input dataset for collaborative merging and seeding, and the experimental results show that the cost is worthwhile.

Meanwhile, there also has been a lot of work on the parallel K-means algorithm and implementation, such as parallel implementation based on SIMD hypercube network [25, 29], master/slave message passing architecture [11, 13, 21, 22, 35], shared memory multi-core processor [30, 3], GPU [12] and MapReduce programming model [36, 2]. Based on the probability-based seeding approach K-Means++ [6], Bahmani et al proposed a parallel seeding approach that can find a good initial set of centers rapidly [8]. Our approach can work together with the parallel seeding technique, with the found seeds being our initial set of centers for iteration.

8. CONCLUSION

This paper presents a collaborative divide-and-conquer algorithm to significantly improve the state-of-the-art divide-and-conquer K-means clustering algorithm, which streams the data from disk into memory and operates on the partitioned streams, improves temporal locality but can misplace objects in clusters since different partitions are processed locally. Our approach is based on two key insights. First, we introduce a break-and-recluster procedure to identify the clusters with misplaced objects. Second, we introduce collaborative seeding between different partitions to accelerate the convergence inside each partition. Compared with the streaming algorithm using a number of wikipedia webpages as our datasets, our collaborative algorithm improves its clustering quality by up to 35.3% with an average of 8.8% while decreasing its execution times from 0.3% to 80.1% with an average of 48.6%.

Acknowledgements

This research is supported in part by the National High Technology Research and Development Program of China (2012AA010902), the National Key Basic Research Program of China (2011CB302504), the National Natural Science Foundation of China (61202055, 60925009, 61100011), the Innovation Research Group of NSFC (60921002), and Australian Research Council Grants (DP0987236, DP110104628 and DP130101970). We would like to thank all the reviewers for their valuable comments and suggestions.

9. REFERENCES

[1] Apache lucene. http://lucene.apache.org/.
[2] Apache mahout. http://mahout.apache.org/.
[3] Parallel k-means data clustering. http://users.eecs.northwestern.edu/ wkliao/Kmeans/.
[4] Wikipedia database download. "http://en.wikipedia.org/wiki/Wikipedia:Database_download".
[5] N. Arlon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. In *NIPS*, 2009.
[6] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the*

eighteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[7] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. In *"Siam Journal of Computing"*, pages 544–562, 2004.

[8] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. "scalable kmeans++". In *VLDB*, 2012.

[9] M. Charikar, L. O. Callaghan, and R. Panigrahy. "better streaming algorithms for clustering problems". In *STOC*, 2003.

[10] W. F. de la Vega, M. Karpinski, C. Kenyon, and Y. Rabani. "approximation schemes for clustering problems". In *STOC*, 2003.

[11] I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, pages 245–260, London, UK, 2000.

[12] R. Farivar, D. Rebolledo, E. Chan, and R. H. Campbell. A parallel implementation of k-means clustering on gpus. In *PDPTA*, pages 340–345, 2008.

[13] G. Forman and B. Zhang. Distributed data clustering can be efficient and exact. In *SIGKDD Explor. Newsl*, pages 34–38, 2000.

[14] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. In *Theoretical Computer Science*, pages 293–306, 1985.

[15] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. on Knowl. and Data Eng.*, 15(3):515–528, Mar. 2003.

[16] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. "clustering data streams". In *FOCS*, 2000.

[17] S. Har-Peled and S. Mazumdar. "on coresets for k-means and k-median clustering". In *STOC*, 2004.

[18] M. R. Henzinger, P. Raghavan, , and S. Rajagopalan. "computing on data streams technical report". In *Digital Equipment Corporation, Systems Research Center*, 1998.

[19] M. Inaba, N. Katoh, and H. Imai. "applications of weighted voronoi diagrams and randomization to variance-based k-clustering". In *SCG*, 1994.

[20] A. K. Jain. "data clustering 50 years beyond k-means". In *Pattern Recognition Letters*, 2009.

[21] D. Judd, P. K. McKinley, and A. K. Jain. Large-scale parallel data clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20:871–876, 1998.

[22] S. Kantabutra and A. Couch. Parallel k-means clustering algorithm on nows. *NECTEC Technical Journal*, 1, 1999.

[23] T. Kanungo, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for k-means clustering. In *"Computational Geometry Theory and Applications"*, pages 89–112, 2004.

[24] A. Kumar, Y. Sabharwal, and S. Sen. "a simple linear time (1+ e)-approximation algorithm for k-means clustering in any dimensions". In *FOCS*, 2004.

[25] X. Li and Z. Fang. Parallel clustering algorithms. *Parallel Comput.*, 11:275–290, 1989.

[26] S. P. Lloyd. Least square quantization in pcm. In *Bell Telephone Laboratories Paper*, 1982.

[27] J. I. Munro and M. S. Paterson. "selection and sorting with limited storage". In *Theoretical Computer Science*, 1980.

[28] R. Ostrovsky, Y. Rabani, L. Schulman, and C. Swamy. "the effectiveness of lloyd-type methods for the k-means problem". In *FOCS*, 2006.

[29] S. Ranka and S. Sahni. Clustering on a hypercube multicomputer. *IEEE Transactions on Parallel and Distributed Systems*, 2:129–137, 1991.

[30] S. T. Rao, E. Prasad, and N. Venkateswarlu. A critical performance study of memory mapping on multi-core processors An experiment with k-means algorithm with large data mining data sets. *International Journal of Computer Applications*, 1:90–98, 2010.

[31] L. Rokach and O. Maimon. *CLUSTERING METHODS, Chapter 15, Data Mining and Knowledge Discovery Handbook*. Springer, 2010.

[32] G. Salton and C. Buckley. erm-weighting approaches in automatic text retrieval. *Journal of Information Processing and Management*, 24:513–523, 1988.

[33] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18:613–620, 1975.

[34] V. Vempala and G. Wang. A spectral algorithm of learning mixtures of distributions. In *FOCS*, pages 113–123, 2002.

[35] Y. Zhang, J. Sun, Y. Zhang, and X. Zhang. Parallel implementation of clarans using pvm. In *International Conference on Machine Learning and Cybernetics*, pages 1646–1649, 2004.

[36] W. Zhao, H. Ma, and Q. He. Parallel k-means clustering based on mapreduce. In *CloudCom*, pages 674–679, 2009.