Parallel Spectral Clustering in Distributed Systems

Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, Edward Y. Chang

Abstract

Spectral clustering algorithms have been shown to be more effective in finding clusters than some traditional algorithms such as k-means. However, spectral clustering suffers from a scalability problem in both memory use and computational time when the size of a data set is large. To perform clustering on large data sets, we investigate two representative ways of approximating the dense similarity matrix. We compare one approach by sparsifying the matrix with another by the Nyström method. We then pick the strategy of sparsifying the matrix via retaining nearest neighbors and investigate its parallelization. We parallelize both memory use and computation on distributed computers. Through an empirical study on a document data set of $193,844$ instances and a photo data set of $2,121,863$, we show that our parallel algorithm can effectively handle large problems.

Index Terms

Parallel spectral clustering, distributed computing, normalized cuts, nearest neighbors, Nyström approximation.

I. Introduction

Clustering is one of the most important subroutines in machine learning and data mining tasks. Recently, spectral clustering methods (e.g., [35] [31] [13]), which exploit pairwise similarities of data instances, have been shown to be more effective than traditional methods such as k-means, which only considers the similarity values from instances to k centers. (We denote k as the number of desired clusters.) Because of its effectiveness in finding clusters, spectral clustering has been widely used in several areas such as information retrieval and computer vision (e.g., [9] [49] [35] [50]). Unfortunately, when the number of data instances (denoted as n) is large, spectral clustering can encounter a quadratic resource bottleneck [13] [25] in computing pairwise similarity among n data instances, and in storing the large similarity matrix. Moreover, the algorithm requires considerable time and memory to find and store the first k eigenvectors of a Laplacian matrix.

The most commonly used approach to address the computational and memory difficulties is to zero out some elements in the similarity matrix, or to sparsify the matrix. From the obtained sparse similarity matrix, one then finds

W.-Y. Chen is with the Department of Computer Science, University of California, Santa Barbara, CA 93106, USA.

Y. Song is with the Department of Automation, Tsinghua University, Beijing, 100084, China.

C.-J. Lin is with the Department of Computer Science, National Taiwan University, Taipei, 106, Taiwan.

H. Bai and E. Y. Chang are with Google Research, USA/China.

the corresponding Laplacian matrix and calls a sparse eigensolver. Several methods are available for sparsifying the similarity matrix [28]. A sparse representation effectively handles the memory bottleneck, but some sparsification schemes still require calculating all elements of the similarity matrix. Another popular approach to speed up spectral clustering is by using a dense sub-matrix of the similarity matrix. In particular, Fowlkes et al. [13] propose using the Nyström approximation to avoid calculating the whole similarity matrix; this approach trades accurate similarity values for shortened computational time. In another work, Dhillon et al. [10] propose a method that does not use eigenvectors, but they assume the availability of the similarity matrix. In this work, we aim at developing a parallel spectral clustering package on distributed environments. We begin by analyzing 1) the traditional method of sparsifying the similarity matrix and 2) the Nyström approximation. While the sparsification approach may be more computationally expensive, our experimental results indicate that it may yield a slightly better solution. This paper presents one of the first detailed comparisons between the effectiveness of these two approaches.

We consider the sparsification strategy of retaining nearest neighbors, and then investigate its parallel implementation. Our parallel implementation, which we call parallel spectral clustering (PSC), provides a systematic solution for handling challenges from calculating the similarity matrix to efficiently finding eigenvectors. Note that parallelizing spectral clustering is much more challenging than parallelizing k-means, which was performed by e.g., [6] [11] [18] [48]. PSC first distributes n data instances onto p distributed machine nodes. On each node, PSC computes the similarities between local data and the whole set in a way that uses minimal disk I/O. Then PSC stores the eigenvector matrix on distributed nodes to reduce per-node memory use. Together with parallel eigensolver and k-means, PSC achieves good speedup with large data sets.

The remainder of this paper is organized as follows: In Section II, we present spectral clustering and analyze its memory bottlenecks when storing a dense similarity matrix. After discussing various approaches to handle this memory challenge, we present sparsification and Nyström approaches in Sections III and IV, respectively. In Section V, we present PSC, which considers sparsifying the matrix via retaining nearest neighbors. We conduct two experiments in Section VI. The first one compares the clustering quality between sparsification and the Nyström approximation. Results show that the former may yield slightly better clustering results. The second experiment investigates the speedup of our parallel implementation. PSC achieves good speedup on up to 256 machines. Section VII offers our concluding remarks.

II. Spectral Clustering

This section presents the spectral clustering algorithm, and describes the resource bottlenecks inherent in this approach. To assist readers, Table I defines terms and notation used throughout this paper.

A. Basic Concepts

Given n data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, the spectral clustering algorithm constructs a similarity matrix $S \in R^{n \times n}$, where $S_{ij} \geq 0$ reflects the relationship between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. It then uses similarity information to group $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ into k

TABLE I

NOTATION. THE FOLLOWING NOTATION IS USED IN THE PAPER.

n	number of data
d	dimensionality of data
k	number of desired clusters
p	number of nodes (distributed computers)
t	number of nearest neighbors
m	Arnoldi length in using an eigensolver
l	number of sample points in Nyström method
$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in R^d$	data points
$S \in R^{n \times n}$	similarity matrix
$L \in R^{n \times n}$	Laplacian matrix
$\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in R^n$	first k eigenvectors of L
$V \in R^{n \times k}$	eigenvector matrix
$\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k$ (in different length)	cluster indicator vectors
$E \in R^{n \times k}$	cluster indicator matrix
$\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k \in R^d$	cluster centers of k-means

clusters. There are several variants of spectral clustering. Here we consider the commonly used *normalized* spectral clustering [32]. (For a survey of all variants, please refer to [28].) An example similarity function is the Gaussian:

$$S_{ij} = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right), \tag{1}$$

where σ is a scaling parameter to control how rapidly the similarity S_{ij} reduces with the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Consider the normalized Laplacian matrix [7]:

$$L = I - D^{-1/2} S D^{-1/2}, \tag{2}$$

where D is a diagonal matrix with

$$D_{ii} = \sum_{j=1}^{n} S_{ij}.$$

Note that $D^{-1/2}$ indicates the inverse square root of D. It can be easily shown that for any S with $S_{ij} \geq 0$, the Laplacian matrix is symmetric positive semi-definite. In the ideal case, where data in one cluster are not related to those in others, nonzero elements of S (and hence L) only occur in a block diagonal form:

$$L = \begin{bmatrix} L_1 & & \\ & \ddots & \\ & & L_k \end{bmatrix}.$$

It is known that L has k zero-eigenvalues, which are also the k smallest ones [28, Proposition 4]. Their corresponding eigenvectors, written as an $R^{n \times k}$ matrix, are

$$V = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k] = D^{1/2} E, \tag{3}$$

where $\boldsymbol{v}_i \in R^n, i = 1, \ldots, k$, and

$$E = \begin{bmatrix} \boldsymbol{e}_1 & & \\ & \ddots & \\ & & \boldsymbol{e}_k \end{bmatrix},$$

where $\boldsymbol{e}_i, i = 1, \ldots, k$ (in different length) are vectors of all ones. As $D^{1/2}E$ has the same structure as E, simple clustering algorithms such as k-means can easily cluster the n rows of V into k groups. Thus, what one needs to do is to find the first k eigenvectors of L (i.e., eigenvectors corresponding to the k smallest eigenvalues). However, practically the eigenvectors we obtained are in the form of

$$V = D^{1/2}EQ,$$

where Q is an orthogonal matrix. Ng et al. [32] propose normalizing V so that

$$U_{ij} = \frac{V_{ij}}{\sqrt{\sum_{r=1}^{k} V_{ir}^2}}, i = 1, \ldots, n, j = 1, \ldots, k. \tag{4}$$

Each row of U has a unit length. Due to the orthogonality of Q, (4) is equivalent to

$$U = EQ = \begin{bmatrix} Q_{1,1:k} \\ \vdots \\ Q_{1,1:k} \\ Q_{2,1:k} \\ \vdots \end{bmatrix}, \tag{5}$$

where $Q_{i,1:k}$ indicates the i^{th} row of Q. Then U's n rows correspond to k orthogonal points on the unit sphere. The n rows of U can thus be easily clustered by k-means [32] or other simple techniques [50] [52].

Instead of analyzing properties of the Laplacian matrix, spectral clustering algorithms can also be derived from the graph cut point of view. That is, we partition the matrix according to the relationship between points. Some representative graph-cut methods are Normalized Cut [35], Min-Max Cut [12], and Ratio Cut [19].

B. Approximation of the Dense Similarity Matrix

A serious bottleneck for spectral clustering is the memory use for storing S, whose number of elements is the square of the number of data points. For instance, storing $n = 10^6$ data instances (assuming double precision storage) requires 8 TBytes of memory, which is not available on a general-purpose machine. Approximation techniques have been proposed to avoid storing the dense matrix. Figure 1 depicts several existing approximation techniques.

In this paper, we study two representative methods. First, the sparsification method retains the "most useful" S_{ij} to form a sparse matrix to reduce memory use. We discuss details in Section III. Second, the Nyström approximation stores only several columns (or rows) of the similarity matrix S. We present the details in Section IV. These two methods represent the two extremes of the choice spectrum depicted in Figure 1. On the one extreme, the sparsification method considers information in the data and hence requires expensive computation for gaining such

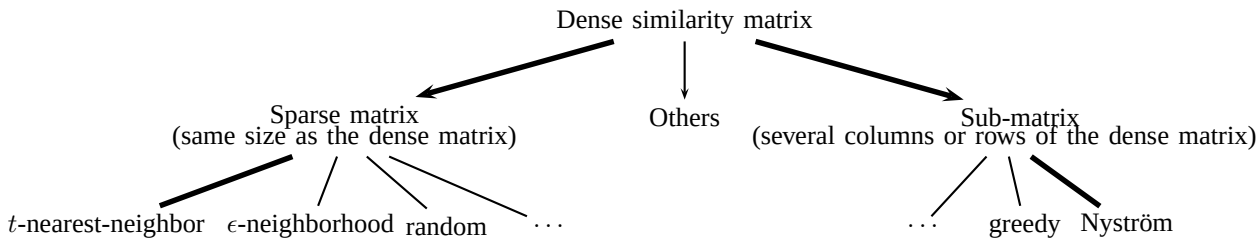


Fig. 1. This figure presents approximation techniques for spectral clustering to avoid storing the dense similarity matrix. In this paper, we mainly investigate two types of techniques: "Sparse matrix" is a technique to retain only certain nonzero S_{ij}'s. The resulting matrix is still a square one, but is sparse. "Sub-matrix" is a technique to use several columns (or rows) of S. That is, a rectangular portion of the original dense matrix is retained. Each type of techniques has several variants. We use thick lines to indicate approaches studied in this paper.

information; on the other extreme, the Nyström approximation considers no information in sampling columns/rows (random samples).

III. Spectral Clustering Using a Sparse Similarity Matrix

To avoid storing the dense similarity matrix, one could reduce the matrix S to a sparse one by considering only significant relationships between data instances. For example, we may retain only S_{ij} where j (or i) is among the t nearest neighbors of i (or j) [2] [28]. Typically t is a small number (e.g., a small fraction of n). We refer to this way as the t-nearest-neighbor approach. Another simple strategy to make S a sparse matrix is to zero out those S_{ij} smaller than a pre-specified threshold ϵ. It is often called the ϵ-neighborhood approach. While these techniques effectively conquer the memory difficulty, they still have to calculate all possible S_{ij}, and hence the computational time is high. Some approaches (e.g., [1]) thus consider zeroing out random entries in the similarity matrix. Though this simplification effectively saves time, experiments in Fowlkes et al. [13] reveal inferior performance. Such a result is predictable as it does not use significant relationships between data points. In this section, we focus on studying the method of using t nearest neighbors.

Algorithm 1 presents the spectral clustering using t-nearest-neighbor method for sparsification. In the rest of this section we examine its computational cost and the memory use. We omit discussing some inexpensive steps.

Construct the similarity matrix. To generate a sparse similarity matrix, we employ the t-nearest-neighbor approach and retain only S_{ij} where i (or j) is among the t nearest neighbors of j (or i). A typical implementation is as follows. By keeping a max heap with size t, we sequentially insert the distance that is smaller than the maximal value of the heap and then restructure the heap. Since restructuring a max heap is on the order of $\log t$, the complexity of generating a sparse matrix S is

$$O(n^2 d) + O(n^2 \log t) \text{ time and } O(nt) \text{ memory.} \tag{6}$$

The $O(n^2 d)$ cost can be reduced to a smaller value using techniques such as KD-trees [4] and Metric trees [42]. However, these techniques are less suitable if d, the dimensionality, is large. To further reduce the cost, one can

Algorithm 1 Spectral clustering using a sparse similarity matrix

Input: Data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$; k: number of desired clusters.

1) Construct similarity matrix $S \in R^{n \times n}$.
2) Modify S to be a sparse matrix.
3) Compute the Laplacian matrix L by Eq. (2).
4) Compute the first k eigenvectors of L; and construct $V \in R^{n \times k}$, whose columns are the k eigenvectors.
5) Compute the normalized matrix U of V by Eq. (4).
6) Use k-means algorithm to cluster n rows of U into k groups.

only find neighbors which are close but not the closest (approximate nearest neighbors). For example, it is possible that one only approximately finds the t nearest neighbors using techniques such as spill-tree [26]. The complexity, less than n^2, depends on the level of the approximation; see the discussion in Section VII. Nevertheless, studying approximate nearest-neighbor methods is beyond the scope of this study. We thus focus only on a precise method to find t nearest neighbors.

The above construction may lead to a non-symmetric matrix. We can easily make it symmetric. If either (i, j) or (j, i) element is nonzero, we set both positions to have the same value S_{ij}. Making the matrix symmetric leads to at most $2t$ nonzero elements per row. As $2t \ll n$, the symmetric matrix is still sparse.

Compute the first k eigenvectors by Lanczos/Arnoldi factorization. Once we have obtained a sparse similarity matrix S and its Laplacian matrix L by Eq. (2), we can use sparse eigensolvers. In particular, we desire a solver that can quickly obtain the first k eigenvectors of L. Some example solvers are SLEPc [21] and ARPACK [22] (see [20] for a comprehensive survey). Most existing approaches are variants of the Lanczos/Arnoldi factorization. These variants have similar time complexity. We employ a popular one called ARPACK and briefly describe its basic concepts hereafter; more details can be found in the user's guide of ARPACK. The m-step Arnoldi factorization gives that

$$L\bar{V} = \bar{V}H + (\text{a matrix of small values}), \tag{7}$$

where $\bar{V} \in R^{n \times m}$ and $H \in R^{m \times m}$ satisfy certain properties. If the "matrix of small values" in (7) is indeed zero, then $\bar{V}$'s m columns are L's first m eigenvectors (details not derived here). Therefore, (7) provides a way to check how well we approximate eigenvectors of L. To know how good the approximation is, one needs all eigenvalues of the dense matrix H, a procedure taking $O(m^3)$ operations. ARPACK employs an iterative procedure called "implicitly restarted" Arnoldi. Users specify an Arnoldi length $m > k$. Then at each iteration (restarted Arnoldi) one uses $\bar{V}$ and H from the previous iteration to conduct the eigendecomposition of H, and find a new Arnoldi factorization. An Arnoldi factorization at each iteration involves at most $(m - k)$ steps, where each step's main computational complexity is $O(nm)$ for a few dense matrix-vector products and $O(nt)$ for a sparse matrix-vector product. In particular, $O(nt)$ is for

$$L\boldsymbol{v}, \tag{8}$$

where $\boldsymbol{v}$ is an $n \times 1$ vector. As each row of L has no more than $2t$ nonzero elements, the cost of this sparse matrix-vector product is $O(nt)$.

After finishing the implicitly restarted Arnoldi procedure, from the final $\bar{V}$, we can obtain the required matrix V. Based on the above analysis, the overall cost of ARPACK is

$$\big(O(m^3) + (O(nm) + O(nt)) \times O(m-k)\big) \times (\text{\# restarted Arnoldi}), \tag{9}$$

where $O(m-k)$ is a value no more than $m-k$. Obviously, the selected value m affects the computational time. One often sets m to be several times larger than k. In a typical run, ARPACK may take a few dozens of restarted Arnoldi iterations. The memory requirement of ARPACK is $O(nt) + O(nm)$.

Use k-means to cluster the normalized matrix U. Let $\boldsymbol{u}_j$, $j = 1, \ldots, n$, be vectors corresponding to U's n rows. Algorithm k-means aims at minimizing the total intra-cluster variance, which is the squared error function in the spectral space:

$$\sum_{i=1}^{k} \sum_{\boldsymbol{u}_j \in C_i} ||\boldsymbol{u}_j - \boldsymbol{c}_i||^2. \tag{10}$$

We assume that data are in k clusters $C_i, i = 1, 2, \ldots, k$, and $\boldsymbol{c}_i \in R^k$ is the centroid of all the points $\boldsymbol{u}_j \in C_i$.

The k-means algorithm employs an iterative procedure. At each iteration, one finds each data point's nearest center and assigns it to the corresponding cluster. Cluster centers are then recalculated. The procedure stops after reaching a stable error function value. Since the algorithm evaluates the distances between any point and the current k cluster centers, the time complexity of k-means is

$$O(nk^2) \times (\text{\# of } k\text{-means}). \tag{11}$$

Note that each point or center here is a vector of length k. In this work, we terminate k-means iterations if the relative difference between the two error function values in consecutive iterations is less than 0.001.

Overall analysis. If one uses a direct method of finding the t nearest neighbors, from (6), (9), and (11), the $O(n^2 d) + O(n^2 \log t)$ computational time is the main bottleneck for spectral clustering. This bottleneck has been discussed in earlier work. For example, Liu et al. [25] state that "The majority of the time is actually spent on constructing the pairwise distance and affinity matrices. Comparatively, the actually clustering is almost *negligible*." A summary of the computational complexity of t-nearest-neighbor is listed on the last column of Table II.

IV. Spectral Clustering Using Nyström Approximation

In Section III we introduced approximate spectral clustering based on sparse similarity values (t-nearest-neighbor). In this section, we describe another approximation technique, the Nyström method, which uses a sub-matrix of the dense similarity matrix.

TABLE II

ANALYSIS OF THE TIME COMPLEXITY FOR THE METHODS DESCRIBED IN SECTIONS III AND IV.

Algorithms	Nyström using (16)	Nyström using (19)	t-nearest-neighbor sparse matrix
Obtaining the similarity matrix	$O(nld)$	$O(nld)$	$O(n^2d + n^2 \log t)$
Finding first k eigenvectors	$O(l^3) + O(nlk)$	$O((n-l)l^2) + O(l^3)$ $+ O(nlk)$	$(O(m^3) + (O(nm) + O(nt)) \times O(m-k))$ $\times$(# restarted Arnoldi)

A. Nyström Method

The Nyström method is a technique for finding an approximate eigendecomposition. Williams [44] applied it to speed up the kernel machines, and this method has been widely applied to areas involving large dense matrices. Some existing work includes Fowlkes et al. [13] for image segmentation and Talwalkar et al. [39] for manifold learning. In this section, we discuss how to apply the Nyström method to general spectral clustering.

Here we denote by S_d a dense $n \times n$ similarity matrix. Assume that we randomly sample $l \ll n$ points from the data*. Let A represent the $l \times l$ matrix of similarities between the sample points, B be the $l \times (n-l)$ matrix of affinities between the l sample points and the $(n-l)$ remaining points, and W be the $n \times l$ matrix consisting of A and B^T. We can rearrange the columns and rows of S_d based on this sampling such that:

$$S_d = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}, \text{ and } W = \begin{bmatrix} A \\ B^T \end{bmatrix} \tag{12}$$

with $A \in R^{l \times l}, B \in R^{l \times (n-l)}$, and $C \in R^{(n-l) \times (n-l)}$. Here C contains the similarities between all $(n-l)$ remaining points.

The Nyström method uses A and B to approximate S_d. Using W of (12), the approximation (denoted by $\tilde{S}$) takes the form:

$$S_d \approx \tilde{S} = WA^{-1}W^T = \begin{bmatrix} A & B \\ B^T & B^TA^{-1}B \end{bmatrix}. \tag{13}$$

That is, the matrix C in S_d is now replaced by $B^TA^{-1}B$. Assume the eigendecomposition of A takes the form $A = V_A \Sigma_A V_A^T$, where Σ_A contains the eigenvalues of A and V_A are the corresponding eigenvectors. The approximate eigenvalues ($\tilde{\Sigma}$) and eigenvectors ($\tilde{V}$) of S_d generated from the Nyström method are:

$$\tilde{\Sigma} = \left(\frac{n}{l}\right)\Sigma_A, \quad \tilde{V} = \sqrt{\frac{l}{n}} W V_A \Sigma_A^{-1}. \tag{14}$$

Moreover, (14) implies that $\tilde{S}$ has the eigendecomposition:

$$\tilde{S} = \tilde{V}\tilde{\Sigma}\tilde{V}^T.$$

*One may use other ways to select samples. See, for example, Figure 1 references a "greedy" method discussed in [33].

B. Application to Spectral Clustering

To apply the Nyström method to spectral clustering, we select $l > k$. The normalized Laplacian matrix for the Nyström method takes the form:

$$\tilde{L} = I - D^{-1/2}\tilde{S}D^{-1/2},$$

where D is a diagonal matrix with

$$D_{ii} = \sum_{j=1}^{n} \tilde{S}_{ij}.$$

Following the procedure in Section II-A, we then find the first k eigenvectors of $\tilde{L}$ and conduct k-means to cluster data. To obtain the matrix D, we compute the row sums of $\tilde{S}$. A direct calculation of $\tilde{S}$ involves the matrix-matrix product $B^T A^{-1} B$, which is an expensive $O(l(n-l)^2)$ operation. Fowlkes et al. [13] thus propose the following procedure:

$$\tilde{S}\mathbf{1} = \begin{bmatrix} A\mathbf{1}_l + B\mathbf{1}_{n-l} \\ B^T\mathbf{1}_l + B^T A^{-1} B\mathbf{1}_{n-l} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a} + \boldsymbol{b}_1 \\ \boldsymbol{b}_2 + B^T\left(A^{-1}\boldsymbol{b}_1\right) \end{bmatrix}, \tag{15}$$

where $\boldsymbol{a}$, $\boldsymbol{b}_1$, $\boldsymbol{b}_2$ represent the row sums of A, B and B^T, respectively, and $\mathbf{1}$ is a column vector of ones. Using (15), the cost of obtaining D is just $O(l(n-l))$. Then $D^{-1/2}\tilde{S}D^{-1/2}$ can be represented in a different form:

$$D^{-1/2}\tilde{S}D^{-1/2} = \begin{bmatrix} \bar{A} & \bar{B} \\ \bar{B}^T & \bar{B}^T\bar{A}^{-1}\bar{B} \end{bmatrix},$$

where

$$\bar{A} = D^{-1/2}_{1:l,1:l} A D^{-1/2}_{1:l,1:l}, \quad \bar{B} = D^{-1/2}_{1:l,1:l} B D^{-1/2}_{l+1:n,l+1:n}.$$

As $D^{-1/2}\tilde{S}D^{-1/2}$ and $\tilde{L}$ share the same eigenspace, following (14) we approximately obtain $\tilde{L}$'s first k eigenvectors via the eigendecomposition $\bar{A} = \bar{V}_A \bar{\Sigma}_A \bar{V}_A^T$ and then

$$\tilde{\Sigma} = \left(\frac{n}{l}\right)\left(\bar{\Sigma}_A\right)_{:,1:k}, \quad \tilde{V} = \sqrt{\frac{l}{n}}\begin{bmatrix} \bar{A} \\ \bar{B}^T \end{bmatrix}\left(\bar{V}_A\right)_{:,1:k}\left(\bar{\Sigma}_A^{-1}\right)_{1:k,1:k}, \tag{16}$$

where we assume that eigenvalues in $\bar{\Sigma}_A$ are arranged in descending order.

However, we explain below a concern that columns of $\tilde{V}$ are not orthogonal. For kernel methods and some other applications of the Nyström method, this concern may not be a problem. For spectral clustering, in (3) of Section II-A, V's columns are orthogonal and then in the ideal situation, the normalized matrix U has rows corresponding to k orthogonal points on the unit sphere. Fowlkes et al. [13] proposed an approach to obtain orthogonal columns of $\tilde{V}$[†]. Let

$$R = \bar{A} + \bar{A}^{-1/2}\bar{B}\bar{B}^T\bar{A}^{-1/2} \tag{17}$$

[†]Instead one may use a method called "column sampling." See the discussion in [39].

Algorithm 2 Spectral clustering using the Nyström method

Input: Data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$; l: number of samples; k: number of desired clusters; $l > k$.

1) Construct $A \in R^{l \times l}$ and $B \in R^{l \times (n-l)}$ so that $\begin{bmatrix} A & B \end{bmatrix}$ contains the similarity between $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.
2) Calculate $\boldsymbol{a} = A\mathbf{1}_l$, $\boldsymbol{b}_1 = B\mathbf{1}_{n-l}$, $\boldsymbol{b}_2 = B^T\mathbf{1}_l$ and $D = \text{diag}\left(\begin{bmatrix} \boldsymbol{a} + \boldsymbol{b}_1 \\ \boldsymbol{b}_2 + B^T A^{-1}\boldsymbol{b}_1 \end{bmatrix}\right)$. Here $\mathbf{1}$ represents a column vector of ones.
3) Calculate $\bar{A} = D_{1:l,1:l}^{-1/2} A D_{1:l,1:l}^{-1/2}$, $\bar{B} = D_{1:l,1:l}^{-1/2} B D_{l+1:n,l+1:n}^{-1/2}$.
4) Construct $R = \bar{A} + \bar{A}^{-1/2}\bar{B}\bar{B}^T\bar{A}^{-1/2}$.
5) Calculate eigendecomposition of R, $R = U_R \Lambda_R U_R^T$. Ensure that the eigenvalues in Λ_R are in decreasing order.
6) Calculate $\tilde{V} = \begin{bmatrix} \bar{A} \\ \bar{B}^T \end{bmatrix} \bar{A}^{-1/2}(U_R)_{:,1:k}(\Lambda_R^{-1/2})_{1:k,1:k}$ as the first k eigenvector of $\tilde{L}$.
7) Compute the normalized matrix $\tilde{U}$ of $\tilde{V}$ by Eq. (4).
8) Use k-means algorithm to cluster n rows of $\tilde{U}$ into k groups.

and its eigendecomposition $R = U_R \Lambda_R U_R^T$. It is proved in [13] that if $\bar{A}$ is positive definite[‡], then

$$\tilde{V} = \begin{bmatrix} \bar{A} \\ \bar{B}^T \end{bmatrix} \bar{A}^{-1/2} U_R \Lambda_R^{-1/2} \tag{18}$$

has orthogonal columns (i.e. $\tilde{V}^T\tilde{V} = I$) and $D^{-1/2}\tilde{S}D^{-1/2} = \tilde{V}\Lambda_R\tilde{V}^T$. Since we require only the first k eigenvectors of $\tilde{L}$, similar to (16), we calculate the first k columns of $\tilde{V}$ via

$$\tilde{V} = \begin{bmatrix} \bar{A} \\ \bar{B}^T \end{bmatrix} \bar{A}^{-1/2}(U_R)_{:,1:k}(\Lambda_R^{-1/2})_{1:k,1:k}. \tag{19}$$

We then normalize $\tilde{V}$ along its rows to get $\tilde{U}$, and use k-means to cluster $\tilde{U}$'s n rows into k groups. A summary of the Nyström method using (19) is presented in Algorithm 2.

C. Computational Complexity and Memory Use

Under the same assumption that calculating each S_{ij} costs $O(d)$, where d is the dimension of data, obtaining matrices A and B costs $O(nld)$. If using (16), the main cost includes $O(l^3)$ operations for the eigendecomposition of $\bar{A}$ and $O(nlk)$ operations for the matrix-matrix product. The first two columns of Table II list the computational complexity of the Nyström method. Regarding the memory use, as l columns of the similarity matrix are constructed, the Nyström method needs $O(nl)$ space.

If using (19), in constructing R, the matrix-matrix product costs $O((n-l)l^2)$. The eigendecomposition of a dense R requires $O(l^3)$. Finally, calculating $\tilde{V}$ via (19) takes $O(nlk)$. As generally $l \ll n$, the $O((n-l)l^2)$ cost is the

[‡]For similarity functions such as the Guassian in (1), if $\boldsymbol{x}_i \neq \boldsymbol{x}_j, \forall i \neq j$, then S is positive definite and so is $\bar{A}$.

dominant term. Moreover, as this term is not needed when using (16), maintaining orthogonal $\tilde{V}$ via (19) is more expensive. Note that in Step 2 of Algorithm 2, one should calculate the diagonal matrix D by

$$B^T(A^{-1}\boldsymbol{b}_1) \text{ instead of } (B^TA^{-1})\boldsymbol{b}_1$$

to avoid a possible $O((n-l)l^2)$ operation.

D. A Comparison Between Two Approximation Methods

We have investigated how spectral clustering is performed with both sparse-matrix and Nyström approximations. Here we discuss their differences and explain why we choose to parallelize the approach of using sparse similarity matrices (t-nearest-neighbor).

From the clustering quality perspective, we suggest that the sparse-matrix approximation may give slightly better results than the Nyström approximation. For the t-nearest-neighbor approach, small similarity values are discarded, so insignificant relationships between data points are ignored. Therefore, using a sparse matrix does not lose much information and may avoid retaining some noisy/inaccurate similarity values. In contrast, for the Nyström method, there is no evidence indicating that approximation may provide higher quality results than using the fully dense matrix. In fact, some studies [33], [43] show that approximation may yield inferior results. Their evaluations apply the Nyström approximation to topics other than spectral clustering. Thus, we conduct in Section VI-A a systematic comparison on clustering performance using real-world data. Results show that in general using the t-nearest-neighbor sparse matrix is either as good as or slightly better than the Nyström approximation.

From the computation time perspective, the main disadvantage of the t-nearest-neighbor approximation is calculating the whole similarity matrix, which is $O(n^2d)$ in complexity (higher than $O(nld)$ of Nyström). Thus Nyström seems to be an attractive approach for large-scale problems. However, the more expensive computation of having a sparse similarity matrix might be worth attempting because:

- The calculation of the similarity matrix can be easily parallelized without any communication cost. See details in Section V-B.
- It is possible to reduce the $O(n^2d)$ cost by approximately finding t nearest neighbors.
- In finding the first k eigenvectors of L, Nyström may not be efficient. For the sparse-matrix approach, we often set $m = 2k$ for the Arnoldi length and invoke a sparse eigensolver. In contrast, for Nyström, we use a dense eigensolver and may need $l \gg k$.

V. Parallel Spectral Clustering (PSC) Using a Sparse Similarity Matrix

We now present PSC using t-nearest-neighbor sparse similarity matrices. We discuss challenges and then depict our solutions. We have both Message Passing Interface (MPI) [37] and MapReduce [8] systems on our distributed environments. Little has been discussed in this community about when to use which. We illustrate their differences and present our implementation of the parallel spectral clustering algorithm.

TABLE III

SAMPLE MPI FUNCTIONS [37].

MPI_Bcast:	Broadcasts information to all processes.
MPI_AllGather:	Gathers the data contributed by each process on all processes.
MPI_Reduce:	Performs a global reduction (e.g., sum) and returns the result to the specified root.
MPI_AllReduce:	Performs a global reduction and returns the result on all processes.

A. MPI and MapReduce

MPI is a message passing library interface specification for parallel programming [37]. An MPI program is loaded into the local memory of each machine, where every processor/process (each processor will be assigned just one process for maximum performance) has a unique ID. MPI allows the same program to be executed on multiple data. When needed, the processes can communicate and synchronize with others by calling MPI library routines. Examples of MPI functions are shown in Table III.

Different from MPI, MapReduce is a Google parallel computing framework [8]. It is based on user-specified map and reduce functions. A map function generates a set of intermediate key/value pairs. In the reduce phase, intermediate values with the same key are passed to the reduce function. As an abstract programming model, different implementations of MapReduce are possible depending on the architecture (shared or distributed environments). The one considered here is the implementation used in Google distributed clusters. For both map and reduce phases, the program reads and writes results to disks. With the disk I/O, MapReduce provides a fault tolerant mechanism. That is, if one node fails, MapReduce restarts the task on another node. This framework allows people with no experience in parallel computing to use large distributed systems. In contrast, MPI is more complicated due to its various communication functions. Instead of using disk I/O, a function sends and receives data to and from a node's memory. MPI is commonly used for iterative algorithms in many numerical packages. Though MPI itself does not perform logging for recovery, an application can employ check-points to achieve fault tolerance. In general, MapReduce is suitable for non-iterative algorithms where nodes require little data exchange to proceed (*non-iterative* and *independent*); MPI is appropriate for iterative algorithms where nodes require data exchange to proceed (*iterative* and *dependent*).

In Algorithm 1, constructing the sparse similarity matrix is a non-iterative and independent procedure, thus we consider MapReduce. As this step may be the most time consuming, having a fault tolerant mechanism is essential. To find the first k eigenvectors, we consider PARPACK [30], a parallel ARPACK implementation based on MPI as eigensolvers are iterative and dependent procedures. For k-means, we consider MPI as well.

We give some implementation details. To ensure fast file I/O, we use Google file system (GFS) [14] and store data in the SSTable file format [5]. In contrast to traditional file I/O, where we sequentially read data from the beginning of the file, using SSTable allows us to easily access any data point. This property is useful in calculating the similarity matrix; see the discussion in Section V-B. Regarding MPI implementations, standard tools such

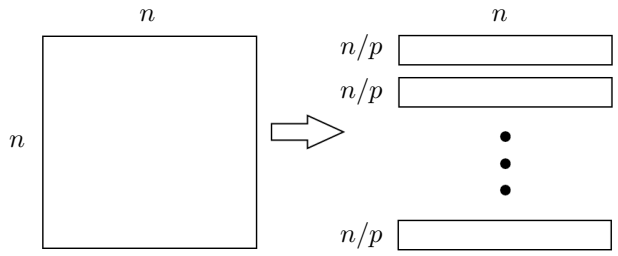


Fig. 2. The similarity matrix is distributedly computed and stored on multiple machines.

as MPICH2[§] [17] cannot be directly ported to our distributed system because the Google system uses its own remote procedure call (RPC) protocol and system configurations. We modify the underlying communication layer of MPICH2 to work in our system.

B. Similarity Matrix and Nearest Neighbors

To construct the sparse similarity matrix using t nearest neighbors, we perform three steps. First, for each data point, we compute distances to all data points, and find its t nearest neighbors. Second, we modify the sparse matrix obtained from the first step to be symmetric. Finally, we compute the similarities using distances. These three steps are implemented using MapReduce, as described below.

Compute distances and find nearest neighbors. In this step, for each data point, we compute the distances (Euclidean or cosine distances) to all data points and find the t nearest neighbors. Suppose p nodes are allocated in a distributed environment. Figure 2 shows that we construct n/p rows of the distance matrix at each node. To handle very large data sets, we need to carefully consider the memory usage in calculating the distances. In our implementation, we do not assume that all data instances can be loaded into the memory of each single node in the distributed environment. However, we require that each node can store n/p instances. This can be easily achieved by increasing p, the number of nodes.

The map phase creates intermediate keys/values so that we make every n/p data points have the same key. In the reduce phase, these n/p points are loaded to the memory of a node. We refer to them as the local data. We then scan the whole data set: given an $\boldsymbol{x}_i$, we calculate $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ for all $\boldsymbol{x}_j$ of the n/p local points. We use n/p max heaps so each maintains a local data point's t nearest neighbors so far. If the Euclidean distance is considered, then

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 = \|\boldsymbol{x}_i\|^2 + \|\boldsymbol{x}_j\|^2 - 2\boldsymbol{x}_i^T\boldsymbol{x}_j.$$

We precompute all $\|\boldsymbol{x}_j\|^2$ of local data to conserve time. The use of SSTable allows us to easily access arbitrary data points in the file. Otherwise, in reading the n/p local points, we must scan the input file to find them. On each node, we store n/p sparse rows in the compressed row format.

[§]`http://www.mcs.anl.gov/research/projects/mpich2`

Modify the distance matrix to be symmetric. The sparse distance matrix computed from the first step is not symmetric. Note that $\boldsymbol{x}_j$ being in the t-nearest-neighbor set of $\boldsymbol{x}_i$ does not imply that $\boldsymbol{x}_i$ is in the t-nearest-neighbor set of $\boldsymbol{x}_j$. In this step, if either (i, j) or (j, i) element of the t-nearest-neighbor distance matrix contains the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, we set both positions to have the same value.

In the map phase, for each nonzero element in the sparse distance matrix, we generate two key/value pairs. The first key is the row ID of the element, and the corresponding value is the column ID and the distance. The second key is the column ID, and the corresponding value is the row ID and the distance. When the reduce function is called, elements with the same key correspond to values in the same row of the desired symmetric matrix. These elements are then collected. However, duplicate elements may occur, so we keep a hash map to do an efficient search and deletion. Each row contains no more than $2t$ nonzero elements after symmetrization. As $t \ll n$, the resulting symmetric matrix is still sparse.

Compute similarities. Although we can easily compute the similarities in the previous step, we use a separate MapReduce step to selftune the parameter σ in (1) [51]. We consider the following similarity function:

$$S_{ij} = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma_i\sigma_j}\right). \tag{20}$$

Suppose $\boldsymbol{x}_i$ has t nearest neighbors. We can define σ_i as the average of t distance values. Alternatively, we can use the median value of each row of the sparse similarity matrix. That is, $\sigma_i = ||\boldsymbol{x}_i - \boldsymbol{x}_{i_t}||$, where $\boldsymbol{x}_{i_t}$ is the $\lfloor t/2 \rfloor$th neighbor of $\boldsymbol{x}_i$ by sorting distances to $\boldsymbol{x}_i$'s neighbors[¶]. For the implementation, in the map phase, we calculate the average distance or the median value of each row of the distance matrix. Each reduce function obtains a row and all parameters. The similarity values are then calculated by (20).

C. Parallel Eigensolver

After we have obtained the sparse similarity matrix, it is important to use a parallel eigensolver. Several works have studied parallel eigendecomposition [21] [29] [30] [45] [46]. We consider PARPACK [30], a parallel ARPACK implementation based on MPI. We let each MPI node store n/p rows of the matrix L as depicted in Figure 2. For the eigenvector matrix $\bar{V}$ (see (7)) generated during the call to ARPACK, we also split it into p partitions, each of which possesses n/p rows. Note that if k (and m) is large, then $\bar{V}$, an $R^{n\times m}$ dense matrix, may consume more storage space than the similarity matrix. Hence $\bar{V}$ should be distributedly stored on different nodes. As mentioned in Section III, major operations at each step of the Arnoldi factorization include a sparse and a few dense matrix-vector multiplies, which cost $O(nt)$ and $O(nm)$, respectively. We parallelize these computations so that the complexity of finding eigenvectors becomes:

$$\left(O(m^3) + (O(\frac{nm}{p}) + O(\frac{nt}{p})) \times O(m-k)\right) \times (\#\text{ restarted Arnoldi}). \tag{21}$$

The communication overhead between nodes occurs in the following three situations:

[¶]In the experiments, we use the average distance as our self-tuning parameter.

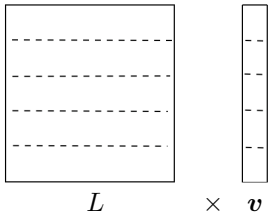


Fig. 3. Sparse matrix-vector product. We assume $p = 5$ here. L and $\boldsymbol{v}$ are respectively separated to five block partitions.

1) Sum p values and broadcast the result to p nodes. Details of these p values are not discussed here.
2) Parallel sparse matrix-vector product (8).
3) Parallel dense matrix-vector product: Sum p vectors of length m and broadcast the resulting vector to all p nodes.

The first and the third cases transfer only short vectors, but the sparse matrix vector product may move a larger vector $\boldsymbol{v} \in R^n$ to several nodes. Due to this high communication cost, we next discuss the parallel sparse matrix-vector product in detail.

Figure 3 shows matrix L and vector $\boldsymbol{v}$. Suppose $p = 5$. The figure indicates that both L and $\boldsymbol{v}$ are horizontally split into five parts and each part is stored on one computer node. Take node 1 as an example. It is responsible for performing

$$L_{1:n/p,1:n} \times \boldsymbol{v}, \tag{22}$$

where $\boldsymbol{v} = [v_1, \ldots, v_n]^T \in R^n$. $L_{1:n/p,1:n}$, the first n/p rows of L, is stored on node 1, but only $v_1, \ldots, v_{n/p}$ are available there. Hence other nodes must send to node 1 the elements $v_{n/p+1}, \ldots, v_n$. Similarly, node 1 should dispatch its $v_1, \ldots, v_{n/p}$ to other nodes. This task is a gather operation in MPI (*MPI_AllGather*, see Table III): data points on each node are gathered on all nodes. Note that one often assumes the following cost model for transferring some data between two nodes [3]:

$$\alpha + \beta \cdot (\text{length of data transferred}),$$

where α, the start-up time of a transfer, is a constant independent of the message size. The value β is the transfer time per unit of data. Depending on α, β of the distributed environment and the size of data, one can select a suitable algorithm for implementing the *MPI_AllGather* function. After some experiments, we consider the *recursive doubling* algorithm [40]. The total communication cost to gather $\boldsymbol{v}$ on all nodes is

$$O\left(\alpha \cdot \log(p) + \beta \cdot \frac{p-1}{p} n\right), \tag{23}$$

where n is the length of the vector $\boldsymbol{v}$. For this implementation, the number of machines must be a power of two. Among various approaches discussed in [40] for the gather operation, (23) has the smallest coefficient related to α. On our distributed environment (cheap PCs in a data center), the initial cost of any point-to-point communication is expensive, so (23) is a reasonable choice.

Further reducing the communication cost is possible if we take the sparsity of L into consideration. The reduction of the communication cost depends on the sparsity and the structure of the matrix. We defer this optimization to future investigation.

D. Parallel k-means

Once the eigensolver computes the first k eigenvectors of the Laplacian matrix, the matrix V is distributedly stored. Thus the normalized matrix U can be computed in parallel and stored on p local machines. Each row of the matrix U is regarded as one data point in the k-means algorithm. We implement the k-means algorithm using MPI. Several prior works have studied parallel k-means. For example, [6] [11] [18] [48].

To start the k-means procedure, the master machine chooses a set of initial cluster centers and broadcasts them to all machines. Revisit Eq. (5). In the ideal case, the centers of data instances calculated based on the matrix U are orthogonal to each other. Thus, an intuitive initialization of centers can be done by selecting a subset of U's n rows whose elements are almost orthogonal [50]. To begin, we use the master machine to randomly choose a point as the first cluster center. Then it broadcasts the center to all machines. Each machine identifies the most orthogonal point to this center by finding the minimal cosine similarity between its points and the center. The cosine similarity is defined as the inner product between two points. By collecting the p minimal cosine similarities, we choose the most orthogonal point to the first center as the second center. This procedure is repeated to obtain k centers.

Once the initial centers are calculated, new labels of each node's local data are assigned to clusters and local sums of clusters are calculated without any inter-machine communication. The master machine then obtains the sum of all points in each cluster to calculate new centers, and broadcasts them to all the machines. Most of the communication occurs here, and this is a reduction operation in MPI (***MPI_AllReduce***, see Table III). The loss function (10) can also be computed in parallel in a similar way. Therefore, the computation time for parallel k-means is reduced to $1/p$ of that in (11). Regarding the communication, as local sums on each node are k vectors of length k, the communication cost per k-means iteration is in the order of k^2. Note that the ***MPI_AllReduce*** function used here has a similar cost to the ***MPI_AllGather*** function discussed earlier. We explain below that the communication bottleneck tends to occur in the sparse matrix-vector product (8) of eigendecomposition instead of the k-means algorithm here. For each sparse matrix-vector product, we gather $O(n)$ values after $O(nt/p)$ operations on each node. From Table II, there are $O(m-k) \times$ (# restarted Arnoldi) sparse matrix-vector products. For k-means, in each iteration we transfer $O(k^2)$ values after $O(nk^2/p)$ operations in calculating the distance between n/p points and k cluster centers. Assume there are (# k-means) iterations. In a typical run, (# restarted Arnoldi) is of the same scale as (# k-means), so the total number of sparse matrix-vector products is bigger than the number of k-means iterations. If $n \geq k^2$, then the communication overhead in eigendecomposition is more serious than k-means. We will clearly observe this result in Section VI-B.

We summarize the computational time complexity of each step of the spectral clustering algorithm before and after parallelization in Table IV.

TABLE IV

ANALYSIS OF THE COMPUTATIONAL TIME COMPLEXITY OF EACH STEP OF THE SPECTRAL CLUSTERING ALGORITHM BEFORE AND AFTER PARALLELIZATION. NOTE THAT COMMUNICATION TIME IS EXCLUDED.

Spectral clustering	Before parallelization	After parallelization
Obtaining the similarity matrix	$O(n^2d + n^2 \log t)$	$O(\frac{n^2 d}{p} + \frac{n^2 \log t}{p})$
Finding first k eigenvectors	$(O(m^3) + (O(nm) + O(nt)) \times O(m-k))$ $\times$(# restarted Arnoldi)	$(O(m^3) + (O(\frac{nm}{p}) + O(\frac{nt}{p})) \times O(m-k))$ $\times$(# restarted Arnoldi)
Performing k-means	$O(nk^2) \times$ (# k-means)	$O(\frac{nk^2}{p}) \times$ (# k-means)

VI. EXPERIMENTS

We designed experiments to evaluate spectral clustering algorithms and investigated the performance of our parallel implementation. Our experiments used three data sets: 1) Corel, a selected collection of $2,074$ images, 2) RCV1 (Reuters Corpus Volume I), a filtered collection of $193,844$ documents, and 3) $2,121,863$ photos collected from PicasaWeb, a Google photo sharing product.

A. Clustering Quality

To justify our decision to sparsify the similarity matrix using t nearest neighbors, we compare it with the Nyström method. As a side comparison, we also report the performance of traditional k-means. We use MATLAB to conduct this experiment, while the parallel implementation (in C++) is used in Section VI-B. The MATLAB code is available at

`http://alumni.cs.ucsb.edu/~wychen/sc.html`

1) Data Sets: We used two data sets with ground truth for measuring clustering quality.

Corel. It is an image data set which has been widely used by the computer vision and image-processing communities. The images within each category were selected based on similar colors and objects. We chose $2,074$ images from the Corel Image CDs to create 18 categories. For each image, we extracted 144 features including color, texture, and shape as the image's representation [24]. In the color channel, we divided color into 12 color bins including 11 bins for culture colors and one bin for outliers [41]. For each color bin, we recorded nine features to capture color information at finer resolution. The nine features are color histogram, color means (in H, S, and V channels), color variances (in H, S, and V channels), and two shape characteristics: elongation and spreadness. Color elongation defines the shape of color, and spreadness defines how the color scatters within the image. In the texture channel, we employed a discrete wavelet transformation (DWT) using quadrature mirror filters [36] due to its computational efficiency. Each DWT on an image yielded four subimages including the scale-down image and its wavelets in three orientations. We then obtained nine texture combinations from subimages for three scales (coarse, medium, fine) and three orientations (horizontal, vertical, diagonal). For each texture, we recorded four features: energy mean,

energy variance, texture elongation and texture spreadness. Finally, we performed feature scaling so features are on the same scale. In conducting spectral clustering, the similarity functions (1) and (20) are considered according to whether one assigns a fixed σ to all data or not.

RCV1. It is an archive of $804,414$ manually categorized newswire stories from Reuters Ltd [23]. The news documents are categorized with respect to three controlled vocabularies: *industries*, *topics* and *regions*. Data were split into $23,149$ training documents and $781,256$ test documents. In this experiment, we used the test set and category codes based on the *industries* vocabulary. There are originally 350 categories in the test set. For comparing clustering results, data which are multi-labeled were not considered, and categories which contain less than 500 documents were removed. We obtained $193,844$ documents in 103 categories. Each document is represented by a cosine normalization of a $\log$ transformed TF-IDF (term frequency, inverse document frequency) feature vector.

2) Quality Measure: We used the *image categories* in the Corel data set and the *document categories* in the RCV1 data set as the ground truths for evaluating cluster quality. We measured the quality via the Normalized Mutual Information (NMI) and Clustering Accuracy between the produced clusters and the ground-truth categories.

NMI. The normalized mutual information between two random variables CAT (category label) and CLS (cluster label) is defined as:

$$\text{NMI(CAT; CLS)} = \frac{\text{I(CAT; CLS)}}{\sqrt{\text{H(CAT)H(CLS)}}}, \tag{24}$$

where I(CAT; CLS) is the mutual information between CAT and CLS. The entropies H(CAT) and H(CLS) are used for normalizing the mutual information to be in the range of $[0,1]$. In practice, we made use of the following formulation to estimate the NMI score [38]:

$$\text{NMI} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k} n_{i,j}\log\left(\frac{n\cdot n_{i,j}}{n_i\cdot n_j}\right)}{\sqrt{\left(\sum_i n_i \log\frac{n_i}{n}\right)\left(\sum_j n_j \log\frac{n_j}{n}\right)}}, \tag{25}$$

where n is the number of images/documents, n_i and n_j denote the number of images/documents in category i and cluster j, respectively, and $n_{i,j}$ denotes the number of images/documents in category i as well as in cluster j. The NMI score is 1 if the clustering results perfectly match the category labels, and the score is close to 0 if data are randomly partitioned [54]. The higher the NMI score, the better the clustering quality.

Clustering Accuracy. Following [47], this measure to evaluate the cluster quality is defined as:

$$\text{Accuracy} = \frac{\sum_{i=1}^{n}\delta(y_i, \text{map}(c_i))}{n}, \tag{26}$$

where n is the number of images/documents, y_i and c_i denote the true category label and the obtained cluster label of image/document $\boldsymbol{x}_i$, respectively. $\delta(y,c)$ is a function that equals 1 if $y=c$ and equals 0 otherwise. map$(\cdot)$ is a permutation function that maps each cluster label to a category label, and the optimal matching can be found by the Hungarian algorithm [34].

3) Results: We compared five different clustering algorithms, including

- k-means algorithm based on Euclidean distance (E-k-means).

TABLE V

NMI COMPARISONS OF FIVE ALGORITHMS. DETAILED SETTINGS ARE DESCRIBED IN SECTION VI-A.

Algorithms	Corel	RCV1
E-k-means	$0.3689(\pm 0.0122)$	$0.2737(\pm 0.0063)$
Nyström without orthogonalization	$0.3496(\pm 0.0140)$	$0.2567(\pm 0.0052)$
Nyström with orthogonalization	$0.3623(\pm 0.0084)$	$0.2558(\pm 0.0031)$
Fixed-σ SC	$0.3811(\pm 0.0050)$	$0.2861(\pm 0.0010)$
Selftune SC	$0.3836(\pm 0.0026)$	$0.2865(\pm 0.0013)$

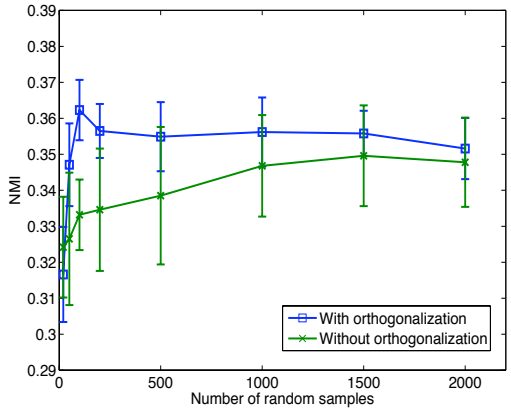
TABLE VI

CLUSTERING ACCURACY COMPARISONS OF FIVE ALGORITHMS. DETAILED SETTINGS ARE DESCRIBED IN SECTION VI-A.

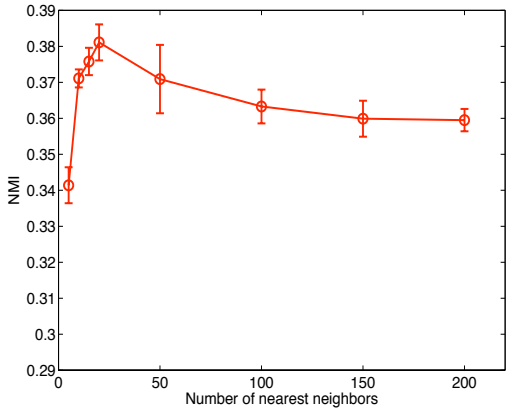
Algorithms	Corel	RCV1
E-k-means	$0.3587(\pm 0.0253)$	$0.1659(\pm 0.0062)$
Nyström without orthogonalization	$0.3622(\pm 0.0186)$	$0.1778(\pm 0.0080)$
Nyström with orthogonalization	$0.3730(\pm 0.0087)$	$0.1831(\pm 0.0051)$
Fixed-σ SC	$0.3826(\pm 0.0086)$	$0.1855(\pm 0.0025)$
Selftune SC	$0.3851(\pm 0.0164)$	$0.1842(\pm 0.0021)$

- spectral clustering using the Nyström method. We apply (16) to obtain the non-orthogonal eigenvectors (Nyström without orthogonalization).
- spectral clustering using the Nyström method. We apply (19) to have orthogonal columns of $\tilde{V}$ (Nyström with orthogonalization).
- spectral clustering using t-nearest-neighbor similarity matrices. The similarity function (1) is used with a given parameter σ (Fixed-σ SC).
- spectral clustering using t-nearest-neighbor similarity matrices. We apply a selftune technique (described in Section V-B) on (20) to adaptively assign σ (Selftune SC).

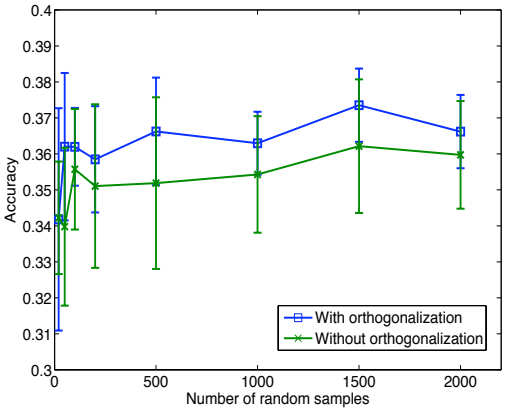
All the above algorithms involve k-means procedures, for which we use the orthogonal initialization discussed in Section V-D. We set the number of clusters to be 18 for the Corel data set and 103 for the RCV1 data set. We reported the best clustering performance of Nyström (without/with orthogonalization) by searching a set of numbers of random samples $(20, \ldots, 2000$ for Corel, and $200, \ldots, 3500$ for RCV1) and a grid of σ values $(10, \ldots, 40$ for Corel, and $0.5, \ldots, 2.5$ for RCV1). Similarly, we reported the best clustering performance of SC (Fixed-σ and Selftune) by searching a set of numbers of nearest neighbors $(5, \ldots, 200$ for Corel, and $20, \ldots, 200$ for RCV1) for both, and a grid of σ values $(10, \ldots, 40$ for Corel, and $0.5, \ldots, 2.5$ for RCV1) for Fixed-σ SC. For Fixed-σ SC and Selftune SC, the Arnoldi space dimension m was set to be two times the number of clusters for each data set. Later in this paper we discuss the sensitivity of using various values of t, the number of nearest neighbors.



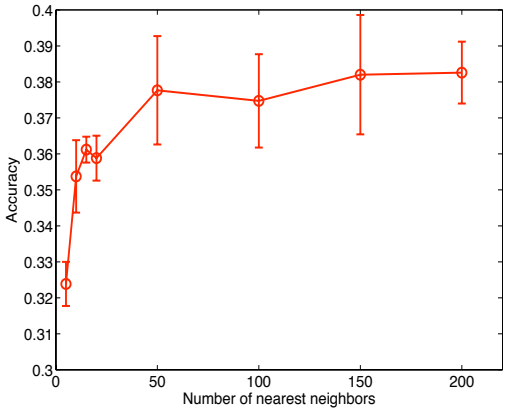
(a) Nyström: NMI score.



(b) Fixed-σ SC: NMI score.



(c) Nyström: accuracy.

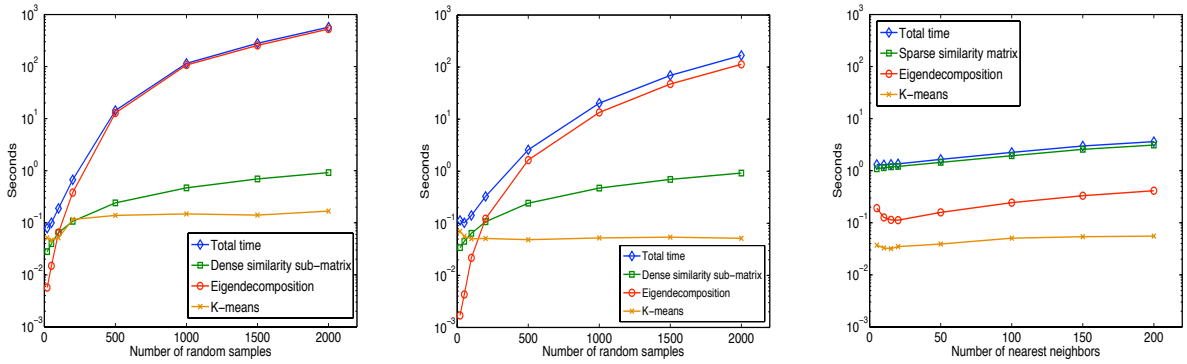


(d) Fixed-σ SC: accuracy.

Fig. 4. A clustering quality comparison between Nyström and Fixed-σ SC using the Corel data set. For Nyström, we use 20, 50, 100, 200, 500, 1000, 1500, and 2000 as the number of random samples. For Fixed-σ SC, we use 5, 10, 15, 20, 50, 100, 150, and 200 as the number of nearest neighbors.

Table V and Table VI present the comparison results. Each result is an average over ten runs. The results show that spectral clustering algorithms using sparse similarity matrices (Fixed-σ SC and Selftune SC) slightly outperform E-k-means and Nyström (without/with orthogonalization). Note that the sparse eigensolver (used for Fixed-σ SC and Selftune SC) in MATLAB is also ARPACK. Using the same two data sets, we further compared Nyström (without/with orthogonalization via (16) and (19), respectively) with Fixed-σ SC from three perspectives: NMI, Clustering Accuracy and runtime. In this comparison, we change the number of selected samples for Nyström and the number of nearest neighbors for Fixed-σ SC. We hope to see how parameters affect the clustering performance for these two types of spectral clustering algorithms.

Figure 4 shows the NMI score and Clustering Accuracy using the Corel data set. When considering the clustering quality in NMI, Nyström achieves stable results once l is large enough; however, more samples may slightly



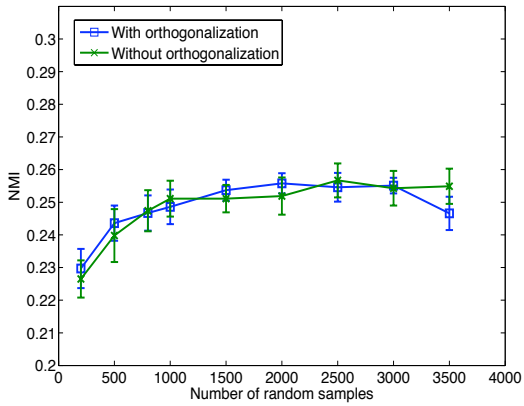
(a) Nyström with orthogonalization. (b) Nyström without orthogonalization. (c) Fixed-σ SC.

Fig. 5. A runtime comparison between Nyström and Fixed-σ SC using the Corel data set. We present the total time as well as the runtime of each important step. For Nyström, we use 20, 50, 100, 200, 500, 1000, 1500, and 2000 as the number of random samples. For Fixed-σ SC, we use 5, 10, 15, 20, 50, 100, 150, and 200 as the number of nearest neighbors.

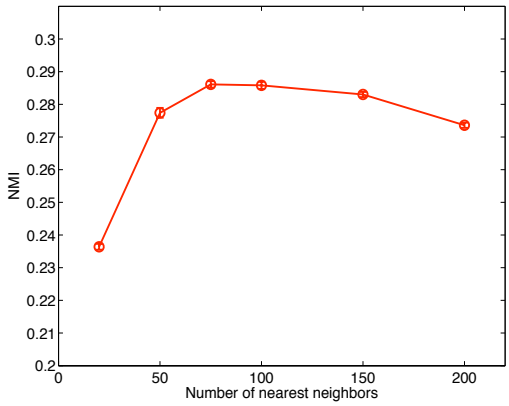
deteriorate the clustering quality because of more noisy data. Similarly, more nearest neighbors may not improve the clustering quality for Fixed-σ SC either, which performs the best when using 20 nearest neighbors. The worst performance of Fixed-σ SC occurs at five nearest neighbors. The poor results obtained for a small number of neighbors is because important relationships between points are not included. In general, Fixed-σ SC performs slightly better than Nyström when an appropriate number of nearest neighbors is set. When considering the clustering quality in Accuracy, Fixed-σ SC achieves a higher accuracy value with more nearest neighbors while Nyström gives rather stable results. Similar to NMI, Fixed-σ SC with an appropriate number of nearest neighbors performs slightly better than Nyström. Besides, Nyström using (16) (i.e., $\tilde{V}$'s columns are not orthogonal) gives markedly worse NMI and Clustering Accuracy than using (19) (i.e., $\tilde{V}$'s columns are orthogonal).

Figure 5 reports the runtime using the Corel data set. We can make the following three observations. First, Nyström is faster for obtaining the similarity matrix. For Nyström, the cost of calculating the similarity sub-matrix is proportional to the number of random samples, but Fixed-σ SC needs to calculate all n^2 similarity values. Second, Nyström needs considerable eigendecomposition time if l, the number of random samples, is large. When l is close to n, $O(l^3)$ becomes the dominant term in Table II. Thus, the total runtime of Nyström can be larger than that of Fixed-σ SC. Third, constructing the similarity matrix dominates the runtime of Fixed-σ SC while eigendecomposition dominates the runtime of Nyström. Additionally, in Fixed-σ, finding eigenvectors takes 6 to 25 restarted Arnoldi iterations based on different numbers of nearest neighbors.

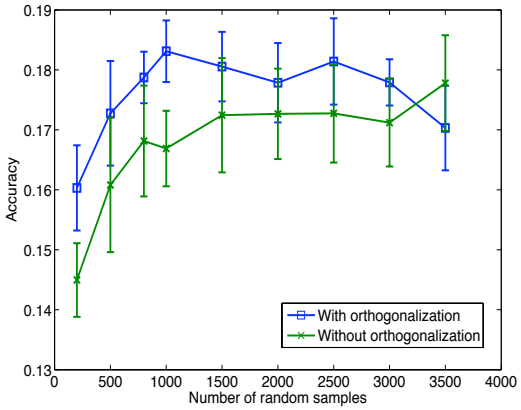
Figure 6 and Figure 7 show the comparison results and runtime using the RCV1 data set, respectively. When considering the clustering quality in NMI, Nyström is slightly worse than Fixed-σ SC if Fixed-σ SC is performed with enough nearest neighbors, (i.e., $\geq$ 20). When considering the clustering quality in accuracy, Fixed-σ SC achieves comparable performance to Nyström with orthogonalization and performs slightly better than Nyström without orthogonalization. When considering the total runtime, Fixed-σ SC takes longer than Nyström as the time



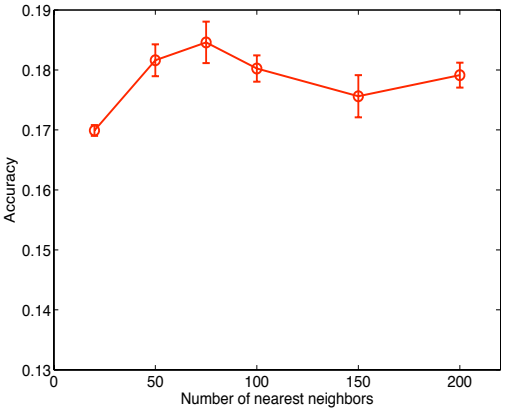
(a) Nyström: NMI score.



(b) Fixed-σ SC: NMI score.



(c) Nyström: accuracy.

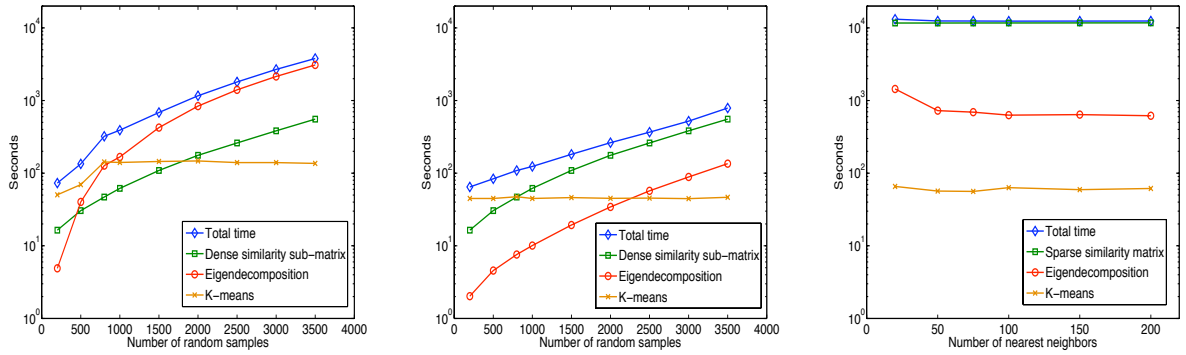


(d) Fixed-σ SC: accuracy.

Fig. 6. A clustering quality comparison between Nyström and Fixed-σ SC using the RCV1 data set. For Nyström, we use 200, 500, 800, 1000, 1500, 2000, 2500, 3000, and 3500 as the number of random samples. For Fixed-σ SC, we use 20, 50, 75, 100, 150, and 200 as the number of nearest neighbors.

for constructing a sparse similarity matrix, $O(n^2 d + n^2 \log t)$, dominates the total runtime. When considering the runtime for finding the first k eigenvectors, Nyström with orthogonal $\tilde{V}$ is expensive if $l \geq 2,000$. With $l \ll n$, $O((n-l)l^2)$ is the dominant term in Table II. In contrast, Nyström without orthogonalization gives comparable NMIs against Nyström with orthogonalization in a short amount of time. The shorter runtime is because the approach without orthogonalization does not need the $O((n-l)l^2)$ operation (see Table II). We also found that Fixed-σ SC using 20 nearest neighbors took longer for the eigendecomposition than using more nearest neighbors. When $t = 20$, we observed that the Laplacian matrix L has many zero eigenvalues. It is known that ARPACK faces difficulties in such a situation[‖]. Additionally, in Fixed-σ, finding eigenvectors takes 9 to 48 restarted Arnoldi iterations based

[‖] This is confirmed through some private communication with one ARPACK author.



(a) Nyström with orthogonalization. (b) Nyström without orthogonalization. (c) Fixed-σ SC.

Fig. 7. A runtime comparison between Nyström and Fixed-σ SC using the RCV1data set. We present the total time as well as the runtime of each important step. For Nyström, we use 200, 500, 800, 1000, 1500, 2000, 2500, 3000, and 3500 as the number of random samples. For Fixed-σ SC, we use 20, 50, 75, 100, 150, and 200 as the number of nearest neighbors.

on different number of nearest neighbors.

Regarding memory use, Nyström consumes $O(nl)$ memory while spectral clustering algorithms using sparse similarity matrices consume $O(nt + nm)$. As l is usually larger than t and m, Nyström may consume more memory. In sum, spectral clustering via sparsifying the similarity matrix takes longer total runtime (including the time for constructing the similarity matrix), but it may be more effective in finding clusters.

B. Speedup and Scalability in Distributed Environments

We used both the RCV1 data set and a PicasaWeb data set to conduct scalability experiments. PicasaWeb is an online platform for users to upload, share and manage images. The PicasaWeb data set we collected consists of $2,121,863$ images. For each image, we extracted 144 features and employed feature scaling as we did for the Corel data set. The RCV1 data set used in Section VI-A can fit into the main memory of a single machine, whereas the PicasaWeb data set cannot. For the PicasaWeb data set, we grouped the data into $1,000$ clusters, and the Arnoldi length m is set to be $2,000$.

We ran experiments on up to 256 machines at our distributed data centers. While not all machines are identical, each machine is configured with a CPU faster than 2GHz and memory larger than 4GBytes. Our experiments begin with detailed runtime and speedup analysis by varying the number of machines. We discuss individual steps of Algorithm 1 as well as the whole procedure. Next, we fix the number of machines and present speedup by varying the problem size. Finally, the scalability of our implementation is investigated.

Calculating the similarity matrix. Table VII and VIII report the speedup for calculating a sparse similarity matrix on RCV1 and PicasaWeb data sets, respectively. For the PicasaWeb data set, storing the similarity matrix and the matrix $\bar{V} \in R^{n \times m}$ with $m = 2,000$ requires more than 32GBytes of memory**. This memory configuration is

** If we assume the double precision storage, we need $2 \times 10^6 \times 2000 \times 8 = 32$ GBytes.

TABLE VII

RCV1 DATA SET. RUNTIME FOR CALCULATING THE SPARSE SIMILARITY MATRIX ON DIFFERENT NUMBER OF MACHINES. n=193,844, k=103, m=206, t=100.

Machines	CompDistance	Symmetric	CompSimilarity	Total	Speedup
1	72088s	655s	182s	72925s	1
2	37084s	324s	100s	37508s	1.94
4	19338s	161s	83s	19582s	3.72
8	9765s	88s	68s	9921s	7.35
16	6376s	33s	28s	6437s	11.33
32	3110s	20s	18s	3148s	23.16
64	2064s	19s	17s	2100s	34.73

TABLE VIII

PICASAWEB DATA SET. RUNTIME FOR CALCULATING THE SPARSE SIMILARITY MATRIX ON DIFFERENT NUMBER OF MACHINES. n=2,121,863, k=1,000, m=2,000, t=100.

Machines	CompDistance	Symmetric	CompSimilarity	Total	Speedup
16	751912s	348s	282s	752542s	16.00
32	376591s	205s	205s	377001s	31.94
64	191691s	128s	210s	192029s	62.70
128	100918s	131s	211s	101260s	118.91
256	54480s	45s	201s	54726s	220.02

not available on off-the-shelf machines. We had to use at least sixteen machines to perform spectral clustering. Therefore, we used sixteen machines as the baseline and assumed a speedup of 16. We separate the running time into three parts according to the discussion in Section V-B: compute distances and find nearest neighbors, modify the distance matrix to be symmetric, and compute similarities. The similarity matrix calculation involves little communication between nodes. Thus the speedup is almost linear if machines have similar configurations and loads (when we ran experiments). By using 256 machines and $t = 100$, the RCV1 data set takes 7.5 minutes[††] and the PicasaWeb data set takes 15.2 hours for obtaining the sparse similarity matrix.

First k eigenvectors and k-means. Here k-means refers to Step 6 in Algorithm 1. In Tables IX and X, we report the speedup on the RCV1 data set for finding eigenvectors and conducting k-means, respectively. We separate running time for finding eigenvectors into two parts: all dense operations and sparse matrix-vector products. Each part is further separated into computation, communication (message passing between nodes) and synchronization

[††] A sharp eyed reader may notice that using one machine may take much longer in data center than via MATLAB. This difference is because we do not assume here that input data can be pre-loaded into the memory of one node (see our implementation details in Section V-A).

TABLE IX

RCV1 DATA SET. RUNTIME FOR FINDING THE FIRST k EIGENVECTORS ON DIFFERENT NUMBERS OF MACHINES. n=193,844, k=103, m=206, t=100.

	Dense matrix operations within ARPACK			Sparse matrix vector product			Total	Speedup
Machines	Comp	Comm	Sync	Comp	Comm	Sync		
1	475s	0s	0s	274s	0s	0s	749s	1
2	234s	2s	6s	150s	16s	6s	414s	1.81
4	112s	4s	5s	75s	24s	6s	226s	3.31
8	57s	5s	6s	39s	29s	5s	141s	5.33
16	28s	8s	5s	20s	34s	4s	99s	7.57
32	16s	11s	7s	12s	40s	5s	91s	8.23
64	8s	13s	7s	6s	44s	5s	83s	9.02

TABLE X

RCV1 DATA SET. RUNTIME FOR k-MEANS ON DIFFERENT NUMBER OF MACHINES. n=193,844, k=103, m=206, t=100.

Machines	Comp	Comm	Sync	Total	Speedup
1	54.27s	0s	0s	54.27s	1
2	27.04s	0.25s	0.23s	27.52s	1.97
4	13.50s	0.45s	0.50s	14.45s	3.76
8	7.23s	0.72s	0.55s	8.50s	6.38
16	3.25s	0.99s	0.84s	5.08s	10.67
32	1.71s	1.46s	1.15s	4.32s	12.56
64	0.80s	1.78s	1.44s	4.02s	13.50

time (waiting for the slowest machine). As shown in Table IX, these two types of operations have different runtime behaviors. Tables IX and X indicate that neither finding eigenvectors nor k-means can achieve linear speedup when the number of machines is beyond a threshold. This result is expected due to communication and synchronization overheads. Note that other jobs may be run simultaneously with ours on each machine, though we chose a data center with a light load.

As shown in Table IX for the RCV1 data set, when the number of machines is small, most of the time is spent on dense matrix operations, which are easy to parallelize. Two reasons explain why dense operations dominate this computation: First, each step of the Arnoldi factorization takes several (usually four) dense matrix-vector products, but only one sparse matrix-vector product. Second, we have $m = 206 > t = 100$, so each dense matrix-vector product may take comparable time to a sparse one. When the number of machines increases, the computational time decreases almost linearly. However, the communication cost of sparse matrix-vector products becomes the bottleneck in finding the first k eigenvectors because a vector $\boldsymbol{v} \in R^n$ is gathered to all nodes. For dense matrix-

TABLE XI

PICASAWEB DATA SET. RUNTIME FOR FINDING THE FIRST k EIGENVECTORS ON DIFFERENT NUMBERS OF MACHINES. n=2,121,863, k=1,000, m=2,000, t=100.

	Dense matrix operations within ARPACK			Sparse matrix vector product			Total	Speedup
Machines	Comp	Comm	Sync	Comp	Comm	Sync		
16	18196s	118s	430s	3351s	2287s	667s	25049s	16.00
32	7757s	153s	345s	1643s	2389s	485s	12772s	31.38
64	4067s	227s	495s	913s	2645s	404s	8751s	45.80
128	1985s	347s	423s	496s	2962s	428s	6641s	60.35
256	977s	407s	372s	298s	3381s	362s	5797s	69.14

TABLE XII

PICASAWEB DATA SET. RUNTIME FOR k-MEANS ON DIFFERENT NUMBER OF MACHINES. n=2,121,863, k=1,000, m=2,000, t=100.

Machines	Comp	Comm	Sync	Total	Speedup
16	18053s	29s	142s	18223s	16.00
32	9038s	36s	263s	9337s	31.23
64	4372s	46s	174s	4591s	63.51
128	2757s	79s	108s	2944s	99.04
256	1421s	91s	228s	1740s	167.57

vector products, the communication cost is less for vectors of size m, but it also takes up a considerable ratio of the total time. For the total time, we can see that when 32 machines were used, the parallel eigensolver achieved 8.23 times speedup. When more machines were used, the speedup decreased. Regarding the computational time for k-means, as shown in Table X, it is less than finding eigenvectors. When using more nodes, the communication time for k-means did not increase as much as it did for finding eigenvectors. This observation is consistent with the explanation in Section V-D.

Next, we looked into the speedup on the PicasaWeb data set. Tables XI and XII report the speedups for finding eigenvectors and conducting k-means, respectively. Compared to the results with the RCV1 data set, here computational time takes a much larger ratio of the total time in obtaining eigenvectors. As a result, we could achieve nearly linear speedups for 32 machines. Even when using 256 machines, a speedup of 69.14 is obtained. As in Table IX, the communication cost for sparse matrix-vector products dominates the total time when p is large. This is due to the large $\alpha \log p$ term explained in (23). If one has a dedicated cluster with a better connection between nodes, then α is smaller and a higher speedup can be achieved. Additionally, finding eigenvectors takes 7 restarted Arnoldi iterations. For k-means, we achieve an excellent speedup. It is nearly linear up to $p = 64$.

End-to-end runtime and speedup. Table XIII and XIV show the end-to-end runtime on RCV1 and PicasaWeb data

TABLE XIII

RCV1 DATA SET. END-TO-END RUNTIME FOR PARALLEL SPECTRAL CLUSTERING ON DIFFERENT NUMBER OF MACHINES. n=193,844, k=103, m=206, t=100.

Machines	SimilarityMatrix	Eigendecomp	k-means	Total	Speedup
1	72925s	749s	54.27s	73728.27s	1
2	37508s	414s	27.52s	37949.52s	1.94
4	19582s	226s	14.45s	19822.45s	3.72
8	9921s	141s	8.50s	10070.50s	7.32
16	6437s	99s	5.08s	6541.08s	11.27
32	3148s	91s	4.32s	3243.32s	22.73
64	2100s	83s	4.02s	2187.02s	33.71

TABLE XIV

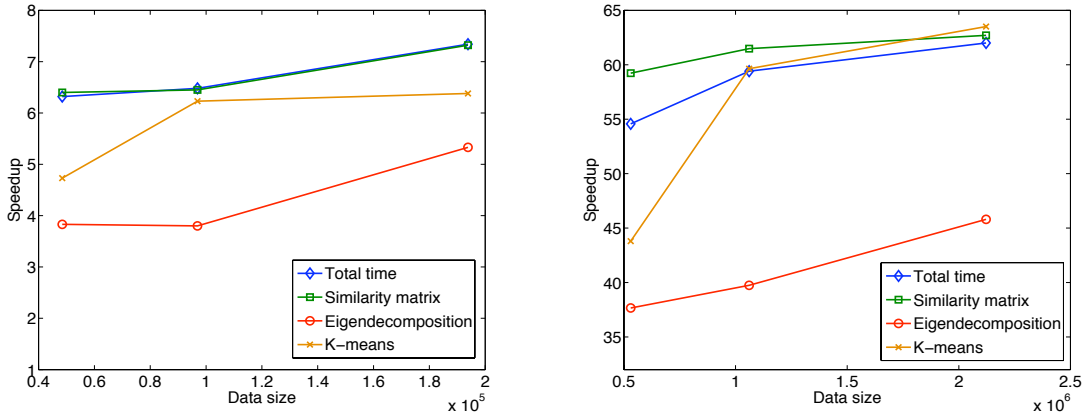
PICASAWEB DATA SET. END-TO-END RUNTIME FOR PARALLEL SPECTRAL CLUSTERING ON DIFFERENT NUMBER OF MACHINES. n=2,121,863, k=1,000, m=2,000, t=100.

Machines	SimilarityMatrix	Eigendecomp	k-means	Total	Speedup
16	752542s	25049s	18223s	795814s	16.00
32	377001s	12772s	9337s	399110s	31.90
64	192029s	8751s	4591s	205371s	62.00
128	101260s	6641s	2944s	110845s	114.87
256	54726s	5797s	1740s	62263s	204.50

sets, respectively. We can achieve near-linear speedup when using 8 machines on RCV1, and using 128 machines on PicasaWeb. Additionally, larger data sets tend to achieve higher speedup when using the same number of machines for parallelization.

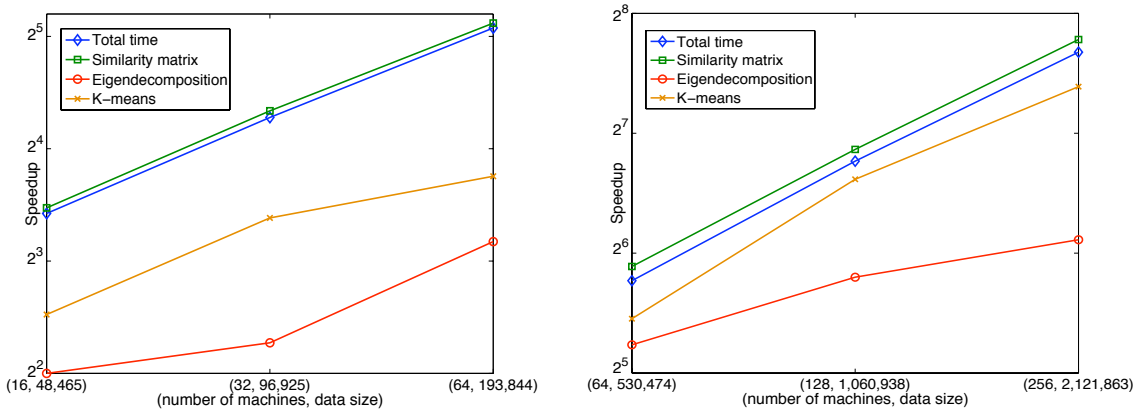
Speedup versus data sizes. Figure 8(a) shows the speedup for varying data sizes on the RCV1 data set using 8 machines. We use three different data sizes: $48,465$, $96,925$, and $193,844$. We observe that the larger the data set, the more speedup we can gain for both end-to-end and three individual steps. Because several computational intensive steps grow faster than the communication cost, the larger the data set, there is more opportunity for the parallelization to gain speedup. However, the speedup of eigendecomposition is lower than others. This is because, when compared to other steps, the communication cost for eigendecomposition takes a higher ratio in the runtime. Figure 8(b) shows the speedup of varying data sizes on the PicasaWeb data set using 64 machines. We use three different data sizes: $530,474$, $1,060,938$, and $2,121,863$. Results are similar to those for the RCV1 data set. However, the speedup line for total time is not as close to that for the similarity matrix. This is because eigendecomposition and k-means steps take a relatively larger portion of the total time on the PicasaWeb data.

Scalability. Due to the communication overhead, we have observed that under a given data size, the speedup goes



(a) RCV1: speedup versus data sizes using 8 machines. (b) PicasaWeb: speedup versus data sizes using 64 machines.

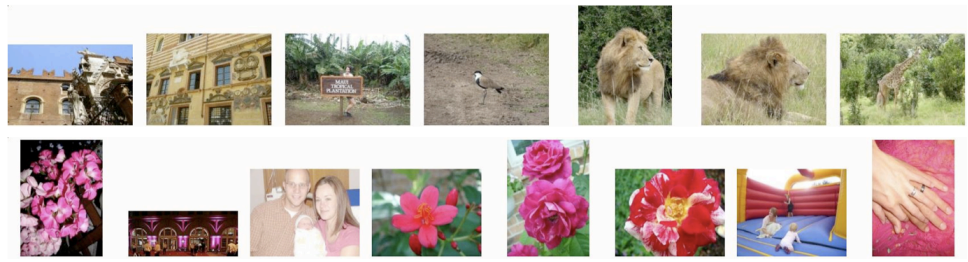
Fig. 8. Speedup versus data sizes. For RCV1, we use 8 machines and three different data sizes: 48,465, 96,925, and 193,844. For PicasaWeb, we use 64 machines and three different data sizes: 530,474, 1,060,938, and 2,121,863.



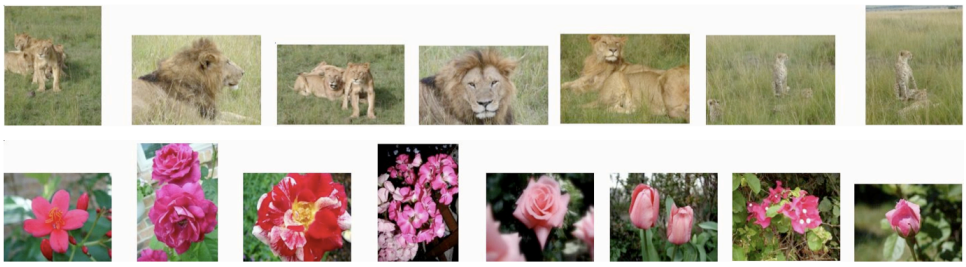
(a) RCV1: scalability. (b) PicasaWeb: scalability.

Fig. 9. Scalability: speedup versus number of machines and data sizes. For RCV1, we use (16, 48465), (32, 96925), and (64, 193844) as pairs of number of machines and data size. For PicasaWeb, we use (64, 530474), (128, 1060938), and (256, 2121863) as pairs of number of machines and data size.

down as the number of machines increases. In the parallel computation community, researchers thus define the scalability by taking the problem size into consideration. As defined in [16] [27], a parallel system is scalable if the speedup can be kept constant as the number of machines and the data sizes are both increased. Figure 9(a) presents the scalability on the RCV1 data set. The y-axis presents the the speedup, but in the x-axis, we check three different pairs of number of machines and data size: $(16, 48465)$, $(32, 96925)$, and $(64, 193844)$. Except for eigendecomposition, the speedup of the whole procedure as well as each step is almost doubled when doubling both number of machines and data size. That is, the curve has a constant slope of 2. Figure 9(b) presents the scalability



(a) Sample images via k-means.



(b) Sample images via spectral clustering.

Fig. 10. Clustering results via k-means and spectral clustering (PicasaWeb data set).

on the PicasaWeb data set. We use three different pairs of number of machines and data size: $(64, 530474)$, $(128, 1060938)$, and $(256, 2121863)$. Similarly, the speedup is doubled except for eigendcomposition. Overall, our parallel implementation scales reasonably well as the number of machines and the data size both increase.

Sample clustering results. Figure 10 shows sample clusters generated by k-means and spectral clustering. The top two rows are clusters generated by k-means, and the bottom two rows are obtained by spectral clustering. We find two observations. First, spectral clustering seems to be better at finding similarities between images (i.e., lions and leopards are clustered together as similar results). Second, spectral clustering discovers similarities between flowers and groups them together more effectively than k-means.

VII. Conclusions

In this paper, we have investigated approaches for large-scale spectral clustering. In particular, we discuss and compare two types of approaches: sparsifying the similarity matrix and the Nyström approximation. We then propose a parallel implementation and evaluate its scalability. A slightly modified version of our code is available at

`http://code.google.com/p/pspectralclustering/`,

in which the construction of similarity matrix is implemented based on MPI instead of MapReduce. No parallel algorithm can escape from Amdahl's law. But we showed that the larger a data set, the greater the number of machines that can be used to apply the parallel spectral clustering algorithm to obtain fast and high-quality clustering performances. Looking forward, we plan to enhance our work to address several research issues.

Very large number of clusters. A large k implies a large m in the process of Arnoldi factorization. Then $O(m^3)$ for finding the eigenvalues of the dense matrix H becomes the dominant term in (9). How to efficiently handle the case of large k is thus an interesting issue.

Clustering without eigendecomposition. As finding the first k eigenvectors of a Laplacian matrix is computationally expensive, Dhillon et al. [10] propose a clustering approach without using eigendecomposition. Their method is related to but different from the standard spectral clustering. Without the eigendecomposition step, it does not produce a low dimensional representation of the data. In other words, their method does not conduct dimension reduction as spectral clustering does. Moreover, they assume the availability of the similarity matrix, so the total computational time may still be high. Since our approach can handle very large data sets, it is interesting to compare the two approaches.

Approximate neighbors in obtaining the sparse similarity matrix. Exactly finding the t nearest neighbors can be quite expensive. Alternatively, one can conduct an approximation. LSH (Locality-Sensitive Hashing) [15] and Spill-tree [26] have been efficient in finding approximate nearest neighbors. However, these methods may trade clustering quality for running time. It is interesting to investigate how these methods perform.

Nyström approximation with an adaptive selection of samples. In Section IV, we discuss a naive implementation using random sampling. Several papers (e.g., [33], [53]) have developed advanced sampling techniques, which can achieve comparable clustering results via a smaller subset of samples. It is unclear yet how they are compared with spectral clustering using sparse similarity matrices. The parallelization of these adaptive sampling approaches is also an interesting and challenging research issue.

In summary, this paper gives a general and systematic study of parallel spectral clustering. Despite the communication and synchronization overheads, we build a system to effectively cluster large-scale data in a distributed computing environment.

ACKNOWLEDGMENTS

The authors thank Sanjiv Kumar, Xiaofei He, and Ling Huang for their helpful comments. The first author is supported by NSF under grant IIS-0535085.

REFERENCES

[1] D. Achlioptas, F. McSherry, and B. Schölkopf. Sampling techniques for kernel methods. In *Proceedings of NIPS*, pages 335–342, 2002.

[2] F. R. Bach and M. I. Jordan. Learning spectral clustering. In *Proceedings of NIPS*, 2003.

[3] M. Barnett, S. Gupta, D. G. Payne, L. Shuler, R. Geijn, and J. Watts. Interprocessor collective communication library (intercom). In *Proceedings of the Scalable High Performance Computing Conference*, pages 357–364, 1994.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of OSDI*, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.

[6] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Proceedings of NIPS*, pages 281–288, 2007.

[7] F. Chung. *Spectral Graph Theory*. Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[8] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[9] I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of SIGKDD*, pages 269–274, 2001.

[10] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

[11] I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large-Scale Parallel Data Mining*, pages 245–260, 1999.

[12] C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings of ICDM*, 2001.

[13] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the Nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):214–225, 2004.

[14] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *Proceedings of SOSP*, pages 29–43, New York, NY, USA, 2003. ACM.

[15] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, editors, *Proceedings of VLDB*, pages 518–529, 1999.

[16] A. Grama, G. Karypis, V. Kumar, and A. Gupta. *Introduction to Parallel Computing (2nd Edition)*. Addison Wesley, January 2003.

[17] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI-2: Advanced Features of the Message-Passing Interface*. MIT Press,, 1999.

[18] A. Gürsoy. Data decomposition for parallel k-means clustering. In *PPAM*, pages 241–248, 2003.

[19] L. Hagen and A. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9):1074–1085, 1992.

[20] V. Hernandez, J. E. Roman, A. Tomas, and V. Vidal. A Survey of Software for Sparse Eigenvalue Problems. Technical report, Universidad Politecnica de Valencia, 2005.

[21] V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software*, 31:351–362, 2005.

[22] R. B. Lehoucg, D. C. Sorensen, and C. Yang. *ARPACK User's Guide*. SIAM, 1998.

[23] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[24] B. Li, E. Y. Chang, and Y.-L. Wu. Discovery of a perceptual distance function for measuring image similarity. *Multimedia Systems*, 8(6):512–522, 2003.

[25] R. Liu and H. Zhang. Segmentation of 3D meshes through spectral clustering. In *Proceedings of Pacific Graphics*, 2004.

[26] T. Liu, A. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *Proceedings of NIPS*, 2004.

[27] I. M. Llorente, F. Tirado, and L. Vázquez. Some aspects about the scalability of scientific applications on parallel architectures. *Parallel Computing*, 22(9):1169–1195, 1996.

[28] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[29] O. A. Marques. BLZPACK: description and user's guide. Technical Report TR/PA/95/30, CERFACS, Toulouse, France, 1995.

[30] K. Maschhoff and D. Sorensen. A portable implementation of ARPACK for distributed memory parallel architectures. In *Proceeding of Copper Mountain Conference on Iterative Methods*, 1996.

[31] M. Meila and J. Shi. Learning segmentation by random walks. In *Proceedings of NIPS*, pages 873–879, 2000.

[32] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Proceedings of NIPS*, pages 849–856, 2001.

[33] M. Ouimet and Y. Bengio. Greedy spectral embedding. In *Proceedings of AISTAT*, pages 253–260, 2005.

[34] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Dover, New York, 1998.

[35] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[36] J. R. Smith and S.-F. Chang. Automated image retrieval using color and texture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1996.

[37] M. Snir and S. Otto. *MPI-The Complete Reference: The MPI Core*. MIT Press, Cambridge, MA, USA, 1998.

[38] A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.

[39] A. Talwalkar, S. Kumar, and H. Rowley. Large-scale manifold learning. In *Proceedings of CVPR*, 2008.

[40] R. Thakur, R. Rabenseinfer, and W. Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[41] S. Tong and E. Chang. Support vector machine active learning for image retrieval. *Proceedings of ACM MM*, pages 107–118, 2001.

[42] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.

[43] C. K. I. Williams, C. E. Rasmussen, A. Schwaighofer, and V. Tresp. Observations on the Nyström method for Gaussian process prediction. Technical report, University of Edinburgh, 2002.

[44] C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. *Proceedings of NIPS*, pages 682–688, 2000.

[45] K. Wu and H. Simon. A Parallel Lanczos Method for Symmetric Generalized Eigenvalue Problems. Technical Report LBNL-42953, Lawrence Berkeley National Laboratory, 1997.

[46] K. Wu and H. Simon. TRLAN user guide. Technical Report LBNL-41284, Lawrence Berkeley National Laboratory, 1999.

[47] M. Wu and B. Schölkopf. A local learning approach for clustering. In *Proceedings of NIPS*, pages 1529–1536, 2007.

[48] S. Xu and J. Zhang. A hybrid parallel web document clustering algorithm and its performance study. *Journal of Supercomputing*, 30(2):117–131, 2004.

[49] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of SIGIR*, pages 267–273, 2003.

[50] S. X. Yu and J. Shi. Multiclass spectral clustering. In *Proceedings of ICCV*, page 313, 2003.

[51] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Proceedings of NIPS*, pages 1601–1608. 2005.

[52] H. Zha, C. H. Q. Ding, M. Gu, X. He, and H. Simon. Spectral relaxation for k-means clustering. In *Proceedings of NIPS*, pages 1057–1064, 2001.

[53] K. Zhang, I. Tsang, and J. Kwok. Improved Nyström low-rank approximation and error analysis. In *Proceedings of ICML*, 2008.

[54] S. Zhong and J. Ghosh. A unified framework for model-based clustering. *Journal of Machine Learning Research*, 4:1001–1037, 2003.