PSC: Parallel Spectral Clustering

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

Abstract

Spectral clustering algorithm has 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 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 large document data set of $193,844$ instances and a large photo data set of $2,121,863$, we demonstrate that our parallel algorithm can effectively alleviate the scalability problem.

Note to reviewers: A preliminary version of this work appears in the proceedings of ECML 2008 [29]. This journal version has made three major enhancements over the ECML version. First, we have improved PSC and this paper provides implementation details in using the Message Passing Interface (MPI) and MapReduce framework. Second, we applied PSC on a much larger data set consisting of $2,121,863$ data instances, and report more results. Third, we discussed and compared the Nyström method for spectral clustering.

Index Terms

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

I. Introduction

Clustering is one of the most important subroutines in tasks of machine learning and data mining. Recently, spectral clustering methods, which exploit pairwise similarities of data instances, have been shown to be more effective than traditional methods such as k-means, which considers only the similarity values 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. Unfortunately, when the number of data instances (denoted as n) is large, spectral clustering can encounter a quadratic resource bottleneck in computing pairwise similarity between n data instances, and in storing that large matrix. Moreover, the algorithm requires considerable time and memory to find and store the first k eigenvectors of a Laplacian matrix.

W.-Y. Chen is with the Department of Computer Science, University of California at Santa Barbara, 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 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 the corresponding Laplacian matrix and calls a sparse eigensolver. Several ways, for examplee [21], are available for sparsifying the similarity matrix. While a sparse representation handles the memory bottleneck, the sparsification procedure typically requires calculating all elements of the similarity matrix, and hence the computational time is O(n^2). To alleviate this high cost, methods that approximate the similarity matrix have been proposed. In particular, Fowlkes et al. [10] propose using the Nyström approximation to avoid calculating the whole similarity matrix; this approach trades accurate similarity values for shortened computational time. An additional advantage is that a smaller eigendecomposition problem is solved. In another work, Dhillon et al. [7] 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 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 to handle 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., [4], [8], [39]. 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. For the eigenvector matrix, PSC stores it on distributed nodes to reduce per-node memory use. Together with parallel eigensolver and k-means, PSC scales well 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 of storing a dense similarity matrix. After discussing various approaches to handle this memory challenge, we present sparsification and Nyström approach in Section III and IV, respectively. In Section V, we present PSC, which chooses an approach of sparsifying the matrix via retaining nearest neighbors. We conduct two experiments in Section VI. The first one compares the clustering quality between employing sparsification and Nyström approximation. Results show that the former may yield better clustering results. The second experiment investigates the speedup of our parallel implementation. PSC achieves substantial 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 notations used throughout this paper.

TABLE I

NOTATIONS. THE FOLLOWING NOTATIONS ARE 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
$\mathbf{x}_1, \ldots, \mathbf{x}_n \in R^d$	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
$\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k \in R^d$	cluster centers of k-means

A. Basic Concepts

Given n data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, the spectral clustering algorithm constructs a similarity matrix $S \in R^{n \times n}$, where $S_{ij} \geq 0$ reflects the relationship between $\mathbf{x}_i$ and $\mathbf{x}_j$. It then uses the similarity information to group $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to k clusters. There are several variants of spectral clustering. Here we consider a commonly used *normalized* spectral clustering [23]. (For a survey of all variants, please refer to [21].) An example similarity function is the Gaussian:

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

Consider the normalized Laplacian matrix [5]:

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

where D is a diagonal matrix with

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

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, non-zero 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 [21, Proposition 4]. Their corresponding eigenvectors, written as an $R^{n \times k}$ matrix, are

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

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

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

where $\mathbf{e}_i, i = 1, \ldots, k$ (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 eigenvectors we obtained are in the form of

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

where Q is an orthogonal matrix. Ng et al. [23] propose normalizing V so that

$$U_{ij} = \frac{V_{ij}}{\sqrt{\sum_{r=1}^{k} V_{ir}^2}}, i = 1, \ldots, n, j = 1, \ldots, k. \tag{3}$$

Each row of U has a unit length. Due to the orthogonality of Q, (3) is equivalent to

$$U = EQ = \begin{bmatrix} Q_{1,1:k} \\ \vdots \\ Q_{1,1:k} \\ Q_{2,1:k} \\ \vdots \end{bmatrix}, \tag{4}$$

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 [23] or other simple techniques [40], [42].

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 [26], Min-Max Cut [9], and Ratio Cut [13].

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. Figure 1 depicts all 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

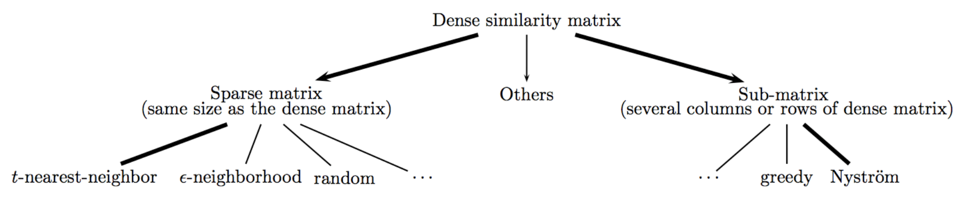


Fig. 1. Approximation techniques for spectral clustering to avoid storing the dense similarity matrix. In this paper, we mainly investigate two types: "Sparse matrix" indicates that we retain only certain nonzero S_{ij}. The resulting matrix is still a square one, but is sparse. "Sub-matrix' is a method to use several columns (or rows) of S (i.e., a retangular portion of the original dense matrix). 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) and requires virtually no computation cost. We will show in Section VI that the information gain from data is worthwhile for improving clustering quality. Again, this paper studies only the two end of the spectrum.

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). 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 considers zeroing out random entries in the similarity matrix. Though this simplification effectively saves time, experiments in [10] reveal inferior performances. Such a result is predictable as it does not use significant relationships between data points. In this section, we focuse 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

$$O(n^2 d) + O(n^2 \log t) \text{ time and } O(nt) \text{ memory.} \tag{5}$$

[*]We do not compare different methods to make a matrix sparse. Nevertheless, our preliminary empirical study shows that the t-nearest-neighbor approach is comparable to or better than the ϵ-neighborhood method.

Algorithm 1 Spectral clustering using a sparse similarity matrix

```
Input: Data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$; $k$: number of clusters to construct.
  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. (3).
  6) Use $k$-means algorithm to cluster $n$ rows of $U$ into $k$ groups.
```

The $O(n^2 d)$ cost can be reduced to a smaller value using techniques such as KD-trees [2] and Metric trees [34]. However, these techniques are less suitable if d, the dimensionality, is large. To further reduce the cost, one can 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 [20]. 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. Besides, the eigensolver remains to be a computation bottleneck, and we must devise parallel algorithms to alleviate that bottleneck. We thus focus only a precise method to find t nearest neighbors.

Compute the first k eigenvectors by Lanczos/Arnoldi factorization. Once we have obtained a sparse similarity matrix S, 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 [15], [16] (see [14] 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 [16], which implements an implicitly restarted Arnoldi method. We 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

$$L\bar{V} = \bar{V}H + (\text{a matrix of small values}), \tag{6}$$

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 (6) is indeed zero, then $\bar{V}$'s m columns are L's first m eigenvectors (details not derived here). Therefore, (6) 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

$$L\boldsymbol{v}, \tag{7}$$

where $\boldsymbol{v}$ is an $n \times 1$ vector. As on average the number of nonzeros per row of L is $O(t)$, the cost of this sparse matrix-vector multiplication 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

$$\big(O(m^3) + (O(nm) + O(nt)) \times O(m-k)\big) \times (\text{\# restarted Arnoldi}), \tag{8}$$

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. The memory requirement of ARPACK is $O(nt) + O(nm)$.

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:

$$\sum_{i=1}^{k} \sum_{\boldsymbol{u}_j \in C_i} ||\boldsymbol{u}_j - \boldsymbol{c}_i||^2. \tag{9}$$

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 assign 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

$$O(nk^2) \times (\text{\# of } k\text{-means iterations}). \tag{10}$$

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 two error function values is less than 0.001.

Overall analysis. If one uses a direct method of finding the t nearest neighbors, from (5), (8), and (10), 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, the authors of [19] 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.

A. Nyström Method

The Nyström method is a technique for finding a numerical approximation to eigendecomposition. It was applied to speed up the kernel machines [37], and then has been widely applied to areas involving large dense matrices.

However, its use for spectral clustering has not been exploited much. Some existing work includes [10] for image segmentation and [31] for manifold learning. In this section, we discuss how to apply the Nyström method to general spectral clustering.

Here we denote S_d as 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:

$$S_d = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}, \text{ and } W = \begin{bmatrix} A \\ B^T \end{bmatrix} \tag{11}$$

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 (11), the approximation (denoted as $\tilde{S}$) takes the form:

$$S_d \approx \tilde{S} = WA^{-1}W^T = \begin{bmatrix} A & B \\ B^T & B^T A^{-1} B \end{bmatrix}. \tag{12}$$

That is, the matrix C in S_d is now replaced by $B^T A^{-1} B$. 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:

$$\tilde{\Sigma} = \left(\frac{n}{l}\right)\Sigma_A, \quad \tilde{V} = \sqrt{\frac{l}{n}} W V_A \Sigma_A^{-1}. \tag{13}$$

Moreover, (13) easily implies that $\tilde{S}$ has the eigendecomposition:

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

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 sum of $\tilde{S}$. A direct calculation of $\tilde{S}$ involves the the matrix-matrix

[†]One may use other ways to select samples. See, for example, the "greedy" way mentioned in Figure 1 [24].

Algorithm 2 Spectral clustering using the Nyström method

```
Input: Data points $\mathbf{x}_1, \dots, \mathbf{x}_n$; $l$: number of samples; $k$: number of clusters to construct; $l > k$.
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 $\mathbf{x}_1, \dots, \mathbf{x}_l$ and $\mathbf{x}_1, \dots, \mathbf{x}_n$.
2) Calculate $\mathbf{a} = A\mathbf{1}_l$, $\mathbf{b}_1 = B\mathbf{1}_{n-l}$, $\mathbf{b}_2 = B^T\mathbf{1}_l$ and $D = \text{diag}\left(\begin{bmatrix} \mathbf{a}+\mathbf{b}_1 \\ \mathbf{b}_2 + B^TA^{-1}\mathbf{b}_1 \end{bmatrix}\right)$. Here $\mathbf{1}$ represents a column vector of ones.
3) Calculate $\bar{A} = D^{-1/2}_{1:l,1:l} A D^{-1/2}_{1:l,1:l}, \bar{B} = D^{-1/2}_{1:l,1:l} B D^{-1/2}_{l+1:n,l+1:n}$.
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 $\Lambda_R$ are in a decreasing order.
6) Calculate
       $$\tilde{V} = \begin{bmatrix} \bar{A} \\ \bar{B}^T \end{bmatrix} \bar{A}^{-1/2}(U_R)_{:,1:k}(\Lambda_R^{-1/2})_{1:k,1:k} \quad (16)$$
   as the first $k$ eigenvector of $\tilde{L}$.
7) Compute the normalized matrix $\tilde{U}$ of $\tilde{V}$ by Eq. (3).
8) Use $k$-means algorithm to cluster $n$ rows of $\tilde{U}$ into $k$ groups.
```

product $B^TA^{-1}B$, which is an expensive $O(l(n-l)^2)$ operation. The authors of [10] thus propose the following procedure:

$$\tilde{S}\mathbf{1} = \begin{bmatrix} A\mathbf{1}_l + B\mathbf{1}_{n-l} \\ B^T\mathbf{1}_l + B^TA^{-1}B\mathbf{1}_{n-l} \end{bmatrix} = \begin{bmatrix} \mathbf{a}+\mathbf{b}_1 \\ \mathbf{b}_2 + B^T\left(A^{-1}\mathbf{b}_1\right) \end{bmatrix}, \tag{14}$$

where $\mathbf{a}$, $\mathbf{b}_1$, $\mathbf{b}_2$ represent the row sums of A, B and B^T, respectively, and $\mathbf{1}$ is a column vector of ones. Using (14), 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 (13) we obtain $\tilde{L}$'s first k eigenvectors via the eigendecomposition $\bar{A} = \bar{V}_A\bar{\Sigma}_A\bar{V}_A^T$ and then

$$\tilde{\Sigma} = \left(\frac{n}{l}\right)\left(\bar{\Sigma}_A\right)_{:,1:k}, \quad \tilde{V} = \sqrt{\frac{l}{n}} \begin{bmatrix} \bar{A} \\ \bar{B}^T \end{bmatrix} \left(\bar{V}_A\right)_{:,1:k}\left(\bar{\Sigma}_A^{-1}\right)_{1:k,1:k}, \tag{15}$$

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 Section II-A

TABLE II

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

Algorithms	Nyström using (15)	Nyström using (16)	t-nearest-neighbor sparse matrix
Obtaining the similarity matrix	$O(nld)$	$O(nld)$	$O(n^2d + n^2 \log t)$
Finding first k eigenvectors	$O(l^3) + O(nlk)$	$O((n-l)l^2) + O(l^3)$ $+ O(nlk)$	$(O(m^3) + (O(nm) + O(nt)) \times (m-k))$ $\times$(# restarted Arnoldi)

we have orthogonal V in (II-A) and then in the ideal situation, its normalized matrix U has rows corresponding to k orthogonal points on the unit sphere. In [10], the authors proposed an approach to have orthogonal columns of $\tilde{V}$ [‡]. Let

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

and its eigendecomposition $R = U_R\Lambda_R U_R^T$. It is proved in [10] that

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

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 (15), we calculate the first k columns of $\tilde{V}$ via (16).

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 (16) is presented in Algorithm 2.

C. Computational Complexity and Memory Usage

Under the same assumption that calculating each S_{ij} costs $O(d)$, where d is the dimension of data, obtaining matrices A and B takes $O(nld)$. If using (15), 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 $O(nl)$ spaces.

If using (16), in constructing R, the matrix-matrix multiplication costs $O((n-l)l^2)$. The eigendecomposition of a dense R requires $O(l^3)$. Finally, calculating $\tilde{V}$ via (16) takes $O(nlk)$. As generally $l \ll n$, the $O((n-l)l^2$ cost is the dominant term. Moreover, as this term is not needed when using (15), maintaining orthogonal $\tilde{V}$ via (16) is more expensive. Note that in Step 2 of Algorithm 2, one should calculate the diagonal matrix D by

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

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

[‡]Instead one may use a method called "column sampling." See the discussion in [31]

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 sparse-matrix approach (t-nearest-neighbor).

From the clustering quality perspective, we suggest that the sparse-matrix approximation may give 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. Experiments in Section VI-A confirm that using a sparse similarity matrix is better than using a dense one. 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 [24], [36] 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 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 may 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 using approximation t-nearest-neighbor methods.
- In finding the first k eigenvectors of L, Nyström may not be efficient. For the sparse-matrix approach, we often set $m = 2k$ or $5k$ for the Arnoldi length. In contrast, for Nyström, we may need $l \gg k$.

V. Parallel Spectral Clustering 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) [28] and MapReduce [6] 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.

A. *MPI and MapReduce*

MPI is a protocol for parallel programming [28]. An MPI program is loaded into the local memory of each node, where every local process 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.

TABLE III

SAMPLE MPI FUNCTIONS [28].

MPI_Bcast:	Broadcasts information to all machines.
MPI_AllGather:	Gathers the data contributed by each machine on all machines.
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 machines.

Different from MPI, MapReduce is a Google parallel computing framework [6]. 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. However, a limitation of using MPI is that fault tolerance is not supported.

In Algorithm 1, calculating the similarity and finding the t-nearest-neighbor is not an iterative procedure, so 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 [22], a parallel ARPACK implementation based on MPI. 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) [11] and store data in the SSTable file format [3]. 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. Since standard MPI implementations such as MPICH2[§] [35] cannot be directly ported to our system, we implement our own MPI system by modifying MPICH2.

B. Similarity Matrix and Nearest Neighbors

To compute the similarity matrix based on a t-nearest-neighbor graph, 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.

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

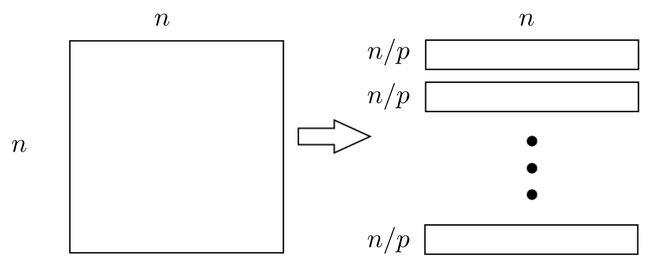


Fig. 2. The similarity matrix is distributedly computed and stored on multiple machines.

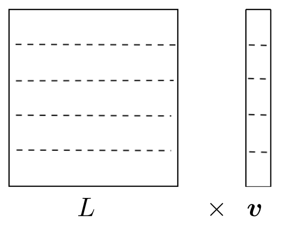


Fig. 3. Sparse matrix-vector multiplication. We assume $p = 5$ here. L and $\boldsymbol{v}$ are respectively separated to five block partitions.

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 cannot assume that all data instances can be loaded into the memory of each single node in the distributed environment. We thus need to carefully consider the memory usage in calculating the distances. 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 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 $\mathbf{x}_i$, we calculate $\|\mathbf{x}_i - \mathbf{x}_j\|$ for all $\mathbf{x}_j$ 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

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

We precompute all $\|\mathbf{x}_j\|^2$ 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.

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

In the map phase, for each non-zero 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.

Compute similarities. Although we can easily compute the similarities in the previous step, we use a separate MapReduce step to self-tune the parameter σ in (1) [41]. We consider the following similarity function:

$$S_{ij} = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma_i\sigma_j}\right). \tag{19}$$

Suppose $\mathbf{x}_i$ 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, $\sigma_i = ||\mathbf{x}_i - \mathbf{x}_{i_t}||$, where $\mathbf{x}_{i_t}$ is the $\lfloor t/2 \rfloor$th neighbor of $\mathbf{x}_i$ by sorting distances to $\mathbf{x}_i$'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 (19).

C. Parallel Eigensolver

After we have obtained the sparse similarity matrix, it is important to use a parallel eigensolver. 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 (6)) 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:

$$\left(O(m^3) + (O(\frac{nm}{p}) + O(\frac{nt}{p})) \times O(m-k)\right) \times (\text{\# restarted Arnoldi}). \tag{20}$$

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

1) Sum p values and broadcast the result to p nodes.
2) Parallel sparse matrix-vector product (7).
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

$$L_{1:n/p,1:n} \times \boldsymbol{v}, \tag{21}$$

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

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:

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

where α, the startup 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 [32]. The total communication cost to gather $\boldsymbol{v}$ on all nodes is

$$O\left(\alpha \cdot \log(p) + \beta \cdot \frac{p-1}{p} n\right), \tag{22}$$

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 [32] for the gather operation, (22) 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 (22) 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.

To start the k-means procedure, the master machine chooses a set of initial cluster centers and broadcasts them to all machines. Revisit Eq. (4). 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 [40]. 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 distance between its points and the center. The cosine distance is difined as the inner product between two points. By collecting the p minimal cosine distances, 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 communication

TABLE IV

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 V

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	$0.3851(\pm 0.0164)$	$0.1842(\pm 0.0021)$

occurs here, and this is a reduction operation in MPI (*MPI_AllReduce*, see Table III). The loss function (9) can also be computed in parallel in a similar way. Therefore, the computational time of parallel k-means is reduced to $1/p$ of that in (10). 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. Therefore, if k is not large, the communication cost of k-means is usually smaller than that in finding the first k eigenvectors.

VI. EXPERIMENTS

We designed experiments to evaluate spectral clustering algorithms and investigated the scalability 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://www.cs.ucsb.edu/~wychen/sc.html`

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

Corel 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 [18]. In the color channel, we divided color into 12 color bins including 11 bins for culture colors and one bin for outliers [33]. 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 [27] 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 (19) are considered according to whether one assigns a fixed σ to all data or not.

RCV1 is an archive of $804,414$ manually categorized newswire stories from Reuters Ltd [17]. 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 between two random variables CAT (category label) and CLS (cluster label) is defined as:

$$\text{NMI(CAT; CLS)} = \frac{\text{I(CAT; CLS)}}{\sqrt{\text{H(CAT)H(CLS)}}}, \tag{23}$$

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 [30]:

$$\text{NMI} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k} n_{i,j} \log\left(\frac{n \cdot n_{i,j}}{n_i \cdot n_j}\right)}{\sqrt{\left(\sum_i n_i \log\frac{n_i}{n}\right)\left(\sum_j n_j \log\frac{n_j}{n}\right)}}, \tag{24}$$

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 0 if data are randomly partitioned. The higher the NMI score, the better the clustering quality.

Clustering Accuracy is another measure to evaluate the cluster quality and is defined as [38]:

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} \delta(y_i, map(c_i))}{n}, \tag{25}$$

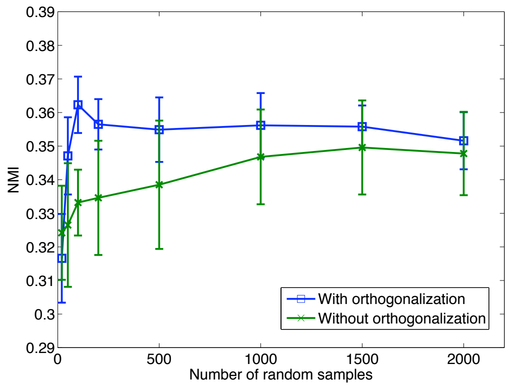
where n is the number of images/documents, y_i and c_i denote the true category label and obtained cluster label of image/document $\mathbf{x}_i$, respectively. $\delta(x, y)$ is a function that equals 1 if $x = y$ and equals 0 otherwise. $map(\cdot)$ is a permutation function that maps each cluster label to a category label, and the optimal matching can be found by Hungarian algorithm [25].

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

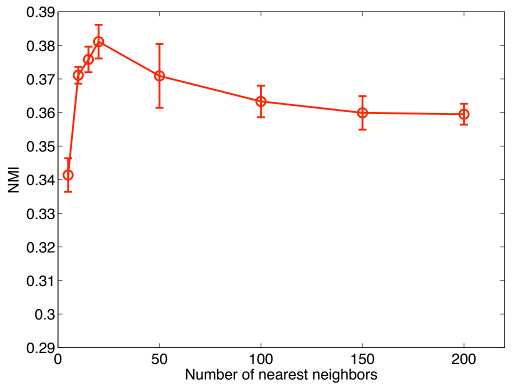
- k-means algorithm based on Euclidean distance (E-k-means).
- spectral clustering using Nyström method. We apply (15) to obtain the non-orthogonal eigenvectors (Nyström without orthogonalization).
- spectral clustering using Nyström method. We apply (16) to have orthogonal columns of $\tilde{V}$ (Nyström 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 (19) 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. For Nyström (without/with orthogonalization) and Fixed-σ SC, we searched a grid of σ values ($10, \ldots, 40$ for Corel data set, and $0.5, \ldots, 2.5$ for RCV1 data set) and reported the best clustering performance. 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 we discussed the sensitivity of using various values of t, the number of nearest neighbors. Table IV presents the comparison results. Each of the reported numbers is an average of ten runs. Note that the sparse eigensolver (used for Fixed-σ SC and Selftune SC) in MATLAB is also ARPACK.

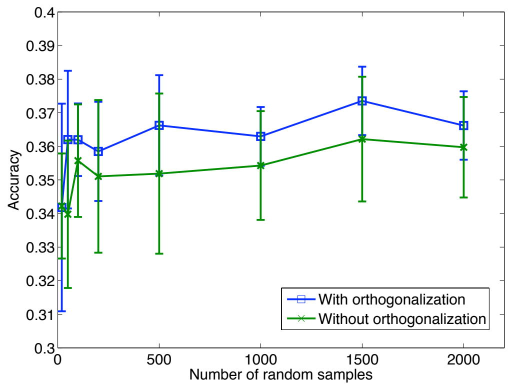
Tables IV and V report that spectral clustering algorithms using sparse similarity matrices (Fixed-σ SC and Selftune SC) slightly outperform E-k-means and Nyström (without/with orthogonalization). Using the same two data sets, we then comprehensively compared Nyström (without/with orthogonalization via (15) and (16), respectively) with Fixed-σ SC from three perspectives: NMI, Clustering Accuracy and (eigendecomposition) runtime. We investigated only the time required to find eigenvectors as Nyström is clearly faster for obtaining the similarity matrix. To generate a sparse similarity matrix, the t-nearest-neighbor approach using $t = 200$ takes 2.89 seconds and 3.2 hours for Corel and RCV1, respectively. As Nyström uses only a few rows of the matrix, the cost of



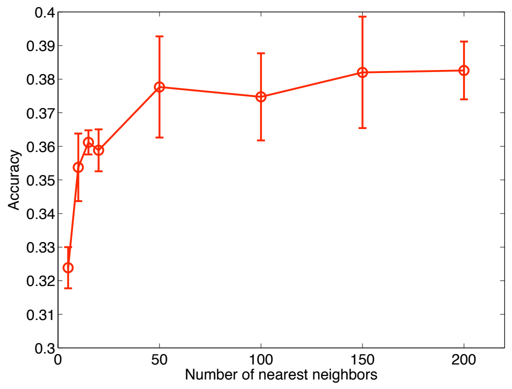
(a) Nyström: NMI score.



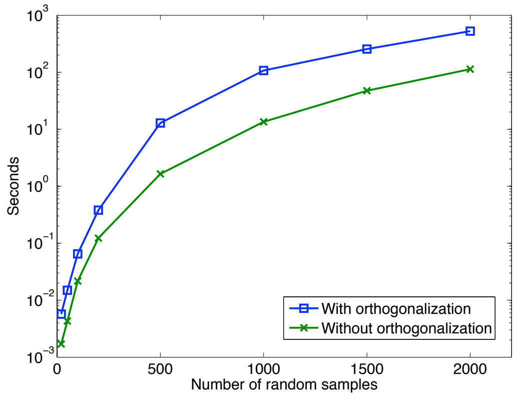
(b) Fixed-σ SC: NMI score.



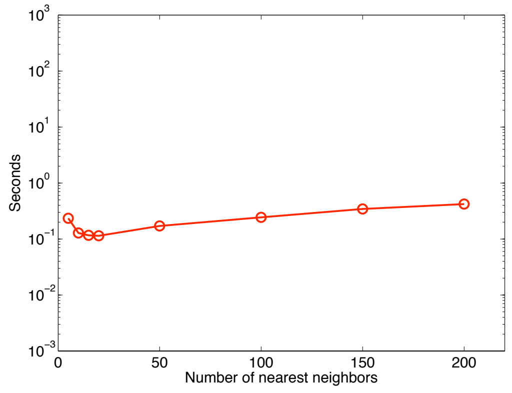
(c) Nyström: accuracy.



(d) Fixed-σ SC: accuracy.

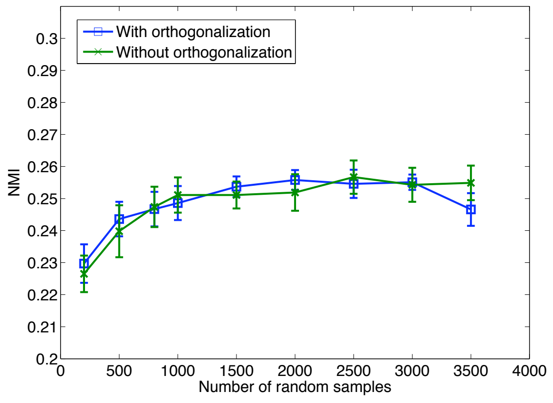


(e) Nyström: eigendecomposition time

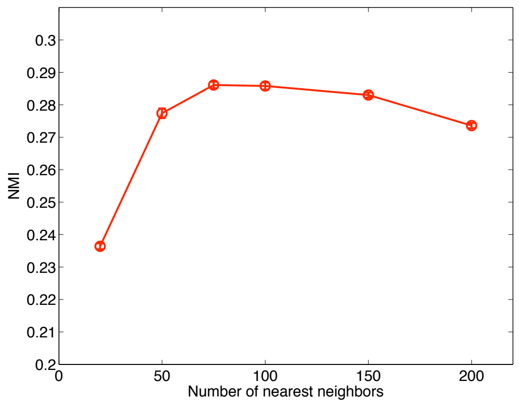


(f) Fixed-σ SC: eigendecomposition time.

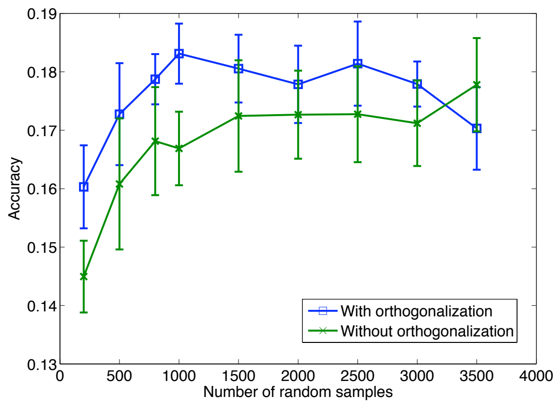
Fig. 4. A comparison between Nyström and Fixed-σ SC using the Corel data set. For Nyström, we use 20, 50, 100, 200, 500, 1000, 1500, and 2000 as the number of random samples. For Fixed-σ SC, we use 5, 10, 15, 20, 50, 100, 150, and 200 as the number of nearest neighbors.



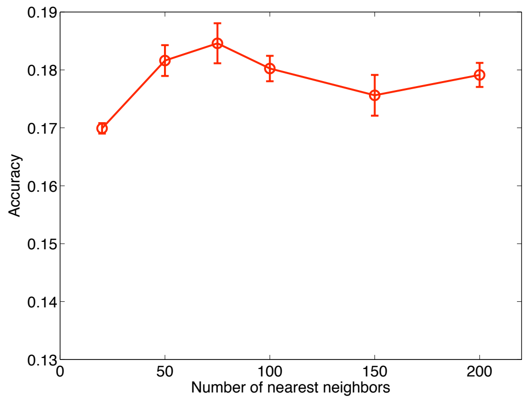
(a) Nyström: NMI score.



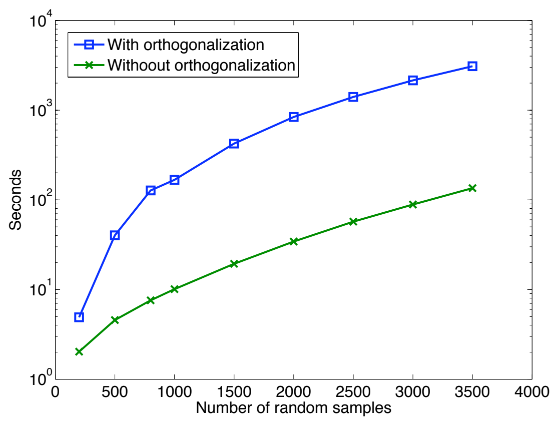
(b) Fixed-σ SC: NMI score.



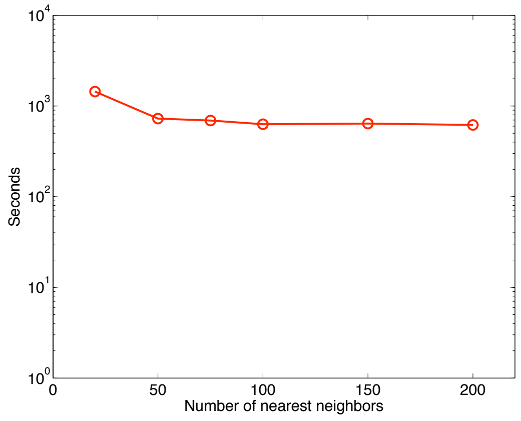
(c) Nyström: accuracy.



(d) Fixed-σ SC: accuracy.



(e) Nyström: eigendecomposition time.



(f) Fixed-σ SC: eigendecomposition time.

Fig. 5. A 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.

TABLE VI

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	720s	0s	0s	270s	0s	0s	990s	1
2	322s	5s	3s	141s	21s	0s	492s	2.01
4	154s	12s	11s	71s	34s	1s	283s	3.50
8	79s	19s	12s	36s	40s	3s	189s	5.24
16	41s	26s	13s	19s	46s	2s	147s	6.73
32	19s	34s	17s	9s	47s	3s	129s	7.67
64	9s	44s	21s	5s	48s	3s	130s	7.62

TABLE VII

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	53s	0s	0s	53s	1
2	26s	0.2s	0.4s	26.6s	2
4	13s	0.5s	0.3s	13.8s	3.8
8	6.6s	0.7s	0.4s	7.7s	6.9
16	3.3s	0.9s	0.6s	4.8s	11
32	1.6s	1.3s	1.0s	3.8s	14
64	0.8s	1.5s	1.1s	3.4s	15

calculating the similarity sub-matrix is proportional to the number of random samples. 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, Clustering Accuracy and runtime for finding the first k eigenvectors using the Corel data set. The result is an average of ten runs. We can make four observations. First, from the NMI perspective, Nyström achieves stable results once l is large enough; however, more samples may slightly 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 may be 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. Second, from the Accuracy perspective, 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. Third, Nyström needs considerable eigendecomposition time if l, the number of random samples, is large. When l is close to n, $O(l^3)$ is the dominant term in Table II. Last, Nyström using (15) (i.e., $\tilde{V}$'s columns are not orthogonal) gives markedly worse NMI and Clustering Accuracy than using (16).

Figure 5 reports the comparison results of using the RCV1 data set. When considering the clustering quality in NMI, Nyström is slightly worse than Fixed-σ SC if Fixed-σ SC is with enough nearest neighbors, (i.e., ≥ 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 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. We also found that Fixed-σ SC using 20 nearest neighbors took longer time for the eigendecomposition. 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[‖]. Regarding memory use, Nyström consumes $O(nl)$ memory while the 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 a longer total runtime (including the time for constructing the similarity matrix), but it tends to be more effective in finding clusters.

B. Scalability: Runtime Speedup 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 emploied 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.

Our parallel spectral clustering code was implemented in C++. 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.

Calculating the Similarity Matrix. The similarity matrix calculation does not involve any communication between nodes. Thus the speedup is almost linear if machines have similar configurations and there are no other jobs running. By using 256 machines and $t = 100$, the RCV1 data set takes four minutes[**] and the PicasaWeb data set takes seven and half hours for obtaining the sparse similarity matrix.

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

[**] A sharp eyed reader may notice that using one machine takes more than 10 hours, longer than three hours reported in Section VI-A 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 VIII

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	24484s	202s	896s	2798s	2335s	104s	30819s	16
32	11727s	275s	971s	1438s	2527s	132s	17070s	29
64	5967s	355s	319s	758s	2776s	50s	10225s	48
128	3241s	656s	539s	438s	3805s	113s	8792s	56
256	1758s	721s	870s	265s	4894s	110s	8618s	57

TABLE IX

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

Machines	Comp	Comm	Sync	Total	Speedup
16	11076s	18s	220s	11314s	16
32	5580s	23s	112s	5715s	32
64	2753s	30s	248s	3031s	60
128	1429s	39s	225s	1693s	107
256	742s	71s	483s	1296s	140

First k eigenvectors and k-means. Here k-means refers to Step 6 in Algorithm 1. In Tables VI and VII, 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 to computation, communication (message passing between nodes) and synchronization time (waiting for the slowest machine). As shown in Table VI, these two types of operations have different runtime behaviors. Tables VI and VII 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 VI 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 multiplication 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 $\boldsymbol{v} \in R^n$ are gathered to all nodes. For dense matrix-vector products, the communication 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 7.62 times speedup. When more machines were used, the speedup decreased. Regarding the computational time of k-means, as shown in Table VII, 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. We grouped the data into $1,000$ clusters, where the corresponding Arnoldi space is set to be $2,000$. Note that 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 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. This assumption is reasonable since our parallelization can achieve approximately linear speedup on up to 32 machines (for finding eigenvectors) and 128 machines (for k-means).

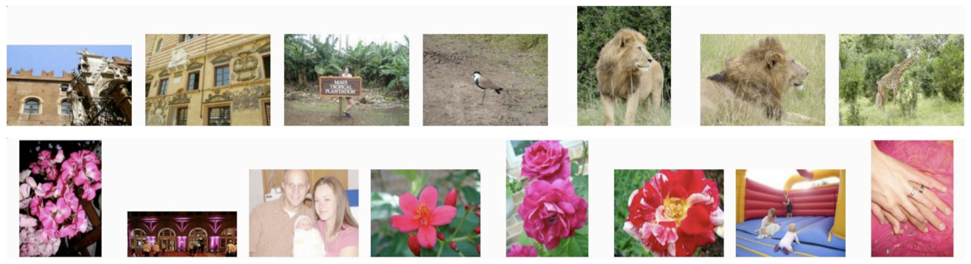
Tables VIII and IX report the speedups for finding eigenvectors and conducting k-means, respectively. Compared to the results with the RCV1 data set, here computational time takes a much larger ratio of the total time in obtaining eigenvectors. As a result, we could achieve nearly linear speedups for 32 machines. Even when using 256 machines, a speedup of 57 is obtained. As in Table VI, 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 (22). If one has a dedicated cluster with a better connection between nodes, then α is smaller and a higher speedup can be achieved. For k-means, we achieve an excellent speedup. It is nearly linear up to $p = 128$.

Figure 6 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 does 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.

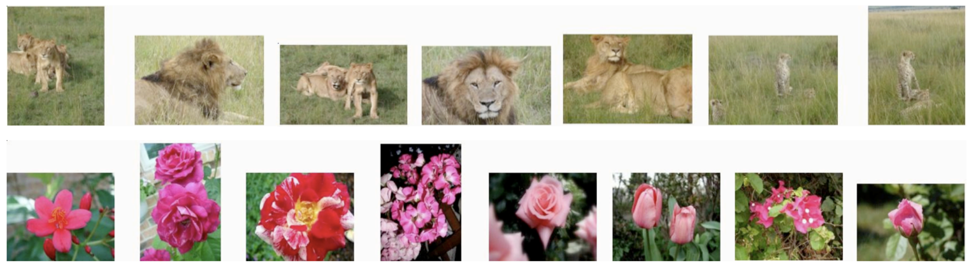
VII. Conclusions

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 demonstrate its scalability. 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 a couple of research issues.

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



(a) Sample images via k-means.



(b) Sample images via spectral clustering.

Fig. 6. Clustering results via k-means and spectral clustering (PicasaWeb data set).

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 (8). 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. [7] propose a clustering approach without using an eigendecomposition. Their method is related to but different from the standard spectral clustering. Without the eigendecomposition, 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) [12] and Spill-tree [20] 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., [24], [43]) 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, and describes how we build a system to efficiently cluster large-scale data in a distributed computing environment.

ACKNOWLEDGMENTS

The authors thank Sanjiv Kumar, Xiaofei He, and Ling Huang for their helpful comments.

REFERENCES

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

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

[3] 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.

[4] 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.

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

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

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

[8] 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.

[9] 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.

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

[11] 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.

[12] 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 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 518–529. Morgan Kaufmann, 1999.

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

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

[15] V. Hernandez, J. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31:351–362, 2005.

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

[17] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, 2004.

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

[19] R. Liu and H. Zhang. Segmentation of 3D meshes through spectral clustering. In *Proceedings of Pacific Conference on Computer Graphics and Applications*, 2004.

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

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

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

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

[24] M. Ouimet and Y. Bengio. Greedy spectral embedding. In *Proceedings of Workshop on Artificial Intelligence and Statistics (AISTAT)*, pages 253–260, 2005.

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

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

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

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

[29] Y. Song, W.-Y. Chen, H. Bai, C.-J. Lin, and E. Y. Chang. Parallel spectral clustering. In *Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*, 2008.

[30] A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3:583–617, 2002.

[31] A. Talwalkar, S. Kumar, and H. Rowley. Large-scale manifold learning. In *IEEE International Conference on Vision and Pattern Recognition (CVPR)*, 2008.

[32] 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.

[33] S. Tong and E. Chang. Support vector machine active learning for image retrieval. *Proceedings of the ninth ACM international conference on Multimedia*, pages 107–118, 2001.

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

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

[36] 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.

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

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

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

[40] S. X. Yu and J. Shi. Multiclass spectral clustering. In *Proceedings of ICCV*, page 313, Washington, DC, USA, 2003. IEEE Computer Society.

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

[42] 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.

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