Parallel Spectral Clustering Algorithm for Large-Scale Community Data Mining

Gengxin Miao
Department of ECE
University of California
Santa Barbara, CA
miao@umail.ucsb.edu

Yangqiu Song
Department of Automation
Tsinghua University
Beijing, China, 100084
songyq99@mails.tsinghua.edu.cn

Dong Zhang, Hongjie Bai
Google Research China
Beijing, China, 100084
dongzhang,hjbai@google.com

ABSTRACT

The spectral clustering algorithm has been shown to be very effective in finding clusters of non-linear boundaries. Unfortunately, spectral clustering suffers from the scalability problem in both memory use and computational time. In this work, we parallelize the algorithm by dividing both memory use and computation on distributed machines. Empirical study on some small datasets shows the accuracy of our parallelization scheme. Empirical study on a large community dataset obtained from Orkut demonstrates the scalability of our parallel spectral clustering algorithm.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Information Search and Retrieval-*Clustering*; I.2 [**Artificial Intelligence**]: Learning; I.5 [**Pattern Recognition**]: Applications; J.4 [**Social and Behavioral Sciences**]

General Terms

Algorithms, Experimentation, Data Mining

Keywords

Spectral Clustering, Parallel Computing, Social Network

1. INTRODUCTION

The fast emerging Web 2.0 allows users to engage each other through both information and application sharing. For instance, users share data via Blog, Wiki, or BBS services. Users share applications via social platforms such as Facebook and OpenSocial. Communities are formed by users of similar interests. Being able to discover communities of joint interests is of the greatest importance for maintaining high viral energy on social networks. Such discoveries can assist effective friend suggestion, topic recommendation, and ads matching, just to name a few.

One approach to discover communities is through clustering. The biggest challenge that a clustering algorithm faces is scalability: an algorithm must be able to handle millions of data instances in a relatively short period of time. Take Orkut [1] as an example; it consists of more than 20 million communities and more than 50 million users. Performing clustering on such a large-scale dataset on a single machine is prohibitive in both memory use and computational time.

In this paper, we propose to parallelize the spectral clustering algorithm to run on distributed computers. With the increasing popularity of Google-like, distributed data centers that contain millions of machines, this parallel approach can scale up to solve any large-scale clustering problem.

We select spectral clustering as our base algorithm because of its well-known effectiveness. The graph cut can be formulated as an eigenvalue decomposition problem of the graph Laplacian [5] by relaxing the labels to be real values. The graph Laplacian can be seen as an approximation of the Laplace-Beltrami operator on the manifold [3]. Representative spectral clustering methods include *Min Cut* [27], *Normalized Cut* [25], *Radio Cut* [15], *Min-Max Cut* [13] and *Co-Clustering* [6, 30]. Moreover, in a general relaxation view, graph cut, k-means, Principle Component Analysis (PCA) and Nonnegative Matrix Factorization (NMF) [18] (and their corresponding kernel versions) can be seen as a unified framework [7, 11, 12]. Many practical applications, such as image segmentation [25] and text categorization [6, 30], have proven to be well-suited spectral clustering problems.

Unfortunately, eigenvalue decomposition and k-means calculations are a bottleneck for spectral clustering. The memory use of eigenvalue decomposition is $\mathcal{O}(n^2)$, where n is the number of data instances. The time complexity for eigenvalue decomposition is $\mathcal{O}(n^3)$ at the worst case. When n is very large, say beyond a million, traditional single-machine speedup schemes [19, 14, 8, 24] still suffer from either memory or CPU limitations. Our parallel algorithm employs a parallel ARPACK algorithm (PARPACK) [21] to perform parallel eigenvalue decomposition. Although there have been several parallel eigenvalue or singular value decomposition techniques [20, 16, 17], the PARPACK algorithm has the following advantages: (1) It can be computed on distributed machines instead of only multi-core systems, and (2) it is fast when the matrix is sparse. Moreover, we implement a parallel k-means algorithm to cluster data in the eigenvector space. To reduce the memory use, our algorithm loads onto each machine only the necessary rows of data for conducting parallel computation. Empirical study shows our proposed algorithm to be both accurate and efficient.

Chu et al. [4] employed map reduce on multi-core computers and parallelized a variety of learning algorithms including k-means to get potential speedup. However, these solutions are implemented based on a shared memory, multi-core system. The limit of memory space is still there. The closest work to this paper is [9]. In [9], a parallelized k-means clustering algorithm which is also based on distributed memory is implemented. However, using k-means alone, it is not possible to deal with non-linearly separable data sets. Moreover, the time complexity of k-means grows linearly with the dimensionality of data, while spectral clustering does not have this problem. The eigenvalue decomposition procedure has the virtue of reducing dimensionality for k-means.

The rest of this paper is organized as follows: Section 2 introduces the spectral clustering and co-clustering algorithms. Section 3 describes our parallel spectral clustering algorithm. The experiment results are shown in Section 4. Section 5 discusses promising directions for future research.

2. SPECTRAL CLUSTERING

In this section, we briefly review the eigenvalue decomposition problem involved in both spectral clustering and co-clustering. This review introduces notations that are used in the remainder of the paper.

2.1 Spectral Analysis of Graph Cut

Consider $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as a weighted neighborhood graph which is constructed by the point cloud $X = (x_1, ..., x_n)$, where n is the point number. $\mathcal{V}$ is the vertex set of graph. $\mathcal{E}$ is the edge set which contains the pairs of neighboring vertices (x_i, x_j). A typical similarity matrix S of a neighborhood graph can be defined as:

$$S_{ij} = \begin{cases} S(x_i, x_j) & \text{if } (x_i, x_j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $S(x_i, x_j)$ is a similarity score such as a Gaussian kernel function. Then the graph Laplacian of a neighborhood graph is $L = D - S$, and the normalized graph Laplacian is $\bar{L} = I - D^{-\frac{1}{2}} S D^{-\frac{1}{2}}$, where the diagonal matrix D satisfies $D_{ii} = d_i$, and $d_i = \sum_{j=1}^{n} S_{ij}$ is the degree of vertex x_i [5].

Consider the normalized cut. We need to find subsets $\mathcal{A}$ and $\mathcal{B}$ such that the normalized cut criterion $\mathcal{J}_{NCut}(\mathcal{A}, \mathcal{B}) = \frac{cut(\mathcal{A},\mathcal{B})}{assoc(\mathcal{A},\mathcal{V})} + \frac{cut(\mathcal{B},\mathcal{A})}{assoc(\mathcal{B},\mathcal{V})}$ is minimized. It has been shown that the solution is given by optimizing the following criterion [25]:

$$f_L^* = \underset{f^T f_0 = 0}{\operatorname{argmin}} \frac{f^T L f}{f^T D f} \tag{2}$$

where $f = (f(x_1), f(x_2), ..., f(x_n))^T \in \mathcal{R}^{n \times 1}$. It can be solved by the second smallest eigenvector of the generalized system $Lf = \lambda D f$ where $f_0 = \vec{1}$ is the eigenvector corresponding to the smallest eigenvalue $\lambda_0 = 0$. Note that if we use the normalized graph Laplacian instead of the unnormalized one, it gives the solution as $f_{\bar{L}}^* = \underset{f^T f_0 = 0}{\operatorname{argmin}} \frac{f^T \bar{L} f}{f^T f}$. This solution is further related to (2) because $f_{\bar{L}}^* = D^{\frac{1}{2}} f_L^*$.

Note the following fact:

$$\operatorname{argmin} \frac{f^T \bar{L} f}{f^T f} = \operatorname{argmin} \frac{f^T (I - D^{-\frac{1}{2}} S D^{-\frac{1}{2}}) f}{f^T f} = \operatorname{argmax} \frac{f^T \bar{S} f}{f^T f}$$

where $\bar{S} = D^{-\frac{1}{2}} S D^{-\frac{1}{2}}$. Therefore, the spectral clustering problem can be solved in the scaled kernel PCA (KPCA) framework. The difference is that KPCA uses full connection graphs, while spectral clustering methods can use neighborhood graphs. The advantage of using neighborhood graphs is that their corresponding similarity matrices are sparse, and therefore many fast algorithms can be introduced.

2.2 Co-Clustering

For text categorization or community analysis problems, the word-by-document or user-by-community co-occurrence matrices can be used to generate a bipartite graph. Taking user-by-community co-occurrence as an example, the graph is defined as $\mathcal{G} = (\mathcal{U}, \mathcal{C}, \mathcal{E})$, where $\mathcal{U}$ denotes the set of user vertices, $\mathcal{C}$ denotes the set of community vertices and $\mathcal{E}$ is still the edge set. We can make use of co-clustering techniques to cluster users and communities simultaneously [6, 30]. Unlike the edges of traditional graphs, the edges of bipartite graph are only related to the co-occurrences, such that if a user i joins the community j, we set an edge connecting them.

It is not difficult to verify that the similarity matrix can be calculated based on the adjacency matrix

$$S = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}, \tag{3}$$

where $A \in R^{n \times n'}$ is the adjacency matrix that indicates the co-occurrence of the users and communities. n and n' are the number of communities and users respectively. For example, $A_{ij} \neq 0$ only if user i joins the community j.

Then the normalized graph Laplacian is

$$\bar{L} = \begin{bmatrix} I & -D_1^{-1/2} A D_2^{-1/2} \\ -D_2^{-1/2} A^T D_1^{-1/2} & I \end{bmatrix}, \tag{4}$$

where D_1 and D_2 are diagonal matrices, calculated as $(D_1)_{ii} = \sum_{j=1}^{n'} A_{ij}$ and $(D_2)_{jj} = \sum_{i=1}^{n} A_{ij}$.

By using eigenvalue decomposition of the normalized graph Laplacian $\bar{L} f = \lambda f$ where $f = (f_1^T, f_2^T)^T \in \mathcal{R}^{(n+n') \times 1}$, we obtain

$$\begin{aligned} D_1^{-1/2} A D_2^{-1/2} f_1 &= (1 - \lambda) f_2, \\ D_2^{-1/2} A^T D_1^{-1/2} f_2 &= (1 - \lambda) f_1. \end{aligned} \tag{5}$$

Performing the SVD technique shows that f_1 and f_2 are just the left and right singular vectors of matrix $D_1^{-1/2} A D_2^{-1/2}$.

The above analysis is all about the 2-way clustering problem. For the k-way (k is the number of clusters) clustering problem, many approaches have been proposed. For example, we can use the 2-way clustering algorithm to recursively partitioning the data $k-1$ times [25]. Other clustering algorithms, e.g. k-means, can be used to cluster the embedded points in the eigenvector space [23]. Moreover, eigenvectors can be discretized into class indicators by means of matrix decomposition [29]. Since k-means is a fast way to cluster data and can be easily parallelized, we select this way to obtain the final k-way clustering results.

3. PARALLEL SPECTRAL CLUSTERING

This section depicts our proposed parallel scheme to scale up spectral clustering.

3.1 Parallel Matrix Decomposition

Parallel matrix decomposition includes eigenvalue decomposition (EVD) and parallel singular value decomposition (SVD). We first present the EVD problem, and then we show how the SVD problem can be converted to the EVD problem.

Table 1: The traditional ARPACK algorithm, as used on a single machine to determine approximate eigenvectors for a large matrix S.

1. Input: an $n \times n$ matrix S.

2. Start: Build a length m Arnoldi factorization

$$SV_m = V_m H_m + f_m e_m^T \tag{6}$$

with the starting vector v_1, where V_m is an $n \times m$ matrix, with normalized orthogonal columns derived from the Krylov subspace. H_m is the projection matrix (upper Hessenberg). $f_m e_m^T$ is the residual vector with length n.

3. Iteration: Until convergence.

3.1. Compute the eigenvalues $\{\lambda_j : j = 1, 2, ...m\}$ of H_m. Sort these eigenvalues according to the user selection criterion into a wanted set $\{\lambda_j : j = 1, 2, ...k\}$, and an unwanted set $\{\lambda_j : j = k+1, k+2, ..., m\}$.

3.2. Perform $m - k = l$ steps of the QR iteration with the unwanted eigenvalues $\{\lambda_j : j = k+1, k+2, ..., m\}$, as shifts to obtain $H_m Q_m = Q_m H_m^+$, where H_m^+ is the projection matrix in the next iteration.

3.3. Restart: Postmultiply the length m Arnoldi factorization with the matrix Q_k consisting of the leading k columns of Q_m to obtain the length k Arnoldi factorization $SV_m Q_k = V_m Q_k H_k^+ + f_k^+ e_k^T$ where is H_k^+ is the leading principal submatrix of order k for H_m^+. Set $V_k \leftarrow V_m Q_k$.

3.4. Extend the length K Arnoldi factorization to a length m factorization.

4. Calculate the eigenvalues and eigenvectors of the small matrix H_k: The eigenvalues of H_k, $\{\lambda_j : j = 1, 2, ..., k\}$, is the approximation of S's eigenvalues. The eigenvectors of H_k is $\{e_j : j = 1, 2, ..., k\}$, and E_k is the matrix formed by e_j.

5. Given $SV_k \approx V_k H_k$, we can derive the approximate eigenvectors of S, $\{u_j : j = 1, 2, ..., k\}$, where u_j is the j^{th} column of matrix $V_k \cdot E_k$.

3.1.1 Parallel EigenValue Decomposition (EVD)

The traditional ARPACK algorithm (shown in Table 1) [19] calculates the approximated top k eigenvalues and the corresponding eigenvectors of a giant matrix. Given a matrix $S \in R^{n \times n}$, we build a length m Arnoldi factorization [2] as

$$SV_m = V_m H_m + f_m e_m^T \tag{7}$$

where $V_m \in R^{n \times m}$; $H_m \in R^{m \times m}$; $f_m e_m^T$ is the residual orthogonal to V_m. H_m is the projection of S in the space $Range(V_m)$. When $f_m e_m^T$ is small, H_m can be viewed as an approximation of S of dimension $m \times m$. Eigenvalues and eigenvectors of S can be calculated from H_m's eigenvalue decomposition:

$$\begin{aligned} SV_m &\approx V_m H_m \\ \lambda_j &\approx \delta_j, j \in \{1, 2, ..., m\} \\ u_j &\approx V_m e_j, j \in \{1, 2, ..., m\} \end{aligned} \tag{8}$$

where λ's are the eigenvalues of matrix S; δ's are the eigenvalues of matrix H_m; u_j is the j^{th} eigenvector of matrix S and e_j is the j^{th} eigenvector of matrix H_m.

To parallelize the process, the data and work space are segmented and loaded onto parallel machines:

- S is distributed across machines in a row-based, round-robin fashion.
- H_m is replicated on every machine.
- V_m is distributed across machines in a row-based, round-robin fashion.
- f_m and work space are distributed accordingly.

3.1.2 Distributed Matrix-Vector Multiplication

Our parallel algorithm, as compared to the single-machine algorithm, has the features that the local block of the set V_m^{local} is passed in place of V_m, and the dimension of the local block n_m^{local} is passed instead of n. Thus, we need to implement a matrix-vector multiplication to calculate the Krylov vectors. In our case, we divide the similarity matrix S into rows.

Figure 1 illustrates that matrix-vector multiplication on distributed computers. In each step, we first reduce each column of the Arnoldi vectors to a replicated vector using the standard message-passing interface. Then although the rows of the similarity matrix are distributedly stored on different machines, the products of each local row by the replicate Arnoldi vector can be locally computed. Therefore, the updated Arnoldi vectors are actually distributedly stored. The elements that correspond to the local rows of the similarity matrix are nonzero, whereas the other elements are still zeroes. By summing the results from all machines, matrix-vector multiplication is obtained.

Besides the matrix-vector multiplication, our algorithm requires two communications: Computing the L_2-norm of the distributed vector f_m, and orthogonalizing f_m to V_m. These can be performed by using the parallel computing summing interface.

3.1.3 Parallel Singular Value Decomposition (SVD)

For every rectangular matrix $A \in R^{n \times n'}$, a singular value decomposition exists:

$$A = USV^T, \tag{9}$$

where U (the left singular vectors) and V^T (the right singular vectors) are matrices with orthonormal columns and S (with singular values as the diagonal elements) is a diagonal

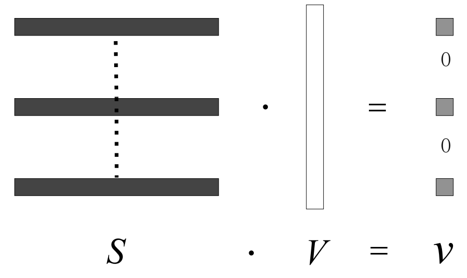


Figure 1: Illustration of the distributed matrix-vector multiplication.

matrix.

Given the Parallel EVD algorithm described in Section 3.1.1, we can calculate the SVD as follows:

$$A^T A = V S^2 V^T, \tag{10}$$

$$U = A V S^{-1}. \tag{11}$$

By calculating EVD on matrix $A^T A$ using Equation (10), we can obtain the right singular vectors in the matrix V^T and the singular values in the matrix S. Equation (11) gives a solution of left singular vectors U.

3.2 Parallel K-Means

The input to the k-means algorithm is the eigenvectors generated by the parallel EVD/SVD algorithm described in Section 3.1. The output of k-means is the cluster labels of each data point in the original data space.

Here, the k-means algorithm aims at minimizing the total intra-cluster variance, that is, the squared error function in the spectral space:

$$V = \sum_{i=1}^{k} \sum_{x_j \in C_i} ||x_j - \mu_i||^2, \tag{12}$$

where there are k clusters $C_i, \{i = 1, 2, ..., k\}$, and μ_i is the centroid or mean point of all the points $x_j \in C_i$.

We implemented the parallel k-means algorithm in such a way to minimize communication and maximize parallel computation. The algorithm's flowchart is shown in Fig. 2. In the parallel EVD algorithm, the output matrix U has been formed by the eigenvectors and distributed on all machines based on rows. Each row of the matrix U is regarded as one data point in the k-means algorithm. Thus, these data points are naturally distributed on the machines and we don't need to move them for k-means. To initialize the process, the master machine chooses a set of initial cluster centers and broadcasts the coordinates of the centers to all the machines. Every machine can work on its local data independently. New labels are assigned and local sums of clusters are calculated without any inter-machine communication. We again make use of the message-passing interface to combine the local information after each local machine has finished the computation. By gathering the statistical information (including the sum of data points in each cluster, the cluster numbers and the local cost values), every machine can update the cluster center coordinates and

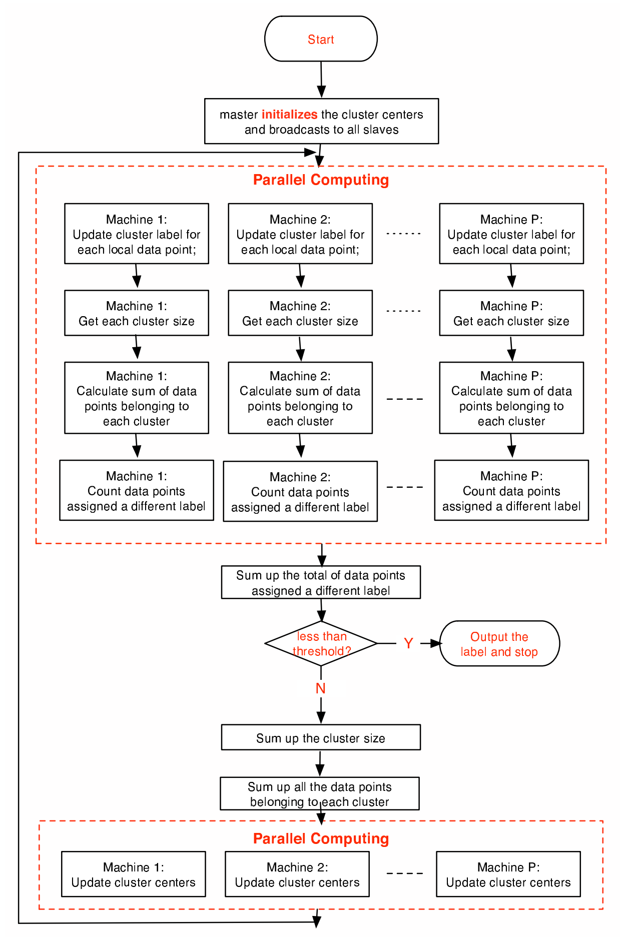

Start
master initializes the cluster centers and broadcasts to all slaves
Parallel Computing
Machine 1: Update cluster label for each local data point;
Machine 2: Update cluster label for each local data point;
Machine P: Update cluster label for each local data point;
Machine 1: Get each cluster size
Machine 2: Get each cluster size
Machine P: Get each cluster size
Machine 1: Calculate sum of data points belonging to each cluster
Machine 2: Calculate sum of data points belonging to each cluster
Machine P: Calculate sum of data points belonging to each cluster
Machine 1: Count data points assigned a different label
Machine 2: Count data points assigned a different label
Machine P: Count data points assigned a different label
Sum up the total of data points assigned a different label
less than threshold?
Y
Output the label and stop
N
Sum up the cluster size
Sum up all the data points belonging to each cluster
Parallel Computing
Machine 1: Update cluster centers
Machine 2: Update cluster centers
Machine P: Update cluster centers

Figure 2: The parallel k-means clustering algorithm.

start a new round of computation until the computation converges. The output cluster labels for data points in the spectral space are mapped to the original data space.

3.3 Complexity Comparison

Our algorithm is shown in Table 2. We can see that steps 4 and 5 are the key parallelization steps. For step 3, we have not appointed the form of the scaled similarity matrix $\bar{S}$. If we make use of the original similarity $\bar{S} = X^T X$ we obtain the relaxed version of k-means. If we make use of $\bar{S} = G$ where G is the Gram matrix computed by the kernel function, then we have the relaxed kernel k-means algorithm. If the matrix $\bar{S}$ is constructed by a graph similarity matrix, which can be either fully connected (can be the same as kernel k-means) or a neighborhood graph, it is the min-cut algorithm. If we make use of the normalized similarity matrix $\bar{S} = D^{-\frac{1}{2}} S D^{-\frac{1}{2}}$, it is the normalized cut algorithm. For the co-clustering problem, we input the matrix $\bar{A} = D_1^{-\frac{1}{2}} A D_2^{-\frac{1}{2}}$, then compute $\bar{A}\bar{A}^T$ as $\bar{S}$. We summarize the above analysis in Table 3.

Now, we analyze memory use and computation intensity. We use n to denote the number of total data points, d to denote the dimensionality and k to denote the number of clusters. Here we introduce a new variable z. Since we assume the data similarity matrix is sparsely stored, we use

Table 2: The parallel spectral clustering algorithm

1. Each computer loads a set of rows of the similarity matrix S into memory.
2. Multiply the matrix S with vector $\vec{1} = [1, 1, ..., 1]^T$. The product vector is the diagonal elements of the matrix D.
3. Calculate the scaled similarity matrix $\bar{S}$.
4. Compute the approximated eigenvalue decomposition of $\bar{S}$ using parallel matrix decomposition.
5. Use parallel k-means to cluster the rows of matrix U.
6. Map the cluster labels to original data points.

Table 3: Spectral clustering matrix comparison

Form of $\bar{S}$	Method
X^TX	Relaxed k-means
Gram matrix G	Relaxed kernel k-means
Similarity matrix on graph	Min-cut
$D^{-\frac{1}{2}}SD^{-\frac{1}{2}}$	Normalized cut
$\bar{A}\bar{A}^T$ where $\bar{A} = D_1^{-\frac{1}{2}}AD_2^{-\frac{1}{2}}$	Co-clustering

z to denote the average number of rows in the similarity matrix. For the iterated algorithms, we let i_{iter} denote the iteration time. If we have p distributed machines, the computational complexity of the key steps are as follows:

k-means. For the traditional k-means algorithm, the memory requirement is $\mathcal{O}(nd)$ and the computational complexity is $\mathcal{O}(ndk \cdot i_{iter})$, in which we need to compute the Euclidean distance between every point and every cluster center.

Parallel k-means. For parallel k-means, the memory requirement is reduced to $\mathcal{O}(\frac{nd}{p})$ for each computer and the computation time reduced to $\mathcal{O}(\frac{ndk}{p} \cdot i_{iter})$. Since the parallel algorithm also involves communication among computers, we need to estimate the communication time. Most of the calculation is done in parallel. Only summation is performed repeatedly on each machine. Therefore, the communication time is $\mathcal{O}(pkd \cdot i_{iter})$.

Spectral Clustering. For spectral clustering based on the Arnoldi method, the memory complexity of loading the similarity matrix and eigenvectors is $\mathcal{O}(n(z+k))$. The time complexity of calculating the eigenvalue decomposition of the similarity matrix is $\mathcal{O}(nzk \cdot i_{iter})$.

Parallel Spectral Clustering. For our parallel spectral clustering algorithm, the memory usage on each machine is $\mathcal{O}(\frac{n(z+k)}{p})$ and the computation cost is $\mathcal{O}(\frac{nzk \cdot i_{iter}}{p})$. Moreover, since we should compute the Arnoldi vector using the message-passing interface, the communication cost is $\mathcal{O}(pnk \cdot i_{iter})$.

All costs are summarized in Table 4.

4. EMPIRICAL STUDY

We first conducted experiments on artificial datasets to examine the accuracy and time cost of our parallel algorithm; we then performed scalability experiments on a large real-world dataset. We ran all our experiments on Google's distributed production data centers.

Table 4: Computation cost comparison. "P. k-means" represents parallel k-means," S. C." represents spectral clustering and "P. S. C." represents parallel spectral clustering.

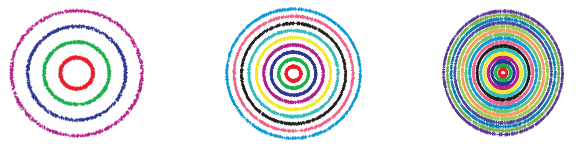
Method	Memory	Comp. Time	Comm. Time
k-means	$\mathcal{O}(nd)$	$\mathcal{O}(ndk \cdot i_{iter})$	-
P. k-means	$\mathcal{O}(\frac{nd}{p})$	$\mathcal{O}(\frac{ndk}{p} \cdot i_{iter})$	$\mathcal{O}(pdk \cdot n_{iter})$
S. C.	$\mathcal{O}(n(n+k))$	$\mathcal{O}(nzk \cdot i_{iter})$	-
P. S. C.	$\mathcal{O}(\frac{n(z+k)}{p})$	$\mathcal{O}(\frac{nzk \cdot i_{iter}}{p})$	$\mathcal{O}(pnk \cdot i_{iter})$

4.1 Accuracy Study

For accuracy experiments, we collected nine datasets with different sizes and numbers of clusters. These nine datasets consist of 1k, 10k, and 100k data points distributed on 4, 9 and 16 non-overlapping circles, as shown in Table 5. We denote these datasets to be from C1 to C9.

Table 5: Description of data sets

	4 clusters	9 clusters	16 clusters
1K data points	C1	C4	C7
10K data points	C2	C5	C8
100K data points	C3	C6	C9



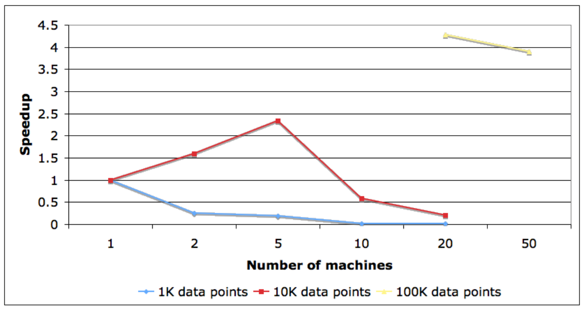
(a) 4 classes. (b) 9 classes. (c) 16 classes.

Figure 3: Artificial testing data sets.

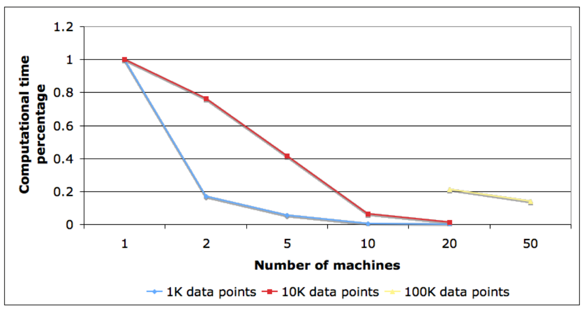
Fig. 3 shows three of the above nine datasets for the purpose of illustration. Pair-wise similarity between two data points is calculated using an RBF kernel function. The width of the RBF kernel is tuned by the self-tuning technique of [28]. Then the RBF is modified as

$$S_{ij} = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma_i\sigma_j}\right), \qquad (13)$$

where σ_i is computed as the distance between x_i and the k's neighborhood of x_i: $\sigma_i = ||x_i - x_{i_k}||$. In this paper, for neighborhood graphs, we set k as half of the neighborhood number.



(a) Algorithm speedup of different scale of data.



(b) Ratio between computation time and communication time.

Figure 4: Time analysis of parallel spectral clustering.

4.1.1 The Speedup Measure: Speedup Factor

The running time using p machines is shorter than that of using one machine, ideally in a linear manner. However, because of the communication overhead, the speedup is usually not linear. We define the speedup factor as follows:

$$speedup = \frac{T_1}{T_p}, \qquad (14)$$

where T_1 is the execution time using one machine, and T_p is the execution time using p machines.

4.1.2 Results

We applied parallel spectral clustering on all the artificial data sets. Again, the purpose of this experiment is to evaluate the accuracy of the clustering results. (Using multiple machines on a small dataset does not yield much benefit as we will show shortly.) We compared the clusters generated by the original spectral clustering algorithm and our parallel version, and they yield the identical results.

We document the running time of these nine datasets in Table 6[1]. Each dataset was run on 1, 2, 5, 10, 20, and 50 machines, respectively. As predicted, when the dataset size is very small, the running time of C1, C4, and C7 shows that adding machines actually increases total running time. This is because inter-machine communication incurs higher cost than that the parallelization can save. When the dataset size grows from $1k$ to $10k$, parallelization yields benefit. When using up to 10 machines, C8 enjoys a speedup of about 2.2 times. When the dataset continues to grow beyond what the main memory of one machine can store, we have to employ enough machines to do the job. On datasets C3, C6, and C9, we can complete the clustering task only when 20 or 50 machines are used.

Given the total time spent on each task, we can calculate the speedup using Equation (14). The results are shown in Fig. 4(a). When the problem scale grows, the speedup can be more significant. This implies that our parallel spectral clustering algorithm is more efficient in large-scale problems than small ones. Fig. 4(b) shows the percentage of time spent on computation. The main factor, which affects the percentage of computation time, is the problem scale. Using a fixed number of machines, the percentage of computation time of problem-scale $10k$ is larger than that of the three $1k$ datasets. This again justifies that our algorithm is more efficient on large-scale problems.

[1]Since we conducted experiments on Google's production data centers, we could not ensure the quiesce of each machine. Therefore, the running time is partially dependent on the slowest machine being allocated for the task.

Table 6: Algorithm running time on different data sets using multiple machines

Data	Number of machines					
	1	2	5	10	20	50
C1	$2.952s$	$7.709s$	$21.70s$	$465.0s$	$503.2s$	
C2	$199.5s$	$139.8s$	$58.70s$	$62.13s$	$589.1s$	
C3	NA	NA	NA	NA	NA	$343.4s$
C4	$1.936s$	$5.548s$	$21.89s$	$120.2s$	$232.1s$	
C5	$140.96s$	$67.63s$	$51.71s$	$283.6s$	$91.72s$	
C6	NA	NA	NA	NA	$558.5s$	$348.8s$
C7	$1.570s$	$5.452s$	$20.43s$	$17.65s$	$52.36s$	
C8	$281.22s$	$255.80s$	$185.92s$	$132.77s$	$491.9s$	
C9	NA	NA	NA	NA	$757.3s$	$820.4s$

4.2 Experiments on Text Data

In this experiment, we made use of the pre-processed 20-newsgroups data set given in [31] to test the accuracy of our parallel algorithm. The data set originally included $20,000$ messages within 20 different newsgroups. The data was pre-processed by the Bow toolkit [22]. It was chopped off the headers, removed stop words and also the words that occurred in fewer than three documents [31]. Thus, the document is represented by a feature which is a $43,586$ dimensional sparse vector. Several empty documents were also removed [31]. Finally we obtained $19,949$ examples.

For comparison of different results, we used $Normalized\ Mutual\ Information$ (NMI) to evaluate the algorithms. NMI between two random variables Y_1 and Y_2 is defined as $NMI(Y_1;Y_2) = \frac{I(Y_1;Y_2)}{\sqrt{H(Y_1)H(Y_2)}}$, where $I(Y_1;Y_2)$ is the mutual information between Y_1 and Y_2. The entropies $H(Y_1)$ and $H(Y_2)$ are used for normalizing the mutual information to be in the range $[0,1]$. In practice, we made use of the following formulation to estimate the NMI score [26, 31]:

$$NMI = \frac{\sum_{s=1}^{K}\sum_{t=1}^{K} n_{s,t}\log\left(\frac{n n_{s,t}}{n_s \cdot n_t}\right)}{\sqrt{\left(\sum_s n_s \log\frac{n_s}{n}\right)\left(\sum_t n_t \log\frac{n_t}{n}\right)}} \qquad (15)$$

Table 7: Comparison result for text categorization

Method	NMI
E-k-means	0.10±7.0e-05
S-k-means	0.30±1.6e-06
Co-clustering	0.54±3.6e-06
Normalized cut	0.55±4.9e-05

where n is the number of data points, n_s and n_t denotes the number of data points in class s and cluster t, $n_{s,t}$ denotes the number of data points in class s as well as in cluster t. The NMI score is 1 if the clustering results perfectly match the category labels; it is 0 means the clustering algorithm returns a random partition. Thus, the larger this score, the better the clustering results.

We compared the following algorithms: relaxed k-means algorithm based on Euclidean distance (E-k-means); relaxed spherical k-means based on cosine distance (S-k-means) [10]; co-clustering algorithm [6]; normalized cut algorithm using 30-neighborhood adjacency graph (without weights on graph edges) [25]. The results are shown in Table 7. We see that normalized cut performs the best. (We applied parallel normalized cut on the 20k documents using 5 machines and it only took around 10 seconds to complete.)

4.3 Experiments on Orkut Data

Web communities have become increasingly popular. The development of those Web communities has facilitated people in knowing new friends with common interests. User can create communities as well as join existing communities on the Web sites. Orkut is an Internet social network service run by Google. Since October 2006, Orkut has permitted users to create accounts without an invitation, and now Orkut has more than 50 million users and 20 million communities.

We used Orkut's user-by-community co-occurrence data. All the users are anonymized and each community is associated with a name and an optional description. To make the clustering result readable, we first filtered out the non-English-language communities. We also removed those inactive communities containing few users. We obtained $151,973$ communities with more than 10-million users involved.

We ran our parallel algorithm on 90 machines to group the communities into 100 clusters. The program finished within 20 minutes. Communities with similar topics are clustered together. We choose four clusters among the clustering results. Popular communities are listed in Table 8 as representative examples of the clusters.

5. CONCLUSIONS AND FUTURE WORK

This paper presented a parallel approach for spectral graph analysis, including spectral clustering and co-clustering. The scalability problem of spectral methods has been improved in both computation time and memory use per machine. This extension makes it possible to analyze Web-scale data using spectral methods. Experiments show that our parallel approach performs accurately on artificial datasets and real text data. We also applied our algorithm on a large Orkut dataset to demonstrate its scalability.

In future work we plan to further reduce inter-machine communication cost to improve scalability. We also plan to investigate incremental methods for community mining and discovery to further speed up performance.

6. REFERENCES

[1] Orkut: http://www.orkut.com/home.aspx.

[2] W. Arnoldi. The principle of minimized iteration in the solution of matrix eigen value problems. *Quarterly of Applied Mathematics*, 9:17–29, 1951.

[3] M. Belkin and P. Niyogi. Towards a theoretical foundation for laplacian-based manifold methods. In *Proceedings of COLT*, pages 486–500, 2005.

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

[7] I. S. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of KDD*, pages 551–556, 2004.

[8] 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.

[9] 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.

[10] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1–2):143–175, 2001.

[11] C. H. Q. Ding and X. He. K-means clustering via principal component analysis. In *Proceedings of ICML*, pages 225–232, 2004.

[12] C. H. Q. Ding and X. He. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of SDM*, pages 606–610, 2005.

[13] 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*, pages 107–114, 2001.

[14] 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.

[15] 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.

[16] E. R. Jessup and D. C. Sorensen. A parallel algorithm for computing the singular value decomposition of a matrix. *SIAM Journal on Matrix Analysis Applications*, 15(2):530–548, 1994.

[17] T. Konda, M. Takata, M. Iwasaki, and Y. Nakamura. A new singular value decomposition algorithm suited to parallelization and preliminary results. In

Table 8: Cluster examples

Sample Cluster 1: *Cars*		**Sample Cluster 2:** *Food*	
Community ID	Community title	Community ID	Community title
22527	Honda CBR	622109	Seafood Lovers
287892	Mercedes-Benz	20876960	Gol gappe
35054	Valentino Rossi	948798	I LOVE ICECREAM ...YUMMY!!!!!!
5557228	Pulsar Lovers	1614793	Bounty
2562120	Top Speed Drivers	1063561	Old Monk Rum
19680305	The Art of DriftIng!!!!!!	970273	Fast Food Lovers
3348657	I Love Driving	14378632	Maggi Lovers
726519	Luxury & Sports Cars	973612	Kerala Sadya -Mind Blowing
2806166	Hero Honda Karizma	16537390	Baskin-Robbins Icecream Lovers
1162256	Toyota Supra	1047220	Oreo Freax!!

Sample Cluster3: *Education*		**Sample Cluster4:** *Pets, animals and wildlife*	
Community ID	Community title	Community ID	Community title
15284191	Bhatia Commerce Classes	18341	Tigers
7349400	Inderprastha Engineering Cllge	245877	German shepherd
1255346	CCS University Meerut	40739	Naughty dogs
13922619	Visions - SIES college fest	11782689	We Love Street Dogs
2847251	Rizvi College of Engg., Bandra	29527	Animal welfare
6386593	Seedling public school, jaipur	370617	Lion
4154	Pennsylvania State University	11577	Arabian horses
15549415	N.M. College, Mumbai	2875608	Wildlife Conservation
1179183	Institute of Hotel Management	12522409	I Care For Animals
18963916	I Love Sleeping In Class	1527302	I hate cockroaches

Proceedings of the 2nd IASTED international conference on Advances in computer science and technology, pages 79–84, 2006.

[18] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Proceedings of NIPS*, pages 556–562, 2000.

[19] R. B. Lehoucg, D. C. Sorensen, and C.Yang. *ARPACK User's Guide: Solution of large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*. SIAM, 1998.

[20] F. T. Luk. A parallel method for computing the generalized singular value decomposition. *Journal of Parallel and Distributed Computing*, 2(3):250–260, 1985.

[21] 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.

[22] A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. In *Technical Report*. http://www.cs.cmu.edu/ mccallum/bow, 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] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang. Incremental spectral clustering with application to monitoring of evolving blog communities. In *Proceeding of SDM*, 2007.

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

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

[27] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: theory andits application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

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

[29] 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.

[30] H. Zha, X. He, C. H. Q. Ding, M. Gu, and H. D. Simon. Bipartite graph partitioning and data clustering. In *Proceedings of CIKM*, pages 25–32, 2001.

[31] S. Zhong and J. Ghosh. Generative model-based clustering of documents: a comparative study. *Knowledge and Information Systems*, 8:374–384, 2005.