Parallel Spectral Clustering

Yangqiu Song[1,4], Wen-Yen Chen[2,4], Hongjie Bai[4],
Chih-Jen Lin[3,4], and Edward Y. Chang[4]

[1]Department of Automation, Tsinghua University, Beijing, China
[2]Department of Computer Science, University of California, Santa Barbara, USA
[3]Department of Computer Science, National Taiwan University, Taipie, Taiwan
[4]Google Research, USA/China
songyq99@mails.tsinghua.edu.cn; wychen@cs.ucsb.edu; hjbai@google.com;
cjlin@csie.ntu.edu.tw; edchang@google.com

Abstract. Spectral clustering algorithm has been shown to be more effective in finding clusters than most traditional algorithms. However, spectral clustering suffers from a scalability problem in both memory use and computational time when a dataset size is large. To perform clustering on large datasets, we propose to parallelize both memory use and computation on distributed computers. Through an empirical study on a large document dataset of 193,844 data instances and a large photo dataset of 637,137, we demonstrate that our parallel algorithm can effectively alleviate the scalability problem.

Key words: Parallel spectral clustering, distributed computing

1 Introduction

Clustering is one of the most important subroutine in tasks of machine learning and data mining. Recently, spectral clustering methods, which exploit pairwise similarity of data instances, have been shown to be more effective than traditional methods such as k-means, which considers only the similarity 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 encounters a quadratic resource bottleneck in computing pairwise similarity between n data instances and storing that large matrix. Moreover, the algorithm requires considerable computational time to find the smallest k eigenvalues of a Laplacian matrix.

Several efforts have attempted to address aforementioned issues. Fowlkes et al. propose using the Nyström approximation to avoid calculating the whole similarity matrix [8]. That is, they trade accurate similarity values for shortened computational time. Dhillon et al. [4] assume the availability of the similarity matrix and propose a method that does not use eigenvectors. Although these methods can reduce computational time, they trade clustering accuracy for computational speed gain, or they do not address the bottleneck of memory use. In

Table 1. 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
$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in R^d$	data points
$S \in R^{n \times n}$	similarity matrix
$L \in R^{n \times n}$	Laplician matrix
$\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in R^n$	first k eigenvectors of L
$V \in R^{n \times k}$	eigenvector matrix
$\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k \in R^n$	cluster indicator vectors
$E \in R^{n \times k}$	cluster indicator matrix
$\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k \in R^d$	cluster centers of k-means

this paper, we parallelize spectral clustering on distributed computers to address resource bottlenecks of both memory use and computation time. Parallelizing spectral clustering is much more challenging than parallelizing k-means, which was performed by e.g., [2, 5, 25].

Our parallelization approach first distributes n data instances onto p distributed machine nodes. On each node, we then compute the similarities between local data and the whole set in a way that uses minimal disk I/O. These two steps, together with parallel eigensolver and distributed tuning of parameters (including σ of the Gaussian function and the initial k centers of k-means), speed up clustering time substantially. Our empirical study validates that our parallel spectral clustering outperforms k-means in finding quality clusters and that it scales well with large datasets.

The remainder of this paper is organized as follows: In Section 2, we present spectral clustering and analyze its memory and computation bottlenecks. In Section 3, we show some obstacles for parallelization and propose our solutions to work around the challenges. Experimental results in Section 4 show that our parallel spectral clustering algorithm achieves substantial speedup on 128 machines. The resulting cluster quality is better than that of k-means. Section 5 offers our concluding remarks.

2 Spectral Clustering

We present the spectral clustering algorithm in this section so as to understand the bottlenecks of its resources. To assist readers, Table 1 defines terms and notations used throughout this paper.

2.1 Basic Concepts

Given n data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, the spectral clustering algorithm constructs a similarity matrix $S \in R^{n \times n}$, where $S_{ij} \geq 0$ reflects the relationships between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. It then uses the similarity information to group $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ into k clusters. There are many variants of spectral clustering. Here we consider a commonly used *normalized* spectral clustering [19]. (For a complete account of all variants, please see [17].) An example similarity function is the Gaussian:

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

In our implementation, we use an adaptive approach to decide the parameter σ^2 (details are presented in Section 3.4). For conserving computational time, one often reduces the matrix S to a sparse one by considering only significant relationship between data instances. For example, we may retain S_{ij} satisfying that j (or i) is among the t-nearest neighbors of i (or j). Typically t is a small number (e.g., t a small fraction of n or $t = \log n$)[1].

Consider the normalized Laplacian matrix [3]:

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

where D is a diagonal matrix with

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

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

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

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

$$E = \begin{bmatrix} \boldsymbol{e}_1 & & \\ & \ddots & \\ & & \boldsymbol{e}_k \end{bmatrix}, \tag{4}$$

[1] Another simple strategy for making S a sparse matrix is to zero out those S_{ij} smaller than a pre-specified threshold. Since the focus of this paper is on speeding up spectral clustering, we do not compare different methods to make a matrix sparse. Nevertheless, our empirical study shows that the t-nearest-neighbor approach yields good results.

Algorithm 1 Spectral Clustering

Input: Data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$; 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 $V \in R^{n \times k}$, which columns are the k eigenvectors.
5. Compute the normalized matrix U of V by Eq. (5).
6. Use k-means algorithm to cluster n rows of U into k groups.

where $\boldsymbol{e}_i, i = 1, \ldots, k$ (in different length) are vectors of all ones. As $D^{1/2}E$ has the same structure as E, simple clustering algorithms such as k-means can easily cluster the n rows of V into k groups. Thus, what one needs 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. [19] propose normalizing V so that

$$U_{ij} = \frac{V_{ij}}{\sqrt{\sum_{r=1}^{k} V_{ir}^2}}, i = 1, \ldots, n, j = 1, \ldots, k. \tag{5}$$

Each row of U has unit length. Due to the orthogonality of Q, (5) is equivalent to

$$U = EQ = \begin{bmatrix} Q_{1,1:k} \\ \vdots \\ Q_{1,1:k} \\ Q_{2,1:k} \\ \vdots \end{bmatrix}, \tag{6}$$

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 or other simple clustering algorithms. A summary of the method is presented in Algorithm 1.

Instead of analyzing properties of the Laplacian matrix, spectral clustering algorithms can 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 [20], Min-Max Cut [7] and Ratio Cut [9].

2.2 Computational Complexity and Memory Usage

Let us examine computational cost and the memory use of Algorithm 1. We omit discussing some inexpensive steps.

Construct the similarity matrix. Assume each S_{ij} involves at least an inner product between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. The cost of obtaining an S_{ij} is $O(d)$, where d is the dimensionality of data. Constructing similarity matrix S requires

$$O(n^2 d) \text{ time and } O(n^2) \text{ memory.} \tag{7}$$

To make S a sparse matrix, we employ the approach of t-nearest neighbors and retain only S_{ij} where i (or j) is among the t-nearest neighbors of j (or i). By scanning once of S_{ij} for $j = 1, \ldots, n$ and keeping a max heap with size t, we sequentially insert the similarity that is smaller than the maximal value of the heap and then restructure the heap. Thus, the complexity for one point $\boldsymbol{x}_i$ is $O(n \log t)$ since restructuring a max heap is in the order of $\log t$. The overall complexity of making the matrix S to sparse is

$$O(n^2 \log t) \text{ time and } O(nt) \text{ memory.} \tag{8}$$

Compute the first k eigenvectors. Once that S is sparse, 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 [11, 13] (see [10] for a comprehensive survey). Most existing approaches are variants of the Lanczos/Arnoldi factorization. We employ a popular eigensolver ARPACK [13] and its parallel version PARPACK [18]. ARPACK implements an implicitly restarted Arnoldi method. We briefly describe its basic concepts hereafter; more details can be found in the user guide of ARPACK. The m-step Arnoldi factorization gives that

$$LV = VH + (\text{a matrix of small values}), \tag{9}$$

where $V \in R^{n \times m}$ and $H \in R^{m \times m}$ satisfy certain properties. If the "matrix of small values" in (9) is indeed zero, then V's m columns are L's first m eigenvectors. Therefore, (9) provides a way to check how good we approximate eigenvectors of L. To perform this check, one needs all eigenvalues of the dense matrix H, a procedure taking $O(m^3)$ operations. For quickly finding the first k eigenvectors, ARPACK employs an iterative procedure called "implicitly restarted" Arnoldi. Users specify an Arnoldi length $m > k$. Then at each iteration (restarted Arnoldi) one uses V and H of the previous iteration to conduct the eigendecomposition of H, and finds a new Arnoldi factorization. Each Arnoldi factorization 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

$$L\boldsymbol{v}, \tag{10}$$

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 multiply is $O(nt)$.

Based on the above analysis, the overall cost of ARPACK is

$$\big(O(m^3) + (O(nm) + O(nt)) \times O(m - k)\big) \times (\#\text{ restarted Arnoldi}), \tag{11}$$

where $O(m-k)$ is a value no more than $m-k$. Obviously, the value m selected by users 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. Algorithm k-means aims at minimizing the total intra-cluster variance, which is the squared error function in the spectral space:

$$J = \sum_{i=1}^{k} \sum_{\boldsymbol{u}_j \in C_i} ||\boldsymbol{u}_j - \boldsymbol{c}_i||^2. \tag{12}$$

We assume that data are in k clusters $C_i, \{i = 1, 2, \ldots, k\}$, and $\boldsymbol{c}_i \in R^{k\times 1}$ is the centroid of all the points $\boldsymbol{u}_j \in C_i$. Similar to Step 5 in Algorithm 1, we also normalize centers $\boldsymbol{c}_i$ to be of unit length.

The traditional k-means algorithm employs an iterative procedure. At each iteration, we assign each data point to the cluster of its nearest center, and recalculate cluster centers. The procedure stops after reaching a stable error function value. Since the algorithm evaluates the distance between any point and the current k cluster centers, the time complexity of k-means is

$$O(nk^2) \times \#\ k\text{-means iterations}. \tag{13}$$

Overall analysis. The step that consumes the most memory is constructing the similarity matrix. For instance, $n = 600,000$ data instances, assuming double precision storage, requires 2.8 Tera Bytes of memory, which is not available on a general-purpose machine. Since we make S sparse, $O(nt)$ memory space may suffice. However, if n is huge, say in billions, no single general-purpose machine can handle such a large memory requirement. Moreover, the $O(n^2d)$ computational time in (7) is a bottleneck. This bottleneck has been discussed in earlier work. For example, the authors of [16] 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*."

3 Parallel Spectral Clustering

Based on the analysis performed in Section 2.2, it is essential to conduct spectral clustering in a distributed environment to alleviate both memory and computational bottlenecks. In this section, we discuss these challenges and then propose our solutions. We implement our system on a distributed environment using Message Passing Interface (MPI) [22].

3.1 Similarity Matrix and Nearest Neighbors

Suppose p machines (or nodes) are allocated in a distributed environment for our target clustering task. Figure 1 shows that we first let each node construct n/p rows of the similarity matrix S. We illustrate our procedure using the first node, which is responsible for rows 1 to n/p. To obtain the i^{th} row, we use Eq. (1) to

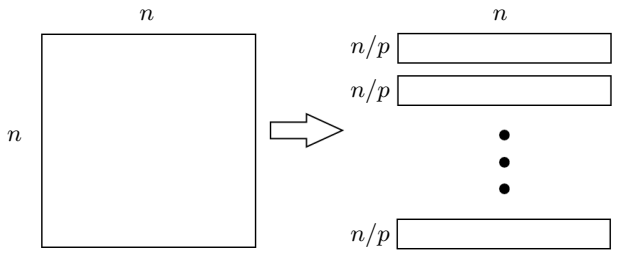


Fig. 1. The similarity matrix is distributedly stored in multiple machines.

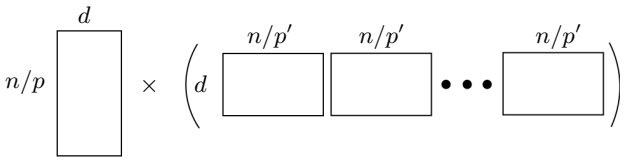


Fig. 2. Calculating n/p rows of the similarity at a node. We use matrix-matrix products for inner products between n/p points and all data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. As data cannot be loaded into memory, we separate $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ into p' blocks.

calculate the similarity between $\boldsymbol{x}_i$ and all the data points, respectively. Using $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 = \|\boldsymbol{x}_i\|^2 + \|\boldsymbol{x}_j\|^2 - 2\boldsymbol{x}_i^T\boldsymbol{x}_j$ to compute similarity between instances $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, we can precompute $\|\boldsymbol{x}_i\|^2$ for all instances and cache on all nodes to conserve time.

Let $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in R^{d \times n}$ and $X_{1:n/p} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n/p}]$. One can perform a matrix-matrix product to obtain $X_{1:n/p}^T X$. If the memory of a node cannot store the entire X, we can split X into p' blocks as shown in Figure 2. When each of the p' blocks is memory resident, we multiply it and $X_{1:n/p}^T$.

When data are densely stored, even if X can fit into the main memory, splitting X into small blocks takes advantage of optimized BLAS (Basic Linear Algebra Subroutines) [1]. BLAS places the inner-loop data instances in CPU cache and ensures their cache residency. Table 2 compares the computational time with and without BLAS. It shows that blocking operation can reduce the computational time significantly.

3.2 Parallel Eigensolver

After we have calculated and stored the similarity matrix, it is important to parallelize the eigensolver. Section 3.1 shows that each node now stores n/p rows of L. For the eigenvector matrix V (see (3)) generated during the call to ARPACK, we also split it into p partitions, each of which possesses n/p rows. As mentioned in Section 2.2, major operations at each step of the Arnoldi factorization include a few dense and a sparse matrix-vector multiplications, which cost $O(mn)$ and $O(nt)$, respectively. We parallelize these computations so that the complexity of

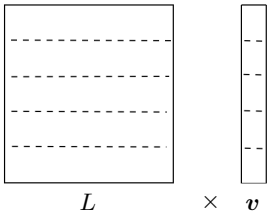


Fig. 3. Sparse matrix-vector multiplication. We assume $p = 5$ here. L and $\boldsymbol{v}$ are respectively separated to five block partitions.

Table 2. Computational time (in seconds) for the similarity matrix ($n = 637,137$ and number of features $d = 144$).

1 machine without BLAS	1 machine with BLAS	16 machines with BLAS
3.14×10^5	6.40×10^4	4.00×10^3

finding eigenvectors becomes:

$$\left(O(m^3) + (O(\frac{nm}{p}) + O(\frac{nt}{p})) \times O(m-k)\right) \times (\#\text{ restarted Arnoldi}). \quad (14)$$

Note that 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 (10).
3. 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. We next discuss how to conduct the parallel sparse matrix-vector product to reduce communication cost.

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

$$L_{1:n/p,1:n} \times \boldsymbol{v}, \quad (15)$$

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 at node 1, but only $v_1, \ldots, v_{n/p}$ are available at node 1. 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: data at each node are gathered on all nodes. We apply this MPI operation on distributed computers by following the techniques in MPICH2[2] [24], a popular implementation of MPI. The communication cost is $O(n)$, which cannot be reduced as a node must get n/p elements from the other $p-1$ nodes.

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

Further reducing the communication cost is possible if we reduce n to a fraction of n by taking 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.

3.3 Parallel k-means

After the eigensolver computes the first k eigenvectors of Laplacian, 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. (See next section for our distributed initialization procedure.) At each node, new labels of its data are assigned 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. The loss function (12) 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 (13). The communication cost per iteration is on broadcasting k centers to all machines. If k is not large, the total communication cost is usually smaller than that involved in finding the first k eigenvectors.

3.4 Other Implementation Details

We discuss two implementation issues of the parallel spectral clustering algorithm. The first issue is that of assigning parameters in Gaussian function (1), and the second is initializing the centers for k-means.

Parameters in Gaussian function. We adopt the self-tuning technique [27] to adaptively assign the parameter σ in (1). The original method used in [27] is

$$S_{ij} = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma_i\sigma_j}\right). \tag{16}$$

Suppose $\boldsymbol{x}_i$ has t nearest neighbors. If we sort these neighbors in ascending order, σ_i is defined as the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i_t}$, the $\lfloor t/2 \rfloor$th neighbor of $\boldsymbol{x}_i$: $\sigma_i = ||\boldsymbol{x}_i - \boldsymbol{x}_{i_t}||$. Alternatively, we can consider the average distance between $\boldsymbol{x}_i$ and its t nearest neighbors[3]. In a parallel environment, each local machine first computes σ_i's of its local data points. Then σ_i's are gathered on all machines.

Initialization of k-means. Revisit (6). 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 $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ whose elements are almost orthogonal [26]. 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

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

center by finding the minimal cosine distance between its points and the center. By gathering the information of different machines, we choose the most orthogonal point to the first center as the second center. This procedure is repeated to obtain k centers. The communication involved in the initialization includes broadcasting k cluster centers and gathering $k \times p$ minimal cosine distances.

4 Experiments

We designed our experiments to validate the quality of parallel spectral clustering and its scalability. Our experiments used two large datasets: 1) RCV1 (Reuters Corpus Volume I), a filtered collection of 193,844 documents, and 2) 637,137 photos collected from PicasaWeb, a Google photo sharing product. We ran experiments on up to 256 machines at our distributed data centers. While not all machines are identical, each machine was configured with a CPU faster than 2GHz and memory larger than 4GBytes. All reported results are the average of nine runs.

4.1 Clustering Quality

To check the performance of spectral clustering algorithm, we compare it with traditional k-means. We looked for a dataset with ground truth. RCV1 is an archive of 804,414 manually categorized newswire stories from Reuters Ltd [14]. 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 and 103 categories. Each document is represented by a cosine normalization of a log transformed TF-IDF (term frequency, inverse document frequency) feature vector.

For both spectral and k-means, we set the number of clusters to be 103, and Arnoldi space dimension m to be two times the number of clusters. We used the *document categories* in the RCV1 dataset as the ground truth for evaluating cluster quality. We measured quality via using the Normalized Mutual Information (NMI) between the produced clusters and the ground-truth categories.

NMI between two random variables CAT (category label) and CLS (cluster label) is defined as $\mathrm{NMI(CAT;\ CLS)} = \frac{\mathrm{I(CAT;\ CLS)}}{\sqrt{\mathrm{H(CAT)H(CLS)}}}$, 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 [23]:

$$\mathrm{NMI} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k} n_{i,j} \log\left(\frac{n \cdot n_{i,j}}{n_i \cdot n_j}\right)}{\sqrt{\left(\sum_i n_i \log\frac{n_i}{n}\right)\left(\sum_j n_j \log\frac{n_j}{n}\right)}}, \tag{17}$$

Table 3. NMI comparisons for k-means, spectral clustering with 100 nearest neighbors.

Algorithms	E-k-means	S-k-means	Spectral Clustering
NMI	0.2586($\pm$0.0086)	0.2702($\pm$0.0059)	0.2875($\pm$0.0011)

where n is the number of documents, n_i and n_j denote the number of documents in category i and cluster j, respectively, and $n_{i,j}$ denotes the number of 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 this NMI score, the better the clustering quality.

We compared k-means algorithm based on Euclidean distance (E-k-means), spherical k-means based on cosine distance (S-k-means) [6], and our parallel spectral clustering algorithm using 100 nearest neighbors. Table 3 reports that parallel spectral clustering algorithm outperforms E-k-means and S-k-means. This result confirms parallel spectral clustering to be effective in finding clusters.

4.2 Scalability: Runtime Speedup

We used both the RCV1 dataset and a PicasaWeb dataset to conduct a scalability experiment. The RCV1 can fit into main memory of one machine, whereas the PicasaWeb dataset cannot. PicasaWeb is an online platform for users to upload, share and manage images. The PicasaWeb dataset we collected consists of $637,137$ images accompanied with $110,342$ tags.

For each image, we extracted 144 features including color, texture, and shape as its representation [15]. In the color channel, we divided color into 12 color bins including 11 bins for culture colors and one bin for outliers [12]. 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 [21] due to its computational efficiency. Each DWT on an image yielded four subimages including scale-down image and its wavelets in three orientations. We then obtained nine texture combinations from subimages of 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.

We first report the speedup on the RCV1 dataset in Table 4. As discussed in Section 3.1, the computation of similarity matrix can achieve linear speedup. In this experiment, we focus on the time of finding the first k eigenvectors and conducting k-means. Here the k-means is referred to Step 6 in Algorithm 1. It is important to notice that we could not ensure the quiesce of the allocated machines at Google's distributed data centers. There were almost always other

Table 4. RCV1 data set. Runtime comparisons for different number of machines. n=193,844, k=103, m=206.

	Eigensolver		k-means	
Machines	Time (sec.)	Speedup	Time (sec.)	Speedup
1	9.90×10^2	1.00	4.96×10^1	1.00
2	4.92×10^2	2.01	2.64×10^1	1.88
4	2.83×10^2	3.50	1.53×10^1	3.24
8	1.89×10^2	5.24	1.10×10^1	4.51
16	1.47×10^2	6.73	9.90×10^0	5.01
32	1.29×10^2	7.67	1.05×10^1	4.72
64	1.30×10^2	7.62	1.34×10^1	3.70

jobs running simultaneously with ours on each machine. Therefore, the runtime is partially dependent on the slowest machine being allocated for the task. (We consider an empirical setting like this to be reasonable, since no modern machine is designed or expected to be single task.) When 32 machines were used, the parallel version of eigensolver achieved 7.67 times speedup. When more machines were used, the speedup actually decreased. Similarly, we can see that parallelization sped up k-means more than five times when 16 machines were used. The speedup is encouraging. For a not-so-large dataset like RCV1, the Amdahl's law kicks in around $p = 16$. Since the similarity matrix in this case is not huge, the communication cost dominates computation time, and hence further increasing machines does not help. (We will see next that the larger a dataset, the higher speedup our parallel implementation can achieve.)

Next, we looked into the speedup on the PicasaWeb dataset. We grouped the data into $1,000$ clusters, where the corresponding Arnoldi space is set to be $2,000$. Note that storing the eigenvectors in Arnoldi space with $2,000$ dimensions requires 10GB of memory. This memory configuration is not available on off-the-shelf machines. We had to use at least two machines to perform clustering. We thus used two machines as the baseline and assumed the speedup of two machines is 2. This assumption is reasonable since we will see shortly that our parallelization can achieve linear speedup on up to 32 machines.

Table 5 reports the speedups of eigensovler and k-means. We can see in the table that both eigensolver and k-means enjoy near-linear speedups when the number of machine is up to 32. For more than 32 machines, the speedups of k-means are better than that of eigensolver. However both speedups became sublinear as the synchronization and communication overheads started to slow down the speedups. The "saturation" point on the PicasaWeb dataset is $p = 128$ machines. Using more than 128 machines is counter-productive to both steps.

From the experiments with RCV1 and PicasaWeb, we can observe that the larger a dataset, the more machines can be employed to achieve higher speedup. Since several computation intensive steps grow faster than the communication

Table 5. Picasa data set. Runtime comparisons for different numbers of machines. n=637,137, k=1,000, m=2,000.

	Eigensolver		k-means	
Machines	Time (sec.)	Speedup	Time (sec.)	Speedup
1	–	–	–	–
2	8.074×10^4	2.00	3.609×10^4	2.00
4	4.427×10^4	3.65	1.806×10^4	4.00
8	2.184×10^4	7.39	8.469×10^3	8.52
16	9.867×10^3	16.37	4.620×10^3	15.62
32	4.886×10^3	33.05	2.021×10^3	35.72
64	4.067×10^3	39.71	1.433×10^3	50.37
128	3.471×10^3	46.52	1.090×10^3	66.22
256	4.021×10^3	40.16	1.077×10^3	67.02

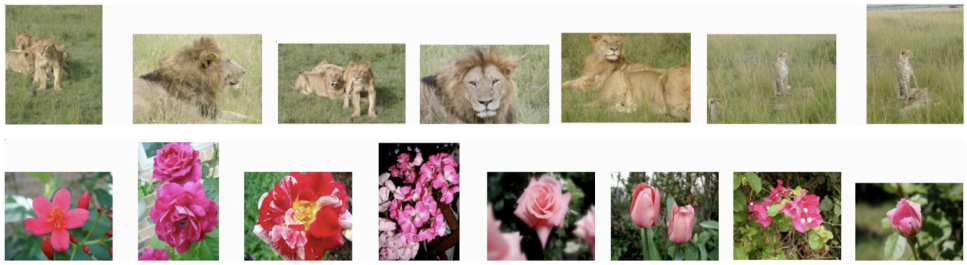
cost, the larger the dataset is, the more opportunity is available for parallelization to gain speedup.

Figure 4 shows sample clusters generated by k-means and spectral clustering. The top two rows are clusters generated by k-means, the bottom two rows are by spectral clustering. First, spectral clustering finds lions and leopards more effectively. Second, in the flower cluster, spectral clustering can find flowers of different colors, whereas k-means is less effective in doing that. Figure 5 provides a visual comparison of the clustering results produced by four different clustering schemes (of ours). On the top is our parallel k-means. Rows 2 to 4 display results of using parallel spectral clustering with different tag weighting settings (α). In addition to perceptual features, tags are useful for image searching and clustering. We use the tag weighting factor α to incorporate tag overlapping information in constructing the similarity matrix. The more tags are overlapped between images, the larger the similarity between the images. When the tag weighting factor is set to zero, spectral clustering considers only the 144 perceptual features depicted in the beginning of this section. When tag information is incorporated, we can see that the clustering performance improves. Though we cannot use one example in Figure 5 to prove that the spectral clustering algorithm is always superior to k-means, thanks to the kernel, spectral clustering seems to be more effective in identifying clusters of non-linear boundaries (such as photo clusters).

5 Conclusions

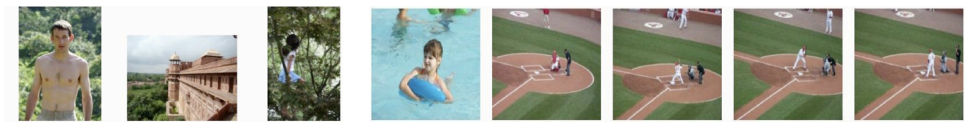
In this paper, we have shown our parallel implementation of the spectral clustering algorithm to be both correct and scalable. No parallel algorithm can escape from the Amdahl's law, but we showed that the larger a dataset, the more machines can be employed to use parallel spectral clustering algorithm to enjoy fast and high-quality clustering performance. We plan to enhance our work to address a couple of research issues.

(a) Sample images of k-means.



(b) Sample images of spectral clustering.

Fig. 4. Clustering results of k-means and spectral clustering.



(a) Sample images of k-means clustering.

(b) Sample images of spectral clustering with tag weighting factor $\alpha = 0.0$.

(c) Sample images of spectral clustering with tag weighting factor $\alpha = 0.5$.

(d) Sample images of spectral clustering with tag weighting factor $\alpha = 1.0$.

Fig. 5. Clustering results of k-means and spectral clustering. The cluster topic is "baseball game."

Nyström method. Though the Nyström method [8] enjoys a better speed and effectively handles the memory difficulty, our preliminary result shows that its performance is slightly worse than our method here. Due to space limitations, we will detail further results in future work.

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 (11). How to efficiently handle the case of large k is thus an interesting issue.

In summary, this paper gives a general and systematic study on parallel spectral clustering. We successfully built a system to efficiently cluster large image data on a distributed computing environment.

References

1. L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of basic linear algebra subprograms (BLAS). *ACM Trans. Math. Software*, 28(2):135–151, 2002.
2. 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.
3. F. Chung. *Spectral Graph Theory*. Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.
4. 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.
5. 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.
6. I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1–2):143–175, 2001.
7. 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.
8. 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.
9. 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.
10. V. Hernandez, J. Roman, A. Tomas, and V. Vidal. A Survey of Software for Sparse Eigenvalue Problems. Technical report, Universidad Politecnica de Valencia, 2005.
11. 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.
12. K. A. Hua, K. Vu, and J.-H. Oh. Sammatch: a flexible and efficient sampling-based image retrieval technique for large image databases. In *Proceedings of ACM MM*, pages 225–234, New York, NY, USA, 1999. ACM.
13. R. B. Lehoucg, D. C. Sorensen, and C. Yang. *ARPACK User's Guide*. SIAM, 1998.

14. 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.
15. 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.
16. R. Liu and H. Zhang. Segmentation of 3D meshes through spectral clustering. In *Proceedings of Pacific Conference on Computer Graphics and Applications*, 2004.
17. U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.
18. 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.
19. A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Proceedings of NIPS*, pages 849–856, 2001.
20. J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Tran. on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.
21. J. R. Smith and S.-F. Chang. Automated image retrieval using color and texture. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1996.
22. M. Snir and S. Otto. *MPI-The Complete Reference: The MPI Core*. MIT Press, Cambridge, MA, USA, 1998.
23. A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3:583–617, 2002.
24. R. Thakur and W. Gropp. Improving the performance of collective operations in MPICH. *Proceedings of European PVM/MPI User's Group Meeting*, 2003.
25. S. Xu and J. Zhang. A hybrid parallel web document clustering algorithm and its performance study. *Journal of Supercomputing*, 30(2):117–131.
26. S. X. Yu and J. Shi. Multiclass spectral clustering. In *Proceedings of ICCV*, page 313, Washington, DC, USA, 2003. IEEE Computer Society.
27. L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Proceeding of NIPS*, pages 1601–1608. 2005.