Distributed Approximate Spectral Clustering for Large-Scale Datasets

Fei Gao
School of Computing Science
Simon Fraser University
Surrey, BC, Canada

Wael Abd-Almageed
Institute for Advanced
Computer Studies
University of Maryland
College Park, MD, USA

Mohamed Hefeeda
Qatar Computing Research
Institute
Qatar Foundation
Doha, Qatar

ABSTRACT

Data-intensive applications are becoming important in many science and engineering fields, because of the high rates in which data are being generated and the numerous opportunities offered by the sheer amount of these data. Large-scale datasets, however, are challenging to process using many of the current machine learning algorithms due to their high time and space complexities. In this paper, we propose a novel approximation algorithm that enables kernel-based machine learning algorithms to efficiently process very large-scale datasets. While important in many applications, current kernel-based algorithms suffer from a scalability problem as they require computing a kernel matrix which takes $O(N^2)$ in time and space to compute and store. The proposed algorithm yields substantial reduction in computation and memory overhead required to compute the kernel matrix, and it does not significantly impact the accuracy of the results. In addition, the level of approximation can be controlled to tradeoff some accuracy of the results with the required computing resources. The algorithm is designed such that it is independent of the subsequently used kernel-based machine learning algorithm, and thus can be used with many of them. To illustrate the effect of the approximation algorithm, we developed a variant of the spectral clustering algorithm on top of it. Furthermore, we present the design of a MapReduce-based implementation of the proposed algorithm. We have implemented this design and run it on our own Hadoop cluster as well as on the Amazon Elastic MapReduce service. Experimental results on synthetic and real datasets demonstrate that significant time and memory savings can be achieved using our algorithm.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering, Search process*

HPDC'12, June 18–22, 2012, Delft, The Netherlands.

Keywords

Distributed clustering, large data sets, kernel-based algorithms, spectral clustering

1. INTRODUCTION

The rapidly declining cost of sensing technologies has led to proliferation of data in virtually all fields of science. Consequently, the sizes of datasets used in all aspects of data-driven decision making, inference, and information retrieval tasks have exponentially grown. For example, datasets for applications such as image clustering [35] and document retrieval [22] have substantially increased with the widespread usage of web contents and the inexpensive cost of capturing and sharing images. In many science and engineering applications, machine learning algorithms are frequently used to process data and perform various tasks such as clustering and classification. However, some of these algorithms do not support the increasing volumes of data being available.

The goal of this paper is to enable an important class of machine learning algorithms to efficiently process very large-scale datasets. We achieve this goal using two techniques: controlled approximation and elastic distribution of computation. In the first technique, we carefully apply approximation schemes on the datasets, which substantially reduce the time and space complexities of the considered class of machine learning algorithms without significantly impacting the accuracy of the produced results. The level of approximation can be controlled to gauge the accuracy of the results versus the required computing resources. In the second technique, we design distributed versions of the machine learning algorithms that can utilize the flexibility offered by cloud computing platforms.

We consider the scalability of kernel-based machine learning algorithms, which have an increasing importance in many fields, such as computer vision, bioinformatics, and natural language processing. The performance of most kernel-based machine learning algorithms significantly improves as the size of training datasets increases. For example, in [26], Munder and Gavrila show that the false negative rate of their image-based human detection algorithm is reduced by approximately 50% by only doubling the size of training dataset for their Support Vector Machine (SVM) classifier, while maintaining the same false alarm rate. Current kernel-based algorithms, however, suffer from an inherent limitation: they require computing a kernel matrix which stores pair-wise similarity values among all data points. That is, the kernel matrix has $O(N^2)$ complexity in both time and space. This is clearly not feasible for large datasets

with millions, and soon billions, of data points. We address this limitation by designing an approximation algorithm for constructing the kernel matrix for large datasets, which promises significant reduction in computational and memory overhead required to compute the kernel matrix.

In particular, the main contributions of this paper are:

- We propose a novel approximation algorithm for computing the kernel matrix for large-scale data sets. The algorithm is designed such that it is independent of the subsequently used machine learning algorithm. Thus, it can be used to scale many kernel-based machine learning algorithms.
- We design a variant of the spectral clustering algorithm [9] on top of the proposed approximation algorithm to show its effect on the performance and accuracy. We theoretically analyze the expected reduction in computation time and memory requirements resulted from our approximation algorithm.
- We present a distributed design of the proposed approximate spectral clustering algorithm in the MapReduce programming model, and we implement and run it on a Hadoop cluster in our lab as well as on the Amazon Elastic MapReduce (EMR) service.
- We conduct extensive experimental study using various synthetic and real datasets. The real dataset contains more than three million documents that we collected from Wikipedia. Our experimental results show substantial (multiple orders of magnitude) saving in the computing time and memory requirements can be achieved by our algorithm when applied on large-scale datasets, without significantly impacting the accuracy of the results. In addition, we demonstrate how the proposed algorithm can efficiently utilize variable computing resources offered by the Amazon cloud platform. Furthermore, we compare the proposed algorithm against three recent algorithms in the literature and show that it outperforms them.

The remainder of this paper is organized as follows. Section 2 summarizes the related work. Section 3 describes the proposed approximation algorithm and its distributed implementation. Section 4 presents the accuracy and complexity analysis of the proposed algorithm. The experimental platform and results are presented in Section 5. The paper is concluded in Section 6.

2. RELATED WORK

Massive data sets are common nowadays in many domains. To process such large data sets, distributed and cloud platforms are employed. For example, Matsunaga et al. propose cloudBLAST [23], which is a cloud version of the BLAST similarity search algorithm of DNA/RNA sequences, and Kang et al. [17] develop a distributed system for mining massive graphs.

Several machine learning algorithms have also been implemented in distributed manner. For example, the open-source Apache Mahout library implements important machine learning algorithms such as K-Means, Singular Value Decomposition and Hidden Markov Models using the MapReduce model. Even with distributed implementations, some machine learning algorithms may not be able to handle large-scale data sets, because of their time and space complexities. The important kernel-based machine learning algorithms fall into this category, because they require $O(N^2)$ time and space complexities to construct similarity matrices for N data points. Our work addresses this problem by proposing an approximation method to reduce the time and space complexities of constructing kernel matrices and efficiently processing the approximated matrices on cloud infrastructures with varying resources.

Previous works addressing the limitations of large-scale kernel-based machine learning algorithms can be broadly classified into two main categories: (1) methods that construct low-rank approximations of the kernel matrix, and (2) efficient implementations for computing the kernel matrix, including implementations on modern computing platforms such as Graphics Processing Units (GPUs).

Low-rank methods depend on the observation that the eigen-spectrum of the kernel matrix rapidly decays, especially when the kernel function is a Radial Basis Function (RBF) [33] [36]. Consider a kernel matrix K with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N \geq 0$ and corresponding eigenvectors $\mathbf{v}_i$, $K = \sum_i^N \lambda_i \mathbf{v}_i \mathbf{v}_i^T$. Since the eigen-spectrum decays rapidly, i.e., most of the information is stored in the first few eigenvectors, the kernel matrix can be approximated by $\tilde{K} = \sum_i^M \lambda_i \mathbf{v}_i \mathbf{v}_i^T$, where $M \ll N$. Williams and Seeger [37] use the Nystrom method [8] to compute the most significant M eigenvalues and eigenvectors. The number of computed eigenvectors is inversely proportional to the approximation error. Nystrom-based methods have complexity of $O(M^2 N)$, where M is the number of computed eigenvectors.

Kernel-based methods have also been a target of efficient implementations employing customized data structures and modern computing platforms. For example, Ohmer et al. [28] use GPUs to implement the classification step of the SVM classifier, in which the kernel values are computed between the input *test* vector and the set of support vectors. It should be noted, however, that the bottleneck of kernel methods is usually encountered during the training phase, rather than the testing phase, due to the large number of training vectors and the dimensionality of the vector space. In SVM classifiers, for example, the number of support vectors is much smaller than the training vectors and the kernel values are only computed against a small number of test vectors.

Our proposed algorithm benefits from the advantages of both categories, in which we exploit the rapidly decaying eigen-spectrum of the kernel matrix and also develop a distributed computing implementation using MapReduce.

3. DISTRIBUTED APPROXIMATE SPECTRAL CLUSTERING (DASC)

This section presents the proposed algorithm. We start with an overview, followed by the details. Then, we present the distributed implementation.

3.1 Overview

Kernel methods have gained significant attention in the machine learning community and other applied fields for more than a decade. Applications involving kernel methods include classification [5], dimensionality reduction [31] and data clustering [27]. Kernel methods offer a modular

framework to do data analysis, in which the fundamental is to compute the kernel matrix. The kernel matrix represents kernelized distances between pairs of data points. The kernel matrix is sometimes referred to as the similarity matrix and Gram matrix. We will use the terms kernel, similarity, and Gram matrix interchangeably. The performance of kernel methods improves as the number of input points N increases [26]. However, since the cost of computing the similarity matrix is $O(N^2)$ in both time and space, it becomes computationally infeasible to handle large- and web-scale datasets, such as Amazon's customer statistics dataset [20], (three million records) and Wikipedia Talk network dataset [21] (two million instances) .

We propose a novel approximation algorithm for computing the similarity matrix. This approximation algorithm can be used with any kernel-based machine learning algorithm to make it scalable to large-scale datasets. We apply our approximation algorithm to the spectral clustering algorithm [27], which is based on eigen-decomposition of kernel matrices. Spectral clustering performs well with non-Gaussian clusters, and it does not suffer from the problem of local optima [27]. Spectral clustering has been used in various fields such as document clustering [39] and image Segmentation [34].

There are four steps in the proposed distributed approximate spectral clustering (DASC) algorithm. First, the algorithm creates compact signatures for all data points. This is done using locality sensitive hashing. Second, points whose signatures are similar to each other are grouped together. Third, similarity values are computed for points in each group to form portions of the similarity matrix. Fourth, the Spectral Clustering algorithm is performed on the approximated similarity matrix. The first three steps make the proposed approximation algorithm for kernel matrices. The fourth step adds an example kernel-based machine learning algorithm. Spectral clustering can be replaced by any other kernel-based algorithm in the fourth step.

3.2 Details

The first step of the proposed DASC algorithm is to preprocess the dataset using locality sensitive hashing (LSH). LSH is a probabilistic dimension reduction technique [25]. The idea is to hash points such that the probability of collision is much higher for close points than for those that are far apart. Points whose hashing values collide with each other fall into the same bucket.

We have studied various LSH families [12], including random projection, stable distributions, and Min-Wise Independent Permutations [4]. The hash functions we use to generate the signatures belong to the family of random projection. The advantage of this family is that, after applying hashing function once, we only need one bit to store the result, which saves memory. It has been shown that random projection has the best performance in high-dimensional data clustering [10]. Also, with random projection, we can compare the generated signatures using hamming distances for which efficient algorithms are available [2].

Given a set of data points $X_1, \ldots, X_N \in R^d$, we use random projection hashing [2] to generate M-bit binary signature for each data point. Each bit is generated as follows. An arbitrary dimension of the input space is selected and compared with a threshold. If the feature value along this dimension is larger than the threshold, the bit is set so 1. Otherwise, it is set to 0. Details on criterion for choosing hashing dimensions and setting the threshold are discussed in Section 4.2. The time complexity of random projection hashing is $O(M\,N)$.

In the second step of the proposed algorithm, vectors with near-duplicate signatures are grouped into the same *bucket*. Near-duplicate signatures mean that for two M-bit binary numbers, there are at least P bits in common, where $P \leq M$. If the total number of unique signatures generated is T, the complexity of this step is $O(T^2)$.

In the third step, for each bucket (representing unique signature), we compute the similarity matrix for the vectors that belong to this bucket only. Assuming we have T buckets, each of which has N_i points, where $0 \leq i \leq T-1$ and $\sum_{i=0}^{T-1} N_i = N$, the overall complexity of this step is $\sum_{i=0}^{T-1} O(N_i^2)$. In the following, we use a Gaussian kernel to compute the pairwise similarity S_{lm} between vectors X_l and X_m as shown in Equation (1)

$$S_{lm}^i = \exp(-\frac{||X_l - X_m||^2}{2\sigma^2}), \tag{1}$$

where σ is kernel bandwidth, which controls how rapidly the similarity S_{lm}^i decays.

The final step in the proposed DASC algorithm is applying spectral clustering. Spectral clustering computes the Laplacian matrix L and the eigenvectors of L. It then performs K-means clustering on the eigenvectors matrix. For the DASC algorithm, spectral clustering is applied on the approximated similarity matrix, which is composed of the smaller similarity matrices computed from different hashing buckets. Thus, we compute the Laplacian matrix on each similarity matrix S^i as shown in Equation (2)

$$L^i = D^{i^{-1/2}} S_i D^{i^{-1/2}}, \tag{2}$$

where $D^{i^{-1/2}}$ is the inverse square root of D^i and is diagonal matrix. For an $N_i \times N_i$ diagonal matrix, the complexity of finding the inverse square root is $O(N_i)$. Moreover, the complexity of multiplying an $N_i \times N_i$ diagonal matrix with an $N_i \times N_i$ matrix is $O(N_i^2)$. Therefore, the complexity of this step is $O(\sum_{i=0}^{T-1} N_i^2)$.

We then find the first K_i eigenvectors of L^i, $V_1^i, V_2^i, \ldots, V_{K_i}^i$ and form the matrix $X^i = [V_1^i V_2^i \cdots V_{K_i}^i] \in R^{N_i \times K_i}$ by stacking the eigenvectors in columns. The eigenvectors are computed using QR decomposition [6], which takes $O(K_i^3)$ steps. To reduce the computational complexity, we transform L^i into a $K_i \times K_i$ symmetric tridiagonal matrix A^i. The complexity of the transformation is $O(K_i\,N_i)$. QR decomposition is then applied to A^i, which is $O(K_i)$. Therefore, the complexity of this step is $O(\sum_{i=0}^{T-1} (K_i\,N_i))$. The input vectors X_i are normalized to have unit length such that $Y_{ij} = X_{ij}/(\sqrt{\sum_j X_{ij}^2})$ and Y_i is treated as a point in R^{K_i} and is clustered into K_i clusters using K-means [13]. The complexity of this step is $O(\sum_{i=0}^{T-1} (K_i\,N_i))$.

Adding the time cost of all the above steps, the overall time complexity for the DASC algorithm is given by:

$$T_{DASC} = O(M\,N) + O(T^2) + \sum_{i=0}^{T-1} \left[2\,O(N_i^2) + 2\,(K_i\,N_i)\right] + 2\,N. \quad (3)$$

3.3 MapReduce Implementation

The fundamental concept behind MapReduce is to break up algorithm execution into two phases: map and reduce. The inputs and outputs of each phase are defined by key-value pairs. We divide the proposed DASC algorithm into two MapReduce stages. The first MapReduce stage applies LSH on the input data and produces vector signatures. In the map phase of the this stage, the input vectors are loaded as *(index, inputVector)* pairs, where *index* is the index of a data point, and *inputVector* is a numerical array associated with the point. The output key-value pair is *(signature, index)*, where *signature* is a binary sequence, and *index* is the same as the input notation.

The input to the reducer of the first stage is the *(signature, listof(index))* pair, where *signature* is a binary sequence, and *listof(index)* is a list of all vectors that share the same signature indicating that these vectors are near-duplicates. The reducer computes the sub-similarity matrix following Equation (1). The pseudo-codes for the mapper and reducer functions are shown in Algorithms 1 and 2, respectively.

Algorithm 1: mapper(index, inputVector)

```
/*a mapper is fed with one inputVector at a time
String Sig = "" ;
for i = 1; i ≤ M; i++ do
    Threshold = get_threshold(i) ;
    Hyperplane = get_hyperplane(i) ;
    if inputVector[Hyperplane] ≤ Threshold then
        p = 1 ;
    else
        p = 0 ;
    Convert p to String and add to the tail of Sig ;
emitPair(Sig, index) ;
```

Algorithm 2: reducer (signature, ArrayList indexList)

```
/*Compute the sub similarity matrix */
Length = getLength(indexList) ;
for i = 1; i ≤ Length; i++ do
    for j = 1; j ≤ Length; j++ do
        if i ≠ j then
            subSimMat[i, j] = simFunc(i, j) ;
        else
            subSimMat[i, j] = 0 ;
Output_to_File(subSimMat) ;
```

In Algorithm 1, we note that the hyperplane and threshold values are important factors in the hash function. The threshold value controls at which point we separate the original dataset apart, and the hyperplane value controls which feature space to compare with the corresponding threshold. We use the principle of k-dimensional tree (k-d tree) [18] to set hyperplane and threshold values. The k-d tree is a binary tree in which every node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as subspaces. Points to the left of this hyperplane are represented by the left subtree of that node and points right of the hyperplane are represented by the right subtree. The hyperplane direction is chosen in the following way: every node in the tree is associated with one of the k-dimensions, with the hyperplane perpendicular to that dimension's axis. For example, if for a particular split, the "x" axis is chosen, all points in the subtree with a smaller "x" value than the node will appear in the left subtree and all points with larger "x" value will be in the right subtree. In such a case, the hyperplane would be set by the x-value of the point, and its normal would be the unit x-axis.

To determine the *hyperplane* array, we look at each dimension of the dataset, and calculate the numerical span for all dimensions (denoted as $span[i], i \in [0, d)$). The numerical span is defined as the difference of the largest and the smallest values in this dimension. We then rank the dimensions according to their numerical spans. The possibility of one $hyperplane[i]$ being chosen by the hash function is:

$$prob = span[i] / \sum_{i=0}^{d-1} span[i], \quad (4)$$

which ensures that dimensions with large span have more chances to be selected. For each dimension space $Dim[i]$, the associated *threshold* is determined as follows: between the minimum (denoted as $min[i]$) and maximum (denoted as $max[i]$) in $Dim[i]$, we create 20 bins (denoted as $bin[j], j \in [0, 19]$), $bin[j]$ will count the number of points whose ith dimension fall into the range $[min[i]+j \times span[i]/20,\ min[i]+(j+1) \times span[i]/20]$. We then find the minimum in array bin (denoted as s), the threshold associated with $Dim[i]$ is set to:

$$Dim[i] = min[i] + s \times span[i]/20. \quad (5)$$

Approximation error can occur if two relatively close points in the original input space are hashed into two different buckets. If a full similarity matrix is computed, the similarity between the two vectors will be significant. However, due to our approximation scheme, this similarity will be zeros. In order to reduce this approximation error, buckets represented by signatures that share no less than P bits. We perform a pair-wise comparison between the M unique signatures and merge the vectors that belong to buckets with signatures no less P similar bits are combined. This step is performed before applying the reducer.

The process of comparing two M-bit signatures A and B is optimized for performance using the bit manipulation:

$$ANS = (A \oplus B)(A \oplus B - 1), \quad (6)$$

where if ANS is 0, then A and B have only 1 bit in difference, thus they will be merged together. Otherwise, A and B are not merged. The complexity of this operation is $O(1)$.

After computing the sub-similarity matrices based on the LSH signatures and combining similar buckets, we use the standard MapReduce implementation of spectral clustering available in the Mahout library [29].

4. ANALYSIS AND COMPLEXITY

In this section, we first analyze the time and space complexities of the proposed DASC algorithm and show its scalability. Then, we analyze the accuracy expected from the proposed approximation method for the kernel matrix.

4.1 Complexity Analysis

The time complexity for computing the full similarity matrix is $O(N^2)$. For the approximated similarity matrix, assume that we have B buckets, and there are N_i number of points in bucket i. The N_i points in bucket B_i form K_i clusters, where $0 \leq i \leq B-1$, and $\sum_{i=0}^{B-1} N_i = N$. Spectral clustering is performed on each bucket. Therefore, the time reduction ratio is:

$$\alpha = \frac{M\,N + B^2 + 2\,N + \sum_{i=0}^{B-1} \left[2\,N_i^2 + 2\,(K_i\,N_i)\right]}{2\,N^2 + 2\,(K\,N) + 2\,N}. \tag{7}$$

To gain insights on the above equation, let us assume that all buckets contain equal number of points, that is $N_i = \frac{N}{B}$ where $0 \leq i \leq B-1$. Equation (7) can be rewritten as:

$$\begin{aligned}
\alpha &= \frac{M\,N + B^2 + 2\,N + \sum_{i=0}^{B-1} \left[2\,N_i^2 + 2\,(K_i\,N_i)\right]}{2\,N^2 + 2\,K\,N + 2\,N} \\
&= \frac{M\,N + B^2 + 2\,N + B\,\left[2\,(\frac{N}{B})^2 + 2\,(\frac{K}{B}\,\frac{N}{B})\right]}{2\,N^2 + 2\,K\,N + 2\,N} \\
&= \frac{M + \frac{B^2}{N} + 2 - \frac{2}{B}}{2\,(1 + N + K)} + \frac{1}{B} \\
&\approx \frac{1}{B},
\end{aligned} \tag{8}$$

since the first item approaches zero as N increases. We note that the above time complexity is an upper bound on the time reduction achieved because of the approximation of the similarity matrix. In the worst case, all points will hash to the same bucket. This, however, is not typical for practical data, because we use multiple hash functions and we apply each function on a different dimension. Further, the dimensions used in the hashing are the ones that have the largest span, i.e., dimensions in which data points are as spread out as possible. We also note that Equation (7) and the proposed DASC algorithm itself *does not* assume or require uniform distribution of data points over buckets. The uniform distribution in Equation (8) is *only* used to illustrate the upper bound on the potential time reduction.

The space complexity for computing the full matrix is $O(N^2)$. Computing the similarity matrices for each bucket individually, the space complexity is reduced to $\sum_{i=0}^{B-1} N_i^2$. Therefore, the space usage reduction ratio is given as:

$$\gamma = \frac{\sum_{i=0}^{B-1} N_i^2}{N^2}. \tag{9}$$

Again for the upper bound, assume that all buckets contain equal number of points, $N_i = \frac{N}{B}$ where $0 \leq i \leq B-1$. Equation (9) can be rewritten as:

$$\gamma = \frac{\sum_{i=0}^{B-1} N_i^2}{N^2} = \frac{B\,(\frac{N}{B})^2}{N^2} = \frac{1}{B}. \tag{10}$$

It is important to note that, the memory needed is distributed across the number of machines running the algorithm, which improves the scalability of the DASC algorithm.

To shed some insights on the above analysis and illustrate the scalability of the DASC algorithm, we numerically analyze its time and space complexities for datasets of different sizes. The processing time of DASC can be written as (in seconds):

$$\begin{aligned}
Time &= \beta\,\frac{M\,N + B^2 + 2\,N + \sum_{i=0}^{B-1} \left[2\,N_i^2 + 2\,(K_i\,N_i)\right]}{C} \\
&= \beta\,\frac{M\,N + B^2 + 2\,N + B\,\left[2\,(\frac{N}{B})^2 + 2\,(\frac{K}{B}\,\frac{N}{B})\right]}{C} \\
&= \beta\,\frac{\log B\,N + B^2 + 2\,N + \frac{2\,N^2 + 34\,N\,(\log N - 9)}{B}}{C},
\end{aligned} \tag{11}$$

where $M = \log B$, $K = 17\,(\log N - 9)$, C is the number of machines, and β is a constant representing the average execution time for machine operations. The exact value for β depends on various issues including the machine architecture, memory, and I/O speed.

Similarly, for memory consumption, assuming single-precision floating point operations, we have (in bytes):

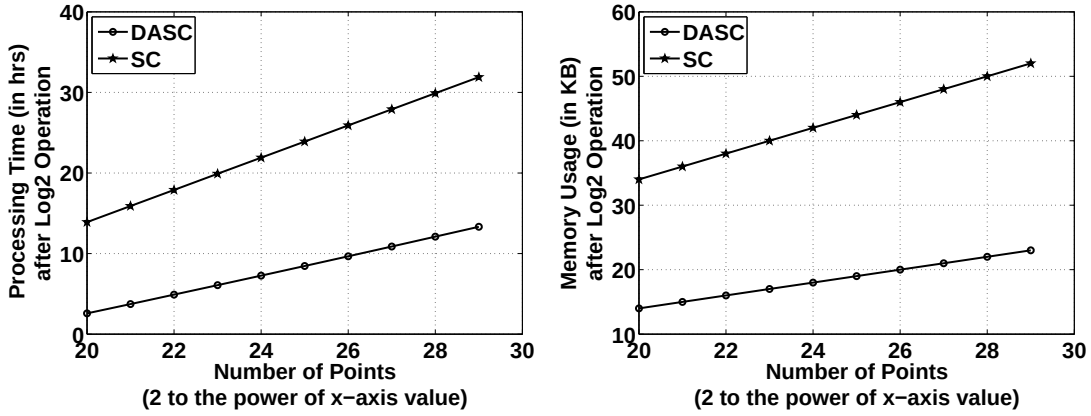
$$Memory = 4\,B\,(\frac{N}{B})^2 = 4\,\frac{N^2}{B}. \tag{12}$$

We plot Equations (11) and (12) in Figure 1 for datasets ranging from 1M to 512M points. To be able to plot the curves, we chose reasonable value for $\beta = 50\mu s$ [15]. We also assume a cluster of $C = 1,024$ nodes. We also plot the standard spectral clustering (SC) algorithm for comparison. Figure 1 shows that DASC is highly scalable to large-scale datasets in both time and space complexities. As the size of the dataset doubles, the increases in both processing time and memory footprint are sub-quadratic. Notice that the points in the x-axis are growing exponentially.

4.2 Accuracy Analysis

Given an $N \times d$ dataset, the dependence of final clustering on any arbitrary dimension i is proportional to the dispersion of data along this dimension. If the data along dimension i is highly dispersed, then this dimension may significantly contribute to the structure of the data, and visa versa. The probability of selecting an arbitrary dimension i for hashing is given by Equation (4). Therefore, since we need to use $M < d$ hash functions in the signature generation stage, we order the importance of the d dimensions based on the numerical span of the data along each dimension and pick the dimensions with highest M spans for applying the hash function. The hash function builds upon the fact that, along an arbitrary dimension, if all data points fall within a small span, the probability of mis-clustering increases.

Along dimension i, the hash function chooses a threshold τ such that if the value of the point along i is below τ,



(a) Time (b) Memory

Figure 1: Scalability of the DASC algorithm to large-scale datasets.

a hash value of 0 is assigned. Otherwise, a hash value of 1 is assigned. The value of the threshold τ is chosen by calculating a histogram of the data along i and setting τ to the lower edge of the histogram bin of the smallest count, which is given by Equation (5).

In a dataset of size N with K clusters, there are, on average, N/K points in each cluster. The dimensionality in each cluster is d. Given any two arbitrary data points that significantly differ in r dimensions, where $r \leq d$, the collision probability of these two points, i.e., the two points have duplicate binary signatures and therefore falling within the same bucket, is given by:

$$P_1 = (\frac{d-r}{d})^M, \tag{13}$$

where $(d-r)/d$ is the probability that the dimension where such two points have similar value is checked. Furthermore, for a set of points, the collision probability is given by:

$$P_2 = P_1^{N/K} = (\frac{d-r}{d})^{M\,N/K}. \tag{14}$$

To study the effect of the number of hash functions M on the tradeoff between the accuracy of clustering and the parallelization of the algorithm, we use a number of Wikipedia datasets as shown in Table 1. We use line fitting to empirically relate the number of clusters to the size of each of the data set. Equation (15) shows the best fitting parameters.

$$K = 17\,(\log_2 N - 9). \tag{15}$$

Every point in the Wikipedia dataset is a document comprised of $d = 11$ terms. We set $r = 5$ in order to ensure that the majority of dimensions, $d - r$, are significantly similar. Therefore, the total number of terms in the dataset is

$$t = 11 - r + \frac{N}{K}\,r. \tag{16}$$

The dimensionality d is given by

$$d = t\,K = K\,(11 - r) + N\,r. \tag{17}$$

Dataset size	Number of categories
1024	17
2048	31
4096	61
8192	96
16384	201
32768	330
65536	587
131072	1225
262144	2825
524288	5535
1048576	14237
2097152	42493

Table 1: Clustering information of Wikipedia dataset.

Equation (14) can now be rewritten as

$$P_2 = (\tfrac{d-r}{d})^{M\,N/K} \tag{18}$$

$$= (1 - \tfrac{5}{17\,(\log_2 N - 9)\,6 + N\,5})^{M\,N/17\,(\log_2 N - 9)}. \tag{19}$$

Figure 2 illustrates the relationship between the collision probability and the number of hash functions used as given by Equation (18), for datasets with different sizes chosen from the Wikipedia dataset. It can be observed that, as the number of hash functions increases, the probability of a group of adjacent points being put into a cluster slowly (sub-linearly) decreases. This means that the incorrectly clustered instances in the dataset increases. On the other hand, increasing the number of hash functions increases the number of instances of Spectral Clustering to run in parallel, improving the parallelization of the algorithm. Moreover, when we use the same M value to do the partitioning, as the size of the dataset increases, the collision probability decreases. Therefore, it can be seen that, through the tuning of the parameter M, we can control the tradeoff between the accuracy of the clustering algorithm and the degree of parallelization.

5. EXPERIMENTAL EVALUATION

In this section, we rigorously evaluate the performance

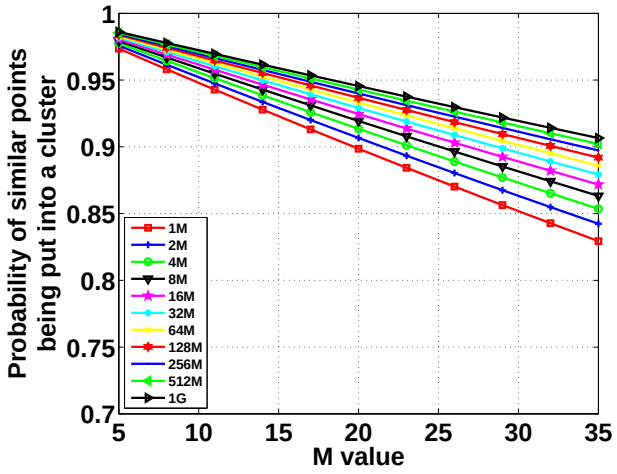


Figure 2: The impact of parameter M on accuracy for datasets of different sizes.

Parameter	Value
Hadoop jobtracker heapsize	768 MB
Hadoop namenode heapsize	256 MB
Hadoop tasktracker heapsize	512 MB
Hadoop datanode heapsize	256 MB
Maximum map tasks in tasktracker	4
Maximum reduce tasks in tasktracker	2
Data replication ratio in DFS	3

Table 2: Setup of the Elastic MapReduce cluster.

of the proposed algorithm and compare it against others. We start by describing our platform and datasets used in the following two subsections. Then, we explain and justify the performance metrics used. This is followed by a brief description of the algorithms implemented and compared against. Finally, the results for accuracy, complexity, and scalability are presented in three subsections.

5.1 Experimental Platform

We have implemented the proposed algorithm and few others in the literature in the Hadoop open-source implementation of the MapReduce framework. We ran our implementation on a local cluster as well as on the Amazon Elastic MapReduce (EMR) service. Our local cluster is composed of five machines, each is equipped with Intel Core2 Duo Processor E6550 (4M Cache, 2.33 GHz, 1333 MHz FSB) and 1 GB DRAM. The cluster runs Hadoop version 0.20.2 on top of Ubuntu Linux 2.6.27-14, and using Java JDK version 1.6.0. One machine in the cluster serves as the master (job tracker) and the others are slaves (task trackers).

For scalability and elasticity experiments, we use the Amazon EMR service with variable number of machines: 16, 32, and 64. Each machine has 1.7 GB memory, 350 GB of disk storage and runs Hadoop 0.20.2 on top of Debian Linux. We list the important parameters and their values in Table 2. EMR works in conjunction with Amazon EC2 (Elastic Compute Cloud) to create a Hadoop cluster, and with Amazon S3 (Simple Storage Service) to store scripts, input data, log files, and output results. To run our experiments, we first upload to Amazon S3 the input data, as well as the mapper and reducer executables that process the data. Then, we send a request to EMR to start a job flow. A job flow is a collection of processing steps that EMR runs on a specified dataset using a set of Amazon EC2 instances. Our job flow is comprised of several steps.

In the first step, we partition the dataset into buckets using locality sensitive hashing. Every bucket is a file stored in S3 containing points in the corresponding bucket. In the second step, Spectral Clustering is applied on individual buckets. The final step produces the results, stores them in S3 and terminates the job flow. We note that the partitioning step allows our DASC algorithm to process very large scale data sets, because the data partitions (or splits) are incrementally processed, split by split, based on the number of available mappers (and physical machines).

Intermediate results of hashing (buckets) are stored on S3 and then incrementally processed by DASC to produce final results. Thus, DASC can handle huge datasets. For very skewed data distributions, we can employ a different hashing function in LSH. There are data-dependent hashing functions (e.g., spectral hashing functions), which will yield balanced partitioning. Their inclusion in DASC is straightforward. Data locality is achieved through the first LSH step, which tries to group near-by points into one bucket. Distributed datasets can be thought of huge datasets with splits stored on different machines, where the output hashes represents the keys that are used to exchange datapoints between different nodes.

5.2 Datasets: Synthetic and Real

We use two datasets in our experiments: synthetic and real (from Wikipedia). We create synthetic datasets with controlled parameters, e.g., number of dimensions, number of data points, and range of values for data points. The size of the synthetic datasets ranges from 1024 to 4 million data points. Each data point is 64-dimension vector, where each dimension takes a real value chosen from the period [0–1]. We use the range [0–1] because dataset normalization is a standard preprocessing step in data mining applications.

Wikipedia has millions of articles, categorized into many groups. We used a dataset of 3,550,567 Wikipedia documents in our experiments. These documents are obtained and cleaned as follows. Note that we refer to a web page as document, and a word that appears in the corpus as a term. We developed a crawler in Python. The crawler started crawling the indexing page: `http://en.wikipedia.org/wiki/Portal:Contents/Categories`, which lists the categories and their links. Each category has sub-categories, where each sub-category can recursively have sub-categories. For these sub-category links, Wikipedia differentiates them into two genres, both are encoded in the HTML files. The first is identified with CategoryTreeBullet, meaning that this link contains its own sub-categories. The second is identified with CategoryTreeEmptyBullet, meaning that this subcategory only contains leaf nodes (HTML files). Our crawler obtained the tree structure of categories and downloaded the content of leaf nodes. At the ended of the crawling process, it downloaded 3,550,567 documents, forming 579,144 categories.

We then processed HTML files as follows: (i) removed all HTML tags keeping only the raw text, (ii) converted characters to lower case, (iii) removed punctuation, (iv) removed stop words, and (v) stemmed all terms. We used Apache

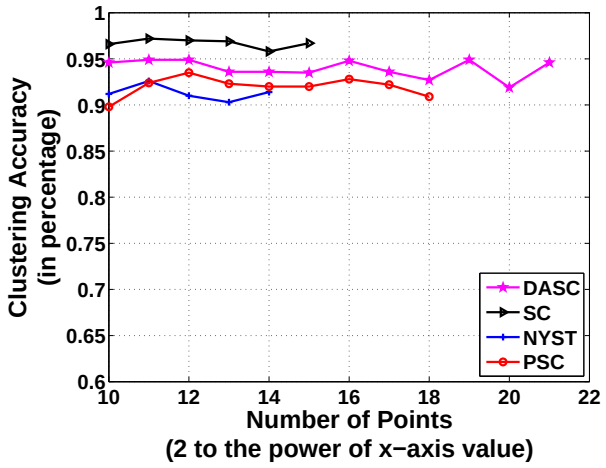


Figure 3: Accuracy of different algorithms using the Wikipedia dataset.

Lucene [14] perform most of these operations. Lucene is a text search engine library. The stop-word list we used is concatenated from several lists to capture the majority of the stop words. Stemming is the process of reducing inflected (or derived) words to their base or root forms. We used the Porter stemming algorithm [30] in Lucene.

We encountered a problem during processing HTML files: The number of terms in the dataset is huge. Terms in a document represent features (or dimensions) of that document, which will make the clustering process extremely computationally expensive. To address this problem, we used only the summary of each document and the important terms in the summary. To rank terms based on their importance, we computed the tf_idf (term frequency, inverse document frequency) value of each term, which is computed for a term t by dividing the total number of documents by the number of documents containing the term t. After ranking all terms based on their tf_idf values, we used the first F terms. We conducted several experiments to choose a good value for F. We randomly chose 1,084 document from the dataset. These documents contain 11,102 different terms. We ran Spectral Clustering on the datasets generated using F from 6 to 16 and compared the clustering accuracy. The clustering accuracy is the ratio of correctly clustered documents to the total number of documents. Our results (figure not shown) indicate that increasing F beyond 11 did not provide any significant improvement in the clustering accuracy. Therefore, we use $F = 11$ in our experiments.

We note that using only some terms and only from the document summaries (not the whole texts) makes the clustering task more challenging for our algorithm, which we compare its results against the ground-truth categorization provided by Wikipedia.

5.3 Performance Metrics

We use several performance metrics to assess the proposed algorithm from different angles. We measure the computational and space requirements of different algorithms. In addition, we assess the clustering accuracy using various metrics. The first metric is comparing the clustering results produced by our algorithm against the ground truth, which is the case for the Wikipedia dataset. In this case, the clustering accuracy is measured as the ratio of correctly clustered number of points to the total number of points. For many datasets in practice the ground truth for clustering is not known. For such cases, we use three metrics: Davies-Bouldin index (DBI), average squared error (ASE), and Frobenius norm (Fnorm). DBI and ASE assess the clustering quality produced by a given clustering algorithm, while the Fnorm provides an estimate on how close the original and approximated Gram matrices are.

DBI [7] is a function of the ratio of the sum of within-cluster scatter to between-cluster separation. It uses both the clusters and their sample means. It is calculated by:

$$DBI = \frac{1}{C}\sum_{i=1}^{C}\max_{j:i\neq j}\{\frac{\sigma_i+\sigma_j}{d(c_i,c_j)}\}, \tag{20}$$

where C is the number of clusters, c_x is the centroid of cluster x, σ_x is the average distance of all elements in cluster x to centroid c_x, and $d(c_i, c_j)$ is the distance between centroids c_i and c_j. The clustering algorithm that produces the smallest index is considered the best algorithm based on this criterion. DBI has been shown to be a robust strategy for the prediction of optimal clustering partitions [7].

ASE [16] serves as a clustering criterion for many classic clustering techniques. For example, it is used in [11] for validation purposes. The squared distance e_k^2 (where $1 \leq k \leq C$) for each cluster k is the sum of the squared Euclidean distances between each point in k and its centroid, and the average squared error for the entire dataset is expressed by:

$$ASE = \frac{1}{N}\sum_k e_k^2 = \frac{1}{N}\sum_k(\sum_{j=0}^{N_k-1} d(X_j, C_k))^2, \tag{21}$$

where N is the size of the dataset, N_k is the number of points in cluster k.

Fnorm is one of the matrix norms [24], which is also called the Euclidean norm. The Fnorm of an $M \times N$ matrix A is defined as:

$$Fnorm = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N}|a_{ij}|^2}. \tag{22}$$

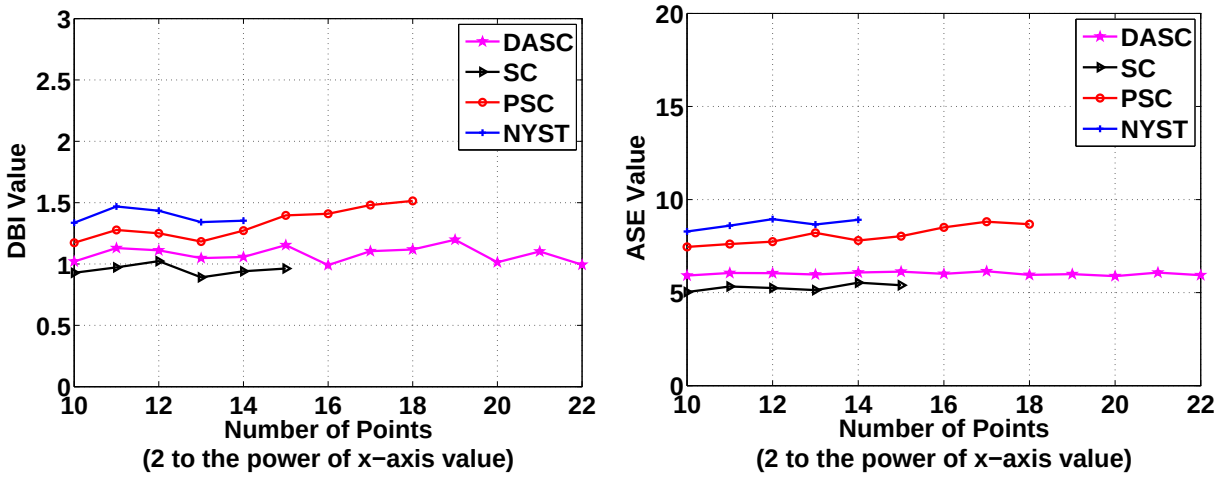
To illustrate the intuition behind this metric, consider an $M \times N$ matrix A. By singular value decomposition, A can be decomposed into the product of three matrices as follows:

$$\begin{aligned} A &= U\Sigma V^H \\ &= U\begin{bmatrix}\Sigma_k & 0\\ 0 & 0\end{bmatrix}V^H, \end{aligned} \tag{23}$$

where U and V are two unitary matrices and Σ_k is a $k \times k$ diagonal matrix containing the k ordered positive definite singular values $\sigma_1 \geq \sigma_2 \cdots \geq \sigma_k \geq 0$. The variable k is the rank of A and represents the number of linearly independent columns in it. The Frobenius norm is invariant under unitary transformations, therefore, we have:

$$Fnorm = \sqrt{\sum_{m=1}^{k}\sigma_m^2}. \tag{24}$$

According to Eq. (24), the larger the ratio of the Fnorm of the approximated matrix to the Fnorm of the original



(a) DBI (b) ASE

Figure 4: Accuracy of different algorithms using the synthetic dataset.

matrix, the closer the sum of singular values in the approximated matrix is to that of the original one. Thus, larger Fnorm ratios imply matrices with similar characteristics. Fnorm is used in [40] to develop a new affinity matrix normalization scheme in Spectral Clustering. Anagnostopoulos et al. [1] propose an approximation algorithm for co-clustering problem and use Fnorm for evaluation. Yang and Yang [38] compare their proposed distance against Fnorm in two-dimensional principal component analysis.

5.4 Algorithms Implemented and Compared Against

We have implemented the proposed DASC algorithm, and compared its performance against the three closest other algorithms: basic Spectral Clustering method (SC), Parallel Spectral Clustering (PSC) [3], and Spectral Clustering using the Nystrom extension (NYST) [32]. We provide brief description on the implementation and configuration below; more details can be found in [12].

We implemented DASC by modifying the Mahout library [29]. We set the number of buckets in the hashing step to $M = \lfloor (\log N)/2 \rfloor - 1$, where N is the dataset size. We set P, which is the minimum number of identical bits in two binary signatures needed to merge their buckets together, to $M-1$. This enables efficient, $O(1)$, comparison operation between signatures. It also increases the degree of parallelization as fewer buckets will be merged.

For SC, we used the basic distributed Spectral Clustering implementation in Mahout. For PSC, we used the C++ implementation by Chen [3], which uses the PARPACK library [19] as the underlying eigenvalue decomposition package. The parallelization is based on MPI. For NYST, we used the existing Matlab implementation by Shi [32].

5.5 Results for Clustering Accuracy

We present the accuracy results in this section. We note that some algorithms we compare against did not scale to support large-scale datasets. Hence, in the figures, some curves do not cover the whole range of the x-axis.

We start by presenting the accuracy results for the Wikipedia dataset in Figure 3. In this figure, we vary the number of

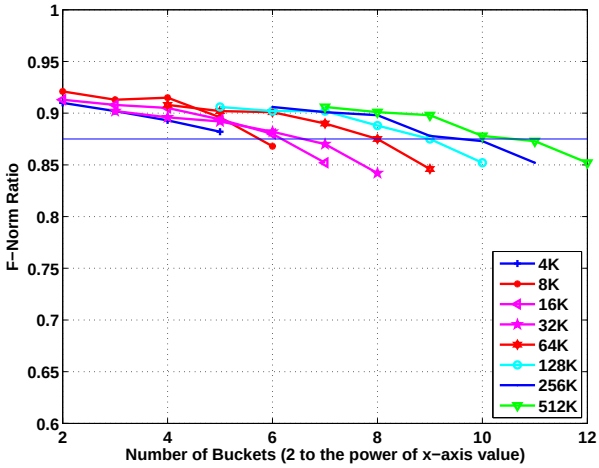


Figure 5: Comparison between approximated and original matrices using Frobenius norm.

randomly selected documents from the dataset, and we plot the ratio of correctly clustered points (documents) to the total number of points. The figure first shows that the three variants of spectral clustering (SC, PSC, and DASC) produce high accuracy: always more than 90%. Recall that these clustering algorithms use only the summaries of documents in the clustering process and the accuracy is computed relative to the pre-defined categorization of topics in Wikipedia. The figure also shows that the proposed DASC algorithm consistently produces better results than the results produced by the PSC algorithm, and its results are very close to the results produced by the basic SC algorithm. This means that the proposed approximation method does not negatively affect the clustering accuracy, while it achieves significant savings as will be shown later.

Next, we show the accuracy results in terms of DBI and ASE for synthetic data in Figure 4. We vary the number of points from 1K to 4M and compute the DBI and ASE values for the clusters produced by four algorithms: DASC,

PSC, SC, and NYST. Figure 4(a) shows that the DBI values achieved by DASC are very close to those produced by SC. When checking the DASC curve alone, the DBI values, although go through some slight ups and downs, in general, they stay in the range between 1 to 1.3 and are always close to the SC's curve. Figure 4(b) shows that DASC outperforms the PSC and NYST and yields close performance to that of SC, in terms of ASE, which measures how close the points in the clusters are to their respective centroids. Small ASE values indicate better clustering result. The results indicate that PSC and NYST are about 30% and 40%, respectively, apart from SC. Moreover, when the dataset grows, the PSC performance tends to deteriorate, as the average distance value grows slightly, which is unlike DASC that produces consistent results.

Finally, we present the results of the low-level Fnorm metric which captures the similarity between the approximated and original Gram matrices. We consider various number of data points: from 4K to 512K. Note that we could not use more than 512K points because we need to compare against the original Gram matrix which requires memory proportional to the square of the number points. We plot the ratio of the Fnorm value computed from the approximated matrix to the Fnorm value computed from the original full Gram matrix in Figure 5. We do this for different number of buckets used in the hashing step; we change the number of buckets from 4 to 4K. Note that larger numbers of buckets allow for higher degrees of parallelization and are desirable. However, larger numbers of buckets imply more partitioning of the dataset which can affect the clustering accuracy. The results shown in Figure 5 indicate that the approximated matrix does not lose significant information compared to the original matrix. The figure also shows that for the same dataset, increasing the number of buckets tends to decrease the Fnorm ratio, which means that the approximated matrix has less resemblance to the original matrix. In addition, for larger datasets, more number of buckets can be used before the Fnorm ratio starts to drop.

5.6 Results for Time and Space Complexities

We measure and compare the processing time and memory requirements for three different algorithms: DASC, PSC, and SC. SC is implemented in the Mahout library in Java using the MapReduce framework. PSC is implemented in C++ using MPI. DASC is implemented in Java using the MapReduce framework. We realize that different implementation languages and parallelization models can impact the running times. However, the orders of magnitudes performance gains observed in our experiments (as shown shortly) clearly overshadow such small differences. We also note that the NYST algorithm is implemented in Matlab and we could not run on the cloud. This experiment is conducted on the five-node cluster and for the Wikipedia and synthetic datasets. We show the results for the Wikipedia dataset in Figure 6; other results are similar [12]. As shown in Figure 6(a), DASC considerably improves the processing time. For example, for a dataset of size 2^{18} points, DASC runs more than an order of magnitude faster than PSC. For datasets larger than 2^{18}, PSC could not even terminate because of the large processing time and memory requirements. The basic SC algorithm in Mahout did not scale to datasets larger than 2^{15} and was orders of magnitudes slower than DASC.

Metric	64 nodes	32 nodes	16 nodes
Accuracy	95.6%	96.4%	96.6%
Memory	29444 KB	29412 KB	28919 KB
Time	20.3 hrs	40.75 hrs	78.85 hrs

Table 3: Results for running DASC on the Amazon cloud with different nodes.

The most important advantage of our proposed DASC algorithm is the substantial memory saving, which is confirmed by Figure 6(b). The numbers shown in the figure are for total memory needed to store the Gram matrix. The figure shows that DASC achieves several orders of magnitude of memory saving compared to the basic SC implemented in Mahout. The figure also shows that although PSC uses sparse matrix representation, DASC requires substantially less memory than it. For example, for a dataset of size 2^{18}, there is a factor of more than 25 reduction in memory usage when comparing DASC versus PSC. More importantly, the memory usage curve for DASC is much flatter than SC and PSC curves. This means that DASC provides much better scalability to process very-large datasets than the other two algorithms.

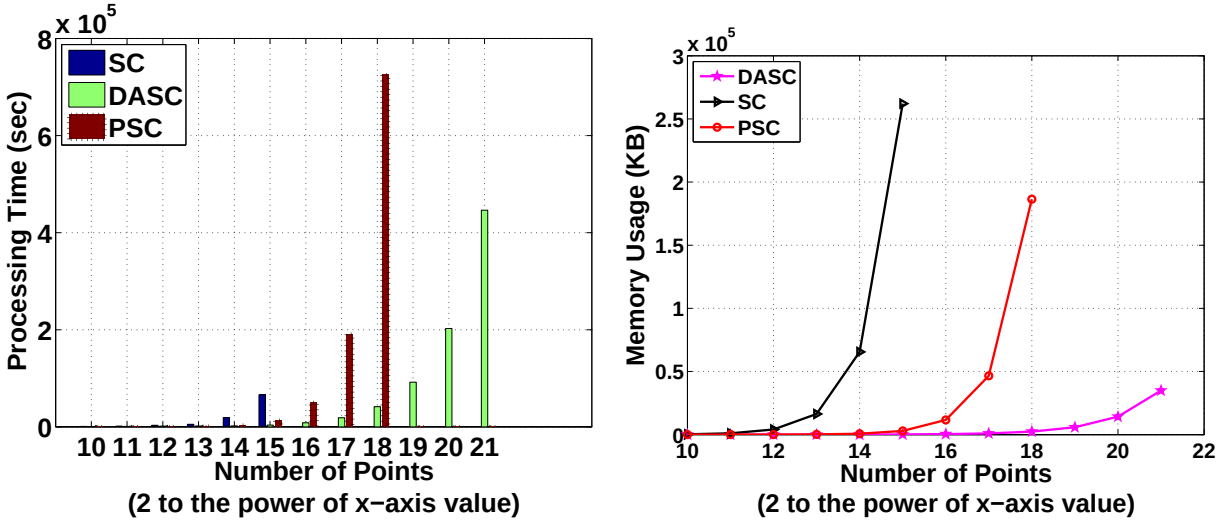
5.7 Elasticity and Scalability

One of the main advantages of using cloud platforms is elasticity, which enables cloud customers to easily request and utilize different amounts of computing resources based on the actual demand. In this section, we demonstrate that the proposed DASC algorithm, and hence the approximation method in general, can benefit from the elasticity offered by cloud platforms.

We run our DASC algorithm on the Wikipedia dataset on Amazon cloud and we vary the number of computing nodes from 16 to 64. We measure the accuracy, running time, and memory usage in each case. We summarize the results in Table 3. The results first demonstrate the scalability of the proposed DASC algorithm, since the running time reduces approximately linearly with increasing the number of nodes, while the memory usage and clustering accuracy stay roughly the same. This scalability is achieved mainly by the proposed preprocessing step, which partitions a given large dataset into independent and non overlapping partitions (hashing buckets). These parts can be allocated to independent computing nodes for further processing. This enables the utilization of various number of computing nodes once they become available, which results in faster processing of the dataset. On the other hand, if the number of computing nodes decreases, the DASC algorithm allocates more partitions per node, yielding correct results but with longer execution time. Therefore, DASC can efficiently and dynamically utilize different number of computing nodes.

6. CONCLUSIONS

We proposed new algorithms to support large-scale data-intensive applications that employ kernel-based machine learning algorithms. We presented an approximation algorithm for computing the kernel matrix needed by various kernel-based machine learning algorithms. The proposed algorithm uses locality sensitive hashing to reduce the number of pairwise kernel computations. The algorithm is general and can be used by many kernel-based machine learning algorithms.



(a) Processing Time

(b) Memory

Figure 6: Processing time and memory requirements for different algorithms.

We designed a distributed approximate spectral clustering (DASC) algorithm based on the approximation algorithm. We showed that DASC can offer a factor of up to $O(B)$ reduction in running time and memory usage compared to the spectral clustering algorithm that uses the full kernel matrix. B is the number of buckets used by the locality sensitive hashing step. B depends on the number of bits in the binary signature generated for each data point by the hashing functions. For large data sets, more bits in the signatures are needed, and thus more buckets. This means that the reduction factor achieved by our algorithm increases as the size of the dataset grows, which is an important and desirable property of our algorithm that enables it to scale to massive data sets.

We implemented DASC in the MapReduce framework, and ran it on a Hadoop cluster in our lab as well as on the Amazon Elastic MapReduce (EMR) service. We used the DASC algorithm to cluster various synthetic and real datasets. The real dataset contains more than three million documents from Wikipedia. We compared the clustering accuracy produced by the DASC algorithm versus the ground truth document categorization provided by Wikipedia. Our results showed that the DASC algorithm achieves high clustering accuracy of more than 90%, and its accuracy is very close to the regular spectral clustering algorithm that uses the full kernel matrix. Whereas the running time and memory usage of the DASC algorithm is several orders of magnitudes smaller than the regular spectral clustering algorithm. We also compared the DASC algorithm versus other algorithms in the literature and showed that it outperforms them in clustering accuracy, running time, and memory requirements.

Acknowledgments

This work is partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada and the British Columbia Innovation Council (BCIC).

7. REFERENCES

[1] A. Anagnostopoulos, A. Dasgupta, and R. Kumar. Approximation algorithms for co-clustering. In *In Proc. of Symposium on Principles of Database Systems (PODS'08)*, pages 201–210, Vancouver, BC, Canada, June 2008.

[2] M. Charikar. Similarity estimation techniques from rounding algorithms. In *In Proc. of ACM Symposium on Theory of Computing (STOC'02)*, pages 380–388, Montréal, Canada, May 2002.

[3] W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):568–586, March 2011.

[4] O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: min-hash and tf-idf weighting. In *In Proc. of British Machine Vision Conference (BMVC'08)*, pages 25–31, Leeds, UK, September 2008.

[5] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[6] J. Cullum and R. Willoughby. Lanczos algorithms for large symmetric eigenvalue computations. *IEEE Transactions on Information Theory*, pages 43–49, 1985.

[7] D. Davies and D. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1:224–227, 1979.

[8] L. M. Delves and J. Walsh, editors. *Numerical Solution of Integral Equations*. Clarendon, Oxford, 1974.

[9] P. Drineas and M. Mahoney. Approximating a gram matrix for improved kernel-based learning. In *In Proc. of Annual Conference on Computational Learning Theory*, pages 323–337, 2005.

[10] X. Fern and C. Brodley. Random projection for high dimensional data clustering: a cluster ensemble

approach. In *In Proc. of International Conference on Machine Learning (ICML'03)*, pages 186–193, 2003.

[11] B. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

[12] F. Gao. Distributed Approximate Spectral Clustering for Large-Scale Datasets. Master's thesis, Simon Fraser University, Canada, 2011.

[13] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, pages 22–29, 1979.

[14] E. Hatcher and O. Gospodnetic. *Lucene in Action*. Manning Publications Co., Greenwich, CT, USA, 2004.

[15] J. Hennessy and D. Patterson. *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann, 2003.

[16] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[17] U. Kang, C. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *In Proc. of IEEE International Conference on Data Mining (ICDM'09)*, pages 229–238, Washington, DC, December 2009.

[18] J. Kubica, J. Masiero, A. Moore, R. Jedicke, and A. Connolly. Variable kd-tree algorithms for efficient spatial pattern search. Technical Report CMU-RI-TR-05-43, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, September 2005.

[19] R. B. Lehoucq, D. C. Sorensen, and C. Yang. Arpack users guide: Solution of large scale eigenvalue problems by implicitly restarted arnoldi methods, 1997.

[20] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. *ACM Transactions on the Web*, 1, May 2007.

[21] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *In Proc. of ACM Conference on World Wide Web (WWW'10)*, pages 641–650, April 2010.

[22] J. Lin, D. Ryaboy, and K. Weil. Full-text indexing for optimizing selection operations in large-scale data analytics. In *In Proc. of International Workshop on MapReduce and its Applications*, pages 59–66, June 2011.

[23] A. Matsunaga, M. Tsugawa, and J. Fortes. Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications. In *In Proc. of IEEE International Conference on eScience*, pages 222–229, Indianapolis, IN, December 2008.

[24] T. Moon and W. Stirling. *Mathematical methods and algorithms for signal processing*. Prentice-Hall, Inc., 2000.

[25] R. Motwani, A. Naor, and R. Panigrahi. Lower bounds on locality sensitive hashing. In *In Proc. of Annual Symposium on Computational Geometry (SCG'06)*, pages 154–157, 2006.

[26] S. Munder and D. Gavrila. An experimental study on pedestrian classification. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(11):1863–1868, Nov. 2006.

[27] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856. MIT Press, 2001.

[28] J. Ohmer, F. Maire, and R. Brown. Implementation of kernel methods on the gpu. In *In Proc. of Conference on Digital Image Computing: Techniques and Applications*, page 78, Washington, DC, USA, December 2005.

[29] S. Owen, R. Anil, T. Dunning, and E. Friedman. *Mahout in Action*. Manning Publications, 2011.

[30] M. F. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.

[31] B. Schlkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, July 1998.

[32] J. Schuetter and T. Shi. Multi-sample data spectroscopic clustering of large datasets using Nystrom extension. *Journal of Computational and Graphical Statistics*, pages 531–542, 2011.

[33] M. Seeger. *Bayesian Model Selection for Support Vector Machines, Gaussian Processes and Other Kernel Classifiers*, volume 12, pages 603–609. The MIT Press, 2000.

[34] Y. Weiss. Segmentation using eigenvectors: A unifying view. In *In Proc. of International Conference on Computer Vision*, pages 975–982, 1999.

[35] B. White, T. Yeh, J. Lin, and L. Davis. Web-scale computer vision using MapReduce for multimedia data mining. In *In Proc. of ACM Workshop on Multimedia Data Mining*, 2010.

[36] C. K. I. Williams and M. Seeger. The effect of the input density distribution on kernel-based classifiers. In *International Conference on Machine Learning*, 2000.

[37] C. K. I. Williams and M. Seeger. *Using Nystrom method to speed up kernel machines*, volume 13 of *Advanced in Neural Information Processing Systems*. MIT Press, 2001.

[38] J. Yang and J.-Y. Yang. From image vector to matrix: a straightforward image projection technique - IMPCA vs. PCA. *Pattern Recognition*, 35:1997–1999, 2002.

[39] D. Yogatama and K. Tanaka-Ishii. Multilingual spectral clustering using document similarity propagation. In *In Proc. of Conference on Empirical Methods in Natural Language Processing*, pages 871–879, 2009.

[40] R. Zass and A. Shashua. Doubly stochastic normalization for spectral clustering. In *Neural Information Processing Systems*, pages 1569–1576, 2006.