Parallel Clustering of Large Document Collections

Deyun Gao *dgao@cs.dal.ca* Xiaohu Li *xiaohu@cs.dal.ca* Zheyuan Yu *zyu@cs.dal.ca*
Faculty of Computer Science
Dalhousie University
Halifax, Canada B3H 1W5

July 17, 2003

Abstract

blah

1 Introduction

Document clustering is the process of organizing documents into clusters so that documents within a cluster have high similarity in comparison to one another, but are very dissimilar to documents in other cluster. It has been studied intensively because of its wide applicability in areas such as web mining, search engines, information retrieval, and topological analysis.

A vector space model, also know as bag-of-words model [22], need to be created before applying document clustering algorithm. A vector is used to represent each document in the document collection. Each direction of the vector space corresponds to a unique term in the document collection. The value of the term direction is a function of term frequency and document frequency (TF-IDF) [22] which reflects the importance of the term in representing the semantics of the document. Similarity between two documents is traditionally measured by the cosine of the angle between their vectors. By using the vector space model, the problem of document clustering becomes the problem of grouping closing document vectors in a vector space.

2 Challenge

The document vectors are very high dimensional because even a small document collection may have thousands unique terms. High dimensionality poses a challenge to document clustering algorithms. K. Beyer et al. [11] have shown that in high dimensional space, the distance to the nearest data point approaches the distance to the farthest data point. In such situation, the similarity measure of the clustering algorithms do not work efficiently, hence the meaningfulness of clustering may be doubtful. This problem was traditionally referred to as dimensionality curse [2].

In addition, large collections of documents are becoming increasingly common. The public internet currently has more than 3 billion web pages, while private intranets also contain an abundance of text data. It is a great challenge to efficiently cluster such huge amount of document collection. The use of parallel computing techniques in large scale document clustering is unavoidable.

In this project, our main concern is in obtaining an effective document clustering algorithm and implemented it in parallel to get a high efficient process for clustering large document collections.

3 Literature Review

There exists a large number of clustering algorithms. J. Han and M. Kamber [15] categorized the major clustering methods into hierarchical methods [14][16][20][26], partition methods [6][8][23], density based methods [10], grid based methods [1] and model based methods [12]. Some clustering algorithms integrate the ideas of several clustering methods [18]. Among these methods, hierarchical and partition methods are two major techniques used in documents clustering.

The hierarchical clustering builds a cluster hierarchy, which graphically displays the merging process and the intermediate clusters, also know as dendrogram. There are two main types of hierarchical clustering algorithms. Agglomerative algorithms initialize each document as a cluster, then recursively merge two or more most similar documents into a new cluster until a certain stopping criterion is met. On the other hand, divisive algorithms start with all documents as one cluster and, at each step, split a cluster until a certain stopping criterion is satisfied. Agglomerative are more common than divisive algorithms. Hierarchical clustering algorithm are effective, but practical infeasible for large document collection because of its quadratic time complexity in the number of documents [23].

In contrast to hierarchical clustering, partition clustering creates one-level partitions such that the documents in a cluster are more similar to each other than to objects in different clusters. The popular K-means method selects K documents in the document collection as initial clustering, then assigns each document to one of the nearest clusters. The cluster centroid's position is recalculated each time when a document is added to the cluster and this continues until all the documents are grouped into the final required number of clusters. K-means and its variants are widely used partition techniques. They are computational cost efficient compared to hierarchical algorithms. For a document set with n documents, m unique terms and k clusters, the time complexity of each iteration of K-means algorithm is $O(nmk)$ [9]. Moreover, Ying Zhao showed that partitional methods can produce hierarchical solutions as good as agglomerative methods [28].

The Principal Direction Divisive Partitioning (PDDP) proposed in [5] is a divisive hierarchical clustering algorithm based on the principal component analysis instead of traditional distance or similarity measure. By calculating the leading principal direction for partitioning and taking the advantage of the sparsity of document vector, PDDP reduces dimensionality of the vector space, hence it can efficiently cluster large data set. Its computational cost is linear in the number of documents, as shown in [4]. It proceeds recursively dividing the document collection into two clusters by using the principal directions. PDDP belongs to the class of SVD-based data-processing algorithms [19][3]. Latent Semantic Indexing (LSI) [24] [7] is also one of this type of algorithms. Unlike LSI, which computing k singular values and vectors of the document matrix, PDDP computes just one leading principal direction. This makes PDDP significantly less computationally demanding than LSI. Also, the sparsity of the document matrix make the Lanczos algorithm [13], which is a fast method for computing a partial SVD of the matrix can be applied in PDDP to reduce the computational cost.

In our project, we are going to implement PDDP based on the message passing model.

A recent paper [25] working on parallel PDDP will be a good reference for our project. The performance of the parallel PDDP will be evaluated. The experiments to test parallel run time and speedup, also entropy and purity [27] are considered to be done in our project.

We introduce the PDDP algorithm in detail in Section 4 and our parallel implementation in Section 5.

4 The PDDP algorithm

The Principal Component Divisive Partition algorithm recursively partitions the vector space with hyperplane. The mean of documents within the cluster and the principal direction with respect to that mean are calculated to define the hyperplane for each iteration. The hyperplane is normal to the principal direction and passes through mean. The result of PDDP is a binary tree of the clusters.

PDDP operates directly on a $n \times m$ term-by-document matrix $M = (d_1, ..., d_m)$ whose i-th column, d_i, is the term vector representing the i-th document.

Initially, the matrix M is viewed as a single cluster. Its centroid or cluster is defined as

$$w = \frac{1}{m}\sum_{i=1}^{m} d_i = \frac{1}{m}Me,$$

where $e = (1, 1, ..., 1)^T$.

The covariance matrix C of the matrix M is computed as

$$C = (M - we^T)(M - we^T)^T = AA^T,$$

Where $A = (M - we^T)$.

The eigenvectors of C correspond to its largest eigenvalues are called the principal components or principal directions. To find the largest eigenvalues of C, we can compute the Singular Value Decomposition (SVD) [13] of A, in the form of $A = U\Sigma V^T$. If M is an $n \times m$ matrix, then Σ is an $n \times m$ diagonal matrix , and U and V are orthonormal unitary square matrices of dimensions $n \times n$ and $m \times m$, respectively. So we have

$$C = AA^T = (U\Sigma V^T)(V\Sigma^T U^T) = U\Sigma^2 U^T.$$

The eigenvectors of C are the columns of U. Assume that the first column vector of U is μ, then the ith document d_i is projected onto the leading singular vector by :

$$\lambda\nu_i = \mu^T(d_i - w),$$

where λ is a positive constant (the largest singular value of C). In the simplest version of the algorithm, if $\nu_i <= 0$, the document d_i is partitioned into the left child of the cluster, otherwise, it is partitioned into the right child of the cluster.

Having described the methods used to split a given node, the remaining question is to decide at each stage which node should be split next. One choice is to try to keep the binary tree balanced by splitting all the nodes at a given level (distance from root) before proceeding to the next level. But the resulting clusters are often imbalanced with a few large clusters and many small clusters, including many singletons. To avoid this situation, PDDP uses a "scatter" value as a measure of the noncohesiveness of a cluster. It uses a norm-based scatter value. The **scat** field is a total scatter value defined to be the Frobenius norm of the corresponding matrix $A = M_p - we^T$. The square of the Frobenius norm of $A = (a_{ij})$ is given by

$$\| A \|_F^2 = \| C \|_F = \sum_{i,j} | a_{i,j} |$$

and equals the Frobenius norm of the covariance matrix C as well as the sum of the eigenvalues σ_i^2 of C [13]:

$$\| A \|_F^2 = \| C \|_F = \sigma_i^2$$

In PDDP algorithm, the total scatter value is used to select the next cluster to split. We choose the cluster with the largest scatter value. The total scatter value reflects the distance between each document in the cluster and the overall mean of the cluster, which is a measure of the cohesiveness of the cluster. Using the total scatter value to choose the next cluster to be split usually results in clusters all having more or less similar numbers of documents. We remark that this scatter value is the only component of this algorithm that is based on a "distance" measure, and it would be just as easy to use other measures not based on a "distance" measure and appropriate for particular data sets.

Algorithm description:

```
0.  Start with $n$ x $m$ matrix $M$ of (scaled)document vectors, and a desired number of clusters $C_{max}$
1.  Initialize Binary Tree with a single Root Node.
2.  For $c = 2, 3, ..., C_{max}$ do
3.      Select node $K$ with largest scat value
4.      Create node L := leftchild(K) and R := rightchild(K).
5.      Set indices(L) := indices of non- positive entries in $rightvec(K)$
6.      Set indices(R) := indices of the positive entries in $rightvec(K)$
7.      Compute all the other fields for the nodes L,R
8.      end
9.  Result: A binary tree with $C_{max}$ leaf nodes forming a partitioning of the entire data set.
```

5 Approach

5.1 Parallel implementation of PDDP algorithm

5.1.1 Data distribution

We store the sparse matrix M in the Compress Sparse Row(CSR)[21] format. Each of the processor has a data structure consisting os three arrays:

1) a real number array MM containing the real values of nonzero elements of M stored on that processor row by row;

2) an integer number array CM for the column indices, corresponding to the array MM;

3) an integer number array RM containing the pointers to the beginning of each row in the arrays MM and CM.

If we use n_z to denote the number of nonzero entries in the matrix M, and n is the number of rows, the storage cost of storing M is n_z real numbers plus $(n_z + n + 1)$ integer numbers. In practice, often less than one percent of M are nonzero. Thus at least 98% of the storage is saved by using the CSR format, and accordingly, the time complexity of performing the matrix vector operations can be greatly reduced.

The distribution of data has an important effect on the performance of the parallel algorithm[17]. Our approach is to distribute different parts of each file to different processors.

5.1.2 Parallel PDDP algorithm

We use C_j, $j = 1, 2, ..., k$,to denote the obtained clusters. At first, we have only root cluster corresponding to the entire document set.

5.1.3 Algorithm description:

Input: tree-height h, matrix M
Output: an array that records the assignment of each document to a cluster

```
1. Read matrix;
2. Distribute matrix;
3. For $i = 1, ..., h$
4.     For $j = 1$,...,the number of clusters at level $i$
5.         if ($C_j$ is a singleton) goto line 4;
6.         $w$ = Mean of vector($C_j$;
7.         $u$ = leading eigenvector($C_j, w$)
8.         For $k = 1$,...,the number of documents in $C_j$
9.             v = Dot product$u, d_k$;
10.            If ($v >= 0$),then
11.                put document $k$ as left child of $C_j$
12.            else
13.                put document $k$ as right child of $C_j$
14.            end if
15.        end for
16.    end for
17. end for
```

In line 1, the matrix M is read from a file . In line 2,the matrix M is distributed to the processors. To every current leaf cluster of the tree,we first check if it is a singleton. A singleton means that its document set is exactly the same as that of its parent cluster, which implied that the cluster can not be partitioned any more. Here we assume that the matrix M is not a singleton, which is always true in practice. If the the cluster C_j is a singleton, we do nothing to it and the leading eigenvector. For every document in the cluster, we compute the dot product of that document and the leading eigenvector. If the result is greater than or equal to 0, we put the document into the left child of that cluster, otherwise,we put it into the right child of that cluster.

The most time consuming part of this parallel algorithm is the function leading_eigenvector, which calculates the leading eigenvector of the covariance matrix C. We will use Lanczos algorithm[13] to compute the leading singular vector. Lanczos is a popular algorithm. for solving large, sparse, symmetric eigenproblems. This method generates a sequence of tridiagonal matrices T_j with the property that the extremal eigenvalues of the $j \times j$ matrix T_j are progressively better estimates of the extremal eigenvalues of the original matrix H. Since the term by document matrix is typically very sparse with almost 99% of the matrix entries being zero. By taking the advantage of the sparsity, the Lanczos algorithm is very efficient, with cost proportional to the number of nonzeroes in the term by document matrix.

References

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *In Proc. SIGMOD'98, Seattle, Washington*, June, 1998.

[2] R. Bellman. *Adaptive Control Processes: A Guided Tour.* Princeton University, Princeton, 1961.

[3] M.W. Berry, Z. Drmac, and E.R. Jessup. Umatrices, vector spaces, and information retrieval. *SIAM Review*, 41:335–362, 1999.

[4] D. Boley, Gini M. Gross, R., Han S., Hastings K. Karypis, and G. Kumar V. Partitioning-based clustering for web document categorization. *Decision Support Systems*, 27:329–341, 1999.

[5] D.L. Boley. Principal direction divisive partitioning. Technical report, Technical Report TR-97-056,University of Minnesota, Minneapolis, 1997.

[6] P. S. Bradley, U. M. Fayyad, and C. A. Reina. Scaling em (expectation maximization) clustering to large database. Technical report, Microsoft Research Technical Report, November 1998.

[7] Deerwester, S. Dumais S., G. Furnas, and R. Harshman. Indexing by latent semantic analysis. *J. Amer. Soc. Inform. Sci*, 41:41–50, 1990.

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

[9] R.C. Dubes and A. K. Jain. *Algorithms for Clustering Data.* Prentice Hall College Div, Englewood Cliffs, NJ, March 1998.

[10] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *In Proc. KDD'96, Portland, Oregon.* AAAI Press, 1996.

[11] K. Beyer et. al. When is nearest neighbor meaningful? In *In proceeding of the 7th ICDT*, Jerusalem, Israel, 1999.

[12] D. Fisher. Improving inference through conceptual clustering. In *In Proc. 1987 AAAI Conf.*, pages 461–465, Seattle, Washington, July 1987.

[13] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins Univ. Press, 3rd edition, 1996.

[14] Sudipto Guha, Rajeev Rastogi, and Kyuscok Shim. Cure: An efficient clustering algorithm for large databases. In *In Proc. of ACM-SIGMOD Int. Conf. Management of Data (SIG-MOD'98)*, pages 73–84, 1998.

[15] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, 2001.

[16] George Karypis, Eui-Hong (Sam) Han, and Vipin Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *Computer*, 32:68–75, 1999.

[17] Y. Kumar, A. Grama, and ect A. Gupta. *Introduction to parallel computing,Design and Analysis of Algorithms*. Benjamin/Cummings Pub. Co.. Don Mills,Ontario, 1994.

[18] GG. Lacey and K. Dawson-Howe. Scatter/gather: A cluster-based approach to browsing large document collections. In *In Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329, 1992.

[19] Berry M.W., S.T. Dumais, and G.W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37:573–595, 1995.

[20] Clark F. Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 21:1313–1325, 1995.

[21] Y. Saad. *Iterative Methods for sparse Linear Systems*. PWS Publishing Co.,Boston, 1996.

[22] G. Salton and M. J. McGill. *Introduction to Modern Retrieval*. McGraw-Hill Book Company, 1983.

[23] Michael Steinbach, George Karypis, and Vipin Kumar. A comparison of document clustering techniques. In *In Text Mining Workshop, KDD*, 2000.

[24] Anderson T. On estimation of parameters in latent structure analysis. *Psychometrica*, 19:1–10, 1954.

[25] ShuTing Xu and Jun Zhang. A hybrid parallel web document clustering algorithm and its performance study. Technical report, Technical Report No. 366-03, Department of Computer Science, University of Kentucky, 2003.

[26] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *In Proc. of ACM-SIGMOD Int. Conf. Management of Data (SIG-MOD'96)*, pages 103–114, 1996.

[27] Y. Zhao and G. Karypis. Criterion function for document clustering experiments and analysis. Technical report, Technical Report No. 01-40, University of Minnesota, Minneapolis, MN, 2001.

[28] Ying Zhao and George Karypis. Evaluation of hierarchical clustering algorithms for document datasets. Technical report, University of Minnesota, 2002.