

A Parallel Learning Algorithm for Text Classification

Canasai Kruengkrai and Chuleerat Jaruskulchai
Intelligent Information Retrieval and Database Laboratory
Department of Computer Science, Faculty of Science
Kasetsart University, Bangkok, Thailand
{g4364115,fscichj}@ku.ac.th

ABSTRACT

Text classification is the process of classifying documents into predefined categories based on their content. Existing supervised learning algorithms to automatically classify text need sufficient labeled documents to learn accurately. Applying the Expectation-Maximization (EM) algorithm to this problem is an alternative approach that utilizes a large pool of unlabeled documents to augment the available labeled documents. Unfortunately, the time needed to learn with these large unlabeled documents is too high. This paper introduces a novel parallel learning algorithm for text classification task. The parallel algorithm is based on the combination of the EM algorithm and the naive Bayes classifier. Our goal is to improve the computational time in learning and classifying process. We studied the performance of our parallel algorithm on a large Linux PC cluster called PIRUN Cluster. We report both timing and accuracy results. These results indicate that the proposed parallel algorithm is capable of handling large document collections.

Keywords

Text classification, parallel expectation-maximization (EM) algorithm, naive Bayes, cluster computing

1. INTRODUCTION

Text classification has become one of the most important techniques in text data mining. The task is to automatically classify documents into predefined classes based on their content. Many algorithms have been developed to deal with automatic text classification. One of the common methods is the naive Bayes. Although the naive Bayes works well in many studies [7][9][10], it requires a large number of labeled training documents for learning accurately. In the real world task, it is very hard to obtain the large labeled documents, which are mostly produced by humans. To overcome this shortage, Nigam et al. [13] apply the Expectation-Maximization (EM) algorithm to the text classification problem. The EM algorithm uses both labeled and unlabeled documents for learning. Their experimental results show that using the EM algorithm with unlabeled documents can reduce classification

error when there is a small number of training data. Unfortunately, the EM algorithm is too slow when it performs on very large document collections. Furthermore, the text data rapidly increase on the World Wide Web. The scalability of the algorithm is required to handle such massive data.

The parallel processing is an interesting technique for scaling up the algorithms. For example, the parallel algorithms are applied for mining association rules [1] and classification based on decision tree [8]. Several researches study techniques for parallelizing clustering algorithms, which can be considered as the unsupervised learning problem. Ruocco and Frieder [15] propose parallel single-link and single-pass algorithm for clustering documents worked on an Intel Paragon. Dhillon and Modha [3] introduce an effective parallelization of the k-means clustering algorithm implemented on an IBM POWERparallel SP2. Forman and Zhang [4] also present a general technique for parallelizing a class of center-based clustering algorithms including k-means, k-harmonic means, and EM algorithm performed their work on existing network structures (LAN). The similar ideas of the last two works are to use data-parallelism and limit the communication among processes to only some parameters.

In this paper, we propose a novel parallel learning algorithm for text classification task. We focus on parallelizing supervised learning algorithm that combines the EM algorithm and the naive Bayes classifier. To the best of our knowledge, the parallel version of the combination algorithm has not been reported in the literature. Our experiments performed on a large Linux PC cluster called PIRUN (Pile of Inexpensive and Redundant Universal Nodes) Cluster. The experimental results show that our parallel implementation has reasonable speedup characteristics.

The paper is organized as follows. In Section 2, we describe how text data is represented. Section 3 briefly reviews the probabilistic framework for text classification task including the naive Bayes classifier and the EM algorithm. In Section 4, we present the parallel implementation of the EM algorithm. Section 5 shows the complexity analysis of the algorithm. In Section 6, we present the experimental results. In particular, we measure the performance of the parallel algorithm by exploring both speedup and accuracy. Finally, Section 7 is the conclusion of our work.

2. TEXT DOCUMENT REPRESENTATION

Typically, text documents are unstructured data. Before learning process, we must transform them into a representation that is suitable for computing. We employ the vector space model, widely used in information retrieval [2], to represent the text documents.

The vector space model is also known as the bag-of-words representation. The representation method is equivalent to an attribute value representation used in machine learning [12]. Each distinct word is a feature (or index term) and the number of times the word occurs in the document is its value (or term frequency).

Let us describe how to construct the vector space model from a document collection. First, we parse the document collection to extract unique words and prune non-content words (or stop-words), low and high frequency words. Then, we obtain $w = (w_1, w_2, \dots, w_k, \dots, w_V)$, where V is the number of the unique words within the collection. In the vector space model, we ignore the sequence in which the word occurs. Next, each document is represented by a vector $d_i = (w_{i1}, w_{i2}, \dots, w_{ik}, \dots, w_{iV})$, where w_{ik} is the frequency of k th word in a document d_i . Finally, we map the entire document vectors to a matrix called the document-word matrix (see Figure 1). Each row of the matrix corresponds to a document vector. The columns of the matrix correspond to the unique terms in the document collection.

	w_1	w_2	...	w_k	...	w_V
d_1	w_{11}	w_{12}	...	w_{1k}	...	w_{1V}
d_2	w_{21}	w_{22}	...	w_{2k}	...	w_{2V}
...
d_i	w_{i1}	w_{i2}	...	w_{ik}	...	w_{iV}
...
d_N	w_{N1}	w_{N2}	...	w_{Nk}	...	w_{NV}

Figure 1. The document-word matrix used for representing a document collection.

3. PROBABILISTIC FRAMEWORK

Suppose that we have training documents $D = \{d_1, \dots, d_N\}$, where each document is labeled by $y_i \in C = \{c_1, \dots, c_M\}$. We write $y_i = c_j$ when a document d_i belongs to a class c_j . We use Θ to denote the parameters of the model. We assume that documents are generated by a mixture of multinomial model, and there is a one-to-one correspondence between mixture components and classes. Each document d_i is generated by choosing a mixture component with the class prior probabilities $P(c_j | \Theta)$, and having this mixture component generate a document according to its own parameters, with distribution $P(d_i | c_j; \Theta)$. Thus we can write:

$$P(d_i | \Theta) = \sum_{j=1}^M P(c_j | \Theta) P(d_i | c_j; \Theta). \quad (1)$$

3.1 Naive Bayes Classifier

The naive Bayes classifier uses the maximum a posteriori (MAP) estimation for learning a classifier. It assumes that the occurrence of each word in a document is conditionally independent of all other words in that document given its class. Using this assumption, the probability of a document given its class becomes:

$$P(d_i | c_j; \Theta) = P(w_{i1}, \dots, w_{i|d_i|} | c_j; \Theta) \propto \prod_{k=1}^{|d_i|} P(w_{ik} | c_j; \Theta). \quad (2)$$

Therefore, the parameters of the model are the conditional probabilities $\hat{\theta}_{w_k | c_j} = P(w_k | c_j; \Theta)$, and the prior probabilities $\theta_{c_j} = P(c_j | \Theta)$. That is:

$$\Theta = \{\theta_{w_1 | c_1}, \dots, \theta_{w_V | c_M}; \theta_{c_1}, \dots, \theta_{c_M}\}. \quad (3)$$

Let $\hat{\Theta}$ be the estimations of Θ . The parameters $\theta_{w_k | c_j}$ can be estimated by using Laplace smoothing that adds one to all the word counts to avoid probabilities of zero. Thus we obtain:

$$\hat{\theta}_{w_k | c_j} = \frac{1 + \phi(w_k, c_j)}{V + \sum_{w' \in w} \phi(w', c_j)}, \quad (4)$$

where $\phi(w_k, c_j)$ is the number of times that a word w_k occurs in the training documents for a class c_j . The parameters θ_{c_j} can be estimated as follows:

$$\hat{\theta}_{c_j} = \frac{\phi(d_i, c_j)}{N}, \quad (5)$$

where $\phi(d_i, c_j)$ is the number of training documents d_i that are assigned to a class c_j . Given estimations of these parameters calculated from the training documents, we can classify an unseen document to a single class that the posteriori probability is highest; $\arg \max_j P(c_j | d_i; \hat{\Theta})$. It can be calculated by applying Bayes' formula:

$$\begin{aligned} P(c_j | d_i; \hat{\Theta}) &\propto P(c_j | \hat{\Theta}) P(d_i | c_j; \hat{\Theta}) \\ &= P(c_j | \hat{\Theta}) \prod_{k=1}^{|d_i|} P(w_k | c_j; \hat{\Theta}) \\ &= P(c_j | \hat{\Theta}) \prod_{k=1}^{|d_i|} P(w_k | c_j; \hat{\Theta})^{\phi(w_k, d_i)}, \end{aligned} \quad (6)$$

where $\phi(w_k, d_i)$ is the number of times that a word w_k occurs in a document d_i .

3.2 Expectation-Maximization Algorithm

One drawback of the naive Bayes classifier is that it requires a large set of the labeled training documents for learning accurately. The cost of labeling documents is expensive, while unlabeled documents are commonly available. By applying the EM algorithm, we can use the unlabeled documents to augment the available labeled documents in the training process.

The EM algorithm is a general technique for maximum likelihood or maximum a posteriori estimation in the incomplete-data problems [11]. In our task, the class labels of the unlabeled documents are considered as the missing values. The document collection D now consists of the disjoint subsets of the labeled and the unlabeled documents: $D_l \cup D_u$. The probability function of all the documents is:

$$P(D|\Theta) = \prod_{d_i \in D_l} P(y_i = c_j | \Theta) P(d_i | y_i = c_j; \Theta) \\ \times \prod_{d_i \in D_u} \sum_{j=1}^M P(c_j | \Theta) P(d_i | c_j; \Theta). \quad (7)$$

For the labeled documents, the generating component is given by labels y_i , we do not need to refer to all mixture components [13]. As described earlier, we use the MAP estimation for learning a classifier, $\arg \max_{\Theta} P(\Theta | D)$. By making use of the Bayes' formula and Equation 7, we obtain the MAP estimation of Θ , which is equivalent to the value of Θ that maximizes the log-posteriori:

$$\log P(\Theta | D) = \sum_{d_i \in D_l} \log(P(y_i = c_j | \Theta) P(d_i | y_i = c_j; \Theta)) \\ + \sum_{d_i \in D_u} \log \sum_{j=1}^M P(c_j | \Theta) P(d_i | c_j; \Theta) + \log(P(\Theta)). \quad (8)$$

It is difficult to compute Equation 8 directly, because the second term contains a log of summations. Here we introduce the class indicator variables Z , where each $z_{ij} \in Z$ is defined to be one or zero according as d_i does or does not come from the j th class. By using the class indicator variables Z , we can write the complete-data log-posteriori in the form:

$$\log P_c(\Theta | D; Z) = \sum_{d_i \in D} \sum_{j=1}^M z_{ij} \log(P(c_j | \Theta) P(d_i | c_j; \Theta)) \\ + \log(P(\Theta)), \quad (9)$$

where the log of the priori $P(\Theta)$ is approximated by using Dirichlet distribution [13]. Since we do not know the exact values of Z , we instead work with their expectation. The algorithm finds a local maximum of the complete-data log-posteriori by iterating the following two steps:

$$\text{E-step: } \hat{Z}_{(k+1)} = E[Z | D; \hat{\Theta}_{(k)}] \\ \text{M-step: } \hat{\Theta}_{(k+1)} = \arg \max_{\Theta} P(\Theta | D; \hat{Z}_{(k+1)}), \quad (10)$$

where the E-step is the current parameter estimations of probabilistic labels for every documents calculated by Equation 6, and the M-step is the new MAP estimations for the parameters calculated by Equation 4 and 5.

4. PARALLEL IMPLEMENTATION

In this section, we present the parallel implementation of the EM algorithm for text classification. We employ the Single Program Multiple Data (SPMD) model in our parallelization. In this model, a single source program is written and each processor executes its personal copy of this program. We assume that we have P processors, where each processor is assigned a unique rank between 0 and $P-1$ and has an individual local memory. The processors communicate with each other by using MPI (Message-Passing Interface) library [6].

The EM algorithm starts by using the naive Bayes classifier to initialize the parameters. The E- and M-step are iterated until the change of $\log P_c(\Theta | D; Z)$ is less than some predefined threshold. The E-step almost dominates the execution time on each iteration,

-
1. Processor P_0 builds the initial global parameters Θ_g from only the labeled documents D_l , and broadcasts them to all processors
 2. Processor P_r reads training documents based on its responsibility from a disk
 3. Iterate until convergence
 - 3.1 E-step: Each processor P_r estimates the class of each document by using the current global parameters Θ_g
 - 3.2 M-step: Each processor P_r re-estimates its own local parameters Θ_l given the estimated class of each document
 - 3.3 Sum up the local parameters Θ_l to obtain the new global parameters Θ_g and return them to all processors
-

Figure 2. The outline of the parallel EM algorithm for text classification.

since it estimates the class labels for all the training documents. Fortunately, we observe that this step is inherently data parallel, because if the parameters Θ are available for each processor, the same operation can be performed on different documents simultaneously. We parallelize the loop by evenly distributing the documents across processors. If we partition the N documents into P blocks, each processor handles roughly N/P documents. In other words, P_r is given a responsibility for documents d_i , where $i = (r)(N/P) + 1, \dots, (r+1)(N/P)$.

Let Θ_l and Θ_g be the local and global parameters of the model, respectively. In the training process, the processor P_0 first computes the global parameters Θ_g from only the labeled training documents. Then, the processor P_0 distributes them to the available processors by using `MPI_Bcast`. We assign this task to only processor P_0 , because the naive Bayes classifier can learn in constant time. Next, each processor P_r uses the current global parameters Θ_g to label the unlabeled documents for its partition. Finally, each processor P_r calculates its local parameters Θ_l and calls `MPI_Allreduce` to sum up the local parameters Θ_l to obtain the new global parameters Θ_g . The algorithm uses these parameters as the current parameters in the next iteration. Since each processor has the same global parameters Θ_g , it can independently decide when it should exit the loop. Figure 2 gives the outline of the parallel EM algorithm.

The `MPI_Allreduce` function is a global communication operation that the result of the reduction operation is available in all processes [6]. The parameters of our model in Equation 3 are the probability estimations consisting of word and document counts among different classes. As a result, the local parameters Θ_l achieve the global parameters Θ_g by simply using `MPI_Allreduce` with the reduction operation `MPI_SUM` as shown in Figure 3. Our algorithm design can avoid the network bottleneck, because there are only the parameters that exchange across processes. In the test process, we use the final global parameters Θ_g to classify the test documents that are evenly partitioned for each processor as in the E-step.

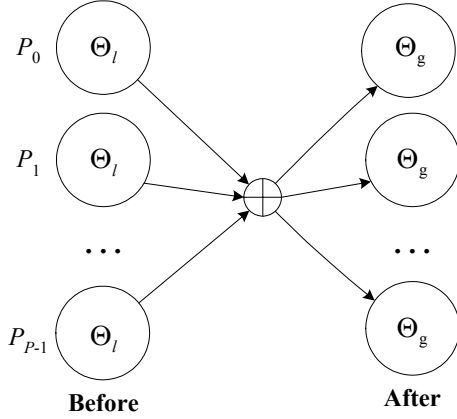


Figure 3. MPI_Allreduce with the reduction operation MPI_SUM.

5. COMPLEXITY ANALYSIS

The time complexity of the sequential EM algorithm can be calculated as follows. In training process, the initial step requires VN_{D_l} for estimating the parameters from the labeled documents. The E-step takes $VMN_{D_{train}}$, since the class estimation is performed on the entire training documents, including the labeled and unlabeled documents. The M-step takes $VN_{D_{train}}$. Let I be the number of iterations. As described earlier, the training process is dominated by the loop. The training process requires $VN_{D_l} + I(VMN_{D_{train}} + VN_{D_{train}})$, which can be approximated by $O(IVMN_{D_{train}})$. In test process, it requires $O(VMN_{D_{test}})$ similar to the E-step. Therefore, we obtain the overall time complexity $O(IVMN_{D_{train}} + VMN_{D_{test}})$.

The space complexity of the algorithm requires $2(VM + M)$ for storing the current and updated parameters and VN_s for storing the subsets of the document collection on demand. We finally obtain the total space complexity $O(2(VM + M) + VN_s)$.

For parallel processing, since each processor handles only N/P documents, the computational time decreases to at most $O(IVM(N_{D_{train}}/P) + VM(N_{D_{test}}/P))$. The communication time for exchanging the parameters is $O(I(VM + M)T_{data})$, where T_{data} is the transmission time for the parameters. Consequently, the overall parallel time complexity is estimated as:

$$O(IVM(N_{D_{train}}/P) + VM(N_{D_{test}}/P) + I(VM + M)T_{data}), \quad (11)$$

and the space complexity reduces to $O(2(VM + M) + V(N_s/P))$ for each node.

6. EXPERIMENTAL RESULTS

In this section, we give the experimental results to provide evidence that our parallel algorithm design can improve both the computational efficiency and the quality of classification. We implemented our parallel algorithm on PIRUN Cluster at Kaset-sart University. PIRUN Cluster consists of 72 nodes connected with Fast Ethernet Switch 3COM SuperStackII. Each node is a 500 MHz Pentium III with 128 Mbytes of RAM and uses Linux

as the operating system. It was constructed by using Beowulf architecture [14]. The configuration of components can be found at <http://pirun.ku.ac.th>.

6.1 Data Set and Performance Measure

The 20 Newsgroups data set was used in our experiments [7][10][13]. It consists of 20000 articles divided evenly among 20 different UseNet discussion groups. We extracted all unique words from the entire documents. After removing stop-words, low and high frequency words, we obtained 11350 unique words. We randomly selected 4000 (20%) of the collection as a test set. The first remaining documents were used to form a labeled training set, containing 6000 documents (30%) drawn at random. The last remaining documents were used as an unlabeled set consisting of 10000 documents (50%). Each set is represented by a document-word matrix.

Normally, the document-word matrix is very large and sparse, having mostly zero entries. For example, we have 10000 documents and extract 20000 unique words from all the documents. If we use 4 bytes for each element, the matrix requires $10000 \times 20000 \times 4 = 800$ Mbytes of main memory. Although we exploit from the distributed memory, reading this matrix can increase disk access costs. Moreover, it also causes network congestion, since we work on cluster. In order to reduce the matrix size, we look at a compression method called Scalar ITPACK [5]. The idea is to store non-zero elements of the matrix with their rows and column indices.

To measure the computational efficiency of our parallel algorithm, we examined the speedup (S). The speedup is the ratio of the execution time for learning and classifying a document collection on a single processor to execution time for the same tasks on P processors. Thus, the speedup of the parallel EM algorithm can be approximated as follows:

$$S = \frac{O(IVMN_{D_{train}} + VMN_{D_{test}})}{O(IVM(N_{D_{train}}/P) + VM(N_{D_{test}}/P) + I(VM + M)T_{data})}, \quad (12)$$

which increases linearly with P , if we have the large numbers of $N_{D_{train}}$ and $N_{D_{test}}$. We measured the elapsed time (disk accesses included) from start to complete the task. The classification result of each parallel execution is equivalent to its sequential execution.

6.2 Results

Figure 4 shows the curves of execution time on the 20 Newsgroups data set. Figure 5 demonstrates the relative speedups. The parallel EM algorithm was run on configurations of up to 16 processors. We varied the number of unlabeled documents to observe the effects of different problem sizes on the performance. Three sets were used with the number of unlabeled documents 2500, 5000, and 10000. For each set, the algorithm performs 6, 7, and 8 iterations, respectively. The number of dimensions V was also varied. The unlabeled documents were fixed at 10000, and the number of dimensions were varied at 6750 and 11350. Figure 6 shows the relative speedups.

The time of the initial step using the naive Bayes classifier does not affect the performance, since the naive Bayes can learn in constant time. From our experiments, it takes less than 18 seconds

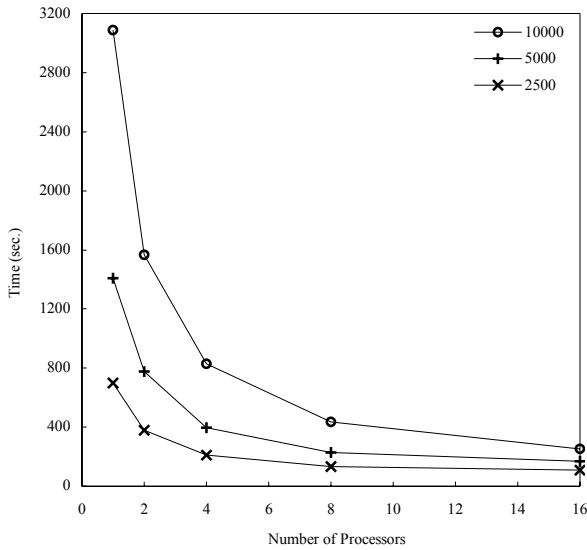


Figure 4. The execution time of the parallel EM algorithm with 2500, 5000, and 10000 unlabeled documents on 20 Newsgroups data set.

even though it uses the largest size of labeled training documents for learning. The computational time of the algorithm is mostly dominated by the loop in the training process. As we analyzed in the previous section, the speedup curves increase linearly in some cases. For example, on the largest unlabeled set, it achieves the relative speedups of 1.97, 3.72, 7.16, and 12.16 on 2, 4, 8, and 16 processors, respectively. When it accesses to a smaller set of unlabeled documents, the speedup curves tend to drop from the linear curve. The algorithm achieves the relative speedups of 1.82, 3.55, 6.18, and 8.38 on 2, 4, 8, and 16 processors, respectively. The smallest unlabeled document sizes give the same trend. If we increase the number of processors further, the speedup curves tend to significantly drop from the linear curve. For a given problem instant, the relative speedups decrease as the number of processors is increased due to increased overheads. This is a normal situation when the problem size is fixed as the number of processors increases. However, it can be solved by scaling the problem size. For example, in Figure 5, the speedups for three sets on 4 processors improve from 3.23 to 3.72, on 8 processors improve from 5.17 to 7.16, and on 16 processors improve from 6.46 to 12.16. It can be seen that our parallel algorithm yields better performance for the larger data sets.

To ensure that the EM algorithm works well with the unlabeled documents, we also re-examined the quality of classification. The number of labeled training documents was varied, and compared the accuracy with the naive Bayes classifier. The parallel EM algorithm accessed to 10000 unlabeled documents in learning process. The parallel execution produced the same classification results as sequential execution. In our experiments, five trials of selecting test/train/unlabeled splits at random were conducted, and each reported accuracy was interpreted as an average over the five

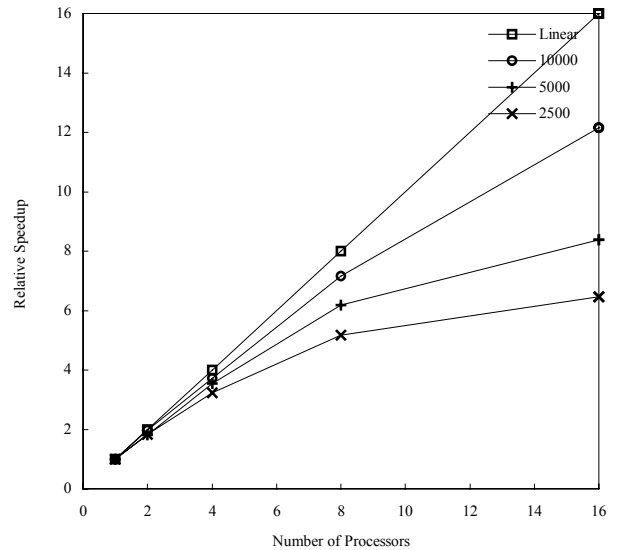


Figure 5. The relative speedup curves of the parallel EM algorithm corresponding to Figure 4.

trials. Figure 7 shows the results on accuracy. We observe that the EM algorithm significantly outperforms the naive Bayes classifier when the amount of the labeled documents is small. For example, the EM algorithm achieves 36% accuracy while the naive Bayes classifier reaches 21% accuracy at 20 labeled documents (or one document per category). With 100 labeled documents, the EM algorithm achieves 59% accuracy while the naive Bayes classifier reaches 35% accuracy. The two approaches begin to converge when the amount of the labeled documents is large. We can see that the EM algorithm constantly outperforms the naive Bayes on the 20 Newsgroups data set. However, work by [13] also shows that the EM hurts accuracy on some data sets. When the labeled documents are large, the accuracy curve drops slightly. The reason is that the data do not fit the assumptions of the generative model. This indicates that using the simple generative model is inadequate to produce accurate classification results. This problem can be solved by using more complex statistical model. We believe that our parallelization strategy can adapt to that model by adding some parameters.

7. CONCLUSIONS

In this paper, we presented the parallelization of the EM algorithm in the area of text classification. Since the EM algorithm uses both the labeled and unlabeled documents for learning, the computational time of the algorithm also increases. Consequently, the parallel processing was applied to the algorithm. We parallelized the EM algorithm by using the idea of data parallel computation. We evaluated our parallel implementation on a large Linux PC cluster called PIRUN Cluster. The experimental results on the efficiency indicate that our parallel algorithm design has good speedup characteristics when the problem sizes are scaled up.

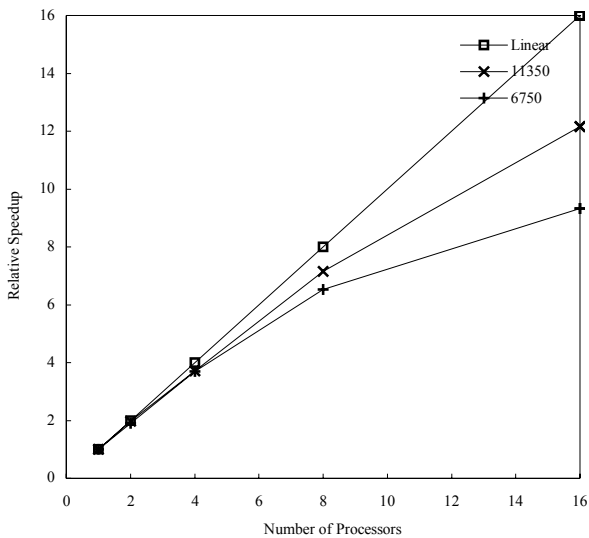


Figure 6. The relative speedup curves using $V = 6750$ and 11350. The unlabeled documents were fixed at 10000.

In future work, we will look at the improvement of the disk I/O. Our current implementation is based on the basic Unix I/O functions. When the problem sizes are scaled up, we require high performance I/O to reduce disk access costs. The parallel I/O defined the MPI-2 standard [16] is one solution. We expect that the parallel I/O will deliver much higher performance.

8. REFERENCES

- [1] Agrawal, R., and Shafer, J. C. Parallel mining of association rules. *IEEE Transaction on Knowledge and Data Engineering*, 1996.
- [2] Baeza-Yates, R., and Ribeiro-Neto, B. *Modern information retrieval*. The ACM Press, New York, 1999.
- [3] Dhillon, I.S., and Modha, D.S. A data-clustering algorithm on distributed memory multiprocessors. *Large-Scale Parallel Data Mining*, pages 245-260, 1999.
- [4] Forman, G., and Zhang, B. Linear speed-up for a parallel non-approximate recasting of center-based clustering algorithms, including k-means, k-harmonic means, and EM. *KDD Workshop on Distributed and Parallel Knowledge Discovery*, 2000.
- [5] Goharian, N., El-Ghazawi, T., Grossman, D., and Chowdhury, A. On the enhancements of a sparse matrix information retrieval approach. *Proceedings of the International Conference on Parallel and distributed Processing Techniques and Applications*, 1999.
- [6] Gropp, W., Lusk, E., and Skjellum, A. *Using MPI: portable parallel programming with the message-passing*. The MIT Press, Cambridge, MA, 1999.
- [7] Joachimes, T. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In

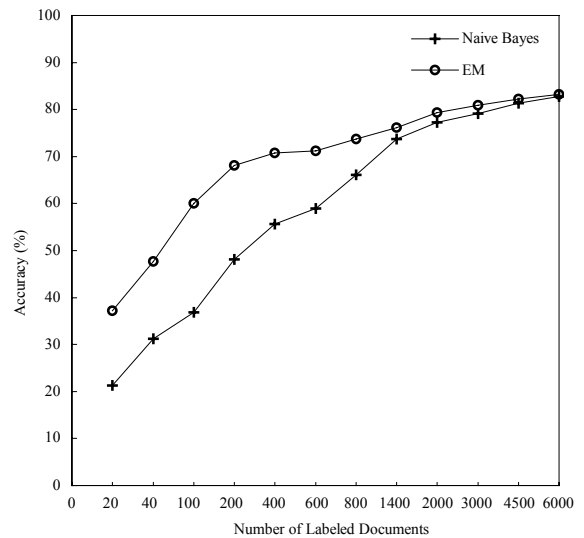


Figure 7. Classification accuracy vs. the number of labeled training documents.

proceedings of the Fourteenth International Conference on Machine Learning, pages 143-151. 1997.

- [8] Joshi, M.V., Karypis, G., and Kumar, V. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proceedings of International Parallel Processing Symposium*, 1998.
- [9] Lewis, D., and Ringuette, M. A comparison of two learning algorithms for text categorization. In *Third Annual Symposium on Document Analysis and Information Retrieval*, pages 81-93, 1994.
- [10] McCallum, A., and Nigam, K. A comparison of events models for naive Bayes text classification. *Papers from the AAAI Workshop*, pages 41-48, 1998.
- [11] McLachlan, G.J., and Krishnan, T. *The EM algorithm and extensions*. John Wiley & Sons, 1997.
- [12] Mitchell, T. *Machine learning*. McGraw-Hill, New York, 1997.
- [13] Nigam, K., McCallum, A., Thrun, S., and Mitchell, T. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, pages 103-134, 2000.
- [14] Ridge, D., Becker D., and Merkey, P. 1997. Beowulf: Harnessing the power of parallelism in a Pile-of-PCs. *Proceedings, IEEE Aerospace*.
- [15] Ruocco, A.S., and Frieder, O. Clustering and classification of large document bases in a parallel environment. *JASIS* 48(10), pages 932-943, 1997.
- [16] Thakur, R., Gropp, W., and Lusk, E. Optimizing noncontiguous accesses in MPI/IO. *Parallel Computing*, pages 83-105, 2002.