High-Performance Free-Text Information Retrieval on a Beowulf PC Cluster

Arnon Rungsawang, Athichat Tangpong, Pawat Laohawee
{fenganr,g4165239,g4165221}@ku.ac.th

KU Text REtrieval Group (KU-TREG)
Department of Computer Engineering
Faculty of Engineering
Kasetsart University, Bangkok, Thailand

Abstract. We present a study concerning the applicability of a distributed computing technique to a million-page free-text document retrieval problem. We propose a high-performance DSIR retrieval algorithm on a Beowulf PC Pentium cluster using PVM message-passing library. DSIR is a vector space based retrieval model in which semantic similarity between documents and queries is characterized by semantic vectors derived from the document collection. Retrieval of relevant answers is then interpreted in terms of computing the geometric proximity between a large number of document vectors and query vectors in a semantic vector space. We test this DSIR parallel algorithm and present the experimental results using a large-scale TREC-7 collection and investigate both computing performance and problem size scalability issue.

1 Introduction

Among the grand challenge applications in high performance computing, free-text retrieval is of particular interest in this information explosion epoch. While the number of new web pages on the Internet, as well as free-text documents in scientific, academic, business, and industrial usages, increase without limit day by day, seeking the relevant information in these million-page documents needs rapid and powerful searching artifact to get through. We believe that recent advances in distributed processing techniques on cluster of workstations or PCs [?,?] is currently mature enough and can provide powerful computing means convenient for overcoming this grand challenge application.

In this paper we focus on using cluster computing technique to implement a parallel free-text retrieval algorithm called "DSIR" [?]. DSIR is a vector space based retrieval model in which semantic similarity between documents is characterized by semantic vectors derived from the document collection itself. Documents and queries whose vectors are closed in a semantic space are assumed to have similar or related contexts. Since DSIR is a quite complex text retrieval system designed for Information Retrieval research usage in our laboratory, indexing and searching time with a large-scale text collection is inevitably much longer than general search engine found in the Internet. To speed up the retrieval process, we thus propose a parallel DSIR retrieval algorithm using PVM message-passing library [?] on a cluster of low-cost Beowulf PC Pentium class machines running Linux operating system [?].

Our Beowulf PC cluster is composed of several Pentium-II 350MHz, 128MB of physical memory machines, interconnected by two-way 10 Mbps and 100 Mbps Ethernet routing networks. Both NFS and message-passing traffics can be configured to pass through these two-way routes in a convenient way depending on the application (i.e. heavy NFS or heavy message-passing traffics) which is running on. In our retrieval experiments, we choose TREC-7 documents [?] as our large test collection. TREC-7 collection consists of more than half-million and varying length free-text documents, 2 GB in total. We index the TREC-7 collection and represent each document with a 500-dimensional vector. During retrieval, that large number of document vectors are then compared with standard TREC-7 50 query vectors using our parallel DSIR algorithm. We also test the problem size scalability issue by duplicating the TREC-7 collection two times and increasing the workload from 50 to 300 queries to search with. We found that our algorithm still scale quite well with this size of collection.

We organize this paper in the following way. Section 2 briefly presents the DSIR retrieval model. Section 3 discusses the proposed parallel algorithm. Section 4 gives more detail about our experimental setup, results, and discussion. Finally, section 5 concludes this paper.

2 DSIR Retrieval Model

In DSIR retrieval model, the distributional semantic information inherently stored in the document collection is utilized to represent document contents in order to achieve better retrieval effectiveness [?,?]. The distributional information used in our current system implementation is characterized by co-occurrence vectors of words selected as index terms for a particular document collection. Both documents and queries input to DSIR are then defined as the weighted vector sum of the co-occurrence vectors corresponding to index terms occurring in those documents and queries.

As documents and queries are represented as vectors in the same vector space, the basic retrieval operation is then very simple; the query vector is compared to every document vector, documents whose vectors locate in the vicinity of that query vector are presented to the user in decreasing order of their closeness as relevant answer. A typical vector similarity measure that we use is either the cosine similarity or dot-product function [?].

Fig. 1 depicts a simple DSIR document indexing and retrieval diagram. First, documents are pre-processed (i.e. eliminating stopwords and performing word stemming) and then put into co-occurrence computational and document statistical extraction routines. Both co-occurrence information and document statistics are then used to compute all document vectors. Here we suppose that query vectors are derived by the same method as documents. Each query vector is

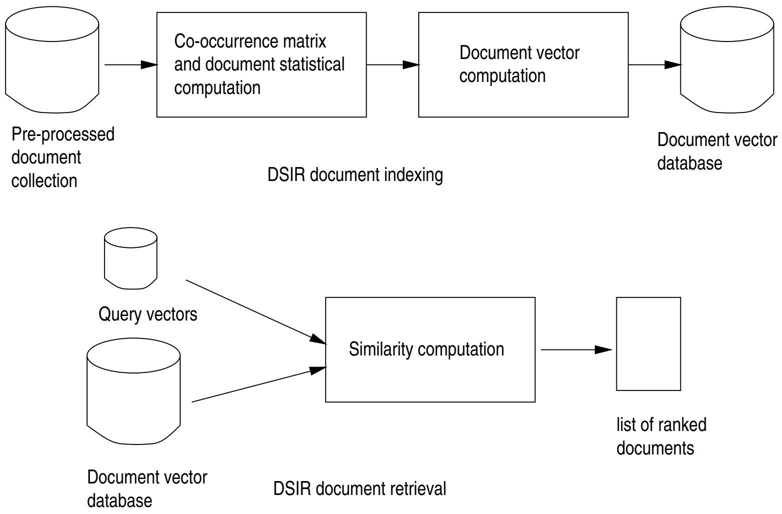

Fig. 1. DSIR Retrieval Model.

searched against document vectors for its related documents in document vector database using similarity computational routine. Output of this routine will be a list of ranked documents in decreasing order of similarity with respect to the input query.

3 Parallel DSIR Retrieval Algorithm

We follow master-slave programming model to develop our parallel retrieval algorithm [**?**]. The algorithm consists of four main steps; partitioning the document vector database to distribute to other computing nodes, sending query vectors to computing nodes, computing vector similarity, and rescoring all similarity scores (see Fig. 2).

We can explain this parallel DSIR retrieval algorithm in more detail as follows; First, a stripping algorithm is responsible for dividing a large document vector database into several chunks. The size of each chunk is depending on how computing nodes can allocate and perform similarity computation without performing local disk paging. Second, the first chunks of document vectors are sent to corresponding computing nodes while in this case master behaves like a file-server. Third, to optimize the massage-passing traffic, master sends all query vectors in advance to all computing nodes. Fourth, each computing node computes similarity scores between document vectors and all query vectors, and sends similarity scores back to master for final rescoring. Fifth, if there are still any chunks of document vectors left, master distributes next chunks of document vectors to computing nodes, and loops back to step four above. Sixth, master

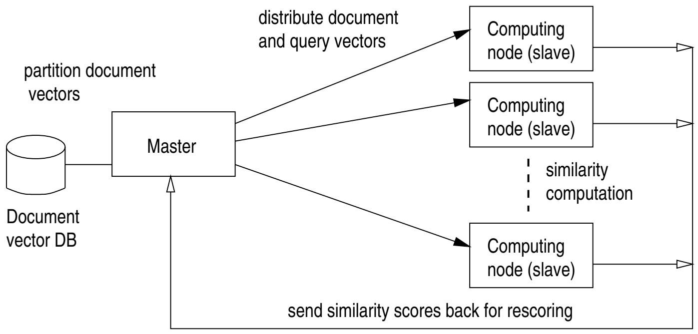

Fig. 2. Parallel DSIR retrieval diagram.

recomputes the final ranking scores and outputs the list of ranked documents as relevant answer.

4 Experimental Results

In this section, we present the results concluding from two sets of experiments. For the first experimental setup, we study the computing performance between "via-NFS retrieval" and "distributed database retrieval" (see Fig. 3). For document retrieval via-NFS, we share the single large document vector database to other computing nodes via 100 Mbps Ethernet switch. For distributed database retrieval, we split the same database into several portions and store those portions to the local disk of the computing nodes[1]. For the second experimental setup, we examine the scalability issue of our implementation by doubling the number of documents in the collection and variating the number of input queries.

4.1 Via-NFS vs. Distributed Database Retrieval

We prepare 528,155 TREC-7 documents to be our test database [?]. Each document is indexed and represented with 503 dimensional vector so that the total document vector database yields around 1 GB. The original 50 TREC-7 queries are indexed and represented by 503 dimensional vectors as well. Retrieval algorithm now can then be interpreted as computing the similarity scores (e.g. vector scalar product) between each query vector and every document vector in 1 GB database, and ranking that 528,155 similarity scores. Several curves from Fig. 4 concludes the results from these experiments.

As we have expected, experimental results show that via-NFS retrieval setup is less effective than distributed database retrieval one. Via-NFS retrieval setup

[1] Evidently we can see that via-NFS retrieval configuration is less expensive to set up than distributed database retrieval one as it is not necessary to have a harddisk installed at every computing node, and all computing nodes can be set up as diskless nodes.

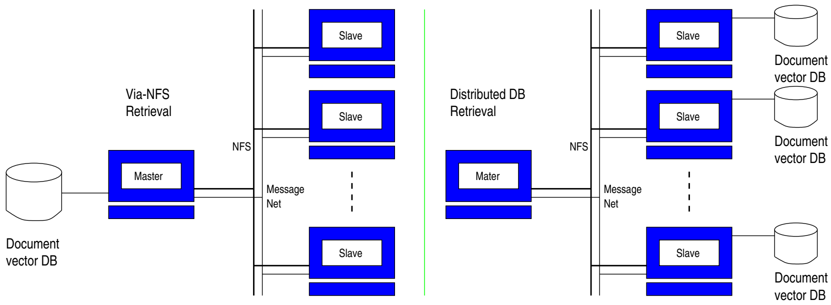

Fig. 3. Via-NFS vs. Distributed Database Retrieval.

gives longer average query response time, lower CPU utilization, and less efficiency. When the number of computing nodes increase from one to two, average CPU utilization decreases sharply from 85.15% to 19.61%, the average query response time increases from 19.98 to 30.74 seconds, and speedup factor decreases almost 50%. We think that the main reason of this performance degradation comes from the fact that during document retrieval each computing node spends almost of its time to wait for document vectors from master via NFS, though that NFS has been routed via an 100 Mbps Ethenet switch (i.e. NFS bottle-neck).

For the distributed database retrieval, each computing node can load up the corresponding document vectors directly from its local disk. When the number of computing nodes increase from one to two, CPU utilization drops from 79% to 62%, and average query response time slowly decreases. But speedup factor increases when the number of computing nodes increase till 5 nodes, and slightly decreases beyond that point.

After examining closely what was the problem with the computing performance when adding more than 5 computing nodes on distributed database retrieval setup, we discover two problems. The first one comes from the fact that we inadvertently let every computing node writes back its similarity scores to master for final rescoring via 10 Mbps NFS route. Thus when more computing nodes have been added, more intense NFS bottle-neck occurs. The second one come from the problem of load unbalancing in our document vector database splitting algorithm. Some computing nodes that have less documents to perform similarity computation will finish their computing faster than the others, and then stay idle. We then correct these two problems by rerouting final similarity scores via 100 Mbps line, fixing the splitting algorithm, and rerunning our tests. Fig. 5 which depicts average query response time and speedup curves supports this finding. Note that we obtain a nearly perfect speedup curve.

4.2 Scalability Issue

To study the scalability issue of the proposed algorithm, we set up the experiments in the following way. We simply doubling the old TREC-7 collection two

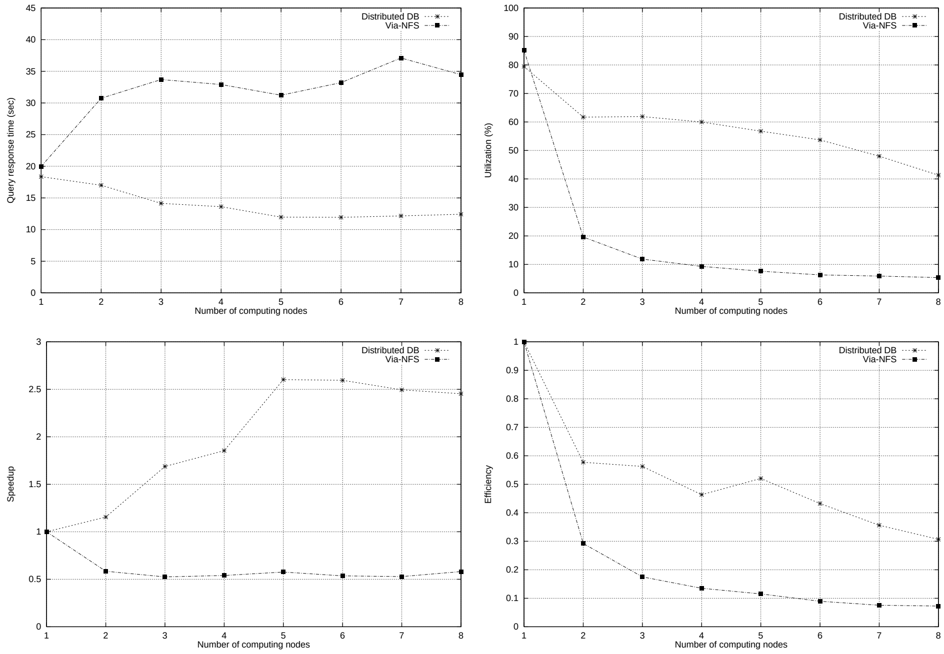

Fig. 4. Via-NFS vs. Distributed database retrieval.

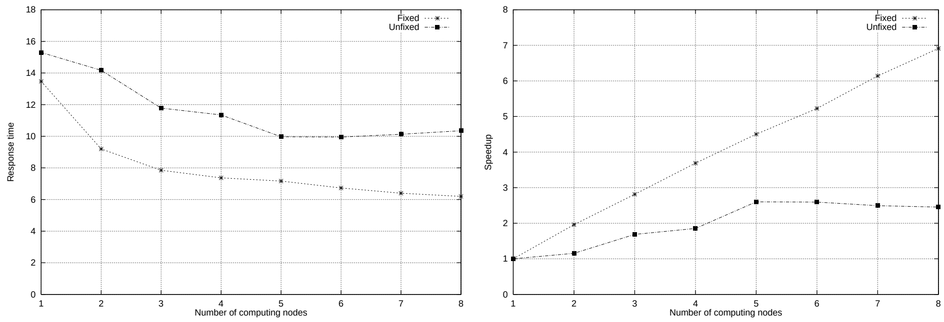

Fig. 5. Distributed database retrieval after fixing the problem of load unbalancing.

times. This new database finally yields more than one million documents, and is around 2 GB of document vectors. We also build the second set of queries by duplicating the old query set 6 times, and rerun more experiments using distributed database retrieval setup. Now, the old 50-query set and the new 300-query set are searched against a million-page document vectors. Results from these experiments have been depicted in Fig. 6.

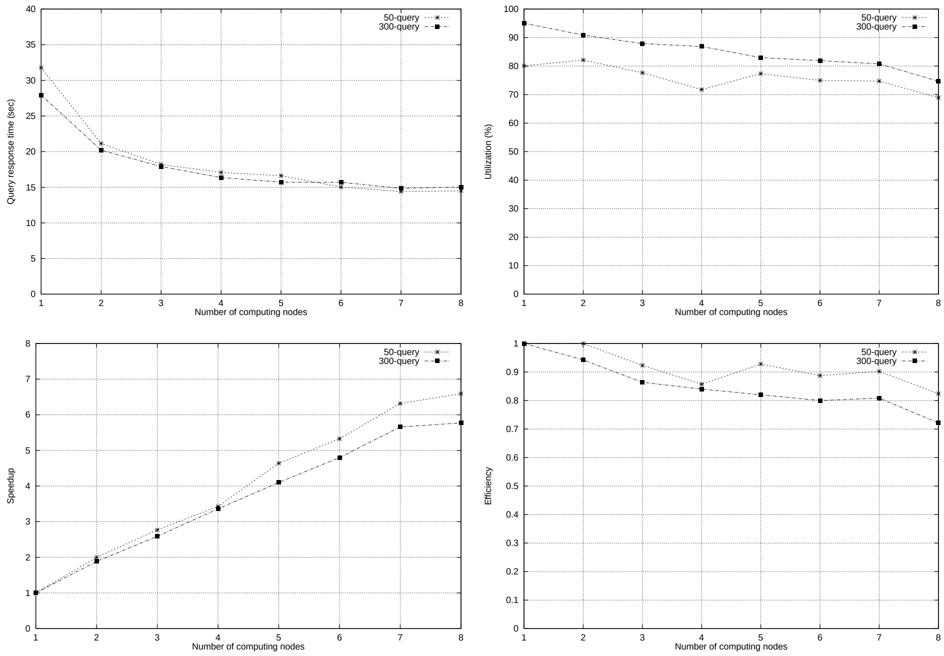

Fig. 6. Problem size scalability when doubling the old document database and increasing the number of input queries.

The results show that our proposed algorithm still scales well though the problem size has been increased two times (i.e. doubling the document vector database), and the work load has been increased six times (i.e. increasing the number of queries to 300). Average query response times similarly decrease to 15 seconds when the number of computing nodes have been added to 8. CPU utilization of 300-query set is higher than another, due to the fact that the timing expense of very long similarity computation which happens in parallel at every computing node compensates much more with the timing expense of rescoring the similarity scores which is the only sequential part in our current implementation. Moreover, speedup factor also linearly increases when the computing nodes are increased.

5 Conclusion

In this paper we have proposed a parallel DSIR text retrieval algorithm, and presented our experimental results using a large TREC-7 document collection. We have learned from this study that applying the distributed computing technique on a low-cost Beowulf class PC cluster is an efficient way to solve a grand challenge problem like large-scale free-text retrieval application.

We have also discovered two main problems that degrade the computing performance when using message-passing technique on this kind of PC cluster. The first one comes from the classical problem of NFS bottle-neck. Since a free-text retrieval is an I/O intensive application in general, partitioning a large retrieval problem into smaller chunks and distributing them to other computing nodes via NFS, though it is less expensive to set up, is very inefficient to do. The second problem comes from the problem of load unbalancing. When a large problem size has not been well divided, some computing nodes will be sometimes left idle if they have less workload to compute.

From our experiments, we also found that our proposed algorithm scales well though we increase our problem size to a million-page documents and double the input query workload from one to six times (i.e. from 50 to 300 queries). However, as we know well, just million-page documents are not a real problem size for an acceptable search engine in the Internet anymore, we thus anticipate to ameliorating our parallel DSIR retrieval algorithm and testing it with several million web-page documents in the near future.