Parallel Implementation of Information Retrieval With Relevance Feedback

Ali M. AlHaj　Eiichiro Sumita　Hitoshi Iida
ATR Interpreting Telecommunications Research Laboratories

1 Introduction

Relevance feedback is a well-known method developed to improve the effectiveness of information retrieval systems. It is based on the automatic and iterative refinement of the textual queries supplied by the users. This method is computationally intensive, and thus it is rarely implemented on the mostly sequential existing information retrieval systems. In this paper, we describe a parallel information retrieval system which efficiently implements relevance feedback. First, a brief description of relevance feedback and its application within the vector processing retrieval algorithm is given in section 2. Next, a parallel implementation of the system is described in section 3, and evaluated in section 4. Finally, concluding remarks are given in section 5.

2 Relevance Feedback

Retrieving textual information using information retrieval systems is inherently more difficult task than retrieving formatted data using database management systems. This is mainly due to the complexity involved in the textual query formulation process. The complexity arises from the users lack of detailed knowledge of the textual database and the retrieval environment. To deal with this problem, the retrieval operation is usually broken down into a sequence of retrieval steps, where the tentatively-formulated users queries are used in the first step, and more efficient system-formulated queries are used in the latter steps.

A well-known method which has proved to be effective in producing efficient system-formulated queries is the relevance feedback [1]. This method is based on the automatic reformulation and improvement of the user's original query by making use of given information about the usefulness of documents previously retrieved in earlier retrieval steps. Relevance feedback utilizes the relevance information in the new query formulation in such a way to enhance the importance of the relevant documents and reduce the importance of the non-relevant documents. The effect of such a query formulation process is to move the original query in the direction of the relevant documents and away from the non-relevant documents.

The relevance feedback method was originally developed to be used within the vector processing document retrieval algorithm. In this algorithm both documents and queries are modeled as vectors of weighted terms, where term weights are real-numbers between 0 and 1. The retrieval operation consists of scoring documents vectors as to how well they match a given query vector, then returning the top ranked documents to the user. Relevance feedback formulates a new query vector by simply utilizing the retrieved documents vectors to expand and re-weight the original query vector.

3 The Implementation

Our implementation consists of four major steps; (1) generating a term weighted vector for each document and query in the text database, (2) performing a parallel score and rank operation to decide on how well each document vector matches a given query vector, (3) retrieving the documents which have the highest similarity scores and judging their relevance to the query vector, and finally (4) formulating a new query and repeating the score and rank operation until all relevant documents are retrieved. These steps are shown in the flowchart of Figure 1, and explained briefly below.

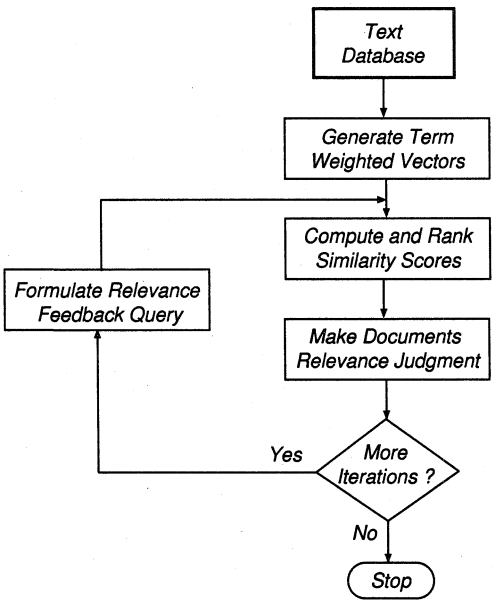

Figure 1: Flowchart of the implementation.

- Generate term weighted vectors: the first step in a vector processing text retrieval system implementation is to represent each document and each query contained in the full-text database by a term weighted vector. The weighted vector generation process is carried out on the host machine by running the following three text pre-processing operations [2]:

  i. *Filtering:* use a stop list of common function terms $(and, of, or, but, the, etc, ...)$ to eliminate from the text of documents and queries such high frequency terms which are insufficiently specific for their representations.

  ii. *Stemming:* use suffix stripping routine to reduce the remaining terms to term stem form.

  iii. *Weight assignment:* for each remaining term stem i occuring in document j, compute a term weighting factor, which is the product of the term frequency of term i in document j multiplied by the inverse document frequency of term j in the documents database as a whole. The term weights are restricted to the range from 0 to 1, where 0 represents a term that is absent from the vector, and 1 represents a fully weighted term.

- Compute documents similarity scores and rank them in decreasing order. This corresponds to (1) loading the documents and query term weighted vectors into the main memory of the parallel machine, (2) distributing the processing of the query vector matching operation on all processing elements of the parallel machine by assigning different documents vectors to different processing elements, (3) performing in parallel the query vector matching operation in such a way that each processing element executes an inner product operation between the query vector and each assigned document vector, and finally (4) ranking the documents in decreasing order of their similarity scores.

- Judge the relevance of the top n scored ranked documents where n is an arbitrary number predetermined by the user. The relevance judgment is made automatically be referring to the relevance information file which contains names of the most relevant documents to the query. If all (or sufficient) relevant documents have been retrieved, exit the retrieval operation and refer to the relevant retrieved documents full text in the disk file of the host machine. Otherwise, move on to the next step to retrieve more (or the remaining) relevant documents by applying the relevance feedback method.

- Formulate a new query vector by expanding and re-weighting the original query. This is done by adding all terms and corresponding weights from all relevant document vectors, and subtracting the weights of all terms found to be existing in the top ranked non-relevant document vector. This query reformulation process is called the Ide dec-hi method and it has proved to be superior to many other relevance feedback methods. After formulating the new query, repeat the above two steps until all (or most) relevant documents are retrieved.

4 Performance Evaluation

We carried out the relevance feedback implementation on the KSR parallel computing system [3]. The KSR system is an MIMD type parallel computer which combines the shared-memory architecture of traditional supercomputers and mainframe systems with the scalability of highly parallel systems. Unlike the typical memory architecture which has large pools of main memory and small caches, all KSR main memory consists of large, communicating local caches each of which physically adjacent to a processor. Communication between the caches is implemented using a slotted, pipelined, rotating ring. A parallel application can be easily implemented on the KSR system by breaking it down into several pieces of work, and assigning each one to a pthread. A pthread is a sequential flow of control within a process that cooperates with other pthreads to solve the application problem. Pthread parallel programming is done using an extended version of the standard C language.

The implementation was evaluated using LISA; an experimental library science documents collection which consists of 6004 documents and 35 queries. First, the host computer generated the documents weighted vectors and the weighted vector of the query which has the largest number of relevant documents. Next, the vectors were read and processed in parallel by a KSR model consisting of 25 processors connected to a distributed shared memory of 800 MB (0.8 GB). Finally, retrieval performance was measured by the recall (R) and precision (P) measures, where recall is defined as the proportion of relevant documents that are retrieved from the collection, and precision is the proportion of retrieved documents that are relevant. The (R) and (P) were measured under the assumption that the top 20 documents retrieved in the initial search are judged for relevance.

Iteration	*Query Size*	*Retrieval Performance*		*Time Performance (msec.)*		
		Recall	*Precision*	*Time(1)*	*Time(25)*	*Speedup*
0	18	0.264	0.700	005.04	00.23	22.37
1	477	0.358	0.475	106.93	04.78	22.33
2	609	0.509	0.450	136.11	06.06	22.42
3	823	0.584	0.387	183.65	08.23	22.31
4	969	0.716	0.380	215.86	09.64	22.38
5	1228	0.735	0.325	273.65	12.23	22.36
6	1268	0.811	0.307	282.49	12.59	22.43
7	1396	0.849	0.281	311.51	13.93	22.35

Table 1: Retrieval and speed performance of the relevance feedback parallel implementation.

Retrieval and time performance data are given in Table 1, where iteration 0 corresponds to the processing of the initial query, and the successive iterations correspond to the processing of the queries formulated by relevance feedback. The number of feedback iterations was determined dynamically by allowing a new feedback iteration only when the previous iteration produced any relevant documents. That is, the relevance feedback was suspended automatically when the top retrieved documents contained no relevant documents. The recall figures in the table prove that the retrieval quality becomes better and better as more feedback iterations are executed. Similarly, the query size figures give a clear evidence of the increased computational demands posed by relevance feedback operation and thus its potential for parallel implementation. Finally, we carried out a sequential implementation of the same retrieval experiment on a single processor, and compared the speed performance with that of the parallel implementation. As given in the table, the parallel implementation was more than 22 times faster than the sequential implementation.

5 Conclusion

The relevance feedback query reformulation method improves information retrieval performance significantly. However, due to its iterative and computation-intensive nature, it is hardly implemented on most commercial information retrieval systems. Therefore, sufficiently powerful systems are needed to implement this method to benefit from its effective performance. In this paper we presented an efficient implementation of relevance feedback on a parallel shared-memory computer system. Based on the performance results we obtained we can easily conclude that parallel machines are indeed well-suited to meet the computational demands posed by the relevance feedback method.

References

[1] Salton, G. and Buckley, C., "Improving Retrieval performance by Relevance Feedback," in *Journal of the American Society for Information Science,* 24, pp. 288-297, 1990.

[2] Frakes, W. and Baeza-Yates, R., *Information Retrieval Data Structures & Algoirthms,* Prentice Hall, New Jersey, 1992.

[3] Kendall Square Research Corporation., Technical Manuals, MA, USA, 1994.