Query Optimization for Vector Space Problems

K. Goda*, T. Tamura**, M. Kitsuregawa*, A. Chowdhury*** and 0. Frieder***
* Institute of Industrial Science, The University of Tokyo
** Mitsubishi Electric Corporation
*** The Illinois Institute of Technology

ABSTRACT

We present performance measurement results for a parallel SQL based information retrieval system implemented on a PC cluster system. We used the Web-TREC dataset under a left-deep query execution plan. We achieved satisfactory speed up.

1. INTRODUCTION

Digital text database are commonplace and searching these database is a daily activity. The continued expansion of these digital repositories necessitates supporting efficient retrieval to maintain acceptable response times. One approach to meeting response time constraints is the use of parallelism. We describe our approach to parallel information retrieval.

Traditional information retrieval systems rely on inverted index implementations to store and retrieve data. While such data structures are efficient, their development and maintenance is difficult. Furthermore, frequent updating of the data introduces processing complexity. Relational database systems, however, are specifically developed to support frequent updates. Until relatively recently[1] [2] [3], however, the database and information retrieval community has resisted the use of relational technology to support information retrieval functionality. This resistance stemmed from the vast overhead in storage and the lengthy response times attributed to relational implementations due to the replication of the index for every entry in the posting list.

Recently, vendors like Oracle are adding index-organized tables where values are stored along with the key in the B+Tree index. These additional storage structures now support the implementation of the vector space model in a manner similar to traditional information retrieval implementations. Unlike traditional inverted index implementations, relational implementations benefit from the standard advantages of database management systems including the support for frequent updates and efficient parallel implementations. We demonstrate the use of our parallel database approach[5] to provide scalable information retrieval.

SIGIR'01, September 9-12, New Orleans, Louisiana, USA..


```
SELECT q_qryid, d_trecdocnum,
       SUM(((1+LOG(i_tf))/((d_logavgtf)*(avgdoclen+(0.20*d_d_doclen))))
FROM  Index i, Document d, Query q, Term t
WHERE d_docid=i_docid  AND  q_term=i_term  AND  t_term=q_term
GROUP BY q_qryid, d_trecdocnum  ORDER BY 1, 2 DESC;
```

2. VECTOR SPACE CALCULATION USING RELATIONS

DBMS packages do not provide full text search capabilities. To address this shortfall, we parsed, stemmed, filtered and loaded the TREC-9(WT10GB) data into the schema shown in Figure 1 (initially presented in [3]). Additionally, the code segment above was used as our vector space ranking model. The SQL implements the pivoted document length normalization calculation[4] shown to be an effective text ranking method.

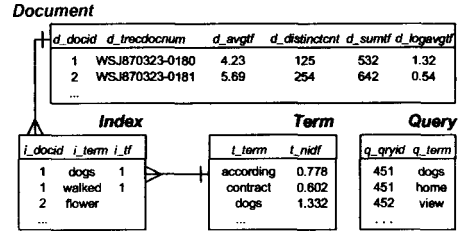

Figure 1: A Relational Design for an Information Retrieval Application

3. PERFORMANCE EVALUATION WITH DIFFERENT QUERY PLANS ON DBKERNEL

We evaluated the query on our home-grown parallel relational database engine, DBKernel[5], to demonstrate the influence of query optimization methods on the performance and the scalability. The DBKernel gives users full control over various query execution strategies including data declustering(placement) methods and query plans. It was designed to run on commodity-based PC clusters. Currently, the third generation system, which consists of 32 PCs and 32 disks connected through SAN(storage area network), is operating. Each PC node runs the Solaris 8 operating system on an 800MHz Pentium III Processor with 128 MBytes of RAM and interface cards for the 100Base-TX Ethernet

and the Fibre Channel. Figure 2 depicts the hardware components of the system.

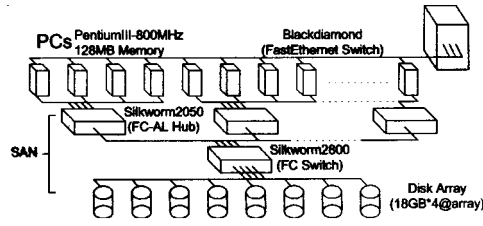

Figure 2: An Overview of the PC Cluster System

We examined the performance of the Vector Space Flanking query using one execution plan(LDh) in the hash declustering and two plans(LDr and LDBr) in the round-robin manner. LDr and LDBr are basically the same but differ in the way the loaded records are distributed, which are described later. These plans all join tables one by one, fully generating intermediate results before proceeding to the next join step. Namely, the order of joins can be denoted as: (((Query ⋈ Term) ⋈ Index) ⋈ Document). Such plans are called *Left Deep* plans in the database community. The rank calculation, aggregation and sort phases follow the joins. For the aggregation, we carried out a two-phase algorithm, where local sums are calculated at each node before being sent to a master node to produce the global results.

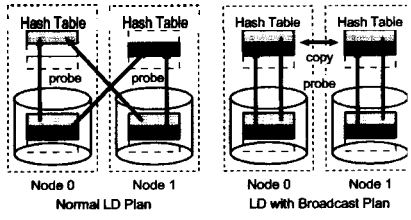

Figure 3: Hash Table Probe Operation in Each Query Plan

Each join operation is based on the hash join algorithm, where each record in one of the two tables is loaded to form a hash table, which is then probed by the records in the other table to produce the results. Usually, the hash tables are fragmented and declustered among nodes just as the tables (relations). In this process, LDh, where there is little data exchange, is theoretically the most efficient, but it is difficult to avoid the data placement skew. Skew can significantly degrade the performance. On the other hand, for the Index table, most of its records must be exchanged among the nodes to probe the appropriate hash table fragments in LDr and LDBr. To eliminate the communication overhead involved in this exchange, an entire hash table built from the results of the Query-Term join can be replicated rather than declustered at the cost of an increased memory requirement. In this case, all probe operations for Index records can be done locally at each node. The differences of these strategies are depicted in Figure 3, where the normal LD plan(LDr) represents the former approach, and the LD with broadcast(LDBr) plan the latter.

Figure 4 shows the elapsed times and speedup ratio of each of the query plans with the number of nodes varying from 4 to 32. With more nodes, the sizes of the table fragments at each node are decreased, resulting in smaller elapsed times. However, LDr suffers from the higher communication overhead of the global data exchange, diminishing the effect of reduced disk I/O times. In contrast, LDBr exhibits a much better speedup due to its optimization in the hash table placement, and its performance is almost equal to LDh.

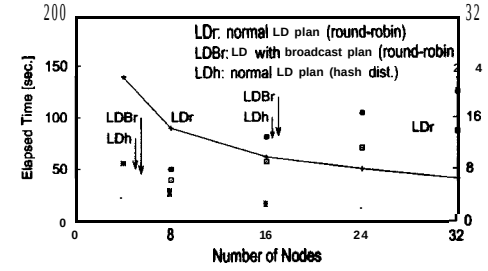

Figure 4: Evaluation Results with Different Query Plans

Even the LDBr and LDh plans are inferior to the ideal speedup. This is due to the overhead of gathering the final results at the master node and transferring them to the client. However, in the actual interactive environment, where response times are significant, we can return only a fraction of the whole query results dimishing the observed transfer time.

4. CONCLUSIONS

We examined SQL based parallel information retrieval over a 32-node PC cluster system. We achieved satisfactory performance in the round-robin declustering manner which was almost equal to the one in the hash distribution. For systems larger than 32 nodes, skew is inevitable, so the LDBr approach potentially is better than the LDr solution. These initial findings motivate us to experiment with more complex queries under both larger datasets and larger machine configurations comprising of greater than 64 nodes.

5. REFERENCES

[1] 0. Frieder, A. Chowdhury, D. Grossman, and M. C. McCabe. On the integration of structured data and text: A review of the SIRE architecture. In *DELOS Workshop* on Information *Seeking, Searching, and Querying* in *Digital Libraries,* December 2000.

[2] T. Grabs, K. Boehm, and H.-J. Schek. A parallel document engine built on top of a cluster of databases - design, implementation, and experiences. In *IEEE Data Engineering Conference,* April 2001.

[3] D. A. Grossman, 0. Frieder, D. 0. Holmes, and D. C. Roberts. Integrating Structured Data and Text: A Relational Approach. *Journal of the American* Society *of Information* Science, 48(2), February 1997.

141 A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *Proc. of ACM SIGIR,* 19th, August 1996.

[5] T. Tamura, M. Oguchi, and M. Kitsuregawa. Parallel database processing on a 100 node PC cluster: Cases for decision support query processing and data mining. In SC97: *High* Performance *Networking* and *Computing,* 1997.