A Parallel Relational Database Management System Approach to Relevance Feedback in Information Retrieval

Carol Lundquist[1], Ophir Frieder[2], David Grossman[3], and David O. Holmes[4]

Abstract. A scalable, parallel, relational-database driven information retrieval engine is described. To support portability across a wide-range of execution environments, including parallel multicomputers, all algorithms strictly adhere to the SQL-92 standard. By incorporating relevance feedback algorithms, accuracy was significantly enhanced over prior database-driven information retrieval efforts. Algorithmic modifications to our earlier prototype resulted in significantly enhanced scalability. Currently our information retrieval engine sustains near-linear speedups using a 24-node parallel database machine. Experiments using the TREC data collections are presented to validate the described approaches.

1. Introduction

The continued growth, acceptance, and public reliance on digital libraries has fostered wide interest in information retrieval systems. Traditionally, customized approaches developed to provide information retrieval services. Recently, however, general solutions that support traditional information retrieval functionality and integration of structured data and text have appeared both in the research community, e.g., (DeFazio95, Grossman94, Grossman97) and in the commercial sector, e.g., Oracle's ConText (Oracle97).

Today's data processing engines must not only provide for the integration of both text and structured data, but do so seamlessly. Given the exponential growth of on-line electronic data, information retrieval engines must be scalable in terms of growth in data volume and retrieval accuracy. Furthermore, with the ever-increasing drive towards Commercial-Off-The-Shelf (COTS) software systems, and solutions that are portable across a wide range of execution environments, modern-day information retrieval engines must be developed using general purpose software components. To address all of the above concerns, we extended our relational database driven information retrieval engine, initially described in (Grossman94) and lately in (Grossman97), to incorporate relevance feedback. We modified our original design to not only support a higher degree of functionality but to also scale to, at least, tens of processors. Throughout our efforts, we continue to rely on strictly unchanged, standard SQL to support a wide range of execution environments. With these enhancements,

[1] George Mason University, Fairfax, Virginia, carol.lundquist@erols.com.

[2] Florida Institute of Technology, Melbourne, Florida, ophir@ee.fit.edu. Ophir Frieder is currently on leave from George Mason University. This work was supported in part by the National Science Foundation under the National Young Investigator Program under contract number IRI-9357785.

[3] Office of Research and Development, Washington, DC, dgrossm1@osf1.gmu.edu

as evidenced by recent experimental results obtained as part of the NIST Text REtrieval Conference (TREC-6) efforts, our current prototype sustains a high retrieval accuracy both in terms of precision and recall while supporting near linear speed-ups using 24 parallel nodes.

The remainder of our paper is organized as follows. We begin by overviewing recent efforts that focus on relevance feedback algorithm implementation, relational database information retrieval approaches, and parallel information retrieval systems. Having reviewed prior efforts, we describe our SQL-driven relevance feedback algorithm. We continue with an experimental evaluation in which we vary the feedback term selection algorithm, the number of top-ranked documents considered, and the number of feedback terms, the feedback scaling, and feedback thresholding approaches used. Throughout all of these experiments, the entire approach uses strictly unchanged SQL. We conclude with a thorough scalability study using the NCR Teradata DBC/1012 Database Computer configured with 24 nodes. As is shown, our original approach described in (Grossman97) was limited in terms of scalability, but with the modifications described in Section 4, a significant improvement in the scalability of the approach was achieved.

2. Prior Work

A. Relevance Feedback

Relevance feedback was first proposed in the late-1960's as a method to increase the number of relevant documents retrieved by a query (Rocchio66). In relevance feedback, a user query is used to search a document collection. The documents obtained from this search are examined and certain terms selected from the documents deemed to be relevant are used to expand the original query. Terms from documents deemed not relevant may also be used to modify the original query. Rocchio's original formula, shown in equation F1, is based on the vector-space model.

$$Q_1 = Q_0 + \beta \sum_{k=1}^{n_1} \frac{R_k}{n_1} - \gamma \sum_{k=1}^{n_2} \frac{S_k}{n_2} \qquad \text{(F1)}$$

[4] NCR Corporation, Rockville, Maryland, David.Holmes@WashingtonDC.NCR.COM

where

Q_1 = new query vector
Q_0 = initial query vector
R_k = vector for relevant document k
S_k = vector for non-relevant document k
n_1 = number of relevant documents
n_2 = number of non-relevant documents
β and γ = weight multipliers to control relative contributions of relevant and non-relevant documents

Much of the prior research on relevance feedback has focused on the impact from the relevant and non-relevant document weight multipliers, β and γ where $(0 \; 2 \; \beta \;, \gamma \; 2 \; 1, \beta + \gamma = 1)$ (Ide71, Rocchio71). In 1990, Salton and Buckley experimented with setting the multiplier for the non-relevant documents to zero and determined that results from this form of relevance feedback were still an improvement over results obtained without relevance feedback (Salton90). In 1992, Harman experimented with term reweighting by query expansion, feedback term selection techniques or sort orders, and the effectiveness of performing multiple iterations of relevance feedback (Harman92). Harman determined that using only selected terms for relevance feedback produced better results than using all of the terms from the relevant documents. She also determined that using a feedback term sort order that incorporated information about the frequency of the term in the document as well as the term's overall collection frequency produced improved results. However, the experiments described above were conducted using small test collections of documents (i.e., collections with less than 15,000 documents) so the effectiveness of these techniques on large document collections was unknown. Results obtained from a small collection do not necessarily hold true for larger collections (Blair85).

The work described in Section 3 expands upon the prior work by using the Tipster document collection (i.e., a large standard collection of more than 500,000 documents) and systematically conducting an experimental study of the effects of the following parameters on relevance feedback:

- number of top-ranked relevant documents used
- number of feedback terms selected
- various types of feedback term selection techniques or sort orders
- feedback term weight multipliers

B. Relational Systems with Information Retrieval

While most information retrieval systems have been implemented using custom software, there have been a few proposals to use a relational database (RDBMS) as the basis for an information retrieval system. Much of the early work focused on designing special extensions to the relational model to incorporate the unique features of text-based data. Examples of this approach can be found in (Blair75, Crawford78, Macleod78, Crawford81, Macleod81a, Macleod81b, Macleod83, Stonebraker83, and DeFazio95).

Several difficulties occur when custom extensions are added to a system, particularly in areas of system integrity, optimization, and portability. For example, custom extensions to an RDBMS may circumvent controls inherent in the RDBMS to ensure data consistency and security, and thus, compromise the integrity of data in the system. In addition, custom extensions implemented and optimized on one system may perform quite differently on another system, and thus, impact system performance. In work done by Grossman, et al, in 1995, the implementation of an information retrieval system using unchanged SQL was first demonstrated (Grossman97). As part of their information retrieval system, they also demonstrated the implementation of basic term weighting and relevance ranking functionality using unchanged SQL. An information retrieval system that is treated as an application ofthe relational model and using standard SQL has the advantages of being portable across RDBMS platforms as well as being able to use the maintenance, security, and data integrity features which are an integral part of the RDBMS system.

C. Parallel Information Retrieval

Recently, several participants in the Text REtrieval Conferences (TREC) have developed parallel information retrieval systems (Hawking96, Mamalis96) to combat the data explosion versus response time concerns. A major drawback to these parallel systems is that they require special purpose hardware and software to implement the information retrieval system which greatly diminishes the portability of these information retrieval systems across platforms. This drawback, however, is avoided when an information retrieval system is designed on the relational model because an information retrieval system designed on the relational model retains the benefits of portability across RDBMS and is able to run virtually unchanged in both the single processor and parallel environments. Information retrieval systems based on the relational model can be easily and transparently ported to a parallel environment by shifting to a parallel RDBMS.

3. Implementing Relevance Feedback in the Relational Model

A. Relevance Feedback Implementation

Using an RDBMS to support information retrieval offers several advantages such as the simplicity of the relational model, user independence from the details of data organization and storage, and portability across different systems and platforms. All major RDBMS vendors (IBM, Informix, Microsoft, NCR, Oracle, and Sybase) offer a parallel version of their database software that allows relational applications to take advantage of a parallel processing environment in a way which is transparent to the application. Since parallel RDBMS also use standard SQL, an information retrieval system designed using unchanged SQL can easily run on a parallel platform as well. In addition, nearly all major, commercial RDBMS products provide extensive mechanisms to handle database maintenance and ensure data integrity through utilities for data recovery, concurrency control, transaction processing, security, etc. Another major advantage to using the relational model for information retrieval is that new features and processes are easily added through minor modifications to the SQL and do not require the recoding, debugging, and testing of large sections of code.

The automatic relevance feedback method developed for the relational model divides the feedback process into five steps. While it is possible to do the entire process with a single query, it would be somewhat cumbersome to implement the SQL to select only the top *n* documents or top *x* terms. For the sake of clarity, the process is separated into five steps. Descriptions of the relational structures used for this system can be found in section 4.

Step 1 - Identify the top *n* documents for each query through relevance ranking.

The first step performs relevance ranking using the *nidf* term weighting method described in (Ballerini96, Singhal96). Using this method, the similarity coefficient between a document and a query is calculated by the following SQL statement. For our experiments, *s* is set to .20 and represents a scaling factor applied to the number of distinct terms in the document being ranked and *p* is set to 282.7 which represents the average number of unique terms across all documents in the collection. To simplify processing, the results from each of these steps are loaded into temporary tables.

```
INSERT INTO temp_table1 VALUES
SELECT a.qryid, d.docid, SUM(((1+LOG(c.termcnt))/
   ((d.logavgtf) * (((1 - s) * p) + (s * d.disterm)))) *
     (a.nidf * ((1 + LOG(a.termcnt))/(b.logavgtf))))
 FROM query a, query_weight b, doc_term c, doc_weight d
WHERE a.qryid = b.qryid AND c.docid = d.docid
    AND a.term = c.term AND a.qryid = query_number
GROUP BY a.qryid, d.docname;
```

Step 2 - Identify the terms from the top *n* documents.

The second step of this process involves the identification of all terms associated with the top documents. The SQL to execute this step is as follows:

```
INSERT INTO temp_table2
SELECT a.qryid, a.docid, b.term, c.nidf
FROM temp_table1 a, doc_term b, nidf c
WHERE a.docid = b.docid AND b.term = c.term;
```

Step 3 - Select the feedback terms.

The third step involves the selection of the feedback terms. Several term selection techniques were tried and one of the more successful sort techniques (*n * term weight*) is similar to a technique suggested in (Harman92) where *n* is equal to the number of top-ranked documents containing the term and *term weight* reflects the collection weight of the term. Other term selection techniques are described in Section 3.B.1. The SQL necessary to implement the (*n * term weight*) selection processes, is shown below. The flexibility of the relational model provides a powerful yet convenient mechanism to easily implement a wide variety of term selection techniques.

```
INSERT INTO temp_table3
SELECT qryid, term, nidf, (nidf * COUNT(*))
FROM temp_table2
WHERE qryid = query_number
GROUP BY qryid, term, nidf
ORDER BY 4 DESC;
```

The SQL for selecting feedback terms using other selection techniques is similar to the SQL for the (*n * term weight*) term selection technique with only minor changes to the grouping clause or the addition of a HAVING clause to specify that a designated number of documents must meet the conditions.

Step 4 - Merge the feedback terms with the original query.

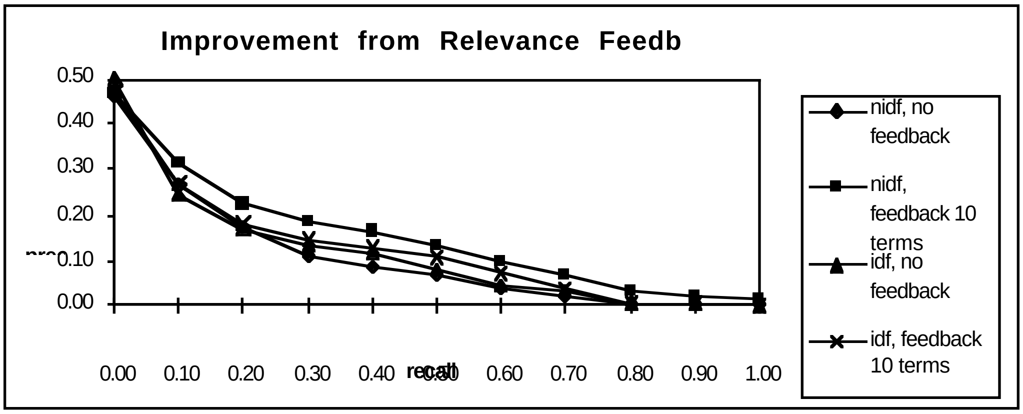
The fourth step in the relevance feedback process merges the terms from the original query with the newly selected terms and builds the aggregate weights for the new query. The SQL statements necessary for this step are shown in the following code segment:

```
INSERT INTO new_query
SELECT a.qryid, 1, a.nidf, a.term
FROM temp_table3 a
WHERE a.term NOT IN
   (SELECT b.term FROM query b
    WHERE b.qryid = temp_table3.qryid);

INSERT INTO new_query
SELECT a.qryid, a.termcnt, a.nidf, a.term
FROM query a;

INSERT INTO new_query_wt
SELECT a.qryid, SUM(a.termcnt)/COUNT(*),
     COUNT(*), (1 + log(SUM(a.termcnt)/COUNT(*)))
FROM new_query a
GROUP BY a.qryid;
```

Given the flexibility of the relational model, scaling the new feedback terms to give them a lesser or greater relative weight in the feedback query can be easily implemented by changing the ‘1’ in the first SQL statement to a scale value, such as 0.4 or 1.4 to give the feedback terms more or less relative weight than the original query terms. Again, the flexibility and data independence of the relational model make it unnecessary to make changes to the application code or underlying data structures to accomplish this change.

Step 5 - Identify the top documents for the modified queries through relevance ranking.

The final step is identical to the first except the query is directed to the new_query tables rather than the table containing the original query terms.



	Average Precision	Percent Change
idf, without feedback	.0966	---
idf, with feedback	.1100	+13.8%
nidf, without feedback	.0914	-5.4%
nidf, with feedback	.1400	+44.9%

--- *Figure 1* ---

Figure 1 illustrates the improvement in precision and recall obtained from relevance feedback using two different types of term weights to show that the improvements to retrieval effectiveness obtained from relevance feedback are not dependent upon a particular term weight method. These experiments were run using a subset of documents from disks 2 and 4 of the Tipster collection and the 50 TREC-5 queries.

B. Relevance Feedback Improvements

Many techniques have been proposed to tailor and improve the relevance feedback results since relevance feedback in information retrieval systems was first proposed in the late-1960's. However, most systems using relevance feedback incorporate a variety of techniques; so the contribution to the overall improvement from any single technique is relatively unknown. Most of the previous work done on comparing different relevance feedback improvements techniques has been done using small, special purpose document collections (Salton90, Harman92), and there has been no systematic comparison of the techniques done against a large, standard collection of research data. To better understand the interaction between the different relevance feedback techniques, the experiments described below focus on a variety of relevance feedback improvement techniques and determine the impact each technique has on improving precision and recall in an information retrieval system.

Our information retrieval system is based on the vector-space model and is implemented in a relational database using unchanged SQL. Details of implementing an information retrieval system using unchanged SQL are found in (Grossman96, Grossman97, Lundquist97a). A four processor Teradata DBC/1012 parallel processing computer is used as the platform for the information retrieval system. The Teradata DBC/1012 Database Computer is a special purpose machine designed to run a relational database management system using standard SQL. To investigate the behavior of relevance feedback in information retrieval, a series of experiments were conducted using various combinations of the TREC queries and the Tipster data collection. The following sections describe these experiments and the improvements in retrieval effectiveness demonstrated by the different relevance feedback techniques. Descriptions of several of these experiments are found in (Lundquist97b).

The following sections focus on discrete components of the relevance feedback process. Because the relevance feedback process consists of several factors working together, the impact from a single factor is investigated by varying that factor while holding all other factors constant.

i. Feedback Term Selection Techniques

The first step of the automatic relevance feedback process identifies the *n* top-ranked documents by conducting relevance ranking between the query and the documents in a collection. These documents are assumed to be relevant and used as the source of the feedback terms. The issue then becomes one of identifying the particular terms, and only those terms, which will improve the precision and recall levels for the query. If "good" terms are chosen, the query will find more relevant documents. However, if "bad" terms are chosen, the feedback terms can potentially re-focus the query onto a topic different from the original query topic and result in fewer relevant documents being identified after relevance feedback. In 1992, Harman experimented with several different feedback term selection techniques using the Cranfield document collection (Harman92). According to her results, the best feedback term selection techniques were those which incorporated information about the total frequency of a term in the collection rather than just the frequency of the term within a document.

To determine the impact on precision and recall from the type of feedback term selection method used, several different feedback term selection methods were developed and tested. These experiments were conducted using a subset of documents from disks 2 and 4 of the Tipster data collection along with the short versions of queries from the TREC-4 and

TREC-5 conferences. The feedback term selection methods are described below and each can easily be implemented using standard SQL.

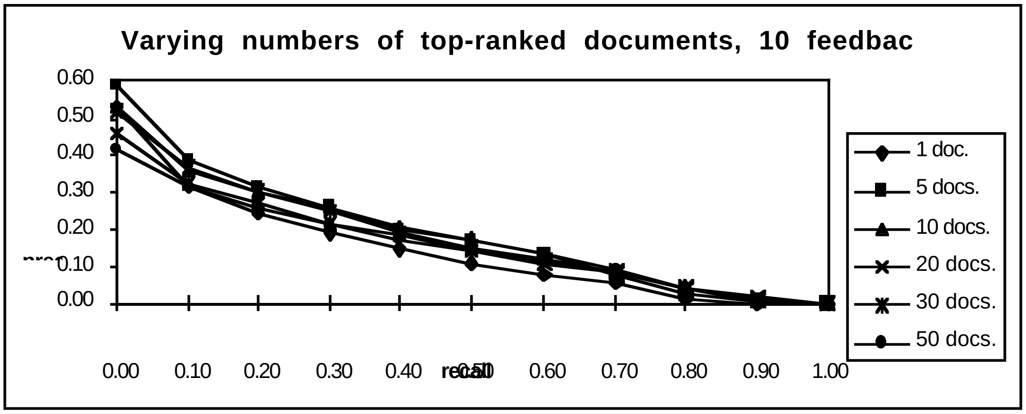
1. *n* * *term weight* (where *n* = number of top-ranked documents containing the term and *term weight* is the *nidf* or *idf* weight of the term in the collection)
2. Modified TermWeight (where modified term weight = (*term weight* * ((1 + log(termcnt))/logavgtf)))
3. Min3Docs (where the top terms occurring in at least 3 of the top-ranked documents are ordered by their *term weight* values)
4. Min4Docs (where the top terms occurring in at least 4 of the top-ranked documents are ordered by their *term weight* values)
5. SUM(termcnt * *term weight*)/reldoc (where reldoc = number of top-ranked documents chosen for relevance feedback)
6. *term weight* * log(*n*) (where *term weight* = the weight of the term in the collection and *n* = number of top-ranked documents containing the term)

The first method, n * term weight, appears to be the most effective feedback term selection technique when compared to the other techniques.

ii. Number of Top-Ranked Documents

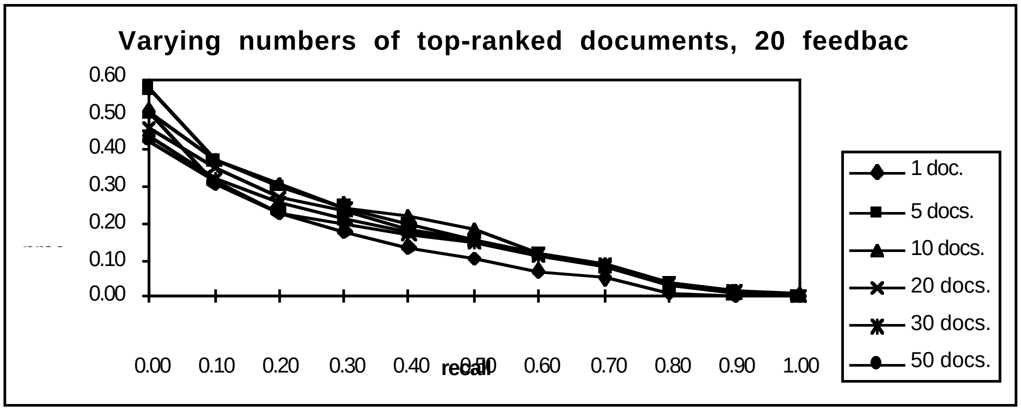
One of the issues in relevance feedback concerns the optimal number of top-ranked documents to use as the source of the feedback terms because in automatic relevance feedback, all of the *n* top-ranked documents are assumed to be relevant to the query. The top-ranked documents for a particular user query are identified by running the SQL relevance ranking query to match terms from the user supplied query against the documents in the collection. The ORDER BY clause in the SQL query returns the relevant documents in ranked order so that the *n* top-ranked documents can be selected.

To investigate the impact on precision and recall for the number of top-ranked documents used, experiments were conducted using the short versions of the 50 TREC-4 queries and documents from disk 2 of the Tipster collection. In these experiments, the *idf* term weight method (described in section 4.5) was used and either 10 or 20 feedback terms were selected from 1, 5, 10, 20, 30, or 50 top-ranked documents. Similar experiments were done by Harman in 1992 but with the small Cranfield document collection and using the probabilistic model (Harman92). Figures 2 and 3 illustrate how the levels of precision and recall are impacted when the number of top-ranked documents used for relevance feedback is varied.



	Average Precision	Percent Change
1 document	.1339	---
5 documents	.1769	+32.1%
10 documents	.1755	+31.1%
20 documents	.1719	+28.4%
30 documents	.1520	+13.5%
50 documents	.1479	+10.5%

--- Figure 2 ---



	Average Precision	Percent Change
1 document	.1283	---
5 documents	.1721	+34.1%
10 documents	.1802	+40.5%
20 documents	.1599	+24.6%
30 documents	.1517	+18.2%
50 documents	.1441	+12.3%

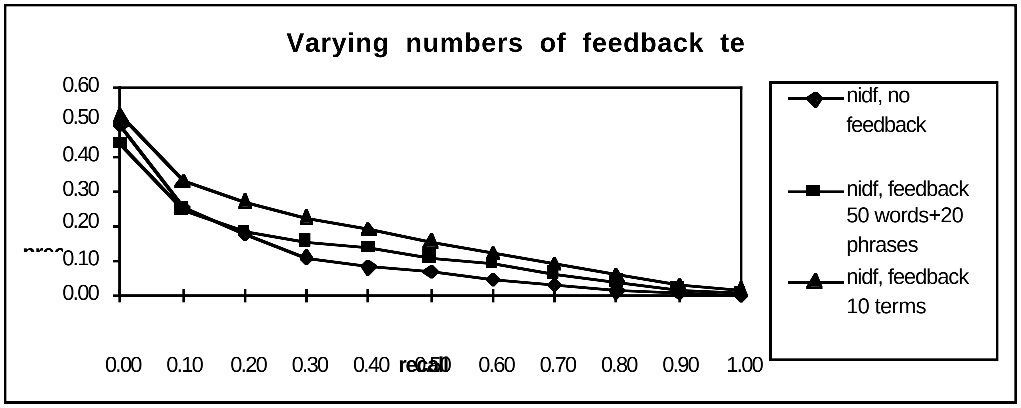
--- Figure 3 ---

The results shown in figures 2 and 3 indicate that the greatest increases in precision and recall are obtained when between 5 and 20 top-ranked documents are used for use for relevance feedback. The results also indicate that selecting a fewer number of documents or a larger number of documents for use in relevance feedback produces less than optimal levels of precision and recall. The results also show that there was a negligible difference when using 10 or 20 feedback terms. The following section provides more detail on the impact on relevance feedback when varying numbers of feedback terms are selected.

iii. Number of Feedback Terms

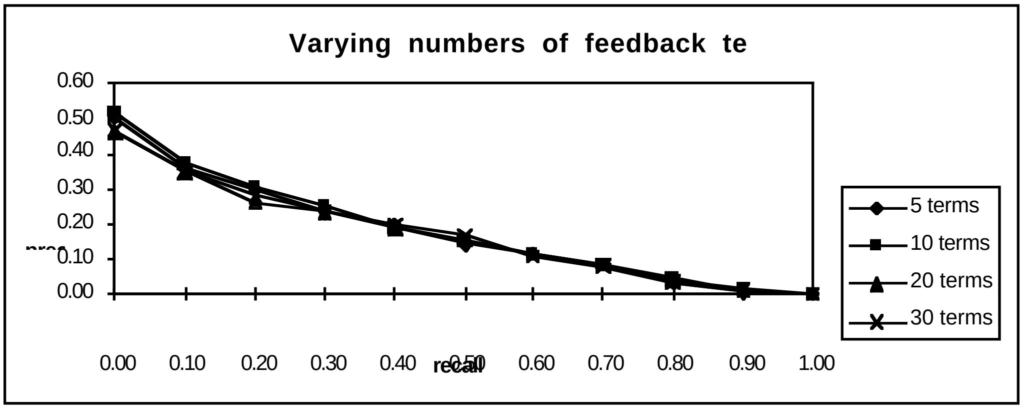
In relevance feedback, a predetermined number (*x*) of feedback terms are selected from the top-ranked documents and added back into the original user query. The goal behind the query expansion is to identify new terms that enhance the retrieval effectiveness of the user supplied query in a particular document collection. The various feedback term selection techniques (described in section 3.B.i) generate a numerical score representing the likelihood that the term is related to the user supplied query terms. The SQL used to implement the feedback term selection techniques generates a list of candidate feedback terms in ranked order which can be used to identify the top *x* feedback terms to add back into the original user supplied query.

To evaluate the impact of the number of terms selected from the top-ranked documents on improving precision and recall, two sets of experiments were conducted. The first set of experiments was conducted using the short versions of the 50 TREC-5 queries and a subset of documents from disks 2 and 4 of the Tipster data collection. In the first set of experiments, the nidf term weighting method was used, the 20 top-ranked documents for each query were identified, and the *n * term weight* feedback term selection technique was used to select 10 terms (either single words or two-word phrases) in one experiment and 50 words plus 20 phrases for the second experiment. Results shown in figure 4 compare the precision and recall levels when the two different numbers of feedback terms are used and shows how much improvement over the baseline is obtained from both methods. Figure 4 illustrates that using a smaller number of feedback terms is clearly better for the short TREC-5 queries than using a larger number of feedback terms although many groups in TREC have used 50 words plus 20 phrases for relevance feedback (Buckley95, Ballerini96).



	Average Precision	Percent Change
no feedback	.0914	---
feedback, 50 words + 20 phrases	.1147	+25.5%
feedback, 10 terms	.1400	+53.2%

--- Figure 4 ---



	Average Precision	Percent Change
5 terms	.1713	---
10 terms	.1755	+2.5%
20 terms	.1802	+5.2%
30 terms	.1746	+1.9%

--- Figure 5 ---

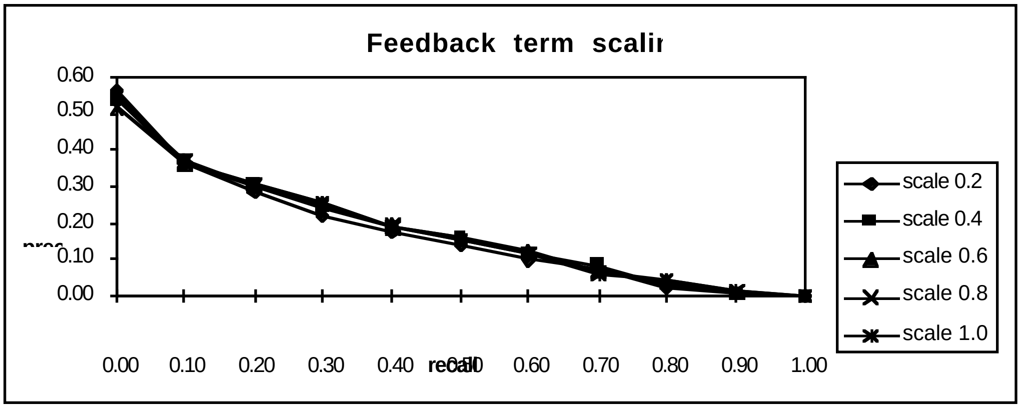
The second set of experiments was conducted using only documents from disk 2 of the Tipster collection, the short versions of the 50 TREC-4 queries, and the *idf* term weighting method. During these experiments the number of feedback terms used was varied when 20top-ranked documents were selected for relevance feedback. Figure 5 shows how the precision and recall levels varied when 5, 10, 20, and 30 feedback terms were added to the original query.

Figure 5 shows that the highest levels of precision and recall are obtained when between 10 and 20 top-ranked documents are selected for relevance feedback and 10 to 20 new terms are added to the short versions of the TREC-5 and TREC-4 queries with terms consisting of either words or phrases. Figure 4 clearly shows that the method commonly used by several TREC participants of expanding the user supplied queries with 50 words and 20 phrases is not as effective as adding only 10 terms (either words or phrases) to the user supplied query. When more feedback terms are used or when fewer feedback terms are used, the precision and recall levels obtained are less than optimal.

iv. Feedback Term Scaling

Rocchio's original formula contained a scaling factor which allowed the relative weights of feedback terms to be either increased or decreased with respect to the original query terms. This scaling factor can be easily implemented in our relational system by applying the scaling factor to the term count (termcnt) field for the feedback terms in the intermediate temporary tables (see step 4 in the relevance feedback process).

To investigate the impact of scaling the weights of the feedback terms, a series of experiments using documents from disk 2 of the Tipster collection along with the short versions of the 50 TREC-4 queries were conducted. Ten feedback terms were chosen from the 20 top-ranked documents and various scaling factors were used to adjust the weight of the feedback terms relative to the original query terms by 0.2, 0.4, 0.6, 0.8, and 1.0. Figure 6 shows the impact scaling has on the overall precision and recall for the queries. The results described in figure 6 indicate that when a scaling factor between 0.4 and 0.6 is used to adjust the weight of the feedback terms relative to the original query terms, the greatest improvement in the precision and recall levels can be obtained.



	Average Precision	Percent Change
scale 0.2	.1580	---
scale 0.4	.1696	+7.3%
scale 0.6	.1741	+10.2%
scale 0.8	.1741	+10.2%
scale 1.0	.1719	+8.8%

--- *Figure 6* ---

v. Relevance Feedback Thresholding

When the precision and recall levels for individual queries were examined, a correlation coefficient of +0.24 was identified between the percentage of improvement in exact precision seen during relevance feedback and the average of the *nidf* term weights of the words (not including any phrases) within the queries. This correlation implies that queries with an average *nidf* of their words below a certain threshold should not undergo relevance feedback. To test this hypothesis, several experiments using the short versions of the TREC-5 queries, a subset of disks 2 and 4 of the Tipster data collection, the *nidf* term weight method, the *n * term weight* feedback term selection method, 10 feedback terms, and 20 top-ranked documents were conducted. These experiments determined that the maximum improvement to precision and recall is realized if relevance feedback is not performed on queries having an average *nidf* < .2175. Further experiments showed that the average *nidf* for a query should be calculated on only the words having an *nidf* <= 0.4. (The 0.4 value roughly corresponds to the word occurring in less than 35,000 documents in the collection. Using these calculations, six of the fifty TREC-5 queries (#251, #263, #268, #270, #272, #293) did not undergo relevance feedback. As a result of relevance feedback thresholding, the average precision increased +1.4% from .1400 to .1421 and the exact precision increased +2.1% from .1755 to .1791. These results demonstrate that improvement, while not dramatic, can be achieved through relevance feedback thresholding.

vi. Combination of Techniques

The experiments demonstrated that for short versions of the TREC-5 and TREC-4 queries, maximum levels of precision and recall were obtained when a relevance feedback process incorporating the following components is used: 1) query expansion by 10 to 20 new terms (either words or phrases); 2) number of top-ranked documents between 5 to 20; and 3) feedback terms scaled by a factor between 0.4 and 0.6. One of the interesting results from these experiments is it appears that certain characteristics of a query (i.e., average *nidf* of words in the query) are a predictor of improvement under relevance feedback.

4. Parallel Relevance Feedback Algorithm

One of the major drawbacks to parallel information retrieval systems is that they require custom hardware and software which greatly reduces the portability of these systems across platforms. In our information retrieval system, parallelism can be easily obtained without requiring any custom hardware or software by treating information retrieval as an application of an RDBMS. This provides a parallel implementation without requiring any changes to the basic information retrieval algorithm.

A. Parallel RDBMS

A major benefit to using the relational model for information retrieval is the ability to exploit parallel processing via the DBMS. To test the effectiveness of parallel processing in information retrieval, an information retrieval system was implemented using Teradata's RDBMS on a 4 processor DBC/1012 Model 4 parallel processing machine. The Teradata DBC/1012 Database Computer is a special purpose machine designed to run a relational database management system using standard SQL.

On the DBC/1012, I/O parallelism is achieved by dividing and physically storing the rows of each table on the different processors. A hashing function is applied to the primary index value for each row, and the output is a 32 bit value which becomes the logical storage address for the row and identifies the processor which will store the row. When rows are retrieved, the hashing function is applied to the key value for the row to determine the logical storage address. An I/O request is then sent to the correct processor, and the row is retrieved. When multiple rows are being retrieved, each processor can work independently to retrieve its rows and send them to a temporary storage area. The resulting dataset from all of the processors is then returned to the user. An interesting side effect of using a hashing design for the indexes instead of an ordered index such as the B-tree index design commonly used in non-parallel RDBMS is that unlike ordered indexes, queries done in the Teradata system cannot make use of partial indexes. Thus, all columns of the index must be contained in the query's WHERE clause or the query cannot use the index. This is different from an ordered B-tree index where if the query's WHERE clause contains only the first column of a multi-column index, the query can still use the index to efficiently access the data.

On the DBC/1012, parallel processing of the queries is transparent to the user. When SQL statements are issued, the parser and query optimizer partitions the request into a series of serial and parallel steps. Parallel steps are are executed concurrently by two or more processors. These parallel steps, which make the best use of the DBC/1012 architecture, are generated by the optimizer whenever possible. The DBC/1012 can execute some statements in a single transaction in parallel. This capability applies both to implicit transactions, such as macros and multi-statement requests, and to transactions explicitly defined with BEGIN/END TRANSACTION statements. Statements in a transaction are broken down by the interface processor into one or more steps that direct the transaction execution performed by the processors. It is these steps, not the actual statements, that are executed in parallel. A handshaking protocol between the interface processor and the processors allows the processors to determine when the interface processor can dispatch the next parallel step (DBC-1).

To show how the query parser and optimizer function on the DBC/1012 to process a query in parallel, the following SQL statement is used as an example. The processing steps outlined below are generated by the "EXPLAIN" function on the DBC/1012 which can be used to obtain information from the query parser and optimizer as to how a particular SQL statement will be processed. These steps illustrate how the query parser and optimizer on the DBC/1012 allow a standard SQL statement to be run in a parallel mode in a way that is totally transparent to the user.

Sample Relational Table Structures:

Doc_Term (docid, termcnt, term)
Doc_Weight (docid, docname, avgtf, disterm, logavgtf)*
Query_Term (qryid, termcnt, nidf, term)
Query_Weight (qryid, avgtf, disterm, logavgtf)*

*Certain aggregate values are pre-computer to optimize performance.

Sample Relevance Ranking Query:

```
SELECT c.qryid, b.docid,SUM(((1+LOG(a.termcnt))/((b.logavgtf) *
        (229.50439 + (.20 * b.disterm)))) * (c.nidf * ((1 + LOG(c.termcnt))/(d.logavgtf))))
FROM disk5_doc_term a, disk5_doc_weight b, query_term c, query_weight d
WHERE a.docid = b.docid AND c.qryid = d.qryid AND a.term = c.term and c.qryid = 301
GROUP BY c.qryid, b.docid
      UNION
SELECT c.qryid, b.docid,SUM(((1+LOG(a.termcnt))/((b.logavgtf) *
        (229.50439 + (.20 * b.disterm)))) * (c.nidf * ((1 + LOG(c.termcnt))/(d.logavgtf))))
FROM disk4_doc_term a, disk4_doc_weight b, query_term c, query_weight d
WHERE a.docid = b.docid AND c.qryid = d.qryid AND a.term = c.term and c.qryid = 301
GROUP BY c.qryid, b.docid
ORDER BY 3 DESC;
```

Breakdown of DBC/1012 Processing Steps:

Step 1 - A read lock is placed on tables disk5_doc_term, disk5_doc_weight, disk4_doc_term, and disk4_doc_weight.

Step 2 - A single processor is used to select and join rows via a merge join from query_term and query_weight where the value of qryid = 301. The results are stored in temporary table1.

Step 3 - An all processor join is used to select and join rows via a row hash match scan from disk5_doc_term and temporary table1 where the values of the term attribute match. The results are sorted and stored in temporary table2.

Step 4 - An all processor join is used to select and join rows via a row hash match scan from disk5_doc_weight and temporary table2 where the values of the docid attribute match. The results are stored on temporary table3 which is built locally on each processor.

Step 5 - The SUM value for the aggregate function is calculated from the data in temporary table3 and the results are stored in temporary table4.

(The next two steps, 6a and 6b, are executed in parallel)

Step 6a - The data from temporary table4 is retrieved and distributed via a hash code to temporary table5 which encompasses all processors.

Step 6b - An all processor join is used to select and join rows via a row hash match scan from disk4_doc_term and temporary table1 where the values of the term attribute match. The results are sorted and stored on temporary table6.

Step 7 - An all processor join is used to select and join rows via a row hash match scan from disk4_doc_weight and temporary table6 where the values of the docid attribute match. The results are stored on temporary table7 which is built locally on each processor.

Step 8 - The SUM value for the aggregate function is calculated from the data on temporary table7 and the results are stored on temporary table8.

Step 9 - The data from temporary table8 is retrieved and distributed via a hash code to temporary table9 which encompasses all processors. A sort is then done to remove duplicates from data on temporary table9.

Step 10 - An END TRANSACTION step is sent to all processors involved and the contents of temporary table9 are sent back to the user.

As seen in the steps described above, the DBC/1012 is able to execute various steps of the query in parallel as well as execute a single step concurrently on multiple processors. All of this, whether it be deciding on the best parallel strategy to process the query or executing the steps in parallel, is accomplished independently of the user and does not require any specialized instructions from the user.

Our hypothesis for the parallel approach was that queries would run in a balanced fashion and the workload for each processor would be approximately equal. This balance is critical if the approach is to be truly scalable. To determine the level of processor imbalance, processor performance data for both the initial relevance ranking to identify the top-ranked documents (Step 1 in Section 3.A) and the relevance ranking after relevance feedback had been accomplished by identifying and adding ten new terms (Step 5 in Section 3.A) was collected. Table 1 illustrates the level of processor imbalance where processor and disk I/O imbalance are calculated with the following equations:

$$\text{processor imbalance} = \frac{(\max\_cpu - \min\_cpu)}{\min\_cpu} \qquad \text{(F2)}$$

$$\text{disk I/O imbalance} = \frac{(\max\_diskio - \min\_diskio)}{\min\_diskio} \qquad \text{(F3)}$$

where

max_*cpu* = maximum cpu time across all processors
min_*cpu* = minimum cpu time across all processors
max_*diskio* = maximum disk I/O time across all processors
min_*diskio* = minimum disk I/O time across all processors

	Percentage of processor imbalance	Percentage of disk I/O imbalance
Step 1 - Relevance Ranking before Relevance Feedback terms added	4.98%	0.69%
Step 5 - Relevance Ranking after Relevance Feedback terms added (10 terms)	9.86%	2.46%

---*Table 1*---

It can be seen that for all workloads, the processors and disk I/O amounts are ten percent or less out of balance. Given that the workload is seemingly balanced evenly among the existing processors, if additional processors are added, response time potentially can be reduced.

B. Parallel Performance

A key aspect of any parallel system are whether the multiple processors are efficiently distributing the workload. One of the methods used in industry to determine the efficiency at which the processors are distributing the workload is to calculate the Parallel Efficiency (PE) (Higa91). The PE measures how well the processors are working together and is an indicator as to how well the data is distributed among the processors. A low PE may indicate that some of the tables being processed by the SQL statement may require a different primary index so the data are hashed and distributed more evenly among the multiple processors. The PE is calculated as follows:

$$\text{parallel efficiency (PE)} = \frac{avg\_cpu}{\max\_cpu} \qquad \text{(F4)}$$

where

max_*cpu* = max. cpu time across all processors
avg_*cpu* = avg. cpu time across all processors

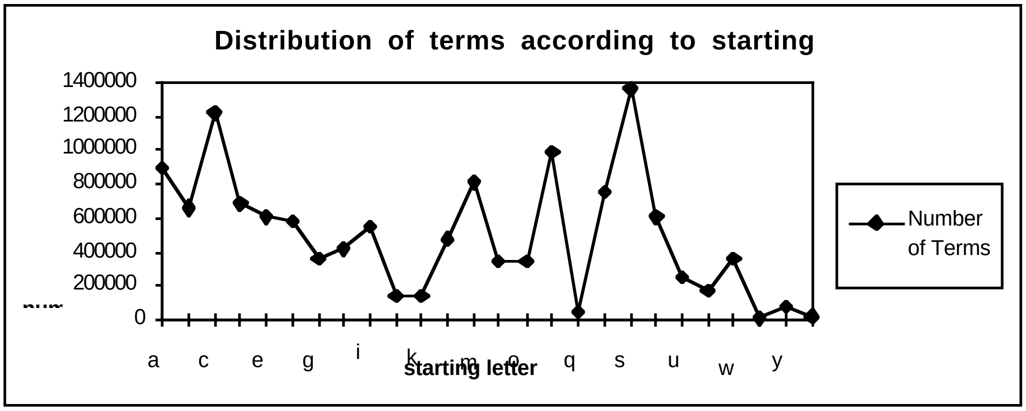
For example, to calculate the PE for the relevance ranking run on the four processor DBC/1012 parallel machine, the average and maximum CPU times across all processors for all 50 queries is determined. When this is done, the PE = 149.45/176 or 85%.

To determine if the information retrieval system was efficiently distributing its workload across multiple processors, several experiments were conducted using both a 4 processor and a 24 processor Teradata DBC/1012 Model 4. Table 2 illustrates the performance and PE for the 50 TREC-5 queries run on these two machines against documents from Tipster disk 2 and disk 4 (approximately 1 gigabyte of data per disk).

	Average CPU time per processor	Maximum CPU time per processor	Data storage imbalance across processors	Parallel Efficiency
4 processors, disk 2 only, primary index on term	149.45	176.00	6.1%	84.9%
4 processors, disk 2 and 4, primary index on term	393.65	453.20	3.1%	86.9%
24 processors, disk 2 only, primary index on term	17.07	42.04	18.9%	40.6%
24 processors, disk 2 and 4, primary index on term	49.11	111.10	19.2%	44.2%

---*Table 2*---

Table 2 illustrates that as the disk storage imbalance increases across the processors, the PE decreases because the workload is not evenly distributed across the processors. Section 4.A described how a hash function based on the primary index is applied to data in the table to distribute it across the processors. So when the primary index is defined to be the term field, the hash function is based on the non-unique term value. This leads to an uneven distribution of the data because terms are more likely to start with certain letters than with others and some terms occur more frequently than others. The uneven distribution across the processors is not as apparent on the 4 processor machine because there are only 4 processors resulting in frequently and infrequently occurring terms being assigned to each processor, thus somewhat balancing out the data storage. However, the uneven distribution is much more apparent on the 24 processor machine because the data are divided among a larger number of processors and there is less opportunity for the frequently and infrequently occurring terms to balance. To highlight the imbalance among the terms in the collection, figure 7 shows the distribution of terms according to their starting letter.



--- Figure 7 ---

To determine if the PE could be improved by redistributing the data across the processors, data from Tipster disks 2 and 4 was loaded into the database using the combination of the docid and term fields to create a unique primary index so that the hash function would be able to more uniformly balance the data across the 24 processors. Table 3 illustrates the difference in CPU time, data storage imbalance, and PE when the data are stored differently on the 24 processors.

	Average CPU time per processor	Maximum CPU time per processor	Data storage imbalance across processors	Parallel Efficiency
24 processors, disk 2 only, primary index on term	17.07	42.04	18.9%	40.6%
24 processors, disk 2 and 4, primary index on term	49.11	111.10	19.2%	44.2%
24 processors, disk 2 only, primary index on docid and term	29.21	31.90	0.8%	91.6%
24 processors, disk 2 and 4, primary index on docid and term	74.18	79.09	0.5%	93.8%

---*Table 3*---

The results listed in table 3 show that when an unique index is used as the primary index, the hashing function is able to uniformly distribute the data across the 24 processors and the data storage imbalance among the processors drops from about 19% to about 1%. Because the data are more evenly distributed, the PE more than doubles and the maximum cpu time per processor decreases by 24% for just disk 2 and 29% for disks 2 and 4 together. The experiments described in this section show that when a system is scaled up to run on additional processors, it may be necessary to restructure the data to maintain an even distribution of data across the processors. If this is not done, then the full extent of the performance improvements obtained by increasing the number of processors may not be realized.

C. Parallel Scalability

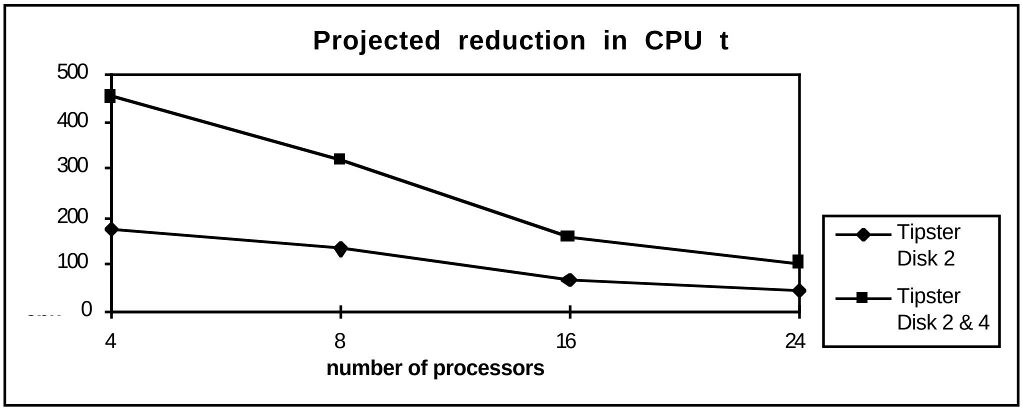
Another key aspect of a parallel system is whether the system demonstrates performance improvements when the number of processors is increased. For example, if a query required 100 seconds to run on a 4 processor machine and the machine was 100% scalable, the query would then only require 50 seconds to run on an 8 processor machine. Most machines, however, do not provide 100% scalability due to hardware limitations, i.e., it takes longer to coordinate messages among 8 processors than among 4 processors. The scalability of a machine can be projected and estimates of the reduction in cpu time and number of disk I/O's per processor can be calculated using the processor imbalance formulae in F2 and F3.

$$\text{projected cpu time per processor} = X\left(\frac{n_1}{((1-PI)n_2)}\right) \qquad \text{(F5)}$$

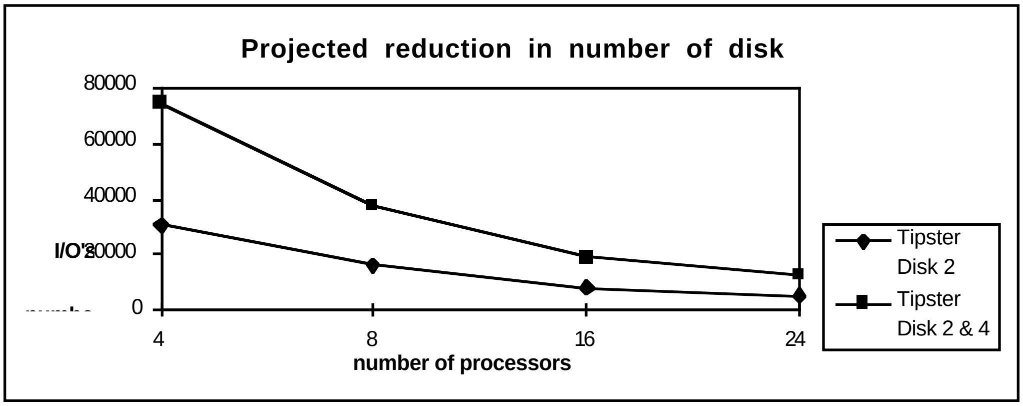
$$\text{projected number of disk I/O's per processor} = Y\left(\frac{n_1}{((1-DI)n_2)}\right) \qquad \text{(F6)}$$

where

PI = processor imbalance calculated by F29
DI = disk I/O imbalance calculated by F30
X = maximum CPU time per processor on n_1 processors
Y = maximum number of disk I/O's per processor on n_1 processors
n_1 = number of original processors
n_2 = number of projected processors



--- *Figure 8* ---



--- *Figure 9* ---

Figures 8 and 9 illustrate the projected reduction in maximum cpu time and number of disk I/O's per processor as the number of processors increases from 4 to 24.

To determine how accurate the estimates of the reduction in average cpu time and number of disk I/O's per processor were, identical amounts of data were loaded into identical relational structures on a 4 processor and a 24 processor Teradata DBC/1012 model 4 parallel machine and identical queries were run on the two machines. The estimated maximum cpu time and number of disk I/O's per processor along with the actual performance results on the 24 processor machine are shown in tables 4 and 5.

	Projected maximum CPU time per processor	Actual maximum CPU time per processor with data storage unbalanced	Actual maximum CPU time per processor with data storage balanced
Tipster disk 2 and 50 TREC-5 queries	45.83	42.04	31.90
Tipster disk 2 & 4 and 50 TREC-5 queries	107.90	111.10	79.09

---*Table 4*---

	Projected maximum number of disk I/O's per processor	Actual maximum number of disk I/O's per processor with data storage unbalanced	Actual maximum number of disk I/O's per processor with data storage balanced
Tipster disk 2 and 50 TREC-5 queries	5197.28	6424.0	9226.0
Tipster disk 2 & 4 and 50 TREC-5 queries	12464.65	14186.0	20772.2

---*Table 5*---

The results shown in tables 4 and 5 indicate that when the number of processors is increased from 4 processors to 24 processors the actual performance is approximately equal to the projected performance, however, when the data stored on the processors is more uniformly distributed, the actual performance is then much better than the projected performance. This difference is due to the improved parallel efficiency when the data stored are balanced among the processors. The number of disk I/O's per processor, however, shows a different pattern because as the data becomes more uniformly distributed among the processors, the number of disk I/O's done by each processor increases. This increase can be explained by having to perform additional disk I/O's to use a secondary index on just the term column. This is done because the query is still searching on the term value, but since the primary index is placed on the combination of docid and term, a second index is necessary. Table 6 shows the actual scalability of the system by directly comparing the CPU times on the 4 processor and the 24 processor machines and shows that when the data is uniformly distributed among the processors, the system achieves a 95.5% level of scalability for disks 2 and 4.

	Scalability when data is unbalanced	Scalability when data is balanced
Tipster disk 2	69.8%	92.0%
Tipster disk 2 & 4	68.0%	95.5%

---*Table 6*---

D. Summary

The preceding sections provide a parallel algorithm for implementing an information retrieval system. The results described above demonstrate that when an information retrieval system is designed on the relational model, it can easily and transparently obtain parallelism by using a parallel RDBMS. It is further demonstrated that such a system is scalable (up to 95.5%) and shows significant performance improvements when scaled up from 4 to 24 processors. These results indicate that an information retrieval system implemented in an RDBMS is scalable and demonstrates significant reductions in processor CPU times and disk I/O's.

5. Conclusions

Information retrieval systems must sustain a high degree of accuracy and must scale in terms of the volume of data they process. We described an information retrieval system that is implemented as a relational database application, and therefore, benefits from all the traditional features supported by commercial database systems. Namely, features such as concurrency, recovery, data integrity constraints, optimization algorithms, and parallel implementations, are all available to the information retrieval application without additional software development.

We described our information retrieval system that is implemented as an application of a relational DBMS. We showed how relevance feedback could be implemented with standard SQL and identified tuning paramters that improved effectiveness of relevance feedback over prior work. Additionally, we implemented relevance feedback on a parallel database machine and, using TREC data, enhanced precision and recall over previously developed database driven approaches while sustaining near linear speed-up using 24 nodes.

It is still an open question what combination of retrieval enhancements and underlying models are best suited for information retrieval. What is clear is that all future information retrieval systems must, at least, follow the technology curve since on-line digital data are exponentially growing. We believe that to support such growth without incurring vast software development costs requires reliance on proven scalable technology. Our relational database-driven information retrieval system is but one attempt at meeting these demands.

References

(Ballerini96) Ballerini, J., M. Buchel, D. Knaus, B. Mateev, E. Mittendorf, P. Schauble, P. Sheridan and M. Wechsler, “SPIDER Retrieval System at TREC-5,” to appear in Text Retrieval Conference, sponsored by National Institute of Standards and Technology and Advanced Research Projects Agency, November 1996.

(Blair75) Blair, D., Square (specifying queries as relational expressions) as a document retrieval language. Unpublished working paper, University of California, Berkeley, 1975.

(Blair85) Blair, D. and M. Maron, “An evaluation of retrieval effectiveness for a full-text document retrieval system,” *CACM*, 28(3), 289-299, 1985.

(Buckley95) Buckley, C., A. Singhal, M. Mitra, and G. Salton, “New Retrieval Approaches Using SMART: TREC 4,” Text Retrieval Conference, sponsored by National Institute of Standards and Technology and Advanced Research Projects Agency, November 1995.

(Crawford78) Crawford, R. G. and Ian A. Macleod, “A Relational Approach to Modular Information Retrieval Systems Design,” *Proceedings of the American Society for Information Systems Annual Meeting*, vol. 15, pp. 83-85, 1978.

(Crawford81) Crawford, R.G., “The Relational Model in Information Retrieval,” *Journal of the American Society for Information Science*, pp. 51-64, January 1981.

(DBC-1) System Manual, Release 5.0, AT&T Global Information Solutions Company, Dayton, Ohio, 1994.

(DeFazio95) Defazio, S., A. Daoud, L. Smith, J. Srinivasan, W. B. Croft, and J. Callan, “Integrating IR and RDBMS Using Cooperative Indexing,” *ACM Eighteenth International Conference on Research and Development in Information Retrieval*, 1995.

(Grossman94) Grossman, D., D. Holmes, and O. Frieder, "A Parallel DBMS Approach to IR in TREC-3," *Overview of the Third Text REtrieval Conference (TREC-3),* November 1994.

(Grossman96) Grossman, D, C. Lundquist, J. Reichert, D. Holmes and O. Frieder, “Using Relevance Feedback within the Relational Model for TREC-5,” to appear in Text Retrieval Conference, sponsored by National Institute of Standards and Technology and Advanced Research Projects Agency, November 1996.

(Grossman97) Grossman, D., D. Holmes, O. Frieder and D. Roberts, “Integrating Structured Data and Text: A Relational Approach,” *Journal of the American Society of Information Science*, February 1997.

(Harman92) Harman, D., “Relevance Feedback Revisited,” *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Ed. Nicholas Belkin, Peter Ingwersen and Annelise Mark Pejtersen, SIGIR Forum, June 21-24, 1992.

(Hawking96) Hawking, D., P. Thistlewaite and P. Bailey, “ANU’s TREC5 Experiments,” to appear in Text Retrieval Conference, sponsored by National Institute of Standards and Technology and Advanced Research Projects Agency, November 1996.

(Higa91) Higa, L., Teradata Resource Utilization Macros, Teradata Corporation, 1991.

(Ide71) Ide, E., “New Experiments in Relevance Feedback,” Gerard Salton, Editor, The SMART Retrieval System, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

(Lundquist97a) Lundquist, C., D. Grossman, O. Frieder, and D. Holmes, “A Parallel Implementation of Relevance Feedback using the Relational Model,” *Proceedings of the World Multiconference on Systemics, Cybernetics, and Informatics*, July 1997.

(Lundquist97b) Lundquist, C., D. Grossman, and O. Frieder, “Improving Relevance Feedback in the Vector-Space Model,” to appear in Proceedings of the Sixth ACM International Conference on Information and Knowledge Management, 1997.

(Macleod78) Macleod, I. A., “SEQUEL as the Language for Document Retrieval,” *Journal of the American Society for Information Science*, pp. 243-249, September 1979.

(Macleod81a) Macleod, I. A., “A Database Management System for Document Retrieval Applications,” *Information Systems*, vol. 6, no. 2, pp. 131-137, 1981.

(Macleod81b) Macleod, I. A., “The Relational Model as a Basis for Document Retrieval System Design,” *The Computer Journal*, vol. 24, no. 4, 1981.

(Macleod83) Macleod, I. A., and R. G. Crawford, “Document Retrieval as a Database Application,” *Information Technology: Research and Development*, vol. 2, pp. 43-60, 1983.

(Mamalis96) Mamalis, B., P. Spirakis and B. Tampakas, “Parallel Techniques for Efficient Searching over Very Large Text Collections,” to appear in Text Retrieval Conference, sponsored by National Institute of Standards and Technology and Advanced Research Projects Agency, November 1996.

(Oracle97) ORACLE, “Managing Text with Oracle8 ConText Cartridge,” An Oracle Technical White Paper, http://www.oracle.com/st/collateral/html/context_collateral.html, June 1997.

(Rocchio66) Rocchio, Jr., J. J., “Document Retrieval Systems - Optimization and Evaluation,” Ph.D. Thesis, Harvard University, March 1966.

(Rocchio71) Rocchio, Jr., J. J., “Relevance Feedback in Information Retrieval,” Gerard Salton, Editor, The SMART Retrieval System, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

(Salton90) Salton, G. and C. Buckley, “Improving Retrieval Performance by Relevance Feedback,” *Journal of the American Society for Information Science*, v. 41, no. 4, pp. 288-297, 1990.

(Singhal96) Singhal, A., C. Buckley, and M. Mitra, “Pivoted Document Length Normalization,” *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Ed. Hans-Peter Frei, Donna Harman, Peter Schauble and Ross Wilkinson, SIGIR Forum, August 18-22, 1996.

(Stonebraker83) Stonebraker, M., H. Stettner, N. Lynn, J. Kalash, and A. Guttman, “Document processing in a relational database system,” *ACM Transactions on Office Information Systems*, vol. 1, no. 2, pp. 143-158, April 1983.