

Parallel, Platform-Independent Implementation of Information Retrieval Algorithms

M. Catherine McCabe
U.S. Government
Washington D.C.
cmccabe@gmu.edu

David O. Holmes
NCR Corporation
Rockville, Maryland
david.holmes@washingtondc.ncr.com

David Grossman, Ophir Frieder
Illinois Institute of Technology
Chicago, IL 60616
{[dagr](mailto:dagr@iit.edu), [ophir](mailto:ophir@iit.edu)} @ ir.iit.edu

Abstract

The relational platform provides a flexible, low maintenance environment for integrating searches of structured and unstructured data. We present relational algebra for the Information Retrieval problem and SQL for leading probabilistic retrieval approaches. We tested 150 standard Text Retrieval Evaluation Conference queries against a collection of half a million documents. Because of the parallel operation features of RDBMS platforms, we achieve subsecond response time on most queries.

1 Introduction

Integrating text data with structured data (e.g. census data, weather data, etc.) is a critical part of getting the most information out of the ever-growing available electronic data. Much work has been done optimizing algorithms for searching text and different algorithms for searching structured data from databases. Combining these disparate algorithms presents complex applications with separate searches. Our approach uses one search for both structured and unstructured data and permits fusion of several text search algorithms. We propose standard relational SQL to implement leading information retrieval algorithms. The application is parallelizable by spreading the data tables across disk RAID and using the parallel query server of the relational database management system (RDBMS). This paper presents relational SQL for several different information retrieval search approaches as well as the fusion of these approaches within SQL. Performance numbers for the information retrieval application are presented.

In this paper, we describe prior work in Section 2. In Section 3, we present relational

algebra for information retrieval and new SQL for probabilistic retrieval approaches. In Section 4, we present performance results on a parallel platform. Finally, in Section 5, we provide conclusions and directions for future work.

2 Prior Work

Significant prior work includes the development of the relational approach to information retrieval and the development of a relational algorithm and SQL for the vector space model [1]. In addition, work has been done to verify the parallelism this approach [2]. We briefly describe this prior work in this section.

2.1 Relational DBMS platform for Information Retrieval

The mainstream approach to information retrieval tasks uses an *inverted index* to store the text to be searched. In contrast, the relational IR approach uses *relations* to model an inverted index [1]. The design of these relations is shown in Figure 1. The main benefit of using a relational platform is the integration of searches across structured and unstructured data. In addition, the application automatically benefits the concurrency control, recovery, security, portability, scalability, and robustness that are standard with any RDBMS. As vendors continuously improve these features and incorporate advances made in hardware and software, the database application simply upgrades versions to keep up with changing technology. The maintenance savings of such an approach are significant.

The relational IR approach has been implemented in the Scalable Information Retrieval Engine (SIRE) system. SIRE used the

vector space method of information retrieval and ranked in the top ten in precision and recall in various tasks of the Text Retrieval Conference (TREC) over the past seven years. The vector space method of information retrieval represents each document and each query as a vector. The components of the vector consist of term weights. A dot product or cosine distance measure between the vectors is used to measure the relatedness of the document to the query.

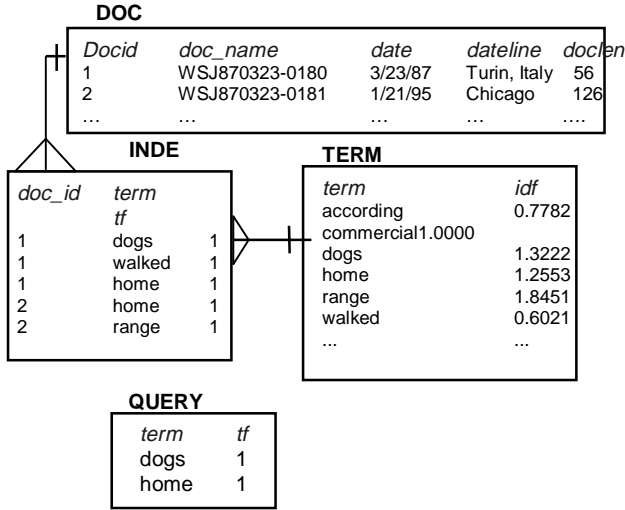


Figure 1: Relational Database Design

This basic vector space retrieval was modified by [3] to account for the observation that longer documents tend to be relevant more often than shorter documents. Singhal introduced a *pivot* formula where term weight in a document vector is d :

$$d_{ij} = \frac{1 + \log tf}{1 + \log(\text{avg}tf)} \text{ where } tf \text{ is the term}$$

frequency within the document. The term weight in a query vector is w which is the same as d but uses the term frequency and average term frequency within queries. The similarity coefficient, or ranking value is defined

$$\text{as: } SC(Q, D_i) = \frac{\sum_{j=1}^t w_{qj} d_{ij}}{((1.0 - s)p + s(|d_i|))} \text{ where } |d_i|$$

is the number of elements in the document vector, p is the pivot which is defined as the average

document length and s is .20 which was experimentally determined by Singhal [3].

Structured Query Language for this formula is shown in SQL2, using the database structure shown in Figure 1. The GROUP BY results in the SUM being calculated across all query terms in each document and thus creates a similarity measure for each document. Note that the number of documents and the average document length are calculated in advance and entered as constants in the SQL. For the TREC6,7,8 collection there are 528,000 documents with an average document length of 606.

```

SELECT d.DocName,
SUM(((1 + LOG(i.tf)) / ((d.LogAvgTF) *
(AvgDocLen + (0.20 * d.DocLen))))
* (t.idf * ((1 + LOG(q.tf)) / (q.LogAvgTF))))
FROM Index i, Doc d, Query q, Term t
WHERE d.Docid = i.Docid
AND q.term = i.term
AND t.term = q.term
GROUP BY d.DocName
ORDER BY 2 DESC;

```

SQL 1: Vector Space Pivoted Normalization

2.2 Parallel Performance of SIRE

A key feature of the relational approach to Information Retrieval is parallel operations – which improves its performance despite the increased workload. While traditional IR approaches have not been effectively parallelized, the relational IR application is highly parallelizable. A 94% Parallel Efficiency (PE) and .05% processor imbalance was achieved for 24 processors [2]. Figure 2 shows the parallelism.

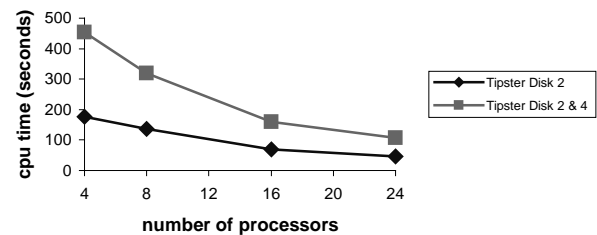


Figure 2: Reduction in CPU time with SIRE

3 Algorithms for Probabilistic Retrieval Strategies

There are several leading approaches or strategies to information retrieval that have not previously been shown to work in the SIRE architecture. These include the probabilistic model introduced by Sparck Jones[4] and currently used in by Robertson et. al. [5], and Kwok’s self-relevance probabilistic model described in [6]. Each of these uses very different representations of the query and the documents and very different measures of similarity. We first present the relational algebra for the Information Retrieval problem. We then describe each probabilistic approach in detail and present SQL for each.

3.1 Relational Algebra for IR

It is possible to formulate the information retrieval problem as a special case of set theory problem – relational sets. The relational manipulation is the same regardless of the information retrieval strategy being applied. We verify this with the presentation of structured query language for each of several most popular strategies – Pivoted Normalized Vector Space model, Okapi probabilistic model, and Kwok probabilistic self-relevance model.

The JOIN pictured is a natural join or an equijoin where the common column is retained only once in the resulting relation [7.] The information retrieval problem can be defined as a series of these relational algebra operations.

Consider the relations proposed in [8] and shown in Figure 1. For this discussion, let us call the QUERY relation, Q , the TERM relation, T , the INDEX relation I and the DOCUMENT relational D . Text documents and queries are parsed and loaded into these relations. Any attributes that are not used in the retrieval are eliminated through a PROJECT operation. A JOIN on the resulting relations results in a relation with all the information required for retrieval. The order of the relations in the JOIN step is theoretically not important – it does not impact the results.

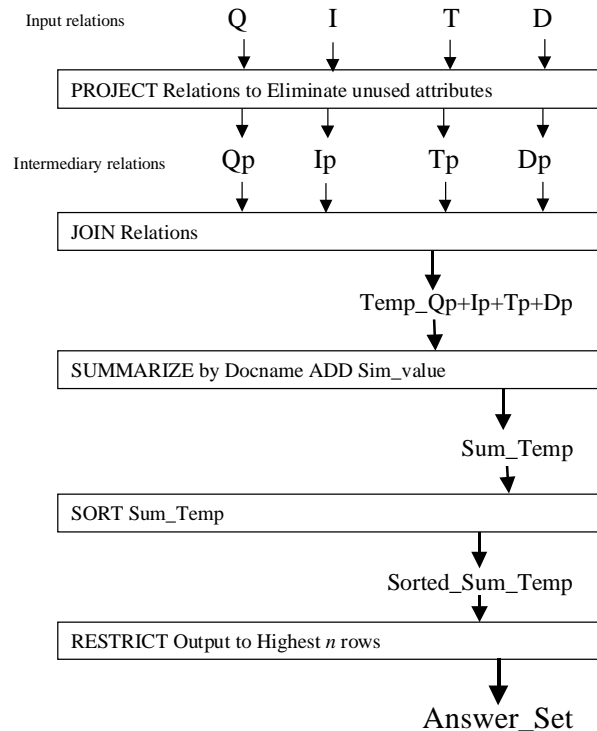


Figure 3: Steps in Relational IR

However, reducing very large relations early in the process eliminates work later and can have a major impact on performance [7]. After the JOIN of the relations, a SUMMARIZE operation calculates the similarity values for each document and generates a temporary relation. Finally, that relation is sorted and restricted for the final output. These steps are outlined in Figure 3.

Using the relational algebra notation of Date in [7], these steps are listed below. In this syntax, T_i indicates a temporary relation, created at the time of assignment and destroyed at an appropriate time such as the end of a session. We have added the $_a$ to indicate the original relation used in the creation the temporary relation.

1. Projections are needed for the Query relation, Q, and the Document relation, D. A projection is denoted by the relation name followed by a list of attributes to keep:

$T1_q := Q [term, tf, LogAvgtf]$
 $T2_d := D [Docid, Doc_name, LogAvgTF, DocLen]$

2. A natural JOIN is simply the relation names with the JOIN operation:

$T3_all := T1_q JOIN I JOIN T JOIN T2_d.$

Alternatively, this could be written

$T3_all := ((T1_q JOIN I) JOIN T) JOIN T2_d).$

Furthermore, a more complex but equivalent expression partitions the JOIN into its primitive components of a PRODUCT:

$T3_all :=$
 $((((T1_q RENAME TERM AS QTERM)$
 $TIMES$
 $((I RENAME TERM AS ITERM) RENAME$
 $DOCID AS IDOCID)$
 $WHERE QTERM = ITERM)$
 $TIMES$
 $(T RENAME TERM AS TTERM)$
 $WHERE TTERM = ITERM)$
 $TIMES$
 $(T2_d RENAME DOCID AS DDOCID)$
 $WHERE DDOCID = IDOCID)$

3. The summarize step groups several rows together and calculates an aggregate attribute.

$T4_all := SUMMARIZE T3_all BY (Doc_name)$
 $ADD aggregate_expression AS RANK$

The aggregate expression will vary depending on the information retrieval model as seen in later sections of this chapter.

Of course, relational algebra contains no notion of sorting. Any sort algorithm may be used to format the result sets for the user of a particular search application. In addition, the end user may require that the number of rows returned be limited to a few of the highest ranking documents. Again, a trivial cut-off may be used to achieve this functionality.

3.2 Okapi Probabilistic Model

The OKAPI system is an implementation of the probabilistic information retrieval model. It has done very well at TreC [5]. The term weight used in the probabilistic model was defined in [4] and is the ratio of the probability of relevance to the probability of nonrelevance. Using the values used in Trec-7 and described in [5], this similarity measure reduces to:

$$\sum \log \left(\frac{N - n + .5}{n + .5} \right) \left(\frac{(k_1 + 1)tf}{K + tf} \right) qtf$$

where:

N is the number of documents in collection
 n is the number of documents containing this term
(document frequency or df)
 $k_1 = 1.2$
 $b = .75$ (This is used to bias the collection based on document size)
 $K = k_1((1-b) + b*dl/avgdl)$
 dl = document length
 $avgdl$ = average document length

So, replacing the variables with their values results in the following formula for probabilistic retrieval. SQL 3 presents SQL for this formula.

$$\sum_{term=i} \log \left(\frac{(N - df_i) + .5}{(df_i + .5)} \right) * \left(\frac{2.2 * tf_{id}}{.3 + (.75 * dl / avgdl) + tf_{id}} \right) * qtf$$

```
SELECT Docid,
SUM(LOG(((NumDocs - t.df) + 0.5) / (t.df + 0.5)) *
((2.2*i.tf) / (.3 + ((.75 * d.dl) / avgdl) + i.tf))
*q.termnt)
FROM INDEX i, TERM t, DOCUMENT d, QUERY q
WHERE i.term = t.term
AND i.docid = d.docid
AND t.term = q.term
GROUP BY DOCID
ORDER BY 2;
```

SQL 2: OKAPI Probabilistic measure

In this SQL, df -- *document frequency*, is stored in the TERM relation, Document length, dl , is stored in the DOCUMENT relation, and the term frequency, tf , is stored in the INDEX relation.

3.3 Self-Relevance Probabilistic Model

A very different probabilistic approach is found in the self-relevance model of LL Kwok [6]. This measure was shown to be highly effective at

TREC-8 where it performed better than any other system [9]. In Kwok's work, like Robertson's, the weight for a given component is defined as the ratio of the probability of the component being relevant to the probability that it is not relevant:

$$w_{ak} = \ln\left(\frac{r_{ak}}{(1-r_{ak})}\right) + \ln\left(\frac{(1-s_{ak})}{s_{ak}}\right)$$

where r is the probability of relevance and s is the probability of nonrelevance. *Self-relevance* is used to generate initial estimates of probabilities. The concept is that each query is relevant to itself. So the probability of relevance, r and nonrelevance, s , are calculated as follows:

$$r_{ak} = \frac{tf_{ak}}{L_a} \quad s_{ak} = \frac{F_k}{N_w}$$

where:

L_a is the number of components (terms) in a document

F_k is the number of occurrences of term k in the collection, and

N_w is the number of distinct components (terms) in the collection.

Once the weights of the components are computed, they are combined to form the overall similarity measure between the document and the query. Kwok proposes two approaches to this – the query-focussed approach and the document-focussed approach. These are combined to obtain the final measure.

$$SC(Q, D_i) = \sum_{i=1}^k \left(\frac{tf_{ik}}{L_i} \right) w_{ik} + \sum_{i=1}^k \left(\frac{qtf_{ik}}{L_i} \right) w_{ik}$$

When the substitutions are made, the measure becomes that shown in Figure 4.

The following SQL will generate Kwok's probabilistic self-relevance measure of relevance.

$$RSV_{(q,d)} = \sum_k \left(\frac{qtf_k}{L_q} \right) * \log \left(\frac{tf_k}{L_d - tf_k} * \frac{N_w - L_d - F_k + tf_k}{F_k - tf_k} \right) + \sum_k \left(\frac{tf_k}{L_d} \right) * \log \left(\frac{qtf_k}{L_q - qtf_k} * \frac{N_w - F_k}{F_k} \right)$$

Figure 4: Self-Relevance Similarity Measure

```
SELECT Docid,
SUM(q.termfreq/q.termcnt *
LOG(((i.tf/(d.DocLen-i.tf) *
(Numwords - d.DocLen - t.sumtf + i.tf)
/ t.sumtf - i.tf))+ (i.tf/d.DocLen) *
LOG((q.termfreq/ (q.termcnt - q.termfreq) *
(Numwords - t.sumtf/t.sumtf)))
FROM INDEX i, TERM t, DOCUMENT d,
QUERY q
WHERE i.term = t.term
AND i.docid = d.docid
AND t.term = q.term
GROUP BY DOCID
ORDER BY 2;
```

SQL 3: KWOK Probabilistic measure

4 Performance and Portability

The standard structured query language described in our approach is portable across relational Database Management Systems. It has been implemented on Oracle, Sybase, MS Access, and Teradata DBMS's. In turn, these DBMS' are portable across various hardware and operating systems. We ran performance measurements on an implementation on a 4 Pentium Pro 200 Mhz CPU with 256MB RAM and a RAID-5 storage device, using Teradata Version 2, release 2. We ran 150 queries from from the TREC collection against the TREC6,7 and 8 document corpus consisting of half a million documents of LA Times, Financial Times, Federal Register and Foreign Broadcast Information Service (FBIS) documents. These documents make up 2GB of text data. We used the Title-only representation of the query, which consists of one to three terms describing the search. We used a variation of the Robertson similarity measure shown above and tuned some parameters using experimentation.

We achieved an average query run time of 2.07 seconds for the 150 queries. Query run time was highly correlated (0.8027) with the number of documents returned. The average number of documents returned for a query was 8,170. Figure 5 displays a histogram of the runtime for queries, rounded up to the nearest second. We also experimented with fifty longer 'description' queries. These had an average of 10.8 terms and

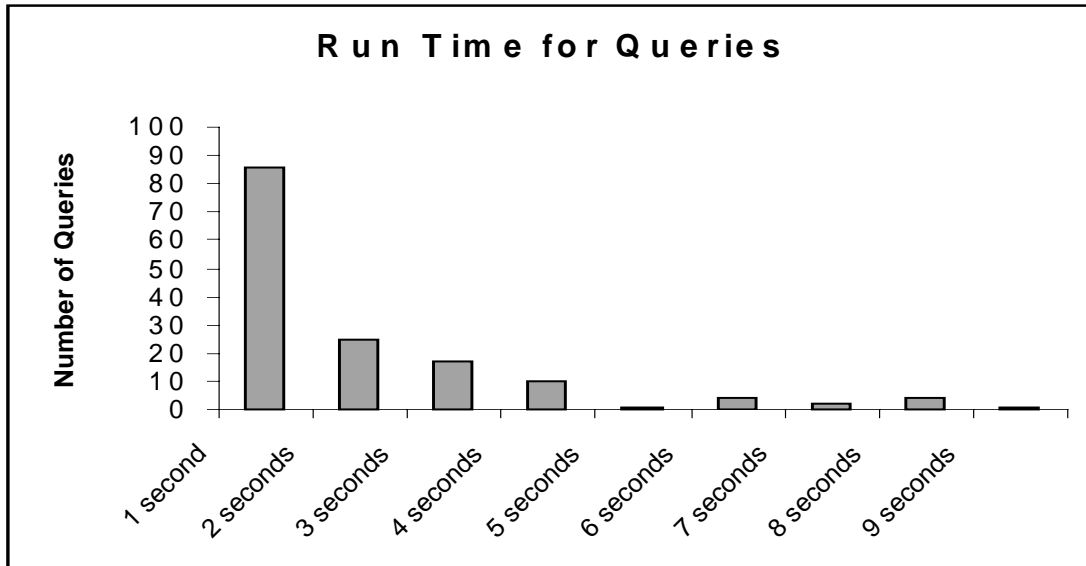


Figure 5: Performance of Parallel SQL for Information Retrieval

ran in an average time of 8.1 seconds. Performance tuning actions such as storing the query terms in memory were not conducted for these runs.

5 Conclusions and Future Work

We presented general relational algebra for the Information Retrieval Problem and show that it applies to state of the art retrieval strategies including vector space model and leading probabilistic models. The relational platform for Information Retrieval has been shown to parallelize well and we show good performance numbers for experimental runs of 150 queries against the 528,000 TREC8 documents, achieving subsecond query time on most queries. Future work includes performance tuning the application and exploration of combinations of retrieval strategies to improve retrieval effectiveness.

6 References

- [1] Grossman, D., O. Frieder, D. Holmes and D. Roberts, "Integrating Structured Data and Text: A Relational Approach," *Journal of the American Society for Information Science*, January 1997.
- [2] C. Lundquist, O. Frieder, D. Holmes, and D. Grossman, "A Parallel Relational Database Management System Approach to Relevance Feedback in Information Retrieval," *Journal of*

the American Society for Information Science, 50(5), April 1999

- [3] Singhal, A., C. Buckley, and M. Mitra, "Pivoted Document Length Normalization," *Proc. of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1996.
- [4] Sparck Jones, K. "Search Term Relevance Weighting given little Relevance Information," *Journal of Documentation*, v. 35, p 30-48, 1979.
- [5] Robertson S., S. Walker and M. Beaulieu, "Okapi at TREC-7: automatic ad hoc, filtering, VLC and interactive," *Proceedings of the 7th Text Retrieval Conference (TREC7)*, 1998.
- [6] Kwok, K. L. "A Network Approach to Probabilistic Information Retrieval." *ACM Journal of Transactions on Information Systems*. Vol. 13. 1995
- [7] Date, C.J. *An Introduction to Database Systems*, 6th edition. Addison-Wesley. 1995.
- [8] Grossman, D. *Integrating Structured Data and Text: A Relational Approach*. PhD Thesis, George Mason University. 1995
- [9] Harman, D., editor *Proceedings of The Third Text Retrieval Conference (TREC-8)*, sponsored by the National Institute of Standards and Technology and Advanced Research Projects Agency, 1999.