On the Integration of Structured Data and Text: *A Review of the SIRE Architecture*

(An Invited Overview)

Ophir Frieder, Abdur Chowdhury, David Grossman, & M. Catherine McCabe
Information Retrieval Laboratory
Illinois Institute of Technology
{frieder, chowdhury, grossman, mccabe}@ir.iit.edu

1.0 Introduction

Over the past decade, members of the Information Retrieval Lab have designed, developed, and deployed a variety of information retrieval systems. A central theme for all of our systems was the integration of structured data and text. One of our more recent efforts, SIRE, a Scalable Information Retrieval Engine [Grossman97, Grossman98, Lundquist99] is the focus of this paper. For completeness, I review some of the functionality of SIRE although it is described, in greater detail, in other forums. We describe the architecture of the prototype developed for the National Institutes of Health (NIH) National Center for Complementary and Alternative Medicine (NCCAM) [Frieder00] by some of the members of the laboratory. The version deployed at NCCAM is a more industrialized version of this prototype.

The mainstream approach in the development of information retrieval systems uses a customized inverted index to represent the text. SIRE, on the other hand, is a relational information retrieval approach and uses relations to model an inverted index. Storing the full text in a relational environment integrates the search of unstructured data with the traditional structured data search of Relational Database Management Systems (RDBMS). By using only standard SQL, SIRE leverages investment of the commercial relational database industry, while providing all capabilities of a more traditional information retrieval approach. RDBMS offer a wide variety of functionality such as concurrency control, recovery, security, portability, scalability, and robustness. RDBMS vendors continuously improve these features and incorporate advances made in hardware and software. Thus, an application using an RDBMS is able to keep up with the technology curve with less investment than a custom solution.

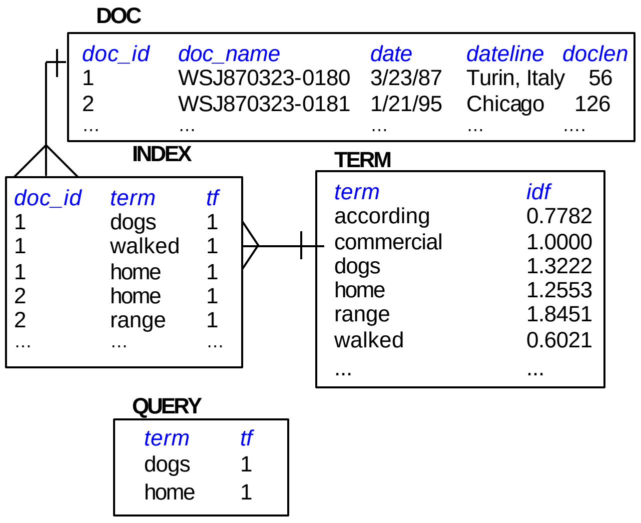
SIRE is implemented strictly as a relational database application. Key information retrieval techniques such as leading similarity measures, proximity searching, n-grams, passages, phrase indexing, and relevance feedback are all implemented using standard SQL. By adhering to strictly standard SQL, SIRE is completely portable across platforms and database management systems. Thus far, either laboratory members or our industrial collaborators or sponsors implemented SIRE on the NCR DBC-1012, Microsoft SQL Server, Sybase, Oracle, IBM DB2 and SQLD/S database management systems.

SIRE achieves good performance and scalability and is currently in production use in a variety of text search applications both commercially as part of multiple industrial efforts and in various government laboratories including the National Institutes of Health National Center for Complementary and Alternative Medicine. The relational platform offers equivalent capabilities to traditional Information Retrieval (IR) systems while providing a parallel, scalable, maintainable architecture with a unified platform for integrating searches of structured and unstructured data.

2.0 SIRE Functionality Overview

In SIRE, relations are used to model an inverted index, see Figure 1[Grossman97]. As shown, three relations, namely DOC, INDEX, and TERM, are used to represent the entire collection. The document (DOC) relation stores all the metadata information about the individual documents. That is, the DOC relation stores the document name, dateline, length, etc. The term (TERM) relation stores all terms in the collection and their corresponding inverted document frequency (idf) score. The example demonstrates only words as terms, but phrases can likewise be stored in the relation. Finally, the index (INDEX) relation itemizes which terms appear in which documents and how many times the term appears within each document.

Storing the full text in a relational environment integrates the search of unstructured data with the traditional RDBMS search of structured data. The retrieval accuracy of SIRE, as measured by the metrics *precision* and *recall*, is comparable with the state-of-the-art systems because leading scoring measures are


DOC

doc_id	doc_name	date	dateline	doclen
1	WSJ870323-0180	3/23/87	Turin, Italy	56
2	WSJ870323-0181	1/21/95	Chicago	126
...	...	...	...	....

INDEX

doc_id	term	tf
1	dogs	1
1	walked	1
1	home	1
2	home	1
2	range	1
...	...	...

TERM

term	idf
according	0.7782
commercial	1.0000
dogs	1.3222
home	1.2553
range	1.8451
walked	0.6021
...	...

QUERY

term	tf
dogs	1
home	1

Figure 1: The Relational Structure of the Scalable IR Engine

implemented in standard SQL within SIRE. An example of such SQL is shown in SQL-1, using the cosine similarity measure.

SQL-1:
```
SELECT          d.doc_name, SUM((i.tf * t.idf * q.tf * t.idf)/d.doclen)
FROM            Index i, Doc d, Query q, Term t
WHERE           d.doc_id = i.doc_id       AND
                q.term = i.term           AND
                t.term = q.term
GROUP BY        d.doc_id
ORDER BY 2 DESC;
```

Modifications to the SUM() element permits implementation of mo st leading similarity measures. For instance, with the additional computation and storage of some document statistics, (log of the average term frequency), some collection statistics (average document length and the number of documents) and term statistic s (document frequency), the following measures are implemented. In the pivoted normalization measure, the constant 0.20 is the pivot value proposed in [Singhal96]. In the OKAPI measure, the constants are the values described in [Robertson98].

SQL-2: Pivoted normalization measure
```
      SUM(((1 + LOG(i.tf)) / ((d.LogAvgTF) * (AvgDocLen + (0.20 * d.doclen))))
                * (t.idf * ((1 + LOG(q.tf)) / (q.LogAvgTF))))
```

SQL-3: OKAPI Probabilistic measure
```
      SUM(LOG((((NumDocs - t.df) + 0.5) / (t.df + 0.5))
                *  ((2.2*i.tf) / (.3 + ((.9 * d.doclen)/AvgDocLen) + i.tf))))
```

Techniques such as relevance feedback, proximity, phrases, and n-grams are common in traditional information retrieval and are therefore also available in the SIRE system. For brevity, we only discuss relevance feedback, as it is the more commonly used information retrieval technique. To implement relevance feedback, the SQL shown is SQL-1 is executed and the best (using N*nidf ranking) terms from the top-ranked documents are added to the original query terms (INSERTED into the Query table) and the query is rerun [Lundquist99].

term	doc_id	tf-idf
dogs	{(1	1.07),
	(7	3.02),
	(19	2.09),
	(44	1.76),
	(45	1.99),
	(63	2.65)}

(A) Term Processing

doc_id	term	tf-idf
1	{(dogs	1.07),
	(walked	3.33),
	(home	2.23),
	(cats	1.58),
	(climbed	1.36),
	(trees	2.11)}

(B) Relevance Feedback Processing

Figure 2. Clustered Indexes

The vector space model that weights terms according to how unique they are within the collection. However, some applications rely either on just the Boolean model or on a combination of both the vector space model and the Boolean model. In the Boolean, one operator TAND (threshold AND) is used to require a certain number of the specified query terms from within the entire query terms to be present in a document for it to qualify as relevant. This feature can be quite complex to implement in traditional information retrieval systems. However, with SQL, TAND is easily achieved by adding a HAVING COUNT(*) >= *threshold_number_of_terms*.

SQL-4:
```
SELECT      d.DocName
FROM        Index i, Doc d, Query q
WHERE       d.Docid = i.Docid        AND
            q.term = i.term
GROUP BY    d.Docid
HAVING COUNT(*) >= threshold_number_of_terms;
```

3.0 Performance Enhancements

The primary performance enhancements supported in SIRE are:

- Clustered Indexes
- Thresholding
    - Index Thresholding
    - Query Thresholding

3.1 Clustered Indexes

Physically implementing the relational structure of SIRE as illustrated in Figure 1 introduces redundancy due to the repetition of the *doc_id* attribute in the RDBMS indexing of the INDEX relation. Furthermore, for relevance feedback processing, the same duplication occurs in the indexing of the *term* attribute. This redundancy increases the storage requirements, hence the I/O processing, and therefore the total processing time. To nullify this situation, the actual SIRE implementation uses clustered indexes as shown in Figure 2. As seen, the redundancy is eliminated. As an aside, also note that instead of the term frequency value, the computed tf-idf value is stored. This reduces the computational time as repeated tf * idf computations are eliminated.

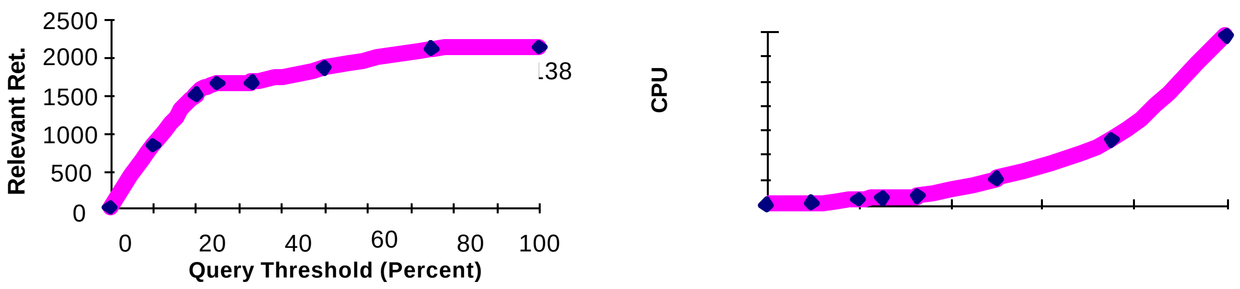


Figure 3. Relevant Retrieved and Computation Time as a Function of Query Threshold

3.2 Thresholding

Thresholding is a technique where terms in certain frequency ranges are eliminated from the search – either removing them from the query or, equivalently, eliminating them from the index, saving space and processing speed. Index Thresholding addresses the possibility that the large term weights of the *infrequently* occurring terms may be artificially inflating the relevance ranking scores of documents. Our experiments with TREC data indicate that eliminating terms occurring in fewer than 75 documents (.014% of the collection) improves the precision/recall by 24%, reduces storage required by 26% and improves runtime

In Query Thresholding, the most *frequent* terms are removed from the query since they are the worst discriminators. In Figure 3, we show that a query threshold of 75% maintains a precision/recall level approximating that obtained with 100% of the query terms but significantly reduces run time for a broad range of queries.

4.0 System Software Architecture

The SIRE software architecture comprises of four parts:

- System Software: Operating Systems, Disk Striping
- Relational Database Management Systems
- Java servlets
- Web server

The system software deals with the operating system layout, namely, the distribution of disk resources, optimization of I/O throughput, memory usage, disk striping to reduce I/O contention, etc. The second components are the use of an RDBMS to store structured and unstructured information. The third and fourth component is the query software architecture, caching algorithms, and other optimizations for efficiency and a web server providing the UI (User Interface) to the client via a traditional web interface, respectively.

4.1 System Software Architecture

The SIRE system uses Solaris 2.7 as its operating system providing the interface to the underlying hardware. The system is running on a Sun ES-450 with four 300MHz processors and 4 Gigabytes of system memory. The system has approximately 300 gigabytes of disk space connected to three SCSI controllers to distribute the I/O load for each controller. To reduce I/O contention for large data files, we use Disk Suit 4.2 to build logical disks that stripe the file system over several controllers and disks in 16K blocks. The disk layout for the data files is described in the next section.

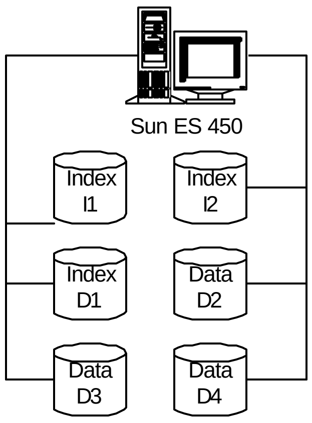


Figure 4: RDBMS Disk Layout

4.2 RDBMS Software Architecture

The SIRE system uses a RDBMS to store data. We currently use Oracle 8.0.5 as the RDBMS. The SIRE system uses indexes to speed up the retrieval of information for queries. We use logical disks to reduce our I/O contention when running multiple queries at the same time as shown in Figure 4. The indexes for each table are striped across two controllers and over two physical disks, I1 and I2. The data files are striped across two controllers and four disks, D1-D4. We model the vector space model to search unstructured text and provide relevance rankings. We search structured information with standard RDBMS techniques.

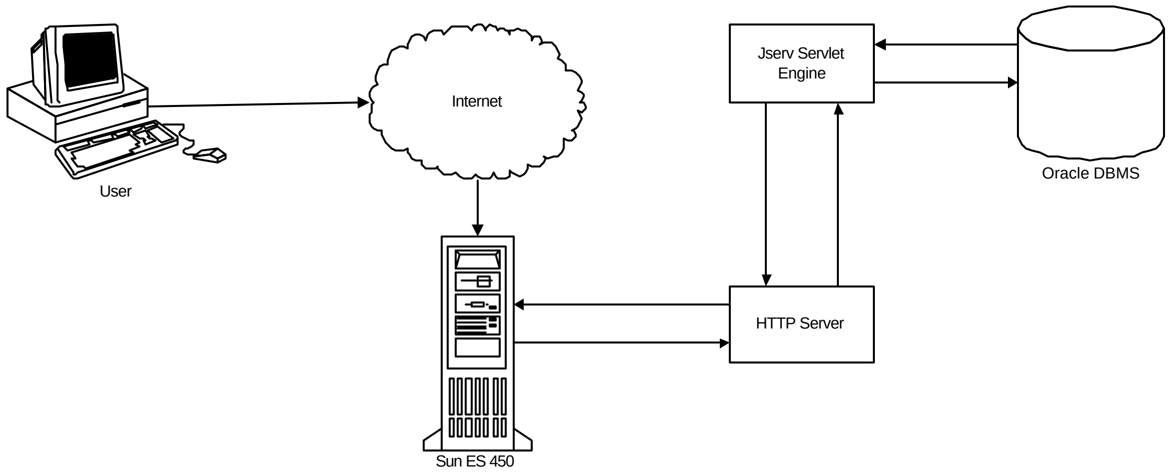


Figure 5. Client Request Flow Chart

4.3 Query Software Architecture

The overall client request flow is illustrated in Figure 5. A user contacts the server via a web browser. The web server provides static information via standard HTML pages. Information requests are static html pages with forms. The form is filled out by the client and sent back to the server via an http get message. The HTTP server passes the message to the Servlet Engine. The Servlet Engine builds a SQL request to

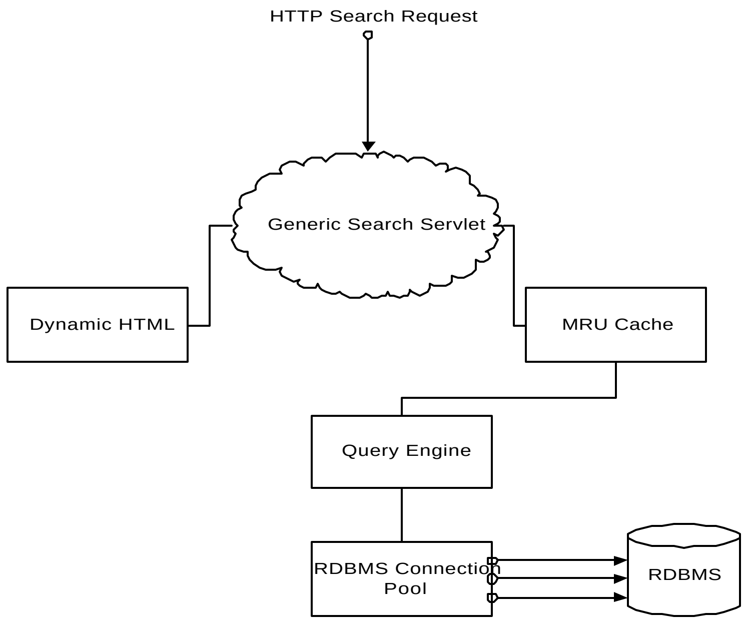


Figure 6. Servlet Flow Chart

the RDBMS. The servlet then queries the database for the requested data and gets the results. The servlet builds a result page and passes it back to the client's browser for display.

As more and more clients access the system, performance becomes a greater and greater issue. To optimize the request process several engineering optimizations were made. As requests come into the servlet, the servlet builds a SQL statement to get the needed information from the RDBMS. That request is passed on to a MRU (Most Recently Used) cache engine. The MRU cache keeps the most recently requested queries in memory. If the current request was previously made and its results are still in the cache, the results are returned to the servlet to be processed into an html results page. If the results are not found the query is passed down to the query engine. The query engine keeps a cache of database connections to speed up the request time. A pre-cached connection is pulled from the cache and the request is sent to the database. The results are returned to the query engine. The engine passes the results up to the MRU cache. The MRU cache stores the results in memory and returns a copy to the servlet. If the same request is made again, the results can be returned right from the MRU cache without going to the RDBMS. After the servlet has received the results, a dynamic HTML page is created. The formatted results are then returned to the client's browser for display. This query processing is illustrated in Figure 6.

5.0 Parallelism

We have investigated the use of a parallel database engine to support the SIRE backend. Experimental evaluation results presented in [Lundquist99] demonstrated that nearly a 92% parallel efficiency was achieved using an NCR Teradata database machine with 24 nodes. That is, roughly a 22-fold speed-up was observed as compared to a single node system.

6.0 Conclusions

We overviewed an implementation SIRE and a corresponding performance analysis. Experimental results demonstrated that SIRE supports a high degree of scalability. The relational environment offers a single platform for the integration of searches across structured and unstructured data. In addition, it has the advantages of portability, scalability and leverages advances of the RDBMS industry.

Acknowledgments

This entire effort would not be possible without the contributions of the entire Information Retrieval Laboratory members. Special additional thanks are extended to Nazli Goharian for her commentary that improved the clarity of this paper. This work was supported in part by grants from the National Science Foundation, the National Institutes of Health, the US Army, and the IIT Research Institute.

References

[Frieder00]	Frieder, O., et. al., "On the Development of the NIH Complementary and Alternative Medicine Digital Library,", www.ir.iit.edu/publications/2000/NCCAM.pdf. *Also available off the CAM Citation Index link of http://nccam.nih.gov/nccam/databases.html.*
[Grossman97]	Grossman, D., O. Frieder, D. Holmes, and D. Roberts, "Integrating Structured Data and Text: A Relational Approach," JASIS, 48(2), February 1997.
[Grossman98]	Grossman, D. and O. Frieder, Information Retrieval: Algorithms and Heuristics, Kluwer Academic Publishers, ISBN 0-7923-8271-4, 1998.
[Lundquist99]	Lundquist, C., O. Frieder, D. Holmes, and D. Grossman, "A Parallel Relational DBMS Approach to Relevance Feedback in IR," JASIS, 50(5), April 1999
[Robertson98]	Robertson S., S. Walker and M. Beaulieu, "Okapi at TREC-7: automatic ad hoc, filtering, VLC and interactive," Proceedings of TREC 7, 1998.
[Singhal96]	Singhal, A., C. Buckley and M. Mitra, "Pivoted Document Length Normalization," Proceedings of Nineteenth SIGIR, 1996.