A Survey on Text Access Methods

Muhammad Rafi[1] and Aslam Parvez Memon
muhammad.rafi@nu.edu.pk, aslam@szabist.edu.pk
SZABIST
Karachi, Pakistan

Abstract: *Text is a general form of information created and stored by computer systems. Special purpose algorithms are required to retrieve, interpret, manipulate and store text repositories. The importance of text retrieval systems has grown dramatically during the recent years due to increase in text based information systems–WWW, Databases, XML, and Document collections. Full text searching, signature files, and inverted indexing are in practice for text retrieval. These methods have failed to give optimized results especially in the case of large compressed/uncompressed text repositories. This study evaluates key features of the above-stated text retrieval methods and develops a comparison matrix for the future. This study also proposes ideal text retrieval systems, which address the deficiencies and shortfall of the existing methods.*

Keywords: *Text processing, indexing, information storage and retrieval*

1. INTRODUCTION

Information retrieval has been facing a new challenge raised by the emergence of massive, complex structural data in the form of large text collections whether in the database or in documents. Major factors that are critical in text retrieval system are search engines and underlying indexing techniques. The process of query evaluation and indexing in high performance text retrieval systems usually consists of several steps. The following steps are typically carried out:

- Query pre-processing, e.g. some kind of linguistic normalization of words used in the query, extension of query with synonyms checking against thesaurus
- Launches of search engines based on pre-build indices
- Post processing of candidate documents/record to filter out set of result relevant to query
- Refinement of the query, based on users' feedback, and reevaluation of the query

All these steps are important for quality of the result, and are very time consuming and computational extensive. Researcher tries to optimize the overall equation by simply tuning up either step of this process. It becomes further complicated when we try to apply these techniques for databases or on document collection, because they both model information in very different ways.

Database vs Document Collection

It is becoming increasingly common practice to store large collection of text as text fields (column) in the table. Database systems normally store and manipulate fixed length formatted records. Relational database theorists like to talk about the "meaning" or "semantics" of data as being in the database (specifically its metadata, and more specifically its constraints). The standard way to manage text is via a full-text index, designed as follows: for hundreds of thousands of words; the index maintains a list of which documents the word appears in, and at which positions in the document it appears. This is a columnar, memory-centric approach, that doesn't work well with the architecture of mainstream relational products. Text search can be carried out against many different kinds of things. One increasingly useful target is the table of a relational database. A standard SQL query might have trouble finding all the references in a whole database to a particular customer organization, product line or whatever; a text search can do a better job. This kind of use is becoming increasingly frequent and easily supported by this approach.

A Document Collection is completely different from a DBMS, as each document is an independent entity and it contains a stream of text within. Several document standards exist and most of them keep some secret header to support their application specific text processing. Search machines parse text files and store associations of lexemes (words) with their parent document. Later, these associations are used to search documents, which contain query words. The process of document indexing usually does parsing of lexemes and it is useful to distinguish various kinds of lexemes; e.g. digits, words, complex words, email address; since different types of lexemes can be processed differently. In principle, the actual types of lexemes depend on specific applications, but for plain search it is desirable to have pre-defined common types of lexemes. Apply linguistic rules to normalize the lexeme to their infinitive form, so one should not bother entering search word in specific form. Taking into account the type of lexeme obtained before, provides rich possibilities for normalization. Store pre-processed document in a way, optimized for searching; for example, represent document

[1] Muhammad Rafi presently working as Lecturer at National University of Computer & Emerging Sciences, and registered as PhD student at SZABIST, Karachi Campus

as a sorted array of lexemes. Along with lexemes itself, it is desirable to store positional information to use it for proximity ranking, so that the document which contains more "dense" region with query words is assigned a higher rank than one with query words scattered all over.

Ordinary full text search engines operate with a collection of documents, where the document is considered as a "bag of words"; i.e., there is minimal knowledge about the document structure and its metadata. Moreover, modern information systems are all database driven and there is a need in IR (Information Retrieval) style full text search inside database, with full conformance to the database principles (ACID). That is why many databases have built-in full text search engines, which allow combining text searching and additional metadata, stored in various tables and available through powerful and standard SQL language. This paper considers all those methods that are equally good for both of the documents/databases text collections.

Problems related to text retrieval

In the preceding section, we have shown that the two word documents/databases are now becoming enriched of text-based information. There are several problems that must be addressed in order to build ideal text retrieval systems. The development of effective retrieval techniques has been the core of IR research for more than 30 years. A number of measures of effectiveness have been proposed, but the most frequently mentioned are recall and precision. Finding text that satisfies a user's information need is not simple, and considerable progress has been made in developing ranking techniques that are significantly more effective than Boolean logic. Information routing, filtering and clipping are all synonyms used to describe the process of identifying relevant documents in streams of information, such as news feeds. Instead of comparing a single query to large numbers of archived documents, as is the case for IR, large numbers of archived profiles are compared to individual documents. Documents that match are sent to the users associated with the profile. A profile is a representation of a long-term information need and is usually more complex than a session-based query.

Effective interfaces for text-based information systems are a high priority for users of these systems. The interface is a major part of how a system is evaluated, and as the retrieval and routing algorithms become more complex to improve recall and precision, more stress is placed on the design of interfaces that make the system easy to use and understandable. Interfaces must support a range of functions including query formulation, presentation of retrieved information, feedback, and browsing. The challenge is to present in this sophisticated functionality in a conceptually simple way. One of the major causes of failures in IR systems is vocabulary mismatch. This means that the information need is often described using different words than are found in relevant documents. Techniques that address this problem by automatic expansion of the query are often regarded as a form of "magic" by users and are viewed as highly desirable. Vocabulary expansion can result from transforming the document and query representations, as with Latent Semantic Indexing, or it can be done as a form of automatic thesaurus built by corpus analysis. Further research in this area will make these techniques more reliable and efficient.

Text retrieval techniques can be used to solve part of an organization's information management problems. Typically, a complete solution requires other text-based tools such as routing and extraction, tools for handling multimedia and scanned documents; such as OCR, a database management system for structured data, and workflow or other groupware systems for managing documents and their use in the organization.

In past, there were three principal indexing methods—full-text searching, inverted files and signature files—that have been proposed for large text databases/document collection. They remain the subject of active research. The question related to which method is better for the current time, where processing power, storage and information gain has been exponentially changed, remain unanswered. This paper is organized as follows: in Section 2, we discuss the previous suggested techniques that fall in the above three methods; in Section 3, we compare and contrast the method for large collection of text in both database/document base repository; Section 4 provides the recommendation and advice for designing and implementing text retrieval system; Section 5 presents our conclusion and future work.

2. PRELIMINARIES

The three methods that were previously used for text retrieval systems [1] are full-text searching, inverted index and signature files. All these methods have been in active research for the last decade and a large number of their derivatives have been suggested and implemented. We start our discussion with Full-Text Searching.

2.1 Full-Text Searching

Full-text search (also called free search text) refers to a technique for searching computer stored documents or databases to retrieve a user given pattern (query text). The search examines all of the text presented in the document/database repository. Full-text searching techniques became common in online bibliographic databases in the 1970s. Most websites, search engines and application programs (such as word processing software) provide full-text search capabilities. The most common approach to full-text search is to generate a complete index or concordance for all the searchable documents or database [4], [5], [6]. For each word (excepting stop words, which are too common to be useful) an entry is

made which lists the exact position of every occurrence of it within the database or document. Although very small document/database full-text searching can be done by serial scanning, indexing is the preferred method for almost all full-text searching. There are currently many problems associated with full-text; some of which are false positive, precision and recall. These problems are due to the fact that text is ubiquitous. The improvement in full-text can be done by improving indexing techniques, collecting keywords and text compression that can support searching. Full-text search also demands the support of field-restricted search, phrase search, proximity search and regular expression. Other improvements can be introduced by making search algorithms more robust, efficient and relevant.

2.2 Inverted Files

An inverted file index [2], [6], [7], has two main parts: a search structure or vocabulary, containing all of the distinct values being indexed; and for each distinct value an inverted list, storing the identifiers of the records containing the value. Queries are evaluated by fetching the inverted lists for the query terms, and then intersecting them for conjunctive queries and merging them for disjunctive queries. To minimize buffer space requirements, inverted lists should be fetched in order of increasing length; thus, in a conjunctive query, the initial set of candidate answers are the records in the shortest inverted list, and processing of subsequent lists only reduces the size of this set. Once the inverted lists have been processed, the record identifiers must be mapped to physical record addresses. This is achieved with an address table, which can be stored in memory or on disk. An effective structure for storing vocabularies is a B1-tree. The high branching factor, typical of these trees, means that the internal nodes are only a small percentage of the total vocabulary size. For example, suppose that in a B1-tree leaves contain pointers to inverted lists, that the vocabulary of some database contains 1,000,000 distinct 12-byte terms, and that the disk being used operates with 8-kilobyte blocks and 4-byte pointers. Then, at most 64 kilobytes are required for the internal nodes. Given this much memory, at most one disk access is required to fetch a vocabulary entry. Since the exact address of the inverted list is then known, a second access suffices to retrieve the corresponding inverted list. Other structures that are suitable for storing vocabularies include arrays and hash tables, with comparable performance.

2.3 Signature Files

In signature files [3], [8], [9], the documents are stored sequentially in the "text file". Their abstractions are stored sequentially in the "signature file". When a query arrives, the signature file is scanned sequentially and a large number of non-qualifying documents are discarded. The rest are either checked (so that the "false drops" are discarded) or they are returned to the user as they are. A document is called a "false drop" if it does not actually qualify in a query, while its signature indicates the opposite. The method is faster than full text scanning, mainly, because the size of the signature file is much smaller.

However, it is expected to be slower than inversion for large files [8]. It requires much smaller space overhead than inversion. If carefully designed, the signature file method can handle queries on part of words and can tolerate errors. One of the difficulties in the comparison of inverted files and signature files is that many variants of signature file techniques have been proposed, and it is possible that some combination of parameters and variants will result in a better method.

2.4 Compressed Inverted vs Compressed Signature Files

The large collection of text generally favors the compression techniques. The inverted lists, themselves, are sequences of record identifiers, sorted to allow fast query evaluation. Sorting of identifiers within inverted lists has another important benefit: the identifiers can be represented using variable-length codes that, for large text databases, compress the index by a factor of about 6 to around 5 to 10% of the data size. This approach has the disadvantage that inverted lists must be decoded as they are retrieved, but such decompression can be fast. An interesting feature of compressed inverted lists is that the best compression is achieved for the longest lists, i.e. the most frequent terms. In the limit (which, in the case of text indexing, is a term such as "the" that occurs in almost every record) at most one bit per record is required.

Compression brings considerable benefit to inverted file indexes, and it is natural to ask if the same improvements can be achieved with signature files. The answer is no. One of the difficulties in the comparison of inverted files and signature files is that many variants of signature file techniques have been proposed, and it is possible that some combination of parameters and variants will result in a better method. But we believe that the methods considered here are at least as good as the best signature file techniques, and are fair representatives.

3. COMPARISON OF TECHNIQUES

The main objective of this study is to outline the tradeoff that can be applied to large text retrieval systems. We have carried out some experiments on different dataset. Let us first agree with our experiment system. The whole system consists of a Centrino 1.3 MHtz Processor, with 512MB RAM and 80GB Hard disk. Our dataset is classified into two different categories: Database and Document base. There are three bags of data: 100MB, 500MB and 1GB. We first apply full-text searching techniques for the random query from database and document base. In database, we have not noticed any significant performance change while document base

system really supports signature file at this level. Similarly, when we execute the same experiment with a data set of 500MB, we came to know that both database and document base are very similar in performance; the result suggests that at this level, it is really insignificant to talk about information platform. The third level of experiment is carried out with a data set of 1GB; it has been observed that for such a large collection it really does matters which techniques you should use.

4. RESULTS

We first present our experimental results as an overall picture of dataset and time to retrieval. The following result demonstrated that at 100MB of document base, all the techniques seem to produce the same time. It is because all the algorithms at this level easily tradeoff the computational requirement. Hence, it is insignificant to discuss the performance of document base at this level.

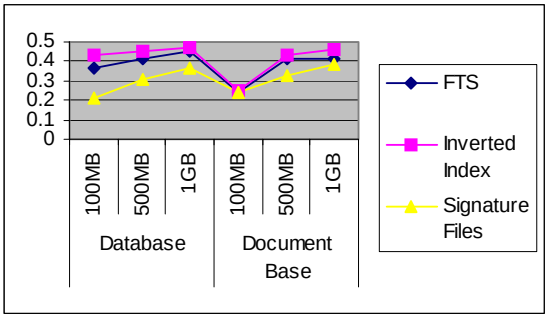

Fig. 1. Results of dataset retrieval response time

Next result of our experiment suggests that, as we increase the data set for our experiment, the performance ratio of inverted index remains the same. Hence, it is a good candidate for implementing text retrieval in large collection of both database and document collection.

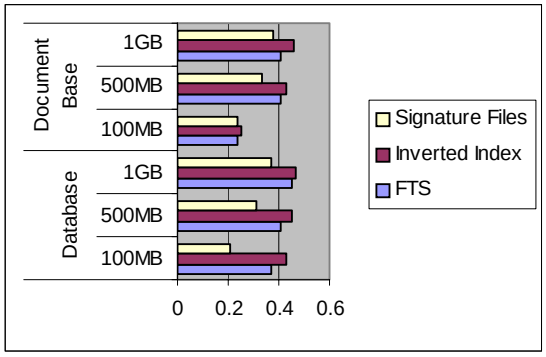

Fig. 2. Results of dataset size vs. performance

5. CONCLUSION

Our conclusions are unequivocal. For typical document indexing applications, current signature file techniques do not perform well compared to current implementations of inverted file indexes. Signature files are much larger; they are more expensive to build and update; they require a variety of parameters to be fixed in advance, involving analysis of the data and tuning for expected queries; they do not support proximity queries other than adjacency; they support ranked queries only with difficulty; they are expensive for disjunctive queries; they are highly intolerant of range in document length; their response time is unpredictable; they do not allow easy addition of functionality; they do not scale well; and, most important of all, they are slow. Even on queries expressly designed to favor them, signature files are slower than inverted files. The current trends in computer technology, in which the ratio of processor speed to disk access time is increasing, further favor inverted files.

ACKNOWLEDGMENT

This work was done for the partial fulfillment of requirement in the course of Independent Study, for which the author is registered as a full-time Ph.D student. Author would like to thank course supervisor, Aslam Parvez Memon, for his valuable advise and guidance throughout the semester in completing this work.

REFERENCES

[1] R.L. Haskin. "*Special Purpose Processor for Text Retrieval*", Database Engineering, 4 (1): pages 16-25, 1981.

[2] A.F. Cardenas. "Analysis and Performance of Inverted Data Base Structures", Communication of the ACM, 18 (5): pages 253-263, 1975.

[3] J. Zobel, Alistair Moffat and Kotagiri. "Inverted Files Versus Signature Files for Text Indexing", ACM Transaction on Database Systems, 23 (4): pages 453-490, 1998.

[4] Bird, R. M., Newsbaum, J. B., and Trefftzs, J. L. 1978. Text File Inversion: An evaluation. In Proceedings of the 4th Workshop on Computer Architecture for Non-Numeric Processing. Blue Mountain Lake, NY, ACM Press, pages 42–50.

[5] Couvreur, T. R., Benzel, R. N., Miller, S. F., Zeitler, D. N., Lee, D. L., Singhal, M., Shivaratri, N., and Wong, W. Y. P. 1994. An analysis of performance and cost factors in searching large text databases using parallel search systems. J. Amer. Soc. Inform. Science 45, 7, pages 443–464.

[6] Faloutsos, C. and Oard, D. W. 1995. A survey of information retrieval and filtering methods. Tech. rep., University of Maryland Institute for Advanced Computer Studies Report, University of Maryland at College Park, MD.

[7] Faloutsos, C. and S. Chrtstodoulakis. “Signature Files: An Access Method for Documents and its Analytical Performance Evaluation”, ACM Trans. on Office & formation Systems, vol. 2, no. 4, Oct. 1984.

[8] Van-Ripbergen, CJ., Information Retrieval, Butterworths, London, England, 1979. 2nd edition.

[9] J. Zobel, Alistair Moffat and Kotagiri. “Guidelines for Presentation and Comparison of Indexing Techniques”, SIGMOD Record, 25 (3): 10-15, 1996.