Indexing Time vs. Query Time Trade-offs in Dynamic Information Retrieval Systems

Stefan Büttcher and Charles L. A. Clarke
University of Waterloo, Canada
{sbuettch, claclark}@plg.uwaterloo.ca

ABSTRACT

We examine issues in the design of fully dynamic information retrieval systems supporting both document insertions and deletions. The two main components of such a system, index maintenance and query processing, affect each other, as high query performance is usually paid for by additional work during update operations. Two aspects of the system – incremental updates and garbage collection for delayed document deletions – are discussed, with a focus on the respective indexing vs. query performance trade-offs. Depending on the relative number of queries and update operations, different strategies lead to optimal overall performance.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Textual databases; H.3.4 [**Systems and Software**]: Performance evaluation

General Terms

Experimentation, Performance

1. INTRODUCTION

Although they are usually studied independently, indexing and query processing are two closely related topics in text retrieval. Many query optimization techniques necessitate additional work at indexing time and thus represent trade-offs between indexing and query processing performance. While this may be ignored in traditional (static) retrieval systems, it is critical in dynamic search environments, in which the underlying text collection is continuously changing and the number of queries to be processed may vary greatly. An example is file system search, where several thousand index updates per day are not unusual [2]. Index updates include two types of update operations: document insertions and document deletions.

Techniques to support document insertions into an existing index usually follow a standard scheme: Two indices are maintained, one in memory, the other on disk. Postings for new documents are accumulated in main memory until it is exhausted, at which point they are transferred to disk and combined with existing on-disk data. This operation can be performed by following an in-place update scheme [4] or by merging the old index with the new data, resulting in a new index that supercedes the old one [3]. Both approaches have in common that the entire on-disk index has to be read/written every time main memory is exhausted, causing performance problems for large collections (*In-place* does not need to read the whole collection. However, it trades read operations for disk seeks, leading to the same type of problem). We show how, by allowing multiple on-disk indices at the same time, the number of disk operations can be significantly reduced.

A thorough evaluation of techniques for document deletions has not been published. Moreover, a general discussion of the trade-offs associated with index maintenance and query optimization techniques does not exist.

2. SUB-INDEX MERGING

We discuss three different strategies to merge sub-indices, representing different trade-off levels.

Strategy 1: Immediate Merge

The first merge strategy has been proposed by Lester et al. [3]. The indexing system maintains one on-disk and one in-memory index. As soon as main memory is full, the in-memory postings are merged with the existing on-disk index, creating a new index. The old index is deleted. This strategy minimizes the number of disk seeks necessary to fetch a posting list. Its disadvantage is that for every merge operation the entire index has to be scanned. Thus, the number of disk operations necessary to index the whole collection is quadratic in the size of the text collection.

Strategy 2: No Merge

The second strategy does not perform any merge operations. When memory is full, postings are sorted and written to disk, creating a new on-disk sub-index. On-disk indices are never merged. When the posting list for a given term has to be retrieved from the index, sub-lists are fetched from all sub-indices. The advantage of No Merge is its high indexing performance (linear number of disk operations). Its disadvantage is that fetching a posting list requires $\Omega(n)$ disk seeks, where n is the size of the text collection.

Strategy 3: Logarithmic Merge

The two strategies described so far represent the two extremes. The third strategy is a compromise: A newly created on-disk sub-index is sometimes merged with an existing

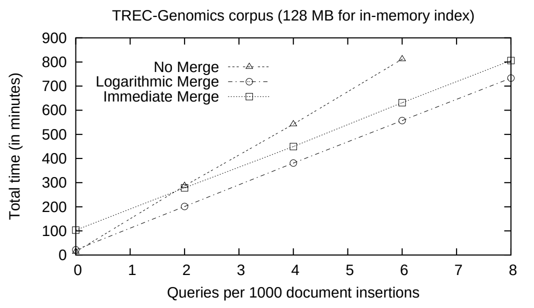

Figure 1: Comparison of sub-index merging strategies for a growing text collection and varying $\mathcal{D}_U^Q$.

one, but not always. We use the concept of *index generation* to decide when to merge sub-indices. An on-disk index that was created directly from in-memory postings is of generation 0. An index that is the result of a merge operation is of generation $g+1$, where g is the highest generation of any input index. If, after creating a new on-disk index, there are two indices of the same generation, they are merged. This is repeated until there are no more such collisions. The number of sub-indices is bounded by $O(\log(n))$, and the total number of disk operations necessary to index the text collection is $O(n \cdot \log(n))$, where n is the size of the collection.

3. GARBAGE COLLECTION

Document deletions are addressed by a garbage collection approach. Postings that belong to deleted documents are filtered and ignored during query processing, using a technique similar to the invalidation scheme proposed by Chiueh and Huang [1]. However, this is expensive in terms of time and space. Therefore, at some point, the garbage collector is started and removes all garbage postings from the index.

Threshold-based Garbage Collection

A simple strategy is to keep track of the relative number of postings in the index that belong to deleted documents:

$$r = \frac{\#\text{deleted postings}}{\#\text{postings}}.$$

As soon as this number exceeds a predefined threshold ρ, the garbage collector is started. $\rho = 0$ guarantees maximum query performance. For $\rho = 1$, index maintenance performance is maximal, but the query processor spends a great amount of time fetching and decompressing postings that belong to deleted documents.

On-the-Fly Garbage Collection

The threshold-based garbage collection strategy descibed above has the disadvantage that it is independent of the sub-index merging strategy employed and thus causes additional disk operations that could have been avoided by integrating the garbage collector into the merge process. We call this integration *on-the-fly garbage collection*: If the combined relative amount of garbage postings in all sub-indices involved in a merge operation exceeds a threshold ρ', the garbage collector is integrated into the merge process.

4. EXPERIMENTAL RESULTS

The retrieval system used in our experiments is the Wumpus search engine. As the underlying text collection, we used the TREC 2004 Genomics corpus, consisting of 4.5 million documents with a total size of 14 GB.

In order to be able discuss indexing time vs. query time trade-offs with respect to the amount of work the indexing subsystem and the query processor have to do, the experiments were conducted with varying relative query and update loads. This ratio is expressed by $\mathcal{D}_Q^U$, the number of update operations per query, and its pendant $\mathcal{D}_U^Q$, queries per update operation. A system with $\mathcal{D}_Q^U = 0$ processes queries for a static text collection. $\mathcal{D}_Q^U = \infty$, on the other hand, describes a system which only performs update operations and never processes any queries. We conducted experiments for $\mathcal{D}_Q^U$ values realistic in file system search.

The first series of experiments, depicted in Figure 1, involves a monotonically growing collection. The whole collection is indexed, with queries being processed concurrently. It can be seen that Logarithmic Merge is the best strategy for a wide range of relative update/query loads. Only for very small $\mathcal{D}_U^Q$, No Merge gives better overall performance.

In the second series of experiments, the system started from an index containing 50% of the collection. Documents were added and deleted randomly. Figure 2 shows that a garbage threshold $\rho = 0.5$ is a good choice under most circumstances. On-the-fly garbage collection increases performance slightly and gives a 3% improvement over the pure threshold-based strategy.

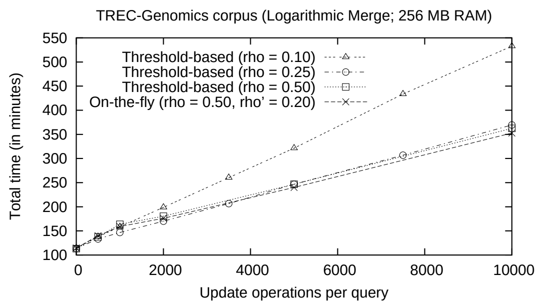

Figure 2: Comparison of garbage collection strategies for a fully dynamic collection and varying $\mathcal{D}_Q^U$.

5. ADDITIONAL RESOURCES

The Wumpus search system, along with technical reports giving more details regarding the issues discussed in this paper, can be found on-line: http://www.wumpus-search.org/.

6. REFERENCES

[1] T. Chiueh and L. Huang. Efficient Real-Time Index Updates in Text Retrieval Systems. Technical report, Stony Brook, New York, USA, August 1998.

[2] T. J. Gibson and E. L. Miller. Long-Term File Activity Patterns in a UNIX Workstation Environment. In *Proceedings of the 15th IEEE Symposium on Mass Storage Systems*, pages 355–371, March 1998.

[3] N. Lester, J. Zobel, and H. E. Williams. In-Place versus Re-Build versus Re-Merge: Index Maintenance Strategies for Text Retrieval Systems. In *Proceedings of the 27th Conference on Australasian Computer Science*, pages 15–23, Darlinghurst, Australia, 2004.

[4] A. Tomasic, H. García-Molina, and K. Shoens. Incremental Updates of Inverted Lists for Text Document Retrieval. In *Proceedings of the 1994 ACM SIGMOD Conference*, pages 289–300, New York, 1994.