Optimized Query Execution in Large Search Engines with Global Page Ordering*

Xiaohui Long **Torsten Suel**

CIS Department
Polytechnic University
Brooklyn, NY 11201
{xlong@cis.poly.edu, suel@poly.edu}

Abstract

Large web search engines have to answer thousands of queries per second with interactive response times. A major factor in the cost of executing a query is given by the lengths of the inverted lists for the query terms, which increase with the size of the document collection and are often in the range of many megabytes. To address this issue, IR and database researchers have proposed pruning techniques that compute or approximate term-based ranking functions without scanning over the full inverted lists.

Over the last few years, search engines have incorporated new types of ranking techniques that exploit aspects such as the hyperlink structure of the web or the popularity of a page to obtain improved results. We focus on the question of how such techniques can be efficiently integrated into query processing. In particular, we study pruning techniques for query execution in large engines in the case where we have a global ranking of pages, as provided by Pagerank or any other method, in addition to the standard term-based approach. We describe pruning schemes for this case and evaluate their efficiency on an experimental cluster-based search engine with 120 million web pages. Our results show that there is significant potential benefit in such techniques.

1 Introduction

Over the last decade, the Web has grown from a few thousand to its present size of several billion pages. Due to this explosion in size, users increasingly depend on web search engines for locating relevant information. Given the large number of web pages on most topics of interest, one of the main challenges for a search engine is to provide a good ranking function that can identify the most useful results from among the many relevant pages. Major search engines need to answer thousands of queries per second on collections of several billion pages. Thus, search engines require significant computing resources, typically provided by large clusters of hundreds or thousands of servers, and have to scale not just with the number of queries, but also with the amount of data that needs to be analyzed per query in order to provide a good answer.

To better understand the challenge, we look at the basic structure of current search engines. These engines, like many other information retrieval tools, are based on an *inverted index*, which is an index structure that allows efficient retrieval of documents containing a particular word (or term). An inverted index consists of many *inverted lists*, where each inverted list I_w contains the IDs of all documents in the collection that contain a particular word w, sorted by document ID or some other measure, plus additional information such as the number of occurrences in each document, the exact positions of the occurrences, and their context (e.g., in the title, in anchor text).

The simplest class of queries that can be implemented on such an inverted index are Boolean queries. Thus, a query (apple AND orange) OR pear for all documents containing both apple and orange, or the word pear, can be implemented by first intersecting the list of document IDs in the inverted lists for apple and orange, and then merging the result with the inverted list for pear. Of course, state-of-the-art search engines are not limited to Boolean queries, which have the following problems:

- Boolean queries do not provide any ranking of results, which is a problem since the result size for common queries may be in the millions of documents. This problem is overcome by applying a ranking function that assigns a numeric score to each document for the given query using term-based techniques (such as the cosine measure), hyperlink analysis, or traffic data.
- As the size of the document collection increases, the inverted lists for common words can become very

*Work supported by NSF CAREER Award NSF CCR-0093400 and the New York State Center for Advanced Technology in Telecommunications (CATT) at Polytechnic University, and by equipment grants from Sun Microsystems and Intel Corporation.

**Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003**

long. Thus, while each inverted list is much smaller than the entire document collection, a terabyte page collection indexed by a large engine results in many lists with sizes in the range of multiple megabytes that may have to be traversed during a query. In general, a doubling of the collection size results in a doubling of the amount of data that is scanned *for each query*.

This second problem typically stays with us as we go from Boolean to other classes of ranked queries. In particular, many engines perform ranking by applying a ranking function to the result of a Boolean AND of the keywords. While this restriction to AND and the low average number of keywords per query make web search queries more efficient than the types of queries found in traditional IR systems, the amount of data that has to be scanned on disk is huge and scales linearly with the size of the collection.

This problem has motivated a number of pruning techniques that attempt to compute or approximate certain classes of ranking functions without scanning the entire inverted lists of the search terms; this is typically based on presorting the inverted lists so that the most promising documents are near the beginning. Most of this work has focused on term-based ranking functions. However, current search engines rely heavily on hyperlink analysis and traffic data (user feedback) for ranking, in addition to term-based techniques. While there has been a large amount of research on link-based ranking techniques, there is very little published work on how to efficiently integrate link-based or traffic-based techniques into query processing on engines of realistic size.

In this paper, we attempt to take a first step towards closing this gap. In particular, we are interested in optimizing query performance, as measured by query throughput and latency, in a large web search engine with term- and link-based ranking, through the use of appropriate pruning techniques that scale well with the collection size. We describe several techniques and perform an initial experimental evaluation on a research prototype engine with 120 million pages that we have built in our group.

The next section gives some technical background. Section 3 describes our contributions, and Section 4 discusses related work. The proposed techniques are described in Section 5 and experimental results are given in Section 6.

2 Preliminaries

In this section, we provide some background on ranking in search engines. We assume that we have a document collection $D = \{d_0, d_1, \ldots d_{n-1}\}$ of n web pages that have already been crawled and are available on disk. Let $W = \{w_0, w_1, \ldots, w_{m-1}\}$ be all the different words that occur anywhere in the collection. Typically, any text string that appears between separating symbols such as spaces, commas, etc. is interpreted as a valid potential word.

Indexes: An *inverted index* I for the collection consists of a set of inverted lists $I_{w_0}, I_{w_1}, \ldots, I_{w_{m-1}}$ where list I_w contains a *posting* for each occurrence of word w. Each posting contains the ID of the document where the word occurs, the (byte or approximate) position within the document, and possibly information about the context (in a title, in large or bold font, in an anchor text) in which the word occurs. The postings in each inverted list are often sorted by document IDs, which enables compression of the list, or by some other measure as described later. Thus, Boolean queries can be implemented as unions and intersections of these lists, while phrase searches (e.g., New York) can be answered by looking at the positions of the two words. We refer to [40] for more details.

Term-based ranking: The most common way to perform ranking in IR systems is based on comparing the words (terms) contained in the document and in the query. More precisely, documents are modeled as unordered sets of words, and a ranking function assigns a score to each document with respect to the current query, based on the frequency of each query word in the page ("higher score for multiple occurrences") and in the overall collection ("rare words are more significant"), the length of the document ("long documents should not have too much of an advantage"), and maybe the context of the occurrence ("higher score if word occurs in title or bold face"). Formally, a *ranking function* is a function F that, given a query consisting of a set of search terms $t_0, t_1, \ldots t_{m-1}$, assigns to each document d a score $F(d, t_0, \ldots, t_{m-1})$. The system then returns the k documents with the highest score.

Many different functions have been proposed, and the techniques in this paper are not limited to any particular ranking function, as long as certain conditions are met as described later. In our experiments, we use a version of the cosine measure [40] defined by

$$\begin{aligned} F(d, t_0, \ldots, t_{m-1}) &= \sum_{i=0}^{m-1} \frac{w(q, t_i) \cdot w(d, t_i)}{\sqrt{|d|}}, \\ w(q, t) &= \ln(1 + N/f_t), \text{and} \\ w(d, t) &= 1 + \ln f_{d,t}, \end{aligned}$$

where $f_{d,t}$ and f_t are the frequency of term t in document d and in the entire collection, respectively. To compute the top k documents, it suffices to assign a score to all documents that contain at least one query word (i.e., the union of the inverted lists). Note that in general it does not suffice to score only documents in the intersection since, e.g., a document containing two out of three query words multiple times or in the title may score higher than a longer document containing all three query words. However, most large search engines default to such an AND semantic, due to reasons involving user expectations, collection size, and the preponderance of short queries. We focus on this case.

Several authors have proposed techniques that can identify (or guess) the top k documents without scoring all documents in the lists; see e.g., [1, 2, 14, 15, 18, 32] for recent work and [38] for an overview of older work. Typically, these techniques reorder the inverted lists such that high-scoring documents are likely to be at the beginning, and then terminate the search over the inverted lists once most

of the high-scoring documents have been scored.

Link-based and other ranking techniques: None of the current major engines perform ranking solely based on term-based techniques. In general, ranking is performed through a combination of term-based and link-based techniques, plus other factors such as user feedback and off-line preprocessing for spam identification and cluster analysis.

A large amount of research has recently focused on link-based ranking techniques, i.e., techniques that use the hyperlink (or graph) structure of the web to identify interesting pages or relationships between pages. One important technique is the *Pagerank* technique underlying the Google search engine [8], which assigns a global importance measure to each page on the web that is proportional to the number and importance of other pages linking to it. A number of other approaches have also been proposed, see, e.g., [6, 13, 22, 24, 25, 33], that perform link-based ranking either at query time or as a preprocessing step.

Integrating term-based and other factors: Despite the large amount of work on link-based ranking, there is almost no published work on how to efficiently integrate the techniques into a large search engine. We are particularly interested in Pagerank and other techniques that precompute a global ranking function for all documents, independent of the query, that is then at query time combined with term-based and other results. While it has been argued that a query-dependent link analysis might give better results, global techniques such as Pagerank that allow precomputation are very attractive for reasons of efficiency and simplicity. Brin and Page allude to the possible advantages in their paper on the architecture of the original Google engine [8], which contains what is essentially a simple term-based pruning technique based on the idea of *fancy hits*.

A natural way to build a combined ranking function is to add up a term-based score and a suitably normalized score derived, e.g., from Pagerank. This is the approach suggested in [33] that we follow in our experiments. Formally, we consider ranking functions of the forms

$$F(d, t_0, \ldots, t_{m-1}) = g(d) + \sum_{i=0}^{m-1} f(d, t_i).$$

In fact, as discussed further below, we use the logarithm of the value returned by the Pagerank computation, rather than the raw value, as input in the above function. Moreover, following [33], before adding up the terms we perform a query-dependent normalization that assigns approximately the same weight to the term-based and link-based contribution, and that is derived from the mean of the x highest term-based and link-based values in each inverted list.[1]

We note that in search engines, the score may also depend on the distance between the query words in the document, with words occurring close to each other getting a higher score. Formally, this adds a third term $h(d, t_0, \ldots, t_{m-1})$ that depends on all query terms. Dealing with this case is an open problem for future research.

Search engine architecture: Answering thousands of queries per second on a terabyte collection requires the equivalent of a small supercomputer, and all current major engines are based on large clusters of servers connected by high-speed LANs or SANs. There are two basic ways to partition an inverted index structure over the nodes: (a) a local index organization where each node builds a complete index on its own subset of documents (used by AltaVista and Inktomi), or (b) a global index organization where each node contains complete inverted lists for a subset of the words.[2] Each scheme has advantages and disadvantages that we do not have space to discuss here; see [4, 28, 37].

In this paper, we assume a local index organization, although some of the ideas also apply to a global index. Thus, we have a number of machines, in our case 16, each containing an index on a subset of the documents. Another machine acts as a *query integrator* that receives queries from the users, broadcasts them, and then merges the returned results into a proper ranking that is sent back to the user; see [26] for details. Thus, one way to answer a top-10 query is to ask for the top-10 results from each machine. However, if documents are randomly assigned, most machines may only have to return their top-k results for $k = 2$ or $k = 3$ in order to determine the global top-10. This is relevant to our work since pruning techniques are much more efficient for small values of k. In fact, some of the proposed methods can be implemented in an incremental fashion, so that each machine returns the top-1, top-2, top-3 results as soon as each is discovered during the computation, and the query integrator can first ask for the top-2 results and then later request additional values to obtain the top-3 or top-4 from some of the nodes as needed.

3 Contributions of this Paper

In this paper, we study optimized query processing in search engines with a locally partitioned inverted index and a ranking function that combines term-based techniques with a global page ordering obtained through link analysis, user feedback, or other offline preprocessing. In particular:

(1) We describe several pruning techniques that can significantly improve query throughput and response times on large document collections.

(2) We perform a preliminary experimental evaluation based on a search engine prototype with 16 nodes and 120 million web pages built in our research group, using a real query trace from the Excite search engine. Our experiments show the limits and benefits of the proposed techniques in terms of query throughput and response time for varying levels of concurrency.

[1] The purpose of choosing not the maximum value, but the mean of the x highest is to discount the effect of outliers. In [33], a value of $x = 10$ is chosen, while we use a larger value due to the much larger data set.

[2] In addition, there are hybrid schemes [37], and replication is often used in conjunction with partitioning for better performance.

3.1 Significance of the work

We believe that our work is interesting from two angles. First, query processing efficiency is a very important issue in large search engines that is not addressed much in the published literature. Increasing efficiency say, by a factor of two, allows a major engine to run on hundreds or thousands of fewer machines. It seems that there are still significant differences in query execution efficiency among the major engines, which are based on different proprietary index organizations and query execution schemes. The problem of scaling with collection size and load is also important in the context of intranet or enterprise search in larger organizations (multinational companies, federal agencies) where query load is smaller but nonetheless significant, requiring a small cluster of machines.[3] We note that pruning schemes such as the ones we propose are particularly attractive during peak times when the query load is significantly larger than average, and they can adapt to the load in a continuous and online fashion. It appears that several search engines are already using techniques for dealing with high loads by modifying query execution [7], though no details have been published.

Second, we believe our results are interesting in the context of the ongoing discussion about different approaches to link analysis, in particular the issues of preprocessing-based [22, 31, 33] vs. online [24, 25] techniques, and global [31] vs. topic-specific [22] vs. query-specific [24, 25, 33] techniques. Our results indicate that a global precomputed ordering is highly desirable from a performance point of view, as it allows optimized index layout and pruning. While it is not clear yet that query-specific and online approaches are really superior in terms of results, such methods could possibly be more efficiently implemented as corrective measures on top of a strong global ordering such as Pagerank that can be used for index layout and pruning.[4]

3.2 Limitations and Loose Ends

There are a number of important issues that we do not address in this paper and that are left for future work. Firstly, our focus here is on query throughput and response time, and in our experiments we try to bypass the issue of result quality which is of course very important. We do so by fixing a particular ranking function from [33] that combines the cosine measure and the Pagerank method of [8], and that we believe to be a reasonable approach to ranking. We then attempt to compute or approximate this function in the most efficient manner. While our techniques do not assume any particular ranking function, the savings obtained in our experimental evaluation clearly depend on the choice of this function. If the function weighs the term-based score much higher than the link-based score, then our approach will give little or no benefit over the standard approach. However, it appears from the Google Toolbar that link-based scores have a significant impact on ranking.

In future work, we intend to investigate what savings would be obtained by pruning a more finely tuned ranking function, involving factors such as term context (title, font size, anchor text), within our search engine prototype. On the other hand, we also plan to evaluate our techniques using the TREC web data set, in order to explore the trade-off between efficiency and the result quality in terms of precision and recall. Other problems for future research are the impact of using distances between terms in the document, and topic-specific link analysis techniques such as [22, 33].

In addition, there are several loose ends in our experimental evaluation that we plan to resolve first. This includes experiments for more than two keywords, results for deterministic pruning with an incremental query integrator, the effects of significantly increasing the collection size on a single node, and an analysis of the impact of query characteristics such as the number of terms and their selectivity and correlation. We intend to include these results in a longer journal version of this paper.

4 Discussion of Related Work

For background on indexing and query execution in IR and search engines, we refer to [3, 5, 40], and for basics of parallel search engine architecture we refer to [7, 8, 26, 34]. Discussions and comparisons of local and global index partitioning schemes and their performance are given, e.g., in [4, 12, 23, 28, 37]. A large amount of recent work has focused on link-based ranking and analysis schemes; see [6, 22, 24, 25, 31, 33] for a small sample.

Previous work on pruning techniques for top-k queries can be divided into two fairly disjoint sets of literature. In the IR community, researcher have studied pruning techniques for the fast evaluation of vector space queries since at least the 1980s. Some early work is described in [11, 21, 38, 41]. Most relevant to our work are the techniques by Persin, Zobel, and Sacks-Davis [32] and the follow-up work in [1, 2], which study effective early termination (pruning) schemes for the cosine measure based on the idea of sorting the postings in the inverted lists by their contribution to the score of the document. A scheme in [32] also proposed to partition the inverted list into several partitions, somewhat reminiscent of our scheme in Subsection 5.2, but for the purpose of achieving good compression of the index, rather than to integrate a global page ordering.

More recently, there has been a lot of work on top-k queries in the database community; see [16] for a survey and [15, 18] for a formal analysis of some schemes. This work was originally motivated by queries to multimedia databases, e.g., to retrieve images. Stated in IR terms, the algorithms also assume that postings in the inverted lists are sorted by their contributions and are accessed in sorted order. However, several of the algorithms proposed in [15, 18, 19, 30, 39] also assume that once a document is encountered in one of the inverted lists, we can efficiently

[3] In some cases, intranet collection sizes may be even larger than the 50 TB size of the Google collection.

[4] We note that this issue is not really limited to link-based techniques, but also includes other approaches such as user feedback or offline text analysis that might be used to arrive at a global ordering.

compute its complete score by performing lookups into the other inverted lists. This gives much better pruning than sorted access alone, but in a search engine context it may not be efficient as it results in many random lookups on disk.[5] Other schemes [10, 18, 20] work without random lookups or in cases where only some of the lists are sorted.

Thus, there are a variety of previous pruning schemes, though many have objectives or assumptions that are different from ours. For example, work in IR is often focused on queries with very large numbers of terms per query (e.g., between 66 and 313 in [32]), or concerned with main memory consumption or CPU resources for evaluating the cosine measure. Other work assumes that random lookups are feasible. Schemes can be either precise or only compute approximate top-k results [14, 18], or use precomputation to reduce the lengths of the stored inverted lists [17]. We also note that much of the previous work performs ranking on the union, rather than intersection, of the inverted lists. This results in increased work for the CPU for evaluating the cosine measure, and some schemes store precomputed floating point scores as part of the inverted lists. One the other hand, using the union can help some pruning schemes by allowing them to lower-bound the total score of a document seen in only some of the lists, by assuming a score of zero for the others. Our approach uses ideas from several schemes, but we are not aware of any previous work considering integration of a global page ordering such as Pagerank into pruning methods.

Finally, in their paper on the original Google architecture [8], Brin and Page discuss an optimization called "fancy hits" that stores term occurrences in special contexts such as title, bold face, or anchortext in a separate much smaller structure that can be quickly scanned for likely good documents. Since these special contexts can be modeled in the vector space model by increasing weights appropriately, this approach is closely related to the one described by us in Subsection 5.2. Note that [8] does not give much detail on how fancy hits are used and we do not know what types of pruning schemes are used in the current Google engine. However, to our knowledge several of the major engines still scan the entire inverted lists for most queries. With the exception of [27], we are not aware of any previous large-scale study on query throughput in large engines under web query loads.

5 Pruning Techniques

We now describe the different pruning techniques that we study in this paper. Recall that we are given a global ordering of the web pages and an associated global score for each page. In our experiments, we are using the Pagerank ordering, with pages numbered from 1 (most important) to n (least important) and with real-valued scores produced through an iterative computation. However, any other global ordering would also work. For simplicity, we describe all techniques for the case of 2 query terms.

One subtle issue is how to utilize the output of the Pagerank computation. Initially, we tried to use the raw Pagerank score. However, the distribution of these values is extremely skewed and very different from that of the term-based values which contain logarithms as part of the cosine measure. We found that the distribution of the logarithm of the Pagerank value is much closer to that of the term values, as shown in Figure 5.1. We thus decided to use the logarithm as input to the normalization procedure below, and throughout the following pr refers to the logarithm of the raw value. We note that the Pagerank scores between 0 and 10 returned by the Google Toolbar are conjectured to also be logarithmic in nature. The base of the logarithm does not matter due to the subsequent normalization, though we could give a higher relative importance to either term-based or global scores by modifying the normalization.

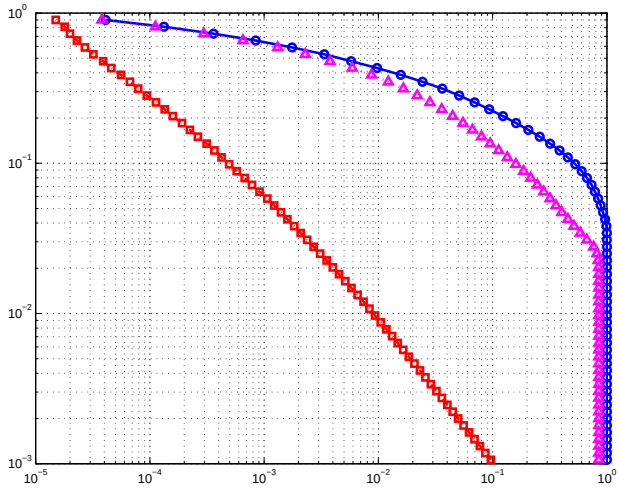

Figure 5.1: Distributions of raw Pagerank, log of Pagerank, and term scores in a subset of the collection (from left to right). For the latter two, between 20 and 50% of the values are within a factor of 10 of the highest score in the inverted list, while Pagerank scores are much more skewed.

We use the cosine measure as defined in Section 2, but we threshold $f(d,t_i)$ at 32 and limit the term $|d|$ in the document length normalization to between 64 and 4096. Following [33] we then define the total score of a document with respect to query $\{t_0,t_1\}$ as

$$\frac{2 * pr(d)}{m_{pr}(t_0)+m_{pr}(t_1)} + \frac{cos(d,t_0)+cos(d,t_1)}{m_{cos}(t_0)+m_{cos}(t_1)}, \qquad (1)$$

where $m_{pr}(t)$ is the mean of the x highest pagerank values for documents in the inverted list for term t, and $m_{cos}(t)$ is the mean of the x highest term cosine values for t. We choose $x = 100$ and in all experiments we use the global mean values over the entire collection on 16 machines; this requires additional precomputation but allows simple merging of results at the query integrator.

We note that the way the Pagerank score is preprocessed and normalized does of course have an impact on performance. In particular, if we use the raw Pagerank score

[5] The situation may be different in highly distributed environments with limited bandwidth [36].

instead of the logarithm and perform normalization using the mean of the top 100 as above, then Pagerank would not have much impact on the result of most queries, and thus no significant pruning based on Pagerank would be possible. This is because for most queries, there would not be any document with significant normalized Pagerank containing both keywords, due to the extreme skew of the values. This would seem to contradict our (unscientific) observation that Pagerank, as displayed by the Google Toolbar, does seem to have a strong influence on query results. If we normalize using average values instead, then the top-10 results would be decided primarily based on Pagerank for most queries, and even the *first-m* heuristic would get great performance benefits without much loss in precision. By using the logarithm of the raw Pagerank value, both terms are important in the ranking of query results.

5.1 A Naive Pruning Heuristic

For the first heuristic, we assume that inverted lists are sorted in descending order by the Pagerank scores of the documents; this is easily achieved by using the global rank as document ID during indexing. The pruning heuristic, called *first-m*, simply scans the inverted lists until it finds the first m pages that contain all query terms. Since our indexes are sorted by Pagerank, these are the m pages with the highest Pagerank that contain all query terms. After scoring these m pages, the top k are returned.

While this technique may not provide the best way to approximate our ranking function, we are interested in it for two reasons. First, for the case of $m = k$, this technique might be considered as a reasonable upper bound on the efficiency gain that can be obtained by pruning, since its cost is essentially that of identifying k pages in the intersections of the lists.[6] Second, in the case where m is a moderate multiple of k, this technique might be seen as a reasonable baseline from which to start.

We note that instead of sorting by Pagerank only and leaving the term values completely unsorted, we could also sort each list by term value and leave the Pageranks unsorted. This approach would be similar to earlier approaches for purely term-based ranking, and it would not perform well for our ranking function. Moreover, it would require additional data structures for matching up postings from the different lists that would slow down computation.

5.2 Separating Large Term Values

The next approach is motivated by the "fancy hits" approach of Brin and Page [8] and the work by Persin et al. in [32]. As discussed, previous pruning techniques for term-based queries sort the postings in each list by their contribution to the final score. In this case, since we are combining term-based and link-based scores according to Equation (1), a natural approach would be to sort the postings in each list by a combination of their term-based and link-based scores. For example, for queries with two terms, we could sort each list by the term value plus half the Pagerank contribution, so that by adding up the values from both lists we get the complete document score. Intuitively, this should introduce a strong correlation between the two lists that makes it more likely to encounter good documents early in both lists.

However, there are problems with this simple idea. First, the above formulation depends on the number of terms in the query. Things are further complicated by the query-specific normalization in Equation (1), and thus it is not clear by what combination of term-based and link-based scores we should sort since we cannot cleanly separate contributions due to different terms. Second, since postings are not ordered by document ID, we need additional data structures to match postings from different lists, and list compression also becomes less effective.

We thus decided to go with the following simpler scheme, which partitions each list into two parts by term value. More precisely, we have a short list containing the h postings in the list with the highest term value according to the cosine measure, and a longer list containing all other postings. The short list, called "fancy list" and inspired by [8], contains maybe a few thousand or more postings, and both lists are sorted by Pagerank. The intuition is that a document scoring high overall should be either in one of the fancy lists or close to the beginning of the longer lists. We utilize this structure in all subsequent heuristics.

5.3 A Better Pruning Heuristic

The next heuristic, called *fancy first-m*, is an extension of *first-m* to the new structure. We first process the two fancy lists. For all documents occurring in both of the lists, we compute their complete score and keep the top-k results. We also maintain two structures D_0 and D_1 for documents that occur only in the first or second fancy list, respectively. Afterwards, we simultaneously scan the two longer lists. Whenever we find a document occurring in both longer lists, or a document in one longer list that has already occurred in the other fancy list, we evaluate its score and consider it for inclusion in the top-k results. We again terminate after encountering m documents in the intersection.

5.4 A Reliable Pruning Technique

The *fancy first-m* approach from the previous section was again a heuristic that does not guarantee the correct top k results. We now describe a technique based on fancy lists that stops only when it can safely determine that it has obtained the top-k results. To do this, we periodically perform the following check while traversing the longer lists, after processing the fancy lists as described before.

Let pr_{last} be the Pagerank score of the last document encountered in the longer lists, and note that all subsequent documents must have lower Pagerank. Also, let t_0 and t_1 be the largest term values in the two longer lists, respectively. Thus, any postings with larger term values are located in the fancy lists. Note that based on the current k-th

[6] It is however not a strict upper bound if query terms are correlated.

largest total score, $top(k)$, and the values t_0 and t_1, we can purge from D_0 and D_1 any documents that cannot make it into the top k. Using $top(k)$, pr_{last}, t_0, and t_1, we can also decide if any documents not yet encountered at all can still make it into the top k. We stop when this test fails and both D_0 and D_1 are empty, and return the current top-k results.

We note that a very similar scheme can also be applied when postings are sorted by Pagerank only, as in Subsection 5.1. However, we observed only negligible performance gains even for top-1 queries and thus decided to not investigate this variant any further.

5.5 Probabilistic Pruning Techniques

We have also studied unreliable techniques that take a similar perspective as the reliable pruning technique. In particular, during the scan of the longer list, we again maintain the sets D_0 and D_1 of candidate documents that have only been encountered in one of the fancy lists and that could still make it into the top-k if they appear in the other list with a sufficient term score. After scanning a small portion of the longer lists, the Pageranks encountered usually become so small that no document not already in D_0 or D_1 can make it into the top-k anymore. At this point, we can assign to each document in D_0 and D_1 a probability that its total score is larger than the current value $top(k)$, under some basic assumptions about the underlying document collection (in particular independence between Pagerank and term values and an upper bound on the correlation between the terms).

Using simple statistics about the distribution of term values in the list where the document has not yet been encountered, we can estimate the likelyhood that the unknown value is larger than the difference between $top(k)$ and the known part of the document score. The procedure terminates whenever the probability of having the correct top-k results goes above some threshold, say 90% or 99%. We note that statistics could be kept in the form of a histogram or a Zipf parameter for each inverted list that upperbounds the value distribution, and that correlations could be estimated based on hashing techniques similar to those in [9].

For the fancy list organization, we have only implemented a very crude version of this idea where we terminate the scan whenever the number of candidates in D_0 and D_1 drops below a fixed threshold, such as 10 or 100. We refer to this technique as *last-m*, where m is the number of remaining candidates. A full implementation of the technique is left for future work. We also note that in some cases it might be useful to perform a limited number of random accesses to resolve some of the remaining candidates.

As before, the techniques are also in principle applicable to the index organization in Subsection 5.1. In fact, we have implemented the more general technique above, assuming term independence and using a simple histogram to upperbound term distributions, for this basic index organization. However, benefits in this case are extremely limited.

6 Experimental Results

We now present the experimental evaluation of the different approaches. We first describe the machine setup, data sets, and preprocessing steps. Subsection 6.2 contains results for the exhaustive baseline method and the *first-m* pruning technique from Subsection 5.1. Subsection 6.3 gives results for the *fancy first-m* scheme. Subsection 6.4 evaluates the reliable and the *last-k* pruning techniques for the fancy list organization. Subsection 6.5 summarizes results for all techniques. All results up to this point are on a single node of the search engine. Subsection 6.6 finally gives results for a 16-node search engine architecture with query integrator frontend. Due to space constraints, we can only include a small sample of the main results.

6.1 Experimental Setup

Hardware: For our experiments, we used a set of 16 Dell Optiplex 240GX machines with 1.6 GHz Pentium-4, 1 GB of memory, and two 80 GB Seagate Barracuda (ST380021A) hard disks, connected by switched fast Ethernet. Each node used only one disk for the index structure.

Software and Data Sets: Our experiments were run on a search engine prototype, named *pingo*, that is currently being developed in our research group. The document collection consisted of about 120 million web pages crawled by the PolyBot web crawler [35] in October of 2002. Not all of the pages are distinct and the set contains a significant number of duplicates due to pages being repeatedly downloaded because of crawl interruptions. The crawl started at a hundred homepages of US Universities, and was performed in a breadth-first manner.[7] As observed in [29], such a crawl will quickly find most pages with significant Pagerank value. The total uncompressed size of the data was around 1.8 TB. An I/O-efficient implementation of Pagerank was used to compute the global ranks and scores.

This data set was distributed over the nodes in a (fairly) random fashion, with each node receiving about 7.5 million pages (120 GB uncompressed) of data. Indexing took about 11 hours per disk, and the resulting inverted lists, sorted according to Pagerank, were stored in Berkeley DB in highly compressed form, using the compression macros available as part of the `mg` system [40]. 5000 queries with two terms were taken from a trace of over 1 million queries issued to the Excite search engine on December 20, 1999. For the experiments, we removed queries with stopwords, although our approach actually does relatively better for queries with very frequent words.

Fancy lists were implemented in a separate Berkeley DB database. Note that we did not delete postings included in the fancy lists from the longer lists in the basic index. This allowed us to experiment with different sizes for the fancy lists without having to rebuild the basic index, but as a consequence the basic index is slightly larger than needed. We also stored position information for all post-

[7] Subject to a $30s$ interval between accesses to the same server that may cause reordering from strict breadth-first.

ings even though this is not needed for the simple ranking functions we consider here. Removing the postings, or using improved compression macros, would increase query throughput. Overall, however, we are confident that our index and query execution implementations are fairly optimized towards high performance.

Evaluation Methodology: We first compare the performance of the various techniques on a single node of the search engine. At the end, we show selected results on 16 machines with a query integrator. The cost of a technique is usually stated in terms of the average number of 4 KB blocks that are scanned per query, but we also investigate how this translates into the query throughput and latency that is achieved under various degrees of concurrency.

The quality of the techniques is always measured relative to the ranking function that we are approximating; thus, the reliable pruning technique has zero error. We consider two measures. An error of 0.01 under the *strict-k* measure means that the technique we consider returns exactly the same top-k results as the ranking function for 99% of all queries. An error of 0.01 under the *loose-k* measure means that 99% of all documents returned within the top-k belong in the top-k. We typically report results for $k = 1$, 4, and 10. We note that the cases of small k are of interest because in a parallel search engine we do not need the correct top-10 from each node to determine the global top-10. In particular, retrieving the top-4 results from each of 16 nodes suffices to reliably determine the global top-10 with probability about 99%, assuming random allocation of documents to nodes.

6.2 Performance for Baseline and *first-m*

In our first set of experiments, we compared the following two approaches:

(a) Baseline (no pruning): We compute top 10 results by scanning the complete inverted lists, except when the inverted lists differ significantly in size in which case a binary search approach kicks in automatically.

(b) *first-m*: We return the top 10 results out of the first m documents in the intersection, for $m = 10, 100, 1000$.

Figures 6.1 and 6.2 show the throughput and average latency achieved for various levels of concurrency, i.e, the number of queries that are allowed to be executed concurrently on a node. As expected, the throughput improves at first with the degree of concurrency, but the average latency soon starts to increase significantly as the CPU is shared between too many active queries. Note that in the case of "AND" queries, the cosine measure has essentially no computational overhead beyond simple list intersection; hence we did not plot the cost of intersection without cosine evaluation as it was almost identical. As we see, the *first-m* methods achieve significantly higher throughput and lower latency than the baseline method. For $m = 1000$ we obtain a throughput of 12 queries per second and half a second latency for concurrency level 4, while the baseline method

top-10	*first-1000*	*first-100*	*first-10*
417.6	110.8	41.0	14.0

Table 6.1: Average number of 4 KB block accesses per node and per query.

is limited to less than 3 queries per second. For $m = 100$ and $m = 10$, throughput further increases and latency decreases, and the optimum degree of concurrency increases as expected for smaller disk accesses.

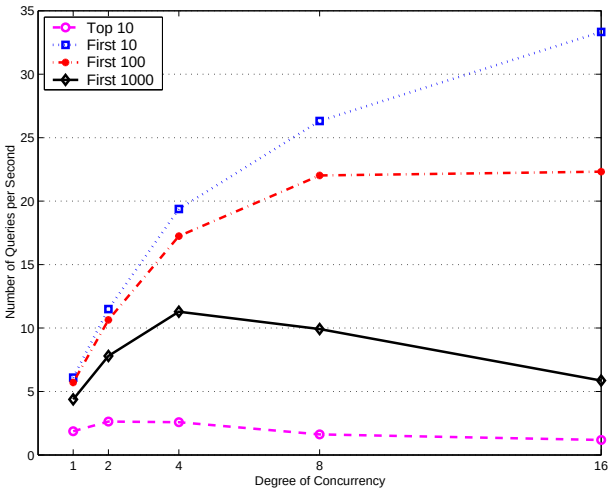

Figure 6.1: Query throughput per second.

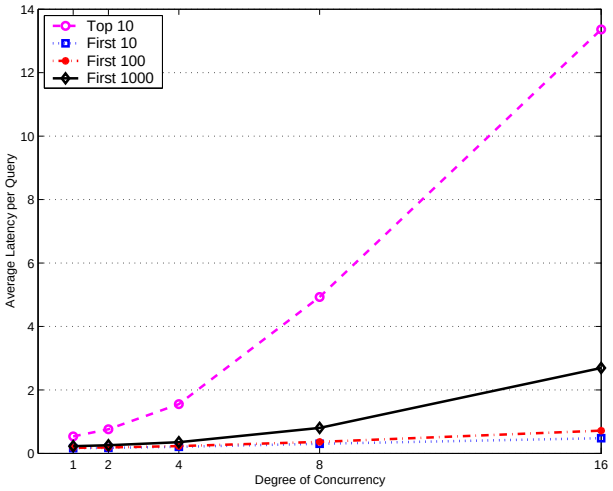

Figure 6.2: Average latency for each query.

If we decrease m to values smaller than 10, there is only very limited additional benefit. This is because for $m = 10$, only a few blocks of each inverted list are retrieved as shown in Table 6.1, implying that disk time is already dominated by seek times. Note that the baseline method scans about 1.67 MB of compressed postings data (about 15 MB uncompressed data) per query, even on a single node of 7.5 million documents.

In Figures 6.3 and 6.4, we show how well the heuristic performs at identifying the correct top-k results. For example, *first-1000* returns the correct top document for almost

90% of all queries, and correctly identifies all top-4 results for 80% of all queries. One the other hand, of all the results returned in the top-4, almost 90% really belong there. However, for *first-100* the results drop to around 81%, 65%, and 78%, respectively.

Thus, the results are at best mixed. While we get great performance benefits, they come at a significant error rate. Our hope is that the error rate will be significantly reduced as we move to the fancy list organization.

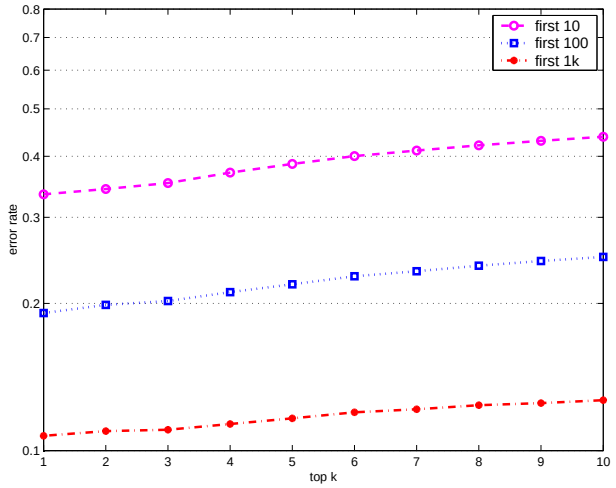

Figure 6.3: Error rate for loose-k, for different k.

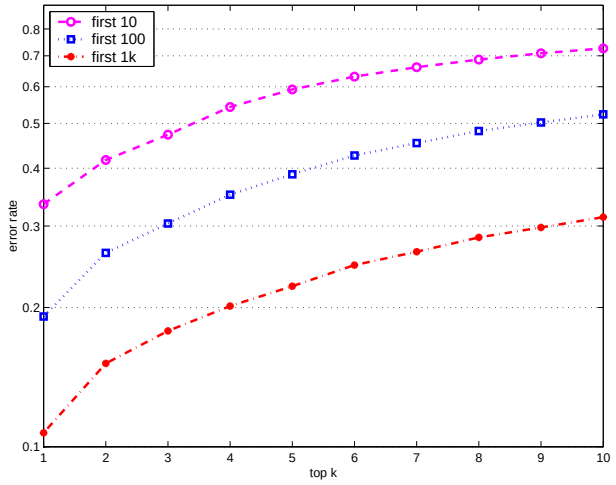

Figure 6.4: Error rate for strict-k, for different k.

6.3 Performance of *fancy first* Schemes

We now look at the *fancy first-m* scheme. Figure 6.5 shows the number of blocks scanned per query for various values of m and sizes of the fancy list. We see that the cost increases moderately with the length of the fancy lists, primarily due to the extra cost of scanning these lists. We do not consider caching effects and simply count the number of blocks that are accessed by our program, though Berkeley DB automatically takes care of caching.

As shown in Figures 6.6 and 6.7, the *fancy first-m* schemes achieve significantly lower error rates than the

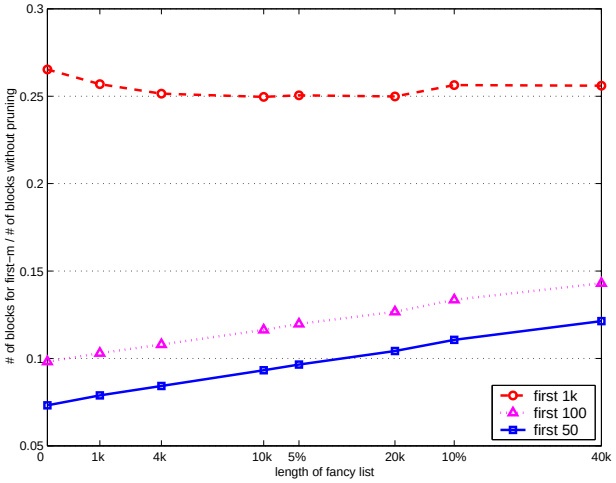

Figure 6.5: Average cost per query, for *fancy first-m* schemes with $m = 1000$, 100, and 50. The length of the fancy lists is varied from 0 to 40000 postings. We also plot results for the case where we choose the fancy lists to be 5% and 10% of the list length; these points are located on the x-axis according to their average list length relative to the others. Cost is plotted as ratio between the number of blocks accessed by *fancy first-m* and the baseline scheme.

first-m schemes. In particular, for a fancy list length of 10%, *fancy first-100* obtains the correct top-4 results for more than 99% of all queries. We note that *fancy first-1000* obtained the correct top-1 result for all queries in our query set except for fancy lists of length 0; thus, only this point is plotted. Also, we see that choosing the lengths of the fancy lists as a percentage of the total list length leads to better error bounds when compared to a fixed length organization with about the same cost.

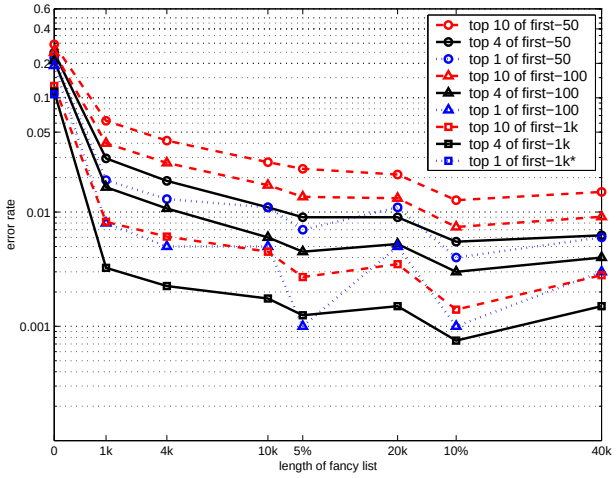

Figure 6.6: Error rate for loose-k measure, for different values of m and k, and different lengths of the fancy lists.

6.4 Reliable and *last-k* Pruning

First, we look at the cost of the reliable pruning technique with different fancy list lengths, shown in Figure 6.8. Note that with a length of 5%, we can determine the top-4 results in about 20% of the cost of the baseline method (with zero

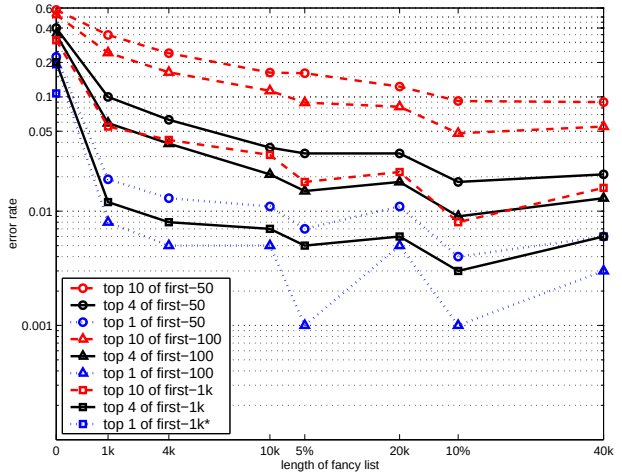

Figure 6.7: Error rate for strict-k measure, for different values of m and k, and different lengths of the fancy lists.

error as the method is reliable).

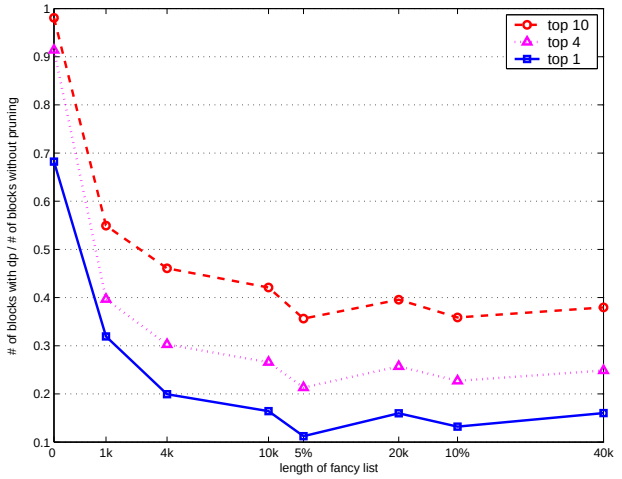

Figure 6.8: Ratio of the number of blocks accessed by reliable pruning and the baseline scheme, for different k and different lengths of the fancy lists.

We also ran experiments for the *last-m* scheme, with $m = 10$ and $m = 100$. Due to space limitations, we cannot include detailed plots. However, both cases are inferior to the *fancy first-m* schemes in terms of the cost/error tradeoff, as shown in the summary plot further below.

6.5 Comparison of the Various Methods

Figure 6.9 finally gives a comparison of the various methods for the case of top-4 queries. We can identify 6 clusters of results: first, on the x-axis we have the reliable pruning technique, which outperforms the *fancy first-1000* schemes located above it to the right. To the left of the reliable scheme are the clusters for *fancy first-100* and *fancy first-50*, which have lower cost but higher error than reliable pruning. Above these two to the right are the *last-10* schemes which are strictly worse. Finally, at the top are the *last-100* schemes which are marginally faster than *fancy first-50*, but have much higher error. In fact, by using, say, *fancy first-25*, we could outperform *last-100* in both cost and error. Thus, the best schemes appear to be reliable pruning and *fancy first-m* with moderate values of m.

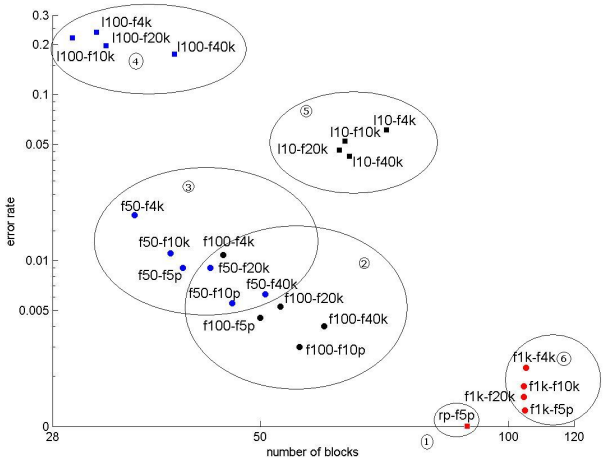

Figure 6.9: Comparison of various pruning techniques. Each technique and parameter setting is plotted as a point showing its cost in disk blocks on the x-axis, and its loose-4 error on the y-axis. Also shown are 6 clusters of points corresponding to different basic methods. Each point is identified by technique (e.g., f100 for *fancy first-m* or l100 for *last-100*) and length of the fancy lists.

6.6 Performance on 16 Machines

We now look at the performance of the best schemes when we run queries on the entire set of 120 million pages partitioned over 16 machines, with a query integrator on a separate machine that merges the results. In particular, we plot results for the baseline scheme, reliable pruning with fancy list length 10%, *fancy first-30* with fancy list length 5%, and *fancy first-100* with fancy list length 10%. All results are for top-10 queries. The concurrency level in this case is the total number of queries that are allowed to be active at the query integrator at any time.

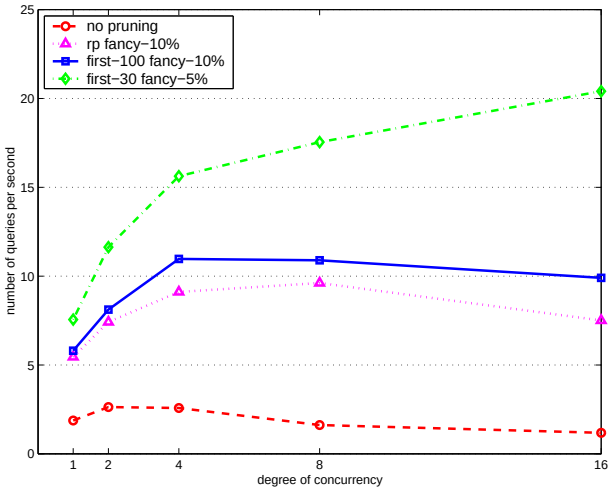

Figure 6.10: Query throughput per second on 16 machines.

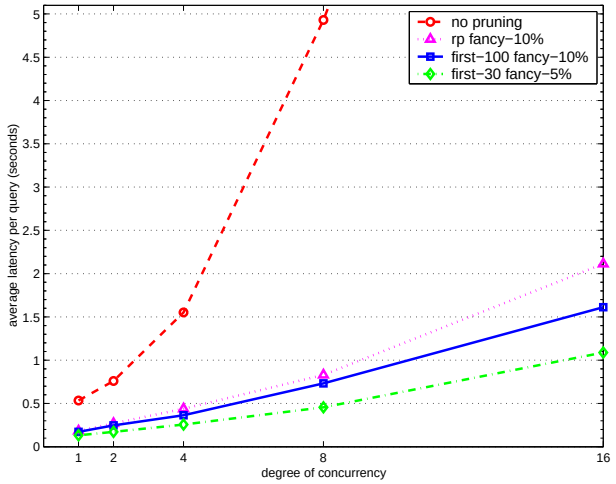

Figure 6.11: Query latency in seconds on 16 machines.

Note that for reliable pruning, the query integrator asks for the same number of results k from all nodes, and for each query we used the minimum k that allowed the query integrator to determine the top-10. This number was precomputed for each query for ease of experimentation. This is not an unrealistic shortcut however, since the reliable pruning scheme can be implemented in an incremental manner, so that the query integrator could ask for the top-$(k+1)$ results after receiving the top-k results (with no extra overhead). In fact, the reliable pruning technique would be improved if the query integrator asks only for the minimum number of results needed from each particular node.

As shown in Figures 6.10 and 6.11, throughput is again limited to less than 3 queries per second for the baseline method. Using reliable pruning, we get a throughput of more than 9 queries per second with latency below one second, at concurrency level 8. For the most aggressive method, *fancy first-30*, we achieve more than 20 queries per second, for concurrency level 16.

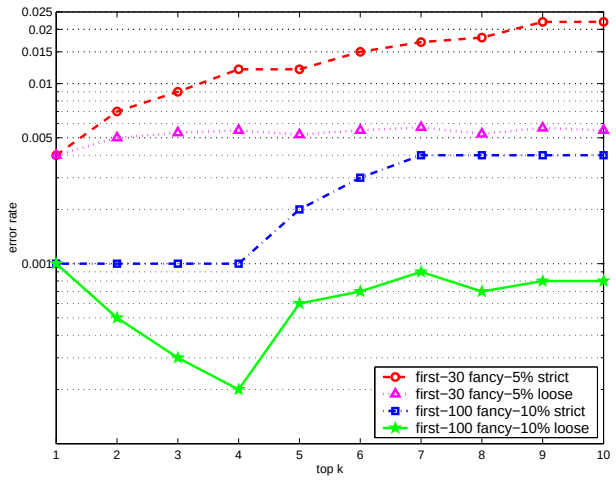

Figure 6.12: Errors for *fancy first-m* on 16 machines.

Looking at the error rate for the *fancy first-m* schemes in Figure 6.12 we get a pleasant surprise. Even for the *fancy first-30* scheme, we obtain exactly the correct top-10 results for almost 98% of all queries, and the error rate is even lower for *fancy first-100*. This is due to our prior observation that we usually only need the correct top-2 or top-3 from each node to determine the global top-10 correctly.

We finish with a few remarks on scaling with problem size, although we have not yet obtained conclusive results. We ran experiments with the above four techniques on a single node with collection sizes from 3 to 12.2 million pages. We observed no significant changes in error rate, but some increase in query cost. Partly this is due to the fact that we chose the lengths of the fancy lists as a fixed percentage of the list lengths; it might be better to decrease the percentage as collection size grows. For search engines we are particularly interested in the following question: if total index size is not an issue, is it better to build a complete index for the entire collection on each node (or subset of nodes) and forward each query to only one node (subset), or should we partition the index and broadcast to all nodes? The answer is still open for the presented pruning techniques, but we hope to resolve this by measuring throughput and latency on significantly larger indexes.

Acknowledgements

Thanks to Jiangong Zhang for help with simulation results and to Yen-Yu Chen for providing the Pagerank values.

References

[1] V. Anh, O. Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proc. of the 24th Annual SIGIR Conf. on Research and Development in information retrieval*, pages 35–42, September 2001.

[2] V. Anh and A. Moffat. Compressed inverted files with reduced decoding overheads. In *Proc. 21st Annual SIGIR Conf. on Research and Development in Information Retrieval*, pages 290–297, 1998.

[3] A. Arasu, J. Cho, H. Garcia-Molina, and S. Raghavan. Searching the web. *ACM Transactions on Internet Technologies*, 1(1), June 2001.

[4] C. Badue, R. Baeza-Yates, B. Ribeiro-Neto, and N. Ziviani. Distributed query processing using partitioned inverted files. In *Proc. of the 9th String Processing and Information Retrieval Symposium (SPIRE)*, September 2002.

[5] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addision Wesley, 1999.

[6] A. Borodin, G. Roberts, J. Rosenthal, and P. Tsaparas. Finding authorities and hubs from link structures on the World Wide Web. In *10th Int. World Wide Web Conference*, 2001.

[7] E. Brewer. Lessons from giant scale services. *IEEE Internet Computing*, pages 46–55, August 2001.

[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. of the Seventh World Wide Web Conference*, 1998.

[9] A. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES'97)*, pages 21–29, 1997.

[10] N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In *Proc. of the 18th Annual Int. Conf. on Data Engineering*, 2002.

[11] C. Buckley and A. Lewit. Optimization of inverted vector searches. In *Proc. 8th Annual SIGIR Conf. on Research and Development in Information Retrieval*, pages 97–110, 1985.

[12] B. Cahoon, K. McKinley, and Z. Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *IEEE Transactions on Information Systems*, 18(1):1–43, January 2000.

[13] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource list compilation by analyzing hyperlink structure and associated text. In *Proc. of the 7th Int. World Wide Web Conference*, May 1998.

[14] S. Chaudhuri and L. Gravano. Optimizing queries over multimedia repositories. *Data Engineering Bulletin*, 19(4):45–52, 1996.

[15] R. Fagin. Combining fuzzy information from multiple systems. In *Proc. of ACM Symp. on Principles of Database Systems*, 1996.

[16] R. Fagin. Combining fuzzy information: an overview. *SIGMOD Record*, 31(2):109–118, June 2002.

[17] R. Fagin, D. Carmel, D. Cohen, E. Farchi, M. Herscovici, Y. Maarek, and A. Soffer. Static index pruning for information retrieval systems. In *Proc. of the 24th Annual SIGIR Conf. on Research and Development in Information Retrieval*, pages 43–50, September 2001.

[18] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. of ACM Symp. on Principles of Database Systems*, 2001.

[19] U. Guntzer, W. Balke, and W. Kiessling. Optimizing multifeature queries in image databases. In *Proc. of the 26th Int. Conf. on Very Large Data Base*, pages 419–428, 2000.

[20] U. Guntzer, W. Balke, and W. Kiessling. Towards efficient multi-feature queries in heterogeneous environments. In *Proc. of the IEEE Int. Conf. on Information Technology, Coding and Computing*, April 2001.

[21] D. Harman and G. Candela. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *Journal of the American Society for Information Science*, 41(8):581–589, August 1990.

[22] T.H. Haveliwala. Topic-sensitive pagerank. In *Proc. of the 11th Int. World Wide Web Conference*, May 2002.

[23] B. Jeong and E. Omiecinski. Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):142–153, 1995.

[24] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Proc. of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, January 1998.

[25] R. Lempel and S. Moran. The Stochastic Approach for Link-Structure Analysis (SALSA) and the TKC Effect. In *Proc. of the 9th Int. World Wide Web Conference*, May 2000.

[26] R. Lempel and S. Moran. Optimizing result prefetching in web search engines with segmented indices. In *Proc. of the 28th Int. Conf. on Very Large Data Bases*, August 2002.

[27] Z. Lu. *Scalable Distributed Architectures For Information Retrieval*. PhD thesis, Univ. of Massachusetts, May 1999.

[28] S. Melnik, S. Raghavan, B. Yang, and H. Garcia-Molina. Building a distributed full-text index for the web. In *Proc. of the 10th Int. World Wide Web Conference*, May 2000.

[29] M. Najork and J. Wiener. Breadth-first search crawling yields high-quality pages. In *10th Int. World Wide Web Conference*, 2001.

[30] S. Nepal and M. Ramakrishna. Query processing issues in image (multimedia) databases. In *Proc. of the 15th Annual Int. Conf. on Data Engineering*, pages 22–29, March 1999.

[31] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University, 1999.

[32] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science*, 47(10):749–764, May 1996.

[33] M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in pagerank. In *Advances in Neural Information Processing Systems*, 2002.

[34] K. Risvik and R. Michelsen. Search engines and web dynamics. *Computer Networks*, 39:289–302, 2002.

[35] V. Shkapenyuk and T. Suel. Design and implementation of a high-performance distributed web crawler. In *Proc. of the Int. Conf. on Data Engineering*, February 2002.

[36] T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. ODISSEA: A peer-to-peer architecture for scalable web search and information retrieval. In *International Workshop on the Web and Databases (WebDB)*, June 2003.

[37] A. Tomasic and H. Garcia-Molina. Performance of inverted indices in distributed text document retrieval systems. In *Proc. of the 2nd Int. Conf. on Parallel and Distributed Information Systems (PDIS)*, 1993.

[38] H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Information Processing and Management*, 31(6):831–850, November 1995.

[39] E. Wimmers, L. Haas, M. Roth, and C. Braendli. Using fagin's algorithm for merging ranked results in multimedia middleware. In *Fourth IFCIS Int. Conf. on Cooperative Information Systems*, pages 267–278, September 1999.

[40] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, second edition, 1999.

[41] W. Wong and D. Lee. Implementations of partial document ranking using inverted files. *Information Processing and Management*, 29(5):647–669, September 1993.