Optimizing Result Prefetching in Web Search Engines with Segmented Indices

RONNY LEMPEL
IBM Research Labs, Haifa
and
SHLOMO MORAN
Technion, Haifa

We study the process in which search engines with segmented indices serve queries. In particular, we investigate the number of result pages that search engines should prepare during the query processing phase.

Search engine users have been observed to browse through very few pages of results for queries that they submit. This behavior of users suggests that prefetching many results upon processing an initial query is not efficient, since most of the prefetched results will not be requested by the user who initiated the search. However, a policy that abandons result prefetching in favor of retrieving just the first page of search results might not make optimal use of system resources either.

We argue that for a certain behavior of users, engines should prefetch a constant number of result pages per query. We define a concrete query processing model for search engines with segmented indices, and analyze the cost of such prefetching policies. Based on these costs, we show how to determine the constant that optimizes the prefetching policy. Our results are mostly applicable to *local index* partitions of the inverted files, but are also applicable to processing short queries in *global index* architectures.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*; H.3.7 [**Information Storage and Retrieval**]: Digital Libraries—*System issues*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Distributed inverted indices, prefetching, search engines

This research was supported by the Technion VPR fund—DENT charitable trust—nonmilitary research fund, and by the Barnard Elkin Chair in Computer Science.

Authors' addresses: R. Lempel, IBM Research Labs, Haifa 31905, Israel; email: rlemplel@il. ibm.com; S. Moran, Department of Computer Science, Technion, Haifa 32000, Israel; email: moran@ cs.technion. ac.il.

1. INTRODUCTION

The sheer size of the World Wide Web (WWW) and the efforts of search engines to index significant portions of it [Lawrence and Giles 1998] have caused many search engines to partition their inverted index of the Web into several disjoint segments (partial indices). The partitioning of the index impacts the manner in which the engines process queries. Most engines also use some form of query result caching, where results of queries that were served are cached for some time. In particular, query results may be *prefetched* in anticipation of user requests. Such scenarios occur when the engine retrieves (for a certain query) more results than will initially be returned to the user.

We examine efficient prefetching policies for search engines. These policies depend on the architecture of the search engine (which, in turn, affects its query processing scheme) and on the behavior patterns of search engine users.

1.1 Search Engine Users

Users interact with search engines in *search sessions*. Sessions begin when users submit *initial* queries to search engines, by typing some search phrase that describes their topic of interest. From the user's point of view, an engine answers each initial query with a linked set of ranked result pages, typically with 10 results per page. All users browse the first page of results, which contains the results deemed by the engine's ranking scheme to be the most relevant to the query. Some users scan additional result pages, usually in the natural order in which those pages are presented. A search session implicitly terminates when the user decides not to browse additional result pages on the topic that initiated the session.

Several studies have analyzed the queries that users submit to search engines, and the length of search sessions [Jansen et al. 2000; Markatos 2000; Silverstein et al. 1998; Lempel and Moran 2003]. Three findings that these studies share are particularly relevant to this work:

- The queries submitted to WWW search engines are very short, averaging less than 2.4 terms per query, with over half of the queries containing just one or two terms. These results were reported by both Silverstein et al. [1998] and Jansen et al. [2000]. While the two studies define *query terms* somewhat differently, the reported term counts may be loosely interpreted as the number of words per query.
- Users browse through very few result pages. These studies differ in the reported distribution of page views, but agree that at least 58% of the users view only the first page (the top-10 results), and that no more than 12% of users browse through more than 3 result pages.

  While the above describes the behavior of users as they browse through multiple result pages, statistics have also been gathered on the browsing patterns of users as they view a single page of results. It has been observed that users are reluctant to scroll beyond the visible part of the page, and so search results that are “above the fold” are viewed (and clicked on) by more users than results at the bottom of the page [Broder 2000].

- The number of distinct information needs of users is very large, as can be seen from the huge variety of queries submitted to search engines. However, popular queries are repeated many times, and the 25 most popular queries account for over 1% of all queries submitted to the engines [Markatos 2000; Silverstein et al. 1998; Lempel and Moran 2003].

1.2 Caching and Prefetching of Search Results

It is commonly believed that all major search engines perform some sort of search result caching and prefetching. Caching of results was noted in Brin and Page's [1998] description of the prototype of the search engine *Google*[1] as an important optimization technique of search engines. Markatos [2000] used a log of a million queries submitted to the search engine Excite[2] to demonstrate that caching search results can lead to hit ratios of close to 30%.

In addition to storing results that were requested by users in the cache, search engines may also *prefetch* results that they predict to be requested shortly. An immediate example is prefetching the second page of results whenever a new session is initiated by a user. Since studies [Jansen et al. 2000; Lempel and Moran 2003; Silverstein et al. 1998] indicate that the second page of results is requested shortly after a new query is submitted in at least 15% of cases, search engines may prepare and cache two (or more) result pages per query. In Lempel and Moran [2003], a log containing over seven million queries submitted to the search engine AltaVista[3] was used to test integrated schemes for the caching and prefetching of search results. Hit ratios exceeding 50% were achieved. Prefetching of results proved to be of major importance, doubling the hit ratios of small caches and increasing those of larger caches by more than 50%.

1.3 Index Structure and Query Processing Models

Inverted indices, or inverted lists/files, are regarded as the most widely applied indexing technique [Arasu et al. 2001; Jeong and Omiecinski 1995; Tomasic and Garcia-Molina 1993; Ribeiro-Neto and Barbosa 1998; Melnik et al. 2001; Ribeiro-Neto et al. 1998], and are believed to be used by the major search engines. As search engines index hundreds of millions of Web pages [Lawrence and Giles 1998], the size of their inverted indices is measured in terabytes.

Ribeiro-Neto and Barbosa [1998] mention three hardware configurations that can handle large digital libraries: a powerful central machine, a parallel machine, or a high-speed network of machines (workstations and high end desktops). However, when considering the size of the indices that search engines maintain, the growth rate of the Web and the large number of queries that search engines answer each day, using a network of machines is considered to be the most cost-effective and scalable architecture [Hawking 1997; Ribeiro-Neto and Barbosa 1998]. Such networks operate in a *shared-nothing* memory,

[1] http://www.google.com/.
[2] http://www.excite.com.
[3] http://www.altavista.com.

organization [Ribeiro-Neto et al. 1998] where each machine has its own processing power (one or several CPUs), its own memory, and its own secondary storage. The machines communicate by passing messages via the high speed network that connects them.

Distributing an inverted index across the separate machines in the network results in a *segmented index*. There are two well-studied schemes for constructing a segmented index:

- *Global index organization*. In this scheme, the inverted index is partitioned by terms. Each machine holds posting lists for a distinct set of terms (the terms may be partitioned by lexicographic order, for example). The posting list for term t holds entries for all documents that include t.
- *Local index organization*. In this scheme, the inverted index is partitioned by documents. Each machine is responsible for indexing a distinct set of documents, and will hold posting lists for all terms that appeared in its set of documents.

Works that have compared the (run-time) efficiency of the above partitioning schemes obtained different results for different query models. Tomasic and Garcia-Molina [1993] simulated throughputs of systems consisting of 4 and 16 machines (hosts), for conjunctive queries. Note that in order to process conjunctive queries, the posting lists of different terms must be intersected at some single point. Their simulations indicate that a local index organization would outperform a globally-partitioned index for both short and long queries. On the other hand, Ribeiro-Neto and Barbosa [1998] used simulations coupled with an analytical model of query evaluation to compare the two architectures for disjunctive queries. In such queries, the contribution of each term to the overall score of the document is independent of the occurrences of other terms, and can be accumulated separately. They found that for TREC-3 queries, the global index architecture outperformed the local architecture. The authors left the performance comparisons on very short, Web-like queries to future work. Parallel generation of a global index has been studied in Ribeiro-Neto et al. [1998], while a system that crawls the Web and builds a distributed local index was presented in Melnik et al. [2001]. Cahoon et al. [2000] evaluated the computational performance of local indices under a variety of workloads, and Hawking [1997] examined scalability issues of local index organizations.

The prototype of Google was reported as using global index partitioning [Brin and Page 1998]. However, as Google scaled up, the architecture changed. In a recent paper, Barroso et al. [2003] reveal that today, each replica of Google's index is locally partitioned. In their terminology, the index is divided into *index shards*, with each shard containing a randomly chosen subset of documents from the full index. Barroso et al., as well as many of the above mentioned works [Cahoon et al. 2000; Ribeiro-Neto et al. 1998; Hawking 1997; Ribeiro-Neto and Barbosa 1998; Tomasic and Garcia-Molina 1993], describe essentially the same model for processing queries in systems with segmented indices:

- User queries arrive at a certain designated machine, which we will call the *Query Integrator*, or QI. This machine was called *home site* in Tomasic and

Garcia-Molina [1993], *central broker* in Ribeiro-Neto et al. [1998] and Ribeiro-Neto and Barbosa [1998], *user interface* (or UIF) in Hawking [1997], *connection server* in Cahoon et al. [2000] and *Google Web server* in Barroso et al. [2003].

- The QI issues each query to the separate index segments in a manner that depends on the partitioning scheme of the index. With local index partitioning, the QI will send the query (as submitted by the user) to all segments. With global index partitioning, the QI sends each segment a partial query consisting only of the set of terms whose posting lists are stored in the segment.
- The QI waits for the relevant segments to return their result sets, and merges these result sets with respect to the system's ranking scheme. The two index partitioning schemes imply different merge operations.

  With local index partitioning, it is usually assumed that each segment has the ability to calculate the global score of each document in its local index with respect to all queries. Since the result sets that are returned by different segments are disjoint, merging the various result sets is straightforward and relatively inexpensive.

  With global index partitioning, each (relevant) segment returns a ranked document list that may overlap lists returned by other segments, and where each score reflects only the score of the document with respect to the partial query that segment received. The QI may need to perform set operations on the partial result sets (for queries containing boolean operators), and might need to weigh the scores returned from each segment differently (for example, according to the different document-frequency values of the terms in each partial query).
- The QI returns the merged results to the users.

We consider a *cache-augmented* process, in which the QI maintains a query-result cache. Upon receiving a query from a user, the QI first checks if the cache contains results for that query. If so, the cached results are returned to the user, without forwarding the query to any of the segments. If the query cannot be answered from the cache, the QI processes the query as described above, and upon completion, caches the merged results.[4]

1.4 This Work

When considering the query processing model described above in the context of Web search engines, we note that merged results are returned to users in small batches (typically 10 at a time), in decreasing order of relevance (as ranked by the search engine). The QI, however, may prepare more results than are needed to populate the first batch, and cache them for future use. This raises the issue of optimizing the number of prefetched results in systems where the cost of processing uncached queries increases with the number of results that are fetched: prefetching a large number of results per query will be costly at

[4]The maintenance of the cache is not considered in this work. In particular, we do not examine how cached entries are replaced or how the freshness of the results is maintained.

Table I. Summary of Notations

Symbol	Denotes
a	shorthand for βA
b	shorthand for $(\omega + 2\alpha m)$
c	shorthand for $(\log C + \alpha m)$
d	shorthand for $\alpha A \log m$
m	number of segments in index
p	probability of viewing result page k when viewing page $k-1$
q	quality criterion of QI
r	number of result pages to fetch
r_{opt}	optimal integral value of r
A	number of results per result page
C	number of relevant results per segment
ω	work needed to identify results in each segment
$W(r)$	work required for fetching r result pages per query
$l_q(r, m)$	number of results to fetch from each segment so that the best rA results are collected with probability at least q; equals $\tilde{l}_q(rA, m)$
α	multiplies the computations of the QI in $W(r)$
β	multiplies the required caching space in $W(r)$

first, but may pay off should the user request additional batches of results (since these will already be cached). The tradeoff between the amount (and cost) of result prefetching and the possibility of serving subsequent queries from the cache is the main topic of this paper. As popular search engines process millions of queries every day, efficient prefetching policies can help reduce both the hardware requirements and the response time of the engines. Note that we also associate the cache space that is occupied by the prefetched results with the cost of prefetching. Assuming a fixed-size cache, increasing the number of prefetched results per query may decrease the number of queries whose results can be simultaneously cached. This may lead to lower cache hit ratios, and to an increase in the load of the engine.

Another issue arising from the query processing model, is the relationship between the number n of results that the QI decides to prefetch per query, and the number l of results that it should ask of each segment. Specifically, consider an engine that uses local index partitioning into m segments, and whose policy is to prefetch n results per query. It may happen that all of the top results reside on a particular segment. Therefore, in order to be *certain* that indeed all top n results are obtained, it is necessary to collect the top-n results of each segment (setting $l = n$). However, assuming that documents are partitioned randomly and independently into the segments, the QI may be able to collect considerably fewer results from each segment and still, with very high probability, obtain all of the top n results. Thus, when optimizing the number of prefetched results n, the behavior of l with respect to n must also be considered.

The rest of this paper is organized as follows. Section 2 formally presents the problems studied and the notations used throughout this paper. The notations are summarized there in Table I. We model both the search engine's query service process and the users' behavior. We then define the cost of prefetching a given number of results in terms of a cost function that is analyzed and

optimized in later sections. Section 3 presents an algorithm that optimizes the prefetch cost function for two special cases. The first case deals with inverted indices that fit on a single machine. This single machine scenario also models serving single-term queries (which are quite common on the Web) with a globally-partitioned index. The second special case deals with a scenario in which the engine guarantees that the users receive absolutely optimal results, using worst-case assumptions on the distribution of relevant documents in local-index partitions. The main body of work is contained in Section 4, which presents algorithms that solve and approximately solve the optimization problem for locally partitioned indices with an arbitrary number of segments, among which the documents are randomly distributed. Sections 5 and 6 tackle several aspects of the combinatorial problem of setting the number of results that should be retrieved from each segment in order to provide quality merged results to the users. Section 7 discusses the practical impact that our results may have on search engine engineering. Conclusions and suggestions for future research are made in Section 8.

2. NOTATIONS AND FORMAL MODEL

2.1 The User

Our work requires a model for the manner in which search engine users conduct search sessions (as defined in Section 1.1). Following evidence that the "deeper" the result page, the less users view it [Lempel and Moran 2003], and similar to the model adopted in Wolf et al. [2002], we chose to model the number of result pages that users view per session as a Geometric random variable $u \sim \mathcal{G}(1-p)$. According to this model, users view result pages in their natural order, and the probability of a user viewing exactly result pages $1, \ldots, k$ (not viewing result pages $k+1$ and beyond) equals $(1-p)p^{k-1}$. In other words, upon viewing a result page, the user requests the next page with probability p.

An important property of the Geometric distribution is the fact that it is memoryless:

$$Pr(u \geq s+t \mid u \geq s) = p^t \quad \forall s, t \in \mathbb{N}$$

Assume that the complexity of retrieving ranked results is also "memoryless", meaning that the complexity of retrieving the results that rank in places $n, n+1, \ldots, n+(k-1)$ depends only on the number of results retrieved, k. As we will see, this assumption holds in our model, provided that the identity of the result that ranks in place $n-1$ is known. Then, the memoryless behavior of the users and the memoryless cost of retrieval implies that the optimal number of result pages r_{opt} that should be prefetched for a query is independent of the number of result pages requested so far: any time a query cannot be served from the cache, the QI should prepare the next r_{opt} result pages.

2.2 The Index Architecture and the Complexity of Processing Queries

The model to which we refer in most of this paper is that of a *local* index partitioning scheme in a *shared-nothing* network. The index is partitioned among

m segments. We assume that documents (URLs) are partitioned into segments by a random process that assigns each document to a segment according to the uniform distribution, and independently of all other documents. Such a partitioning can be achieved by hashing every URL into a fixed-size document ID, and mapping these IDs into segments. Such a scheme was mentioned in Arasu et al. [2001] in the context of building URL repositories, and the same technique can be applied when assigning pages to the segments of an inverted index. Since the number of documents considered is in the hundreds of millions while m is considerably smaller (much less than the square root of the number of documents), the segments will contain roughly the same number of documents (with high probability). The query processing model is as described in Section 1.3. Throughout the discussion we consider the processing of a "broad topic" query that matches C documents in each segment, where C is much larger than the number of the results a user will actually browse.

Let A denote the number of results that the engine presents in each result page (a typical value is $A = 10$). Since results should be prefetched in *page units*, the number of prefetched results per query should be a multiple of A. In what follows we examine the cost of prefetching $n = rA$ results per query, so that in subsequent sections we will be able to optimize the value of r—the number of prefetched result pages. We will denote a user's query by a pair (t, k), where t is the search topic (identified by the search phrase sent by the user) and $k \geq 1$ is the (ordinal) number of the result page requested. A query can either start a new search session (and then $k = 1$), or ask for additional results in an existing session ($k > 1$). The following discussion addresses both query types.

Preliminaries. Upon receiving a query (t, k) that cannot be answered from the cache, the QI needs to fetch n results for t. The first task is to set the value of l, the number of results to retrieve from each of the m segments.

Let $\mathcal{B}(n)$ denote the set of n documents that the engine should ideally retrieve for the query: the n documents that attain the best scores for t (according to the engine's ranking function), out of all documents that have not been retrieved for queries (t, k'), $k' < k$. Let $\mathcal{R}(l, m)$ denote the set of documents that will actually be retrieved for the query (t, k) when each of the m segments returns its l most relevant (and previously unretrieved) matches for t. Ideally, $\mathcal{R}(l, m)$ would contain $\mathcal{B}(n)$, but ensuring that means setting l to equal n.[5] Instead, we assume that the engine employs the following *quality policy*, based on a probability q: The QI sets the value of l with respect to n such that

$$Pr[\mathcal{B}(n) \subseteq \mathcal{R}(l, m)] \geq q.$$

In other words, the QI should collect enough (previously unretrieved) quality results from each segment so that with probability q, the top-n retrieved results will indeed be the best n (previously unretrieved) results for t in the entire index. The relationship between n and l will be studied in Section 5. For the time being, it suffices to note that by the assumption that documents are uniformly

[5]This special case is discussed in Section 3.

distributed among the segments, the above probability depends only on the values of n, m and l, and is independent of the topic t.

Let $\tilde{l}_q(n, m)$ denote the minimal number of documents that should be retrieved by each of the m segments so that the quality criterion is satisfied:

$$\tilde{l}_q(n, m) \stackrel{\triangle}{=} \min\{l \mid Pr[\mathcal{B}(n) \subseteq \mathcal{R}(l, m)] \geq q\}.$$

Collecting results. The QI sends each segment the query (t, k) and a request for its $\tilde{l}_q(n, m)$ top results for the query. Whenever $k > 1$ (this is not the first batch of results to be retrieved for t), segment i also receives the score and id of the lowest ranking document that it had contributed to the results of $(t, 1), \ldots, (t, k-1)$.[6] We now estimate the cost of serving such requests. By our assumption, the query matches C documents in each segment, where C is much larger than the number of results users will actually browse through, and consequently is much larger than $\tilde{l}_q(n, m)$ (since $\tilde{l}_q(n, m)$ is bounded by n, and n is bounded by the number of results that users browse through). Let ω be the work required from each segment for identifying the C-sized set of candidate documents (in the inverted index structure, when the query is the disjunction of several terms, ω is linear in C). Recall that each segment receives the score and id of the lowest ranking document that it had contributed so far for the query, and can thus discard previously retrieved results from the set of candidates.[7] The top-scoring $\tilde{l}_q(n, m)$ documents of the remaining candidates are then found. Each segment will thus spend $\Theta(\omega + \tilde{l}_q(n, m) \log C)$ processing steps (per query) in order to return $\tilde{l}_q(n, m)$ sorted results to the QI.

Merging results. The QI receives m sorted result lists of length $\tilde{l}_q(n, m)$. Reading and buffering these lists takes $\Theta(m\tilde{l}_q(n, m))$ operations. It then partially merges the results until it identifies the top $n = rA$ retrieved results that will populate the r result pages. By using *Tree Selection Sorting* [Knuth 1998] with the m sorted result lists hanging from the leaves of the tree, the merge can be accomplished in time $\Theta(2m + n \log m)$. The overall complexity of this step is thus $\Theta(m\tilde{l}_q(n, m) + 2m + n \log m)$.

Caching results. The r result pages are cached, and the first of those pages is returned to the user. The m scores $s_1(t, k), \ldots, s_m(t, k)$ are also noted. The overall space complexity is thus $\Theta(rA + m)$.

The complexity of the query processing model. Our model requires two messages to be passed between the QI and each of the segments: the QI sends the query to each segment, and each segment returns $\tilde{l}_q(n, m)$ results to the QI. The total number of results received by the QI is $m\tilde{l}_q(n, m)$, and this amount of data impacts its time complexity. Had we allowed more rounds of communication, we could have managed by sending the QI only $m + (n-1)$ results, lowering the complexity of the merge step above to $\Theta(m + n + n \log m)$. We chose not do so since minimizing communication rounds between machines (even at

[6] We assume that the results of the query $(t, k-1)$ are still cached when the query (t, k) arrives.
[7] For the sake of simplicity, we assume that if two documents obtain the same score for a query, the tie is broken by comparing their ids.

the expense of sending larger messages) is likely to improve performance in distributed computations [Hawking 1997; Arasu et al. 2001].

Note that the complexity of the retrieval model described above is indeed "memoryless" (see discussion in Section 2.1). The model implies the following computational loads on the various resources of the engine, when following a policy of prefetching r result pages per query:

- The QI performs $\Theta(m\tilde{l}_q(rA, m) + 2m + rA\log m)$ computation steps.
- Each index segment performs $\Theta(\omega + \tilde{l}_q(rA, m)\log C)$ computations.
- The cache space required is $\Theta(rA + m)$.

Additionally, we introduce two non-negative coefficients α and β that will allow us to assign different weights to the three resources that are consumed during query processing. Specifically, α will multiply the computations of the QI and β will multiply the cache space required.[8] Tuning the values of α and β can emphasize memory (cache) limitations, computational bottlenecks (the QI vs. the segments) and response time per query. More on this in Section 7.2.

We are now ready to formulate $\widetilde{W}(r)$, the expected cost (or *work*) of serving a search session, for geometric users with parameter p, by a policy that prefetches r pages per query execution from a locally segmented index. Note that a session may require multiple query executions, where every execution prepares a batch of r result pages. Accordingly, result pages $ir+1, ir+2, \ldots, (i+1)r$ will be termed as the i'th *batch* of pages. For ease of notation, we introduce $l_q(r, m) \stackrel{\triangle}{=} \tilde{l}_q(rA, m)$.

$$\begin{aligned}
\widetilde{W}(r) &= \text{Caching overhead} + \\
&\quad \sum_{i=1}^{\infty}[Pr(\text{preparing batch } i)\cdot \\
&\qquad \text{(batch preparation complexity)}] \\
&= \beta(Ar + m) + \\
&\quad \sum_{i=0}^{\infty} p^{ir}[\omega + l_q(r, m)\log C + \\
&\qquad \alpha(rA\log m + m l_q(r, m) + 2m)] \\
&= \beta(Ar + m) + \frac{\omega + l_q(r, m)\log C}{1 - p^r} + \\
&\quad \frac{\alpha(rA\log m + m l_q(r, m) + 2m)}{1 - p^r}.
\end{aligned}$$

It is evident that the constant additive term βm does not depend on r and will not affect the optimization of $\widetilde{W}(r)$. Thus, rearranging the terms, optimizing $\widetilde{W}(r)$ is equivalent to optimizing

$$W(r) = \beta Ar + \frac{(\omega + 2\alpha m) + (\log C + \alpha m)l_q(r, m) + (\alpha A\log m)r}{1 - p^r}.$$

[8]The computational loads were expressed using the $\Theta(\cdot)$ notation. For concreteness and simplicity, we will consider the given expressions as the *exact* complexities. This allows us to avoid tedious notations, and does not affect the ensuing analysis (and results) of the paper.

To ease the notation, we define the following constants: $a = \beta A$, $b = (\omega + 2\alpha m)$, $c = (\log C + \alpha m)$ and $d = \alpha A \log m$. With this notation,

$$W(r) = ar + \frac{b + cl_q(r, m) + dr}{1 - p^r}.$$

C, the number of documents per segment that match a query, and consequently ω, the amount of work needed to identify these documents, is typically a large number, while A and m are typically much smaller. Thus, when the proportionality constants α and β are both about 1, typical values of b are large (tens of thousands and beyond), while a, c, d are relatively small (typically less than 100).

For convenience, Table I summarizes the notations we use.

Our mission: Given an m-way locally segmented index, geometric-p users and some quality criterion q, determine r_{opt}, an integral value of r, which minimizes $W(r)$. In doing so, determine $l_q(r_{opt}, m)$. The QI will then prepare r_{opt} result pages whenever a query cannot be answered from the cache, asking each of the m segments to retrieve its top $l_q(r_{opt}, m)$ results (that score below a certain threshold) for the query being processed. We will strive to obtain exact or almost exact values of r_{opt} and $l_q(r_{opt}, m)$.

3. SIMPLE SPECIAL CASES

In this section we show that the problem for a single segment ($m = 1$) and the problem for multiple segments with $q = 1$ behave similarly, and in both cases r_{opt} can be found in $\Theta(\log r_{opt})$ steps.

- When the index is stored in a single segment, we can ignore the terms in the complexity function $W(r)$ that deal with the merging of results from different segments (namely the terms involving α). In addition, $l_q(r, 1) = rA$ regardless of q's value. Thus, $W(r)$ becomes:

  $$W(r) = ar + \frac{\omega + (A \log C)r}{1 - p^r}.$$

  Note that when an index is partitioned globally (each segment holds posting lists for a distinct set of terms), single-term queries are effectively queries to a single segment as described above. Studies [Silverstein et al. 1998; Jansen et al. 2000] indicate that the percentage of single-term queries on the Web is quite large (25%–30%).
- For the case where $q = 1$ we again have $l_1(r, m) = rA$, and the complexity function $W(r)$ takes the following form:

  $$W(r) = ar + \frac{b + (cA + d)r}{1 - p^r}.$$

Both cases imply a complexity function of the form

$$W(r) = ar + \frac{b' + d'r}{1 - p^r} \quad , \quad b', d' > 0.$$

Taking the derivative, we have

$$\begin{aligned} W'(r) &= a + \frac{d'(1-p^r) + [(b'+d'r)\ln p]p^r}{(1-p^r)^2} \\ &= \frac{a(1-p^r)^2 + d'(1-p^r) + [(b'+d'r)\ln p]p^r}{(1-p^r)^2} \\ &= \frac{(1-p^r)(d'+a-ap^r) + [(b'+d'r)\ln p]p^r}{(1-p^r)^2}. \end{aligned}$$

Thus, for $r > 0$, the sign of $W'(r)$ is determined by the sign of

$$(1-p^r)(d'+a-ap^r) + (b'+d'r)(\ln p)p^r$$

or, equivalently, by the sign of

$$f(r) = ap^{2r} - (2a + d' - b'\ln p - (d'\ln p)r)p^r + (d'+a).$$

Note that f is continuous at 0, and that

$$f(0) = a - 2a - d' + b'\ln p + d' + a = b'\ln p < 0.$$

Thus, $W'(r)$ is negative for small positive values of r, so $W(r)$ decreases at first. We now examine the derivative of $f(r)$:

$$\begin{aligned} f'(r) &= (2a\ln p)p^{2r} + (d'\ln p)p^r - [2a + d' - b'\ln p - (d'\ln p)r](\ln p)p^r \\ &= [2ap^r + d' - 2a - d' + b'\ln p + (d'\ln p)r](\ln p)p^r \\ &= [2a(p^r - 1) + \ln p(b' + d'r)](\ln p)p^r \;>\; 0. \end{aligned}$$

Thus, $f(r)$ is negative at zero but monotonically increases. This implies that $W(r)$ (for positive values of r) decreases at first until it reaches its (unique) minimal value, and then increases. Relying on this behavior, an optimal integral value of r (r_{opt}) can be found by applying the following Bracket and Bisection scheme [Press et al. 1988]:

1. Find the minimal natural number n such that $W(2^n) \leq W(2^{n+1})$. This step brackets r_{opt} in the interval $[2^{n-1}, 2^{n+1}]$.
2. Find an optimal value of r, using binary search, in the interval $[2^{n-1}, 2^{n+1}]$.

Since n will not exceed $\lceil \log r_{opt} \rceil$, the complexity of finding r_{opt} is $\Theta(\log r_{opt})$.

4. SOLUTION FOR AN *m*-WAY SEGMENTED LOCAL INDEX

In this section we study the problem of setting the optimal value of r given the quality criterion q ($q < 1$), the engine's architecture parameters A, C and m, and the complexity parameters α and β. Subsection 4.1 presents an algorithm for determining the optimal value of r, which minimizes the retrieval complexity function $W(r)$. Subsection 4.2 presents an approximation algorithm, which finds a value of r for which $W(r)$ is approximately optimal.

4.1 Optimizing r in Indices With m Segments

First, recall the form of the complexity function from Section 2.2:

$$W(r) = ar + \frac{b + cl_q(r, m) + dr}{1 - p^r}.$$

Clearly, the behavior of $W(r)$ depends on the behavior of $l_q(r, m)$. While we will show how to precisely calculate $l_q(r, m)$ in Section 5, for the purpose of this subsection it suffices to note that if $r' \geq r$ then $l_q(r', m) \geq l_q(r, m)$.

In order to facilitate the search for r_{opt}, we now seek to find, for every value of r, an upper bound on the set $\{\bar{r} \mid W(\bar{r}) \leq W(r)\}$.[9]

Definition 1. A function $g(r)$ will be called *W-restrictive* if for all $r' \geq g(r)$, $W(r') > W(r)$.

For example, $g_1(r) \stackrel{\triangle}{=} \frac{W(r)}{a}$ is W-restrictive, since for all $r' \geq g_1(r)$, we have $W(r') > ar' \geq a g_1(r) = W(r)$. Consequently, r_{opt} is not larger than $g_1(1)$.

We will use W-restrictive functions to bound our search space for r_{opt}. For this we now present a W-restrictive function that is better than g_1, providing tighter bounds on the size of the search space.

PROPOSITION 1. *The function*

$$g(r) = r + \frac{p^r(b + cl_q(r, m) + dr)}{(1 - p^r)(a + d)}$$

is W-restrictive.

PROOF. By rearranging terms in the definition of $g(r)$ we get:

$$(a + d)(g(r) - r) - \frac{p^r}{1 - p^r}(b + cl_q(r, m) + dr) = 0 \tag{1}$$

We shall also use the following equality, which holds for all constants X, Y ($Y \neq 1$):

$$\frac{X}{1 - Y} = X + \frac{XY}{1 - Y} \tag{2}$$

Let $r' \geq g(r)$, and let us evaluate $W(r') - W(r)$:

$$\begin{aligned}
W(r') - W(r) &= a[r' - r] + \frac{b + cl_q(r', m) + dr'}{1 - p^{r'}} - \frac{b + cl_q(r, m) + dr}{1 - p^r} \\
&> a[r' - r] + [b + cl_q(r', m) + dr'] - \frac{b + cl_q(r, m) + dr}{1 - p^r} \\
&= a[r' - r] + [b + cl_q(r', m) + dr'] - (b + cl_q(r, m) + dr) - \\
&\quad \frac{p^r}{1 - p^r}(b + cl_q(r, m) + dr) \quad \text{(by equation 2 above)} \\
&\geq a[g(r) - r] + [b + cl_q(r', m) + dg(r)] - (b + cl_q(r, m) + dr) - \\
&\quad \frac{p^r}{1 - p^r}(b + cl_q(r, m) + dr) \quad (\text{since } r' \geq g(r))
\end{aligned}$$

[9] Since $\lim_{r \to \infty} W(r) = \infty$, the set $\{\bar{r} \mid W(\bar{r}) \leq W(r)\}$ is finite for all r.

1. Initializations: $W_{min} \leftarrow \infty$, $r_{opt} \leftarrow 1$, limit $\leftarrow \infty$, $r \leftarrow 1$.
2. While $r <$ limit:
   (a) Calculate $l = l_q(r, m)$, and use the value to set $W \leftarrow W(r)$, $g \leftarrow g(r)$.
   (b) If limit $> g$: limit $\leftarrow g$.
   (c) If $W < W_{min}$: $W_{min} \leftarrow W$, $r_{opt} \leftarrow r$.
   (d) $r \leftarrow r + 1$
3. print r_{opt}.

Fig. 1. Algorithm OP for optimizing the prefetch policy.

$$
\begin{aligned}
&= (a+d)(g(r)-r) + [cl_q(r', m) - cl_q(r, m)] - \\
&\quad \frac{p^r}{1-p^r}(b + cl_q(r, m) + dr) \\
&\geq (a+d)(g(r)-r) - \frac{p^r}{1-p^r}(b + cl_q(r, m) + dr) \\
&= 0 \text{ (by equation 1).}
\end{aligned}
$$

Thus, for all $r' \geq g(r)$, $W(r') > W(r) \implies g(r)$ is W-restrictive. □

Note that $g(r)$ reflects all the architectural parameters of the search engine's index, and also the user's behavior (represented by p) and the desired quality criterion q.

Figure 1 displays Algorithm OP for setting the optimal value of r. The algorithm conducts a linear search for r_{opt}. Starting with $r = 1$, r is incremented as long as it does not exceed the upper bound on r_{opt}, set by the W-restrictive function $g(r)$. That bound is updated (lowered) during the computations. All of the steps except the calculation of $l_q(r, m)$ in 2(a) are trivial; that calculation is the topic of Section 5. The correctness of the algorithm follows from the W-restrictiveness of g(r) (Proposition 1), since we do not need to iterate through values of r for which $W(r)$ is known to be higher than values we have already seen.

The complexity of the algorithm. Algorithm OP needs to be executed only in preprocessing phases, when configuring the prefetching policy of the search engine (see discussion in Section 7.2). Therefore, its own complexity does not impact the performance of the engine. Nevertheless, we now prove that its running time is polynomial. We do so by bounding r_{max}, the maximal number of iterations that OP may require throughout its course. For this, let

$$\tau = \frac{a+d}{b+cA+d}.$$

Note that by our assumptions on the relative values of a, b, c and d (see Section 2.2), τ is a small (positive) constant. Since $l_q(1, m) \leq A$, we have that $g(1) \leq 1 + \frac{p}{(1-p)\tau}$ is a bound on the number of iterations. Thus, r_{max} is bounded by $1 + \frac{2p}{1-p}$ whenever $\tau \geq 0.5$, and by $1 + \frac{1}{1-p}$ whenever $p \leq \tau$. Next, we bound r_{max} when $p > \tau$ and $\tau < 0.5$.

Table II. $r_{opt}(r_{max}^{act})$ Values as a Function of m and p

$m \backslash p$	0.3	0.5	0.7
5	4(5)	7(9)	11(15)
25	4(5)	6(8)	12(14)
50	4(5)	6(8)	10(13)

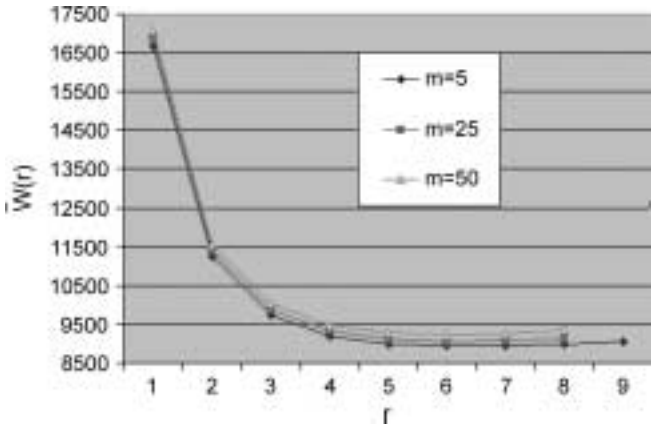

Fig. 2. $W(r)$ as a function of r, for $m = 5, 25$ and 50 ($p = 0.5$).

LEMMA 1. *If $p > \tau$ and $\tau < 0.5$, then $r_{max} < 3\lceil \frac{\log \tau}{\log p} \rceil$.*

PROOF. Let $r = \lceil \frac{\log \tau}{\log p} \rceil$. Then, since $1 > p > \tau$, $r > 1$ and $p^r = 2^{r \log p} \leq 2^{\log \tau} = \tau$. Thus,

$$\begin{aligned} g(r) &= r + \frac{p^r}{(1-p^r)} \frac{b + cl_q(r,m) + dr}{a+d} \\ &\leq r + \frac{\tau}{(1-\tau)} \frac{b + cl_q(r,m) + dr}{a+d} \\ &\leq r + \frac{\tau}{(1-\tau)} \frac{b + crA + dr}{a+d} \\ &\quad \text{(since obviously } l_q(r,m) \leq rA\text{)} \\ &< r + \frac{\tau}{(1-\tau)} \frac{r}{\tau} = \frac{2-\tau}{1-\tau} r < 3r = 3 \left\lceil \frac{\log \tau}{\log p} \right\rceil . \end{aligned}$$

□

To complete the analysis of the complexity of algorithm OP for finding r_{opt}, we show in Section 5 that calculating the values of $l_q(r, m)$ for all $r \in \{1, \ldots, r_{max}\}$ requires $\mathcal{O}(m^2 A^2 r_{max}^2)$ steps (regardless of the value of q). Since we have already bounded r_{max} by simple functions of m, p and τ, bounds on the complexity of the algorithm follow.

Table II gives sample results of the algorithm. For every combination of m and p, r_{opt} and r_{max}^{act} (the highest value of r for which $W(r)$ was actually calculated during execution) are shown. Figure 2 is a plot of $W(r)$ as a function

of r, as calculated during the algorithm for three values of m with $p = 0.5$. For all displayed results, we used $q = 0.99, \alpha = \beta = 1$, $A = 10$ and $C = 2^{13}$.

4.2 Approximating the Optimal Solution

In the previous subsection we have shown how to determine r_{opt}, the number of pages that minimizes the complexity function $W(r)$. However, if we are willing to settle for nearly optimal solutions, namely finding values of r for which $\frac{W(r_{opt})}{W(r)} \geq 1 - \epsilon$ for small values of ϵ, we can use the following algorithm:

1. Let $r_{max} \stackrel{\triangle}{=} \lceil \frac{\log \epsilon}{\log p} \rceil$.
2. Find the value of r in the range $\{1, \ldots, r_{max}\}$ that minimizes $W(r)$.

Note that r_{max} depends on the user's behavior (as modeled by p) but is independent of the engine's architecture and quality policy (which are modeled by a, b, c, d and q). Furthermore, the above algorithm is applicable to *any* work function $\tilde{W}(r)$ such that $(1 - p^r)\tilde{W}(r)$ is an increasing function of r. Note that $W(r)$ satisfies this condition, since

$$(1 - p^r)W(r) = (1 - p^r)ar + b + cl_q(r, m) + dr$$

where a, b, c, d are positive constants, and the functions $(1 - p^r), l_q(r, m)$ are nondecreasing functions of r.

The correctness of the approximation algorithm relies on the following Proposition.

PROPOSITION 2. *Let $W(r)$ be any positive function such that $(1 - p^r)W(r)$ is an increasing function of r. Let $r, t \in \mathbb{N}$ such that $\frac{W(t)}{W(r)} \geq \frac{1}{1-p^t}$. Then, for all $r' \geq t$, $W(r') > W(r)$.*

PROOF. Since $(1 - p^t)W(t) \geq W(r)$, we have for $r' \geq t$

$$W(r') > (1 - p^{r'})W(r') \geq (1 - p^t)W(t) \geq W(r). \qquad \square$$

COROLLARY 1. *Let $0 < s < t$. Then $W(s) < \frac{W(t)}{(1-p^s)}$.*

PROOF. Since $(1 - p^r)W(r)$ increases with r, we get

$$W(s) < \frac{W(s)}{(1 - p^t)} < \frac{W(t)}{(1 - p^s)}. \qquad \square$$

COROLLARY 2. *For all s,*

$$\min\{W(1), \ldots, W(s)\} < \frac{W(r_{opt})}{1 - p^s}.$$

PROOF. If $1 \leq r_{opt} \leq s$, the claim holds. Otherwise, the result is implied by Corollary 1, with $t = r_{opt}$. $\square$

Table III. r_{max} as a Function of p and ϵ

$\epsilon \backslash p$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.1	1	2	2	3	4	5	7	11	22
0.01	2	3	4	6	7	10	13	21	44
0.001	3	5	6	8	10	14	20	31	66

Substituting $s = r_{max} = \lceil \frac{\log \epsilon}{\log p} \rceil$ in the last corollary yields the approximation algorithm:

$$\min\{W(1), \ldots, W(s)\} < \frac{W(r_{opt})}{1 - p^{\lceil \frac{\log \epsilon}{\log p} \rceil}} = \frac{W(r_{opt})}{1 - 2^{\log p \lceil \frac{\log \epsilon}{\log p} \rceil}} < \frac{W(r_{opt})}{1 - \epsilon}.$$

Table III shows the values of r_{max} for $p = 0.1, 0.2, \ldots, 0.9$ and $\epsilon = 0.1, 0.01$ and 0.001. As mentioned earlier, calculating the values of $l_q(r, m)$ for all $r \in \{1, \ldots, r_{max}\}$ requires $\mathcal{O}(m^2 A^2 r_{max}^2)$ computational steps, and thus the time complexity of the approximation algorithm is $\mathcal{O}(m^2 A^2 \lceil \frac{\log \epsilon}{\log p} \rceil^2)$.

Finally, we note that the results of this subsection may be used in practice to improve the running time of Algorithm OP (Figure 1), by checking (between steps 2(*b*) and 2(*c*)) whether $\frac{W}{W_{min}} \geq \frac{1}{1-p^r}$, and setting limit $\leftarrow r$ if so (thus terminating the algorithm). Proposition 2 asserts that all future iterations with larger values of r will result in greater values of $W(r)$, and so OP can safely terminate and output the current value of r_{opt}.

5. CALCULATING $l_q(r, m)$

This section brings recursive formulae with which $l_q(r, m)$ can be calculated in a time that is polynomial in m, r and A.

We model the distribution of the top results in the segments by the following random process: $n \stackrel{\triangle}{=} r A$ different balls (the top results for a query) are thrown randomly and independently into m different bins (the segments), where n_i balls are inserted to bin i (i.e., $\sum_{i=1}^{m} n_i = n$). We model the querying process by taking $\min\{l, n_i\}$ balls from bin i for $i = 1, \ldots, m$. Denote by $e_{n,m,l}$ the number of *excess* balls that remain in the bins after the querying process is completed. In the next two subsections we will calculate the distribution of $e_{n,m,l}$.

In subsection 5.1 we will calculate the probability that $e_{n,m,l} = 0$. This corresponds to the case where no bin contains more than l balls, so that the QI has indeed managed to collect the top n results from the segments. In subsection 5.2 we will calculate the probability that $e_{n,m,l} = k$. This corresponds to the case where the QI managed to collect just $n - k$ of the top n results. We bring this analysis as well since the QI may choose to employ a relaxed quality policy, requiring that with high probability, *most* (but not necessarily all) of the top results are returned to the user. In subsection 5.3 we briefly review previous work on related issues.

We first present a rough bound on $l_q(r, m)$ that may suffice when precise calculations are not essential. Clearly, $\frac{n}{m}$ is a lower bound on $l_q(r, m)$ for all

Table IV. $l_q(r, m)$ for $q = 0.99$, $A = 10$ and Various Values of r and m

$m \backslash r$	1	2	3	4	5	6	7	8	9	10	11	12
5	6	10	13	16	19	22	24	27	30	32	35	37
25	4	5	6	7	8	9	10	10	11	12	13	13
50	3	4	5	5	6	6	7	8	8	8	9	9

$q > 0$. We show that $l_q(r, m)$ need not be larger than

$$\max\left\{\lceil \lambda + \log m \rceil, \left\lceil \frac{2en}{m} \right\rceil\right\} \tag{3}$$

where $\lambda \stackrel{\triangle}{=} -\log_2(1-q)$ (i.e., $q = 1 - \frac{1}{2^\lambda}$): [10]

The probability that *exactly* i of the n results are inserted to a given segment is $\binom{n}{i}(\frac{1}{m})^i(1-\frac{1}{m})^{m-i}$. Since $\binom{n}{i} \le (\frac{ne}{i})^i$ [Motwani and Raghavan 1995],

$$\binom{n}{i}\left(\frac{1}{m}\right)^i\left(1-\frac{1}{m}\right)^{n-i} < \binom{n}{i}\left(\frac{1}{m}\right)^i \le \left(\frac{ne}{i}\right)^i\left(\frac{1}{m}\right)^i = \left(\frac{ne}{mi}\right)^i.$$

Hence, the probability that more than ℓ results are inserted into a given segment is bounded by

$$\sum_{i=\ell+1}^{n}\left(\frac{ne}{mi}\right)^i < \sum_{i=\ell+1}^{\infty}\left(\frac{ne}{m\ell}\right)^i = \left(\frac{ne}{m\ell}\right)^\ell \frac{\frac{ne}{m\ell}}{1-\frac{ne}{m\ell}}.$$

Whenever $\ell \ge \max\{(\lambda + \log m, \frac{2en}{m}\}$, this last expression is bounded from above by $(\frac{1}{2})^{\lambda+\log m} = \frac{1}{m2^\lambda}$. Thus, by the union bound, the probability that at least one of the m segments contains more than ℓ results is smaller than $\frac{1}{2^\lambda}$. The results follow.

5.1 Retrieving All of the Top Results

We now turn to the precise calculation of $l_q(r, m)$. For this we will calculate the probability

$$P(n, m, l) = Pr[e_{n,m,l} = 0],$$

the probability of throwing n different balls into m different bins so that no bin contains more than l balls (see some sample values in Table IV).

The size of the problem space is m^n. We will actually be counting $N(n, m, l)$, the number of ways to throw n different balls into m different bins so that no bin contains more than l balls, and then

$$P(n, m, l) = N(n, m, l)/m^n.$$

The following recursive formulae may be used to calculate the $N(n, m, l)$ values:

$$N(n+1, m, l) = m \cdot \sum_{j=0}^{l-1}\binom{n}{j}N(n-j, m-1, l)$$

$$N(n, m+1, l) = \sum_{j=0}^{l}\binom{n}{j}N(n-j, m, l).$$

[10] Sharper asymptotic bounds on $l_q(r, m)$ are discussed in Section 5.3.

The first formula represents choosing one of m bins for ball number $n+1$, and then putting some number j, $0 \leq j \leq l-1$ of additional balls into the same bin. The rest of the $n-j$ balls then need to be put into $m-1$ bins, respecting the limit of l balls per bin. The second formula represents choosing j, $0 \leq j \leq l$ balls to populate bin $m+1$, and distributing the remaining $n-j$ balls into m bins, respecting the limit of l balls per bin.

However, the recursion that most naturally fits in Algorithm OP from Section 4.1 is

$$N(n, m, l) = \sum_{j=0}^{\lfloor \frac{n}{l} \rfloor} \binom{m}{j} \binom{n}{l, \ldots, l, n-jl} N(n-jl, m-j, l-1).$$

First, we choose some j bins to have exactly l balls. We then choose the balls to populate those bins (the multinomial coefficient has j l-terms). The remaining $n-jl$ balls are distributed to the remaining $m-j$ bins, with each such bin collecting no more than $l-1$ balls.

As r grows in subsequent iterations OP, so will the value of $l_q(r, m)$. This recursion naturally uses results of $N(n, m, l)$ from previous iterations in later iterations. As for the initial values:

1. For all m, l, $N(0, m, l) = 1$.
   Whenever $n > 0$, $N(n, 0, l) = N(n, m, 0) = 0$.
2. For all $n > 0$, $m > 0$:
   - Whenever $l < \lceil \frac{n}{m} \rceil$, $N(n, m, l) = 0$. This implies that the sum over j in the above formula contains no more than $1 + m$ terms.
   - $N(n, m, \lceil \frac{n}{m} \rceil) = \binom{m}{k} \frac{n!}{\lceil \frac{n}{m} \rceil^k \lfloor \frac{n}{m} \rfloor^{m-k}}$, where $k \stackrel{\triangle}{=} n \bmod m$.

Denoting by $n_{max} \stackrel{\triangle}{=} r_{max} A$ the value of n in the last iteration of OP (and by l_{max} the value of l found in that iteration), the total time spent calculating values of $l_q(r, m)$ can be shown to be $\mathcal{O}(n_{max} m l_{max}(m+1)) = \mathcal{O}(n_{max}^2 m^2)$. Furthermore, when q is constant and n_{max} grows, Equation 3 asserts that $l_{max} m \leq 6 n_{max}$ and so $\mathcal{O}(n_{max} m l_{max}(m+1)) = \mathcal{O}(n_{max}^2 m)$. Table IV shows sample values of $l_q(r, m)$.

5.2 Retrieving Most of the Top Results

Consider next the QI's willingness to produce less than optimal results, allowing at most k of the top $n = rA$ results to not appear in the r result pages. In terms of the balls-into-bins terminology, we are allowing some bins to receive more than l balls each (thus losing the excess results), as long as the total excess lost does not exceed k. Our goal in this subsection is to calculate $Pr[e_{n,m,l} \leq k]$.

Denote the number of balls that were thrown into bin i by n_i, and let f denote the number of *overflowing* bins (the number of bins into which more than l balls were thrown):

$$f \stackrel{\triangle}{=} |\{n_i,\ 1 \leq i \leq n\ :\ n_i > l\}|.$$

How many balls were actually thrown into these f bins? Certainly we must allow at least $f(l+1)$ balls to be assigned to these f bins, in order to cause their overflow. However, we cannot assign more than $fl + k$ balls to the overflowing

bins, otherwise the excess will surely exceed k. Therefore, the number of balls in the overflowing bins, t, must respect the following bounds:

$$f(l+1) \leq t \leq fl + k.$$

Once we set the values of f and t, we need to factor in the following:

1. The identities of the f (out of the possible m) bins that will overflow.
2. The identities of the t (out of the possible n) balls that will occupy the overflowing bins.
3. The number of ways to throw t balls into f bins so that *all* bins will receive more than l balls.
4. The number of ways to throw $n - t$ balls into $m - f$ bins so that no bin receives more than l balls.

Denoting the third quantity by $\bar{N}(t, f, l)$, we have:

$$Pr[e_{n,m,l} \leq k] = \frac{1}{m^n} \sum_{f=0}^{k} \binom{m}{f} \sum_{t=f(l+1)}^{fl+k} \binom{n}{t} \bar{N}(t, f, l) N(n-t, m-f, l).$$

So we are left with the task of calculating $\bar{N}(t, f, l)$, the number of ways to throw t balls into f bins so that all bins receive more than l balls. By the inclusion-exclusion principle,

$$\bar{N}(t, f, l) = \sum_{i=0}^{f} (-1)^i \binom{f}{i} \sum_{j=0}^{il} \binom{t}{j} N(j, i, l)(f-i)^{t-j}.$$

In the sign-alternating sum, we first pick i bins that will *not* overflow. Then, we decide how many balls these bins will hold (0 to il). We choose the balls, arrange them legally in $N(j, i, l)$ ways, and distribute the remaining $t - j$ balls into the other $f - i$ bins arbitrarily.

Complexity Issues. Let $l_q^k(n, m)$ denote the smallest integer l such that $Pr[e_{n,m,l} \leq k] \geq q$. We now analyze the complexity of calculating the values of $l_q^k(n, m)$ for a fixed value of m and for all values of n in the range $1, \ldots, n_{max}$. Let l_{max} denote $l_q^k(n_{max}, m)$.

As discussed in the previous subsection, calculating $N(n, m, l)$ for all values of n in the range $1, \ldots, n_{max}$ requires $\Theta(n_{max} m^2 l_{max})$ operations. Turning our attention to the calculation of $\bar{N}(t, f, l)$, we note that f is confined to the range $0, \ldots, k$ and t is confined to the range $0, \ldots, kl_{max}$. Thus, given the values of $N(n, m, l)$ for all $n \leq n_{max}, l \leq l_{max}$, the required values of $\bar{N}$ can be calculated in $\Theta(k^4 l_{max}^2)$.

After accounting for all values of N and $\bar{N}$, we note that calculating the probability $Pr[e_{n,m,l} \leq k]$ requires k^2 operations for specific values of n and l. Thus, we require $\mathcal{O}(n_{max} l_{max} \cdot k^2)$ operations in order to find $l_q^k(n, m)$ for all values of $n \leq n_{max}$, given the values of N and $\bar{N}$.

Overall, the complexity of calculating $l_q^k(n, m)$ for all values of $n \leq n_{max}$ is

$$\mathcal{O}\left(n_{max} l_{max} k^2 + n_{max} m^2 l_{max} + k^4 l_{max}^2\right) = \mathcal{O}\left(n_{max}^2 \left(m^2 + k^4\right)\right).$$

By substituting the known bounds on $n_{max} = r_{max} A$ from Section 4.1, we can bound the complexity of algorithm OP when the engine tolerates (with probability q) up to k non-optimal results in the $r_{opt} A$ pages which it presents to the user.

5.3 Previous Work

The stochastic properties of the process that randomly throws n balls into m bins have been studied extensively. Two good references are Kolchin et al. [1978] and Johnson and Kotz [1977]. Among the properties studied was the distribution of the maximum number of balls in a bin, which we will denote by $L(n, m)$. For example, for $n \geq m$ (more balls than bins), $L(n, m) = \Theta(\frac{\ln m}{\ln \lceil 1+\frac{m}{n} \ln m \rceil} + \frac{n}{m})$ with probability $1-o(1)$ [Czumaj and Stemann 1997]. When $n = m$, $L(n, m)$ behaves asymptotically as $(1 + o(1))\frac{\ln n}{\ln \ln n}$ with probability $1 - o(1)$ [Azar et al. 1999]. In Kolchin et al. [1978], the distribution of $L(n, m)$ is examined with regard to the behavior of the ratio $\frac{n}{m \ln m}$ as $n, m \to \infty$. Separate results are obtained for the three cases $\frac{n}{m \ln m} \to 0$, $\frac{n}{m \ln m} \to \lambda > 0$, and $\frac{n}{m \ln m} \to \infty$. In Johnson and Young [1960] it was shown that the distribution of $L(n, m)$ may be approximated by the the distribution of

$$n \cdot \max \frac{s_j}{\sum_{j=1}^{m} s_j},$$

where each s_j is an independent χ^2 variable with $\frac{2(n-1)}{m}$ degrees of freedom.

6. NON-OPTIMAL MERGE RESULTS

In Section 5 we computed the minimal number of results l that the QI should fetch from each segment per query in order to ensure, with probability no less than $q (q < 1)$, that the n merged results meet some quality criterion. In Section 5.1, the quality criterion was that the n merged results be the top-n results for the query. Section 5.2 relaxed that demand to allow some of the top-n results to be lost; l was set so that with probability q, *most* of the top-n results would be returned. Essentially, the a-priori probabilities of losing quality results during the merge process were studied.

Whether the QI's policy allows some results to be lost or not, l is set so that with probability of about $1-q$, the merge process will fail to meet the QI's standard of quality, losing more of the top-n results than is acceptable. This section deals with such cases. Section 6.1 presents how the QI, as it is merging the $l \cdot m$ results returned by the segments, can estimate *online* how many of the top-n results it may be losing. Section 6.2 discusses an additional measure of non-optimal merge results called *skipped results*, and proves a strong correlation between skipped results and lost results.

6.1 Online Estimation of the Number of Lost Results

We now show that during the merge process, the QI can assess the quality of the merged results. In particular, we show that the QI can determine the probability q_k that at most k of the top-n results have been lost, replaced by nonoptimal results in the merged process.

Let us recall the merge process that takes place in the QI. Results from m sorted lists of size l are merged so as to find the top $n \stackrel{\triangle}{=} r\,A$ results. As long as the merge process does not *exhaust* a list, that is as long as each list contributes to the best results at most $l-1$ results, the optimality of the merge process is certain. Assume, however, that for some T, the $n-T$ best result in the merged list is the l (i.e., the last) result of some segment, and thus that segment's list is exhausted. The optimality of merged results $n-T+1, n-T+2, \ldots, n$ is now in doubt: result $l+1$ of the exhausted segment, which was not retrieved by the QI and does not participate in the merge process, may match the query better than the results that remain in the merge process. In general, result lists of several segments may be exhausted during the merge process, and we would like the QI to be able to assess the optimality of its merged results.

Formal analysis. Let $D_i,\ i = 1, \ldots, m$ denote the set of documents that are indexed on segment i, and let l denote the number of results that each segment contributes to the merge process.

Let $\mathcal{B}$ denote the (ranked) set of the results for the current query in the entire index, and let $\mathcal{B}[t]$ denote the t top-ranking results in $\mathcal{B}$. Similarly, let $\mathcal{R}$ denote the set of lm retrieved results, and let $\mathcal{R}[t]$ denote the set of the top-t results in $\mathcal{R}$, as determined by the merge process.

Let R_t denote the rank of merged result number t with respect to the set $\mathcal{B}$ (the absolute rank of merge result number t). Obviously, $R_1 = 1$ and $R_t \geq t$. The *loss* at time t, $L(t)$, is defined as follows:

$$L(t) \stackrel{\triangle}{=} |\mathcal{B}[t] \setminus \mathcal{R}[t]| = |\mathcal{B}[t] \setminus \mathcal{R}|.$$

$L(t)$ counts the number of top-t results that do not participate in (are lost by) the merge process. Clearly, $R_t = t \iff L(t) = 0$. This is generalized in the following Lemma:

LEMMA 2. *For every $t = 1, \ldots, n$: $R_{t-L(t)} \leq t$ and $R_{t-L(t)+1} > t$.*

PROOF. By definition, exactly $t - L(t)$ results from $\mathcal{B}[t]$ are in $\mathcal{R}$. This means that the top $t - L(t)$ results in $\mathcal{R}$ are in $\mathcal{B}[t]$, but the $(t - L(t) + 1)$-st result in $\mathcal{R}$ is not in $\mathcal{B}[t]$. The lemma follows. □

We next develop a recursive formula that allows the QI to estimate, during the merge process, the probability of losing a certain number of results. For this formula we need the following notations:

Let $t = 1, \ldots n$; then $\eta_t \stackrel{\triangle}{=} |\{\, i \,:\, D_i \cap \mathcal{R}[t] = l\}|$ denotes the number of exhausted segments after merging result t. Similarly, define $\mu_t \stackrel{\triangle}{=} |\{\, i \,:\, D_i \cap \mathcal{B}[t] \geq l\}|$ as the number of segments that store l or more of the best t documents. Note that while the values of η_t, $t = 1, \ldots, n$ are known to the QI during the merge process, the corresponding values of μ_t are not. The following Proposition links the values of $L(t)$, η_t and μ_t.

PROPOSITION 3. *For every $t \in \{1, \ldots, n\}$, $\mu_t = \eta_{t-L(t)}$.*

PROOF. We need to show that the document set D_i contains l or more results of $\mathcal{B}[t] \Leftrightarrow$ segment i is exhausted before merging result $t - L(t) + 1$.

$\Rightarrow$: If D_i contains at least l results of $\mathcal{B}[t]$, then segment i is exhausted before the first result which is not in $\mathcal{B}[t]$ is merged. By Lemma 2 we have that $R_{t-L(t)+1} > t$, hence segment i is exhausted before the $t - L(t) + 1$-st result is merged, as claimed.
$\Leftarrow$: Similarly, if by the time we merge result $t - L(t)$ (whose rank is $R_{t-L(t)}$) segment i is exhausted, then the top l results in D_i are all ranked better than or equal to $R_{t-L(t)}$, which by Lemma 2 is at most t. □

Using the above relationship, the QI can evaluate the probabilities of losing results during the merge process as follows:

PROPOSITION 4. *$L(1) = 0$, and for $t = 1, \ldots, n-1$ and $k = 0, \ldots, t-1$:*

$$Pr[L(t+1) = k] = \frac{\eta_{t-k+1}}{m} Pr[L(t) = k-1] + \frac{m - \eta_{t-k}}{m} Pr[L(t) = k].$$

PROOF. For deductive convenience, assume that the query's results are distributed to the segments from best to worst, where at step t the t-th best result is randomly inserted into one of the segments.

As noted previously, the best result is never lost, hence $L(1) = 0$. For larger values of t, observe that either $L(t+1) = L(t)$ or $L(t+1) = L(t) + 1$. Thus:

$$\begin{aligned} Pr[L(t+1) = k] = {} & Pr[L(t) = k-1] * Pr[L(t+1) = k \mid L(t) = k-1] + \\ & Pr[L(t) = k] * Pr[L(t+1) = k \mid L(t) = k]. \end{aligned}$$

Let d denote the $(t+1)$-st result in $\mathcal{B}$. Assume that $d \in D_j$, and let j_d denote the local rank of d in segment j.

Now, $L(t+1) = L(t) + 1$ if and only if $d \notin \mathcal{R}$, or equivalently if and only if $j_d > l$. This means that j, d's segment, must be one of the μ_t segments to which l or more of the top t results belong. Similarly, $L(t+1) = L(t) \Longleftrightarrow j_d \leq l$, which means that j must be one of the $m - \mu_t$ segments to which less than l of the top t results belong. Therefore,

$$Pr[L(t+1) = k \mid L(t) = k-1] = \frac{\mu_t}{m} = \frac{\eta_{t-k+1}}{m}$$

and

$$Pr[L(t+1) = k \mid L(t) = k] = \frac{m - \mu_t}{m} = \frac{m - \eta_{t-k}}{m},$$

from which the proposition follows. □

In particular, the probability that the merge process produces optimal result pages equals

$$Pr[L(n) = 0] = \prod_{t=1}^{n-1} \left(1 - \frac{\eta_t}{m}\right).$$

The QI may set a policy that result pages should contain at most k nonoptimal results with probability at least q_k. In order to enforce this policy, the QI should assert that

$$\sum_{l=0}^{k} Pr[L(n) = l] \geq 1 - q_k.$$

The QI can calculate these probabilities by updating $Pr[L(t) = l],\ l = 0, \ldots, k$ on the fly. Whenever the merged results fail to comply with the policy, the QI should fetch more data from the segments in order to protect the quality of the merged result pages.

6.2 Skipped Results as a Measure of Sub-Optimality

The absolute rank of the last merged result is another indicator of the quality of the merge process. Obviously, after merging t results, $R_t \geq t$. When $R_t = t$, the top-t merged results are optimal; however, when $R_t > t$, quality results have been *skipped* over by the merge process, allowing lesser results to occupy top-t positions in the merged results. We thus define the number of skipped results at time t by $S(t) \stackrel{\triangle}{=} R_t - t$. This number, which reflects the rank promotion of merged result t with respect to its absolute rank, also counts the number of results in the index that (a) were not merged up to time t and that (b) rank higher than R_t, the rank of the result merged in time t.

We begin by observing the following relationship between our two measures of non-optimal results: lost and skipped results.

PROPOSITION 5. *$L(n) \geq k$ if and only if $S(n-k+1) \geq k$.*

PROOF. Let $L(n) \geq k$. It follows that $R_{n-k+1} > n$. Therefore,

$$S(n-k+1) = R_{n-k+1} - (n-k+1) > n-n+k-1 = k-1 \Longrightarrow S(n-k+1) \geq k.$$

Conversely, Let $S(n-k+1) \geq k$. It follows that

$$R_{n-k+1} = S(n-k+1) + (n-k+1) \geq n+1\ ,$$

and therefore $L(n) \geq k$. □

We cannot use the above relationship to deduce probabilities of the type $Pr[S(n) = k]$ for arbitrary values of k. We therefore propose the following online computation for the probabilities $S(n) = j$ for $j = 0, \ldots, k$. As in the previous subsection, we again consider the query's results to be uniformly and independently distributed to the segments from best to worst. Consider a merge process after $t \geq 1$ steps, that has just merged the result whose rank is R_t. Recall that η_t is the number of exhausted segments at that time. The results ranking in place $1 + R_t, \ldots, j + R_t$ will be skipped if and only if each is assigned to one of the η_t exhausted segments. We can think of each result assignment as a Bernoulli trial, which fails if the result is assigned to an exhausted segment, and succeeds otherwise. This leads to the observation that the increase in skips following merge step $t+1$, $S(t+1) - S(t)$, equals the number of failures in a series of independent Bernoulli trials, each with a success probability of $1 - \frac{\eta_t}{m}$, until the first success. In other words, $S(t+1) - S(t)$ is a geometric random variable with parameter $1 - \frac{\eta_t}{m}$. Thus, for $t > 1$ and $j = 0, \ldots, k$,

$$Pr[S(t+1) = k \mid S(t) = k - j] = \left(\frac{\eta_t}{m}\right)^j \left(1 - \frac{\eta_t}{m}\right),$$

hence

$$Pr[S(t+1)=k] = \sum_{j=0}^{k} \left(\frac{\eta_t}{m}\right)^j \left(1-\frac{\eta_t}{m}\right) Pr[S(t)=k-j].$$

The above derivation, along with the fact that $S(1)=0$ with probability one, allows probabilities of the form $Pr[S(t)=j]$ for $j=0,\ldots,k$ to be computed in $\mathcal{O}(k^2)$ per merge step.[11]

An alternative computation of the probability of skips, which may be more efficient when the merge process is long (n is large), is the following. Let $\alpha \stackrel{\triangle}{=} \eta_{n-1}$ denote the number of exhausted segments before the last (the n'th) merge step. Further, define

$$\tau_i = \min\{t \mid \eta_t = i+1\}, \quad i=0,\ldots,\alpha-1$$

that is, merge step τ_i is the latest that was executed with exactly i exhausted segment. Accordingly, define $\tau_\alpha \stackrel{\triangle}{=} n$.

The first τ_0 merge steps took place while no segment was exhausted, so $R_{\tau_0} = \tau_0$ and $S_{\tau_0} = 0$. The next $\tau_1 - \tau_0$ merge steps took place while one segment was exhausted. In general, let $t_i \stackrel{\triangle}{=} \tau_i - \tau_{i-1},\ i=1,\ldots,\alpha$; there were t_i merge steps that took place while exactly i segments were exhausted; we will refer to those steps as the i'th phase of the merge process. We also define $X_i \stackrel{\triangle}{=} S(\tau_i) - S(\tau_{i-1})$, the increase in the number of skips during phase i; it is easy to see that

$$S(n) = \sum_{i=1}^{\alpha} X(i) = \sum_{i=1}^{\alpha} S(\tau_i) - S(\tau_{i-1}) \tag{4}$$

Phase i of the merge process constitutes a sequence of Bernoulli trials, t_i of which are successful, and whose failures correspond to skipped results. X_i is therefore a negative binomial random variable[12] with parameters $(t_i, 1-\frac{i}{m})$, and

$$Pr[X_i = k] = \binom{k+t_i-1}{k} \left(1-\frac{i}{m}\right)^{t_i} \left(\frac{i}{m}\right)^k \tag{5}$$

By Equation 4 we deduce that $S(n)$ is distributed according to a convolution of a series of α independent (and differently distributed) negative binomial random variables:

$$Pr[S(n)=k] = Pr\left[\sum_{i=1}^{\alpha} X_i = k\right] = \sum_{j=0}^{k} Pr[X_1=j] \cdot Pr\left[\sum_{i=2}^{\alpha} X_i = k-j\right].$$

Since every X_j is a negative binomial random variable, $Pr[X_j = i]$ can be evaluated by expression (5) in $\mathcal{O}(\log t_i + k)$ operations. A naive iterative

[11] The complexity can be lowered to $\mathcal{O}(k \log k)$ per step by applying FFT, see [Cormen et al. 1990] for more details.

[12] A negative binomial random variable with parameters (r, p) counts the number of *failures* in a series of i.i.d. Bernoulli trials, each with a success probability of p, until the r-th success. Negative binomials are often alternatively defined as counting the number of i.i.d. Bernoulli *trials*, each with success probability p, until the r-th success [Feller 1970].

procedure can thus calculate $Pr[S(n) = k]$ in $\mathcal{O}(\alpha k^2) = \mathcal{O}(mk^2)$ evaluations of such expressions, resulting in a total of $\mathcal{O}(mk^3 \log n)$ operations.[13]

7. FROM THEORY TO PRACTICE

This section attempts to bridge the gap between theory and practice by highlighting the possible practical implications of our model and results.

7.1 The Complexity Function $W(r)$

We first revisit two assumptions we have made while formalizing $W(r)$ (Section 2). These assumptions pertain to the manner by which users view result pages and to the memoryless query processing scheme.

1. "Users view search result pages according to a memoryless geometric process." While this assumption is extremely simplistic, the studies cited in Section 2.1 indicate that it might reasonably approximate the aggregate behavior of users.
2. "When a request for result page k arrives, result page $k-1$ is still cached." We used this assumption to send each segment the score of the lowest result it had contributed to page $k-1$. This, in turn, allowed us to formulate a memoryless query processing scheme. While ignoring cache management issues in this work, the following consideration justifies the intuition behind this assumption: the aim of *any* policy that prefetches r pages (numbered $k, \ldots, k+r-1$) when processing a request for result page k of some query, is to rapidly answer (from the cache) subsequent requests for pages $k+1, \ldots, k+r-1$ of that query. Thus, the prefetching policy implicitly assumes that the life expectancy of cached entries will allow page $k+r-1$ to be cached until it is requested. In other words, *every* policy that prefetches r pages assumes that pages will be cached long enough for $r-1$ subsequent requests. We require pages to be cached for r subsequent requests.

The above assumptions allowed us to formulate an exact complexity function to our concrete query processing model. At the end of Section 2.2, the complexity function was abbreviated to the form

$$W(r) = ar + \frac{b + cl_q(r, m) + dr}{1 - p^r}.$$

We claim that this abbreviated form (and our results) can accommodate any retrieval model that incurs the following costs when prefetching r pages:

- Cache space that is linear in r, the number of prefetched result pages.
- Retrieval complexity that is the sum of (1) a term that depends on the query's breadth (number of matching results), (2) a term that is linear in $l_q(r, m)$, and (3) a term that is linear in r.

Thus, our results may apply to index structures and query processing schemes that differ from our model. Furthermore, the results of Section 4.2 apply to

[13]Again, this can be lowered to $\mathcal{O}(mk^2 \log k \log n)$ by applying FFT.

any complexity function $\tilde{W}(r)$ where $(1 - p^r)\tilde{W}(r)$ is an increasing function of r. Finally, the combinatorial results of Sections 5 and 6 are applicable to any search engine that uses a locally segmented index in which documents are partitioned uniformly and independently.

7.2 Implementing a Prefetching Policy

Implementing a prefetching policy for engines with locally segmented indices in the framework of this research requires the following two preprocessing steps:

- Setting the parameters: an approximate value of p is derived from analyzing the engine's query logs, the parameter q is set according to the quality policy, and the values of α, β are set according to the engine's resources. Systems with small caches should set β to a high value; when the QI is heavily loaded, α should be set to a high value.
- For a range of query breadths (a range of values for the parameters C, ω), an algorithm (either optimizing or approximating r_{opt}) is executed. The QI and each segment are then loaded with tables containing the values of $r_{opt}(C, \omega)$ and $l_q(r_{opt}(C, \omega), m)$ for values of C, ω in the range.

Upon receiving a query (t, k), each segment estimates the parameters C_t, ω_t: C_t is simply the number of matches for t in the segment, and ω_t is a measure of the work it took to identify the $|C_t|$ matches (for example, the total length of the posting lists that were scanned). For broad topic queries (when $C \gg m$), the estimates by the various segments should be fairly equal. Note that for disjunctive queries (including all single-term queries), ω_t is linear in C_t and the problem reduces to estimating the breadth of the query. Query breadths may be estimated using *global* term statistics, which are kept in each segment by many local index implementations in order to facilitate term-based scoring [Arasu et al. 2001].

After estimating C_t, ω_t, each segment forwards $l_q(r_{opt}(C_t, \omega_t))$ results to the QI, which merges the retrieved results to produce $r_{opt}(C_t, \omega_t)$ result pages.

8. CONCLUSIONS AND FUTURE WORK

This work examined how search engines should prefetch search results for user queries. We started by presenting a concrete query processing model for search engines with locally segmented inverted indices. We argued that for a model that assumes that the number of result pages that users view is distributed geometrically, the optimal engine policy is to prefetch a constant number of result pages r. We expressed the computational cost of a policy that prefetches r pages, and suggested an algorithm for finding the optimal value of r (that minimizes the expected cost of serving a search session). We also suggested how to find values of r that imply policies whose cost is approximately optimal.

Several extensions of this work are the following:

- The model presented in this paper ignores overlaps in the information needs of different users. We did not consider, for example, that popular queries may be submitted by multiple users during a short time span, increasing

the probability of at least one user requesting additional results. By taking query popularity into account, we may find that popular queries warrant more result prefetching than rare queries do. Note that the impact of prefetching on the cache hit ratios, under real-life, multi-user workloads, was discussed in Section 1.2.

- This work did not address cache replacement policies; in particular, we did not suggest which result pages should be removed from the cache upon prefetching results for a new query. An experimental evaluation of several cache replacement and prefetching schemes, based on real-life search engine query logs, is available in Lempel and Moran [2003]. The analysis in Lempel and Moran [2003] is concentrated on the cache hit ratios that the evaluated policies attain. A possible extension of this research would aim to minimize the aggregate work invested by the various components of the search engine while executing queries.
- An interesting problem that stems from the sequential nature of multi-page search sessions is that of maintaining consistency of the cached entities. In the context of search sessions, consistency has two (sometimes conflicting) meanings: first, cached objects need to be consistent with reality (fresh). This can be of great importance in some vertical search applications such as news search engines. Second, each individual session must be consistent in the sense that returned results should neither be skipped nor repeated. Consider, for example, a user that requested the top 10 results for some query, and after some time requested the next 10 results. If the underlying index has changed, it might be that the previous top 10 results now rank in places 11–20. Should the engine return these results (again) to the user? Will such a policy deliver the best user experience?

  Note that the first issue arises in any system that employs caching (regardless of any prefetching policy that may be used), while the second issue is present even in the absence of caching. Reconciling these two aspects of consistency in caches that serve concurrent, unsynchronized sessions may necessitate caching of separate versions of the same logical resource for different sessions.
- Most of the results in this paper are applicable to locally segmented indices. Only single-term queries to global indices are considered. Additional research is required in order to extend our results to multi-term queries to global indices.

ACKNOWLEDGMENTS

We thank Andrei Broder[14] and Farzin Maghoul from AltaVista for useful discussions and insights on the problems covered in this paper. We thank the anonymous referees for many useful comments and suggestions.

REFERENCES

ARASU, A., CHO, J., GARCIA-MOLINA, H., PAEPCKE, A., AND RAGHAVAN, S. 2001. Searching the web. *ACM Trans. Internet Tech. 1*, 1, 2–43.

[14]Andrei Broder is currently with IBM Research.

AZAR, Y., BRODER, A. Z., KARLIN, A. R., AND UPFAL, E. 1999. Balanced allocations. *SIAM J. Comput. 29*, 1, 180–200.

BARROSO, L. A., DEAN, J., AND HÖLZLE, U. 2003. Web search for a planet: The google cluster architecture. *IEEE Micro 23*, 2 (April), 22–28.

BRIN, S. AND PAGE, L. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the 7th International WWW Conference*. 107–117.

BRODER, A. 2000. Web search. *Invited talk at the 9th Text Retrieval Conference (TREC-9), Gaithersburg, Maryland*, November.

CAHOON, B., MCKINLEY, K. S., AND LU, Z. 2000. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Transactions on Information Systems 18*, 1, 1–43.

CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1990. *Introduction to Algorithms*. MIT Press and McGraw-Hill.

CZUMAJ, A. AND STEMANN, V. 1997. Randomized allocations processes. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*. 194–203.

FELLER, W. 1970. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons.

HAWKING, D. 1997. Scalable text retrieval for large digital libraries. In *First European Conference on Digital Libraries*.

JANSEN, B. J., SPINK, A., AND SARACEVIC, T. 2000. Real life, real users, and real needs: A study and analysis of user queries on the web. *Inf. Proc. Manage. 36*, 2, 207–227.

JEONG, B.-S. AND OMIECINSKI, E. 1995. Inverted file partitioning schemes in multiple disk systems. *IEEE Trans. Parallel Distrib. Syst 6*, 2, 142–153.

JOHNSON, N. L. AND KOTZ, S. I. 1977. *Urn Models and their Application*. John Wiley & Sons, Inc.

JOHNSON, N. L. AND YOUNG, D. H. 1960. Some applications of two approximations to the multinomial distribution. *Biometrika 47*, 463–469.

KNUTH, D. E. 1998. *The Art of Computer Programming, Volume 3*. Addison-Wesley Publishing Company Inc.

KOLCHIN, V. F., SEVAST'YANOV, B. A., AND CHISTYAKOV, V. P. 1978. *Random Allocations*. V. H. Winston & Sons.

LAWRENCE, S. AND GILES, C. L. 1998. Searching the world wide web. *Science 280*, April.

LEMPEL, R. AND MORAN, S. 2003. Predictive caching and prefetching of query results in search engines. In *Proceedings of the 12th World Wide Web Conference (WWW2003), Budapest, Hungary*, May.

MARKATOS, E. P. 2000. On caching search engine query results. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, May.

MELNIK, S., RAGHAVAN, S., YANG, B., AND GARCIA-MOLINA, H. 2001. Building a distributed full-text index for the web. In *Proceedings of the 10th International WWW Conference*.

MOTWANI, R. AND RAGHAVAN, P. 1995. *Randomized Algorithms*. Cambridge University Press.

PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., AND VETTERLING, W. T. 1988. *Numerical Recipes in C—The Art of Scientific Computing*. Cambridge University Press.

RIBEIRO-NETO, B. AND BARBOSA, R. 1998. Query performance for tightly coupled distributed digital libraries. In *Proceedings of the ACM Digital Libraries Conference, 1998*. 182–190.

RIBEIRO-NETO, B. A., KITAJIMA, J. P., NAVARRO, G., SANT'ANA, C. R. G., AND ZIVIANI, N. 1998. Parallel generation of inverted files for distributed text collections. In *Proceedings of the 18th International Conference of the Chilean Computer Science Society*.

SILVERSTEIN, C., HENZINGER, M., MARAIS, H., AND MORICZ, M. 1998. Analysis of a very large altavista query log. Tech. Rep. 1998-014, Compaq Systems Research Center. October.

TOMASIC, A. AND GARCIA-MOLINA, H. 1993. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*. 8–17.

WOLF, J. L., SQUILLANTE, M. S., YU, P. S., SETHURAMAN, J., AND OZSEN, L. 2002. Optimal crawling strategies for web search engines. In *Proceedings of the 11th International World Wide Web Conference (WWW2002)*.

Received October 2002; revised June 2003; accepted June 2003