Partial Collection Replication For Information Retrieval

Zhihong Lu
AT&T Laboratories
200 South Laurel Ave.
Middletown, New Jersey 07748
zhihonglu@att.com

Kathryn S. McKinley
Deptartment of Computer Sciences
University of Texas
Austin, Texas 78712
mckinley@cs.utexas.edu

Abstract

The explosion of content in distributed information retrieval (IR) systems requires new mechanisms in order to attain timely and accurate retrieval of unstructured text. This paper shows how to exploit locality by building, using, and searching *partial replicas* of text collections in a distributed IR system. In this work, a partial replica includes a subset of the documents from larger collection(s) and the corresponding inference network search mechanisms. For each query, the distributed system determines if partial replica is a good match and then searches it, or it searches the original collection. We demonstrate the performance of partial replication is better than systems that use *caches* which only store previous query and answer pairs. We first use logs from THOMAS and Excite to show to build partial replicas and caches from frequent queries. We show that searching replicas can improve locality (from 3 to 20%) over the exact match required by caching. Replicas increase locality because they satisfy queries which are distinct but return the same or very similar answers. We then present a novel inference network replica selection function. We vary its parameters and compare it to previous collection selection functions, demonstrating a configuration that directs most of the appropriate queries to replicas in a replica hierarchy. We then explore the performance of partial replication in a distributed IR system. We compare it with caching and partitioning. Our validated simulator shows that the increases in locality due to replication make it preferable to caching alone, and that even a small increase of 4% in locality translates into a performance advantage. We also show a hybrid system with caches and replicas that performs better than each on their own.

Keywords: Partial replication, Replica Selection, Distributed information retrieval architectures.

1 Introduction

Due to the explosion of information and users on the Internet and intranets, a major challenge for distributed information retrieval systems is providing fast and scalable performance. Information retrieval (IR) systems solve the problem of finding documents related to a query. Users issue a query, and the IR server typically responds with a list of relevant documents and snippets of text from these documents. If any of these documents meet the users information need, they request one or more of them. If not, users refine or change their query. To achieve scalable performance on this workload, IR systems distribute queries, and restrict the search to as

little data as possible while maintaining acceptable retrieval accuracy. Previous work on distributed databases and Web search engines improve performance using multiple cache and/or full replica servers to distribute workloads. When these servers are closer to their users than the source collection, they also reduce network traffic, minimizing network latency.

To achieve accurate results with caches and other mechanisms that restrict query search, IR systems need to exhibit and exploit *query locality*, i.e., users issue the same, or closely related queries, and improvements are of course limited by the locality the queries present. The classic system mechanism to exploit locality is *caching* which in an IR system stores pairs of recently issued queries and results. For example, Web search engines adopt query caches that store a map between queries and result lists. They use a simple, exact test of set membership to determine if the query is in the cache. The cache improves performance when queries *exactly match* a previous query in the cache.

In this work, we propose *partial replicas* which exploit locality from the same *and* from closely related queries. Because information retrieval engines find similarity or relevance between queries and documents, partial replicas build on this functionality to query a subset of the collection and return relevant documents. A partial replica thus includes query logic as well as a subset of the documents. If two queries are not the same, but return the same top documents, partial replicas find and exploit this locality. Partial replicas therefore can more thoroughly exploit locality because the queries need not match. To be more effective than caches, partial replicas need to provide additional locality, an efficient selection mechanism to direct queries to partial replica(s) or the original collection, and a scalable IR architecture. We examine all these components in this paper.

To motivate our approach, we first investigate the type and amount of locality for a few traces. Although others report on query locality (Croft, Cook, & Wilder, 1995; Holmedahl, Smaith, & Yu, 1998), there exist no widely available, shared, or standard query sets. In addition, no results divide locality into exact match and similar queries (we conservatively define similar queries to require the top 20 documents to match). We report locality properties on server logs from THOMAS (THOMAS, 1998) for 62 days and Excite (Excite, 1997) for 1 day. These results show that locality remains high (above 20%) over time (weeks) for the THOMAS logs. For both traces, inexact match increases locality from 3% up to 15%, even with our very restrictive definition of query similarity. These results are due to human variety that results in queries that do not exactly match but return the same results. We then turn to exploiting the additional locality of partial replicas as compared to caches to improve performance.

To maintain retrieval effectiveness, a selection function must determine whether a replica contains all, some, or none of the relevant documents for a query. We describe such a function that uses an inference networks and demonstrate its effectiveness using the 20 GB TREC VLC collections and TREC queries. We vary the selection function parameters and compare it with collection ranking functions. For a given query, this function correctly selects the most relevant of a set of replicas or the original collection, when appropriate. It maintains the highest retrieval effectiveness while searching the least amount of data as compared to the

other replica selection functions we explore.

We then describe our distributed IR architecture and report its performance as a function of locality using a validated simulator (Lu, 1999). We show that the simulator closely matches our prototype system which uses InQuery for the basic IR functionality on all the collections (original and replicated) (Callan, Croft, & Broglio, 1995). We compare the performance of searching a terabyte of text using partial replication to partitioning and caching. Partition simply divides a large collection into multiple parts and stores them on separate servers. We find partial replication is more effective at reducing execution time, even with many fewer resources, and it requires only modest query locality to achieve better, sometimes much better, performance than partitioning and caching.

In summary, this paper is the first to explore partial replication. We first show that there is sufficient query locality in a few real traces to justify this exploration. We then develop a novel and effective replica selection function which is able to choose between a hierarchy of replicas and the original collection to achieve high accuracy and high performance. We furthermore show that partial replication offers a performance benefit over exact match caching, and can work in cooperation with caching to further improve performance.

The remainder of this paper is organized as follows. The next section further compares our work to related work. In Section 3, we characterize the locality and access patterns of our test logs from THOMAS and Excite. We also define our notion of query similarity, and show it increases locality up to 15%. Section 4 describes the replication architecture. Section 5 describes how to select a partial replica based on relevance, and compares its effectiveness with the ranking functions for collection ranking. Section 6 reports on the performance of searching a terabyte of text using partial replication and compares performance with collection partitioning as well as caching. Section 7 summarizes our results and concludes.

2 Related Work

This paper combines and extends our previous work studying query locality (Lu & McKinley, 2000) and developing a replica selection function (Lu & McKinley, 1999). We include here a more detailed presentation of retrieval effectiveness on using partial replicas with varying numbers of top documents, strategies on update replicas, data validating our simulator, and a richer set of performance studies, including using a hierarchy of replicas rather than just one. We discuss below a variety of related research topics by others: architectures for fast, scalable, and effective large scale information retrieval; single query performance; IR versus database systems; IR versus the web; caching; and collection selection.

2.1 Scalable IR Architectures

In this section, we discuss architectures for parallel and distributed IR systems. Our research combines and extends previous work in distributed IR (Burkowski, 1990; Harman, McCoy, Toense, & Candela, 1991; Couvreur et al., 1994; Burkowski, Cormack, Clarke, & Good, 1995; Cahoon & McKinley, 1996; Hawking, 1997; Hawking, Craswell, & Thistlewaite, 1998; Cahoon, McKinley, & Lu, 2000) since we model and analyze a complete system architecture with replicas, replica selection, and collection selection under a variety of

workloads and conditions. We base our distributed system on InQuery (Callan, Croft, & Harding, 1992; Turtle, 1991), a proven, effective retrieval engine. We also model architectures with very large text collections; up to 1 terabyte of data on up to 32 InQuery servers. Much of the prior work on distributed IR architectures has been restricted to small collections, typically less than 1 GB and/or 16 servers. Participants in the TREC Very Large Collection track use collections up to 100 GB, but they only provide query processing times for a single query at a time (Hawking et al., 1998). It is clear that some industrial sites use collections larger than what we simulate, but they choose not to report on them in the literature to maintain their competitive edge, with an exception of Google, the most popular search engine nowadays, which has reports on its technologies when it was a research project (Brin & Page, 1998).

Harman et al. show the feasibility of a distributed IR system by developing a prototype architecture and performing user testing to demonstrate usefulness (Harman et al., 1991). Unlike our research which emphasizes performance, Harman et al. do not study efficiency issues and they use a small text collection (i.e., less than 1 GB).

Burkowski et al. report on a simulation study which measures the retrieval performance of a distributed IR system (Burkowski, 1990; Burkowski et al., 1995). The experiments explore two strategies for distributing a fixed workload across a small number of servers. The first equally distributes the text collection among all the servers. The second splits servers into two groups, one group for query evaluation and one group for document retrieval. They assume a worst case workload where each user broadcasts queries to all servers without any think time. We experiment with larger configurations, and consider collection selection and replicas with replica selection.

Couvreur et al. analyze the performance and cost factors of searching large text collections on parallel systems (Couvreur et al., 1994). They use simulation models to investigate three different hardware architectures and search algorithms including a mainframe system using an inverted list IR system, a collection of RISC processors using a superimposed IR system, and a special purpose machine architecture that uses a direct search. The focus of the work is on analyzing the tradeoff between performance and cost. Their results show that the mainframe configuration is the most cost effective. They also suggest that using an inverted list algorithm on a network of workstations would be beneficial but they are concerned about the complexity. In their work, each query is evaluated against all collections.

Hawking designs and implements a parallel information retrieval system, called PADRE97, on a collection of workstations (Hawking, 1997). The basic architecture of PADRE97, which is similar to ours, contains a central process that checks for user commands and broadcasts them to the IR engines on each of the workstations. The central process also merges results before sending a final result back to the user. Hawking presents results for a single workstation and a cluster of workstations using a single 51 term query. A large scale experiment evaluates query processing on a system with up to 64 workstations each containing a 10.2 GB collection. The experiment uses four short queries of 4 to 16 terms. This work focus on the speedup of a single query, while our work evaluates the performance for a loaded system under a variety of workloads and

collection configurations.

The founders of Google reported Google's architecture overview and core technology: PageRank before Google went to commercial (Brin & Page, 1998). Since the performance of search was not the major focus of their research at that time, the paper just mentions they intend to speed up Google considerably through distribution and hardware, software, and algorithmic improvements, without any supporting experiments. As far as we know, there are no recently published reports on the performance or architecture of their current system.

Cahoon et al. report a simulation study on a distributed information retrieval system based on InQuery (Cahoon & McKinley, 1996; Cahoon et al., 2000). They assume the collections are uniformly distributed, and experiment with collections up to 128 GB using a variety of workloads. They measure performance as a function of system parameters such as client command rate, number of document collections, terms per query, query term frequency, number of answers returned, and command mixture. They demonstrate system organizations for which response time gracefully degrades as the workload increases and performance scales with the number of processors under some realistic workloads. Our work builds on and extends this work by adding replicas, a replica selection function, and caches.

2.2 How to Search Large Collections

The TREC conference recently added the Very Large Collection track for evaluating the performance of IR systems on large text collections (Hawking & Thistlewaite, 1997; Hawking et al., 1998). To handle large collections, participants use shared-memory multiprocessors and/or distributed architectures. The experiments in TREC-7 use 49 long queries on a 100 GB collection of web documents. The Very Large Collection track summary (Hawking et al., 1998) presents precision and query processing time results but does not provide significant details about each system. The experiments report response times for a single query at a time, rather than for a variety of workloads as we do. None of the systems report results for caching or searchable replicas. Two of the participants present details of their distributed systems elsewhere, but neither provide significant performance evaluations (Burkowski et al., 1995; Brown & Chong, 1997).

2.3 Database versus IR Architectures

There is also a large volume of work on architectures for distributed and parallel database systems including research on performance (Stonebraker et al., 1983; DeWitt et al., 1986; Mackert & Lohman, 1986; Hagmann & Ferrari, 1986; DeWitt & Gray, 1992; Bell & Grimson, 1992). Although the fields of information retrieval and databases are similar, there are several distinctions which make studying the performance of IR systems unique. A major difference between database systems and information retrieval systems is structured versus unstructured data. In structured data, the tests resemble set membership. In unstructured data, we measure similarity of queries to documents. The unstructured nature of IR data raises questions about how to create large, efficient architectures. Our work attempts to discover some of the factors that affect performance when searching and retrieving unstructured data. Furthermore, the types of common operations that are

typical to database and IR systems are slightly different. For example, the basic commands in an IR system, query evaluation and document retrieval, differ from those in a database system. In our IR system, we are not concerned with updates (commit protocols) and concurrency control which are important issues in distributed database systems. We assume our IR system performs updates offline.

2.4 Web versus IR Architectures

Although commercial information retrieval systems, such as the web search engines AltaVista and Infoseek exploit parallelism, parallel computers, caching, and other optimizations to support their services, they have not published their hardware and software configurations, which makes comparisons difficult.

There are several important differences between the IR technology we discuss here and the web's implementation. We consider a more static collection of unstructured text on a local area network, such as a collection of case law or journal articles. Whereas the web has more structured text on a wide area network whose content is very dynamic. On the web, most caches are built for specific documents, not for querying against as we do here (Wang, 1999). The web's document cache simply uses set membership tests to determine if the cache has a requested document. Web users do employ search engines for queries, which can maintain query caches, but the query caches seem not to be distributed because of the very dynamic nature of the web's content (as far as we know).

2.5 Caching

Caching in distributed IR systems has a long research history (Simpson & Alonso, 1987; Martin, Macleod, Russell, Lesse, & Foster, 1990; Martin & Russell, 1991; Tomasic & Garcia-Molina, 1992). The client caches data so that operations are not repeatedly sent to the remote server. Instead, the client locally performs frequent operations. These early reports show that the use of caching is significantly beneficial for systems that are distributed over slow networks or that evaluate queries slowly.

Recently, Markatos reports on caching search engine results (Markatos, 1999). He analyzes a trace from the Excite search engine and uses trace-driven simulations to compare several cache replacement policies. He shows that medium-sized caches (a few hundred Mbytes large) can achieve the hit ratio of around 20%, and effective cache replacement policies should take into account both recency and frequency of access in their replacement decisions. Larger caches should of course improve hit rates.

Saraiva et al. report on a two-level caching schema for search engines where one level caches query results, and another level caches inverted lists (Saraiva et al., 2001). They experiment with this caching schema using a set of log queries from a real case search engine, and show that the throughput of the two-level cache is up to 52% higher than the cache of inverted lists and 36% higher than the cache of query results. Our results are complementary to these, and our mechanisms will work in their system as well.

Researchers have also used replication techniques to solve the problem of scale in the Web (Katz, Butler, & McGrath, 1994; Bestavros, 1995; Baentsch, Molter, & Sturm, 1996). Katz et al. report a prototype of a scalable web server (Katz et al., 1994). They treat several identically configured `http` servers as a

cluster, and use the DNS (Domain Name System) service to distribute `http` requests across the cluster in a round-robin fashion. Bestavros proposes a hierarchical demand-based replication strategy that optimally disseminates information from its producer to servers that are closer to its consumers in the environment of the web (Bestavros, 1995). The level of dissemination depends on the popularity of that document (relative to other documents in the system) and the expected reduction in traffic that results from its dissemination. Baentsch9 et al. implement a replication system called CgR/WLIS (Caching goes Replication/Web Location and Information Service) (Baentsch et al., 1996). As the name suggests, CgR/WLIS turns web caches into replicated servers as needed. In addition, the primary servers forward the data to their replicated servers. A name service WLIS is used to manage and resolve different copies of data.

Although we also organize replicas as a hierarchy, our work is different from those above, because our system is a retrieval system that supports queries while their servers contain Web documents and only support document fetching. Our work is different from Web caching, because we use searchable replicas and a replica selector to select a partial replica based on content and load, rather than simple membership test in caching. Compared with caching, selection based on content increases observed locality, and is thus able to offload more work from servers that process original collections. However, there is an additional space overhead with partial replicas compared to caches which will slightly degrade locality as well, since the partial replicas will store less (see Section 3.3 for a discussion and quantitative results for our traces). In addition, the query processing of partial replication is greater than simply an exact match test. These overheads indicate, that the partial replica will need more than a slight locality benefit to improve performance, and that the best system architecture will probably include both caches and partial replicas.

2.6 Collection Selection

A number of researchers have been working on how to select most relevant collections for a given query (Callan, Lu, & Croft, 1995; Chakravarthy & Haase, 1995; Danzig, Ahn, Noll, & Obraczka, 1991; Gravano, Garcia-Molina, & Tomasic, 1994; Voorhees, Gupta, & Johnson-Laird, 1995; Fuhr, 1999; Xu & Croft, 1999). Only this and our previous work (Lu & McKinley, 1999) considers partial replica selection based on relevance.

Danzig et al. use a hierarchy of brokers to maintain indices for document abstracts as a representation of the contents of primary collections (Danzig et al., 1991). They support Boolean keyword matching to locate the primary collections. If users' queries do not use keywords in the brokers, they have difficulty finding the right primary collections. Our approach is thus more general.

Voorhees et al. exploit similarity between a new query and relevance judgments for previous queries to compute the number of documents to retrieve from each collection (Voorhees et al., 1995). Netserf extracts structured, disambiguated representations from the queries and matches these query representations to hand-coded representations (Chakravarthy & Haase, 1995). Both approaches require manual intervention which limits them to relatively static and small collections.

Callan et al. adapt the document inference network to ranking collections by replacing the document

node with the collection node (Callan et al., 1995). This system is called CORI. CORI stores the collection ranking inference network with document frequencies and term frequencies for each term in each collection. Experiments using CORI with the INQUERY retrieval system and the 3 GB TREC Volumes 1+2+3 collection which is basically organized by source show that this method can select the top 50% of subcollections and attain similar effectiveness to searching all subcollections.

GLOSS uses document frequency information for each collection to estimate whether, and how many, potentially relevant documents are in a collection (Gravano et al., 1994; Gravano & Garcia-Molina, 1995). The approach is easily applied to large numbers of collections, since it stores only document frequency and total weight information for each term in each collection. French et al. compare GLOSS with CORI and demonstrate that CORI consistently returns better results while searching fewer collections citeFrench99.

Fuhr proposes a decision-theoretic approach to solve the collection selection problem (Fuhr, 1999). He makes decisions by using the expected recall-precision curve which yields the expected number of relevant documents, and uses cost factors for query processing and document delivery. He does not report on effectiveness.

Xu and Croft propose cluster-based language models for collection selection (Xu & Croft, 1999). They first apply clustering algorithms to organize documents into collections based on topics, and then apply the approach of (Callan et al., 1995) to select the most relevant collections. They find that selecting the top 10% of topic collections can achieve retrieval accuracy comparable to searching all collections.

Our work on partial replica selection reported here and in (Lu & McKinley, 1999) modifies the collection inference network model of (Callan et al., 1995), to rank partial replicas and the original collections, proposes a new algorithm for replica selection, and shows that it is effective and improves performance.

3 Access Characteristics in Real Systems

In this section, we examine query locality in two logs from real systems. We examine query similarity versus exact match, how locality changes over time, and its effect on the size of replicas. We also suggest mechanisms for keeping replicas up to date.

Since currently there exists no widely available, shared, or standard set of queries with locality properties, we obtained our own sets of server logs from THOMAS (THOMAS, 1998) and Excite (Excite, 1997). The THOMAS system is a legislative information service of the U.S. Congress through the Library of Congress. THOMAS contains the full text Congressional Records and bills introduced from the 101st Congress to 105th Congress. We analyze the logs of THOMAS between July 14 and September 13, 1998, during which the Starr Report became available. We obtained full day logs for 40 days, and partial logs for remaining 22 days due to lack of disk space in the mailing system of the library of Congress. The Excite system provides online search for more than 50 million Web pages. The Excite log we obtained contains one day of log information for September 16, 1997.

Since the logs do not contain document identifiers returned from query evaluation, we built our own test databases to cluster similar queries. We define a *topic* as all queries whose top 20 documents completely

Num. queries	Num. unique queries	Topics			
		total	occurring only once	more than once	more than one unique query
8143 (7703)	4876 (4651)	4069	2888 (71%)	1181 (29%)	412
percentages of queries that top topics account for					
100	200	500		1000	2000
21.2%	28.7%	41.5%		54.1%	73.0%
percentages of queries that top unique queries account for					
100	200	500		1000	2000
18.1%	24.5%	36.4%		49.4%	64.5%

(a) Query locality in the THOMAS log

Num. queries	Num. unique queries	Topics			
		total	occurring only once	more than once	more than one unique query
499836 (444899)	365276 (320987)	249405	196672 (79%)	52733 (21%)	32750
percentages of queries that top topics account for					
500	1000	5000		10000	20000
12.3%	16.0%	27.9%		34.4%	42.0%
percentages of queries that top unique queries account for					
500	1000	5000		10000	20000
7.9%	10.4%	18.4%		23.0%	28.2%

(b) Query locality in the Excite log

Table 1: Query locality in the logs

overlap. This definition of query similarity is arbitrary and restrictive; a looser definition would further improve the locality we observe. Exact top 20 overlap is less likely to occur if we increase the number of documents. For large collections, partial overlap would probably yield results almost as accurate as exact overlap, and would also yield better locality than full overlap. For queries from the THOMAS log, we reran all queries against a test database that uses the Congress Record for 103rd Congress (235 MB, 27992 documents). For queries from the Excite log, we reran all queries against a test database using downloads of the websites operated by ten Australian Universities (725 MB, 81334 documents).

3.1 Query Locality

Table 1 shows query locality statistics for our THOMAS and Excite logs. We collect the average number of queries, unique (singleton) queries, topics, topics occurring only once, topics occurring more than once, and topics that contain more than one unique query. We also present the percentages of queries that correspond to the top topics and top unique queries, respectively. Table 1(a) shows the average numbers in the THOMAS logs over 40 days with full day logs. The numbers of queries that actually find matching documents from our test database are in the parentheses in columns 1 and 2. Some queries do not find any matching documents, due to misspelling, or because query terms do not exist in the test database. The statistics show that on the average, 29% of topics occur more than once, and they account for 63% ((7703-2888)/7703) of queries. Among the topics occurring more than once, 35% (412) contain more than one unique query. The top 100

topics (2.5% of topics) and the top 500 topics (12% of topics) account for 21.2% and 41.5% of queries, while the top 100 unique queries and top 500 unique queries account for 18.1% and 36.4%.

The Excite log on September 16, 1997, shown in Table 1(b), demonstrates that the Excite queries also have high query locality: 21% of topics occur more than once, and they account for 56% ((444899-196672)/444899) of queries. Among the topics occurring more than once, 62% (32750) contain more than one unique query. The top 1000 topics and the top 10000 topics account for 16.0% and 34.4% of queries, while the top 1000 unique queries and top 10000 unique queries account for 10.4% and 23.0%. Both sets of logs see a drop in locality between 3 and 14% if we require an exact match.

3.2 Locality as a Function of Time

We also examine the THOMAS logs to see how many queries on a given day match a topic or a query that appears on a previous day or week, in order to examine the overlap as a function of time. Table 2 shows for a number of days between July 15 and September 11, 1998 the percentage of queries that match a top query through topic match and exact query match on a previous day or week. Column 1 lists date. Columns 2 through 4 list the query overlap when we build a replica using top topics or top unique queries on the previous day and update it daily. Columns 5 through 7 list the query overlap when we build a replica using top topics or top unique queries on July 14, 1998, without update afterwards. Columns 8 through 10 list the query overlap when we build a replica using top topics or top unique queries in the week from July 14 to July 20, 1998, without update afterwards. Replicas may actually satisfy more queries than we report, because we do not include queries whose top documents appear in the replica because the response is a combination of two or more other topic queries. Since the logs do not contain document identifiers and our test database is pretty small, we can not obtain accurate figures about this situation.

The statistics also show that topic matching finds up to 15% more overlap than exact query match for the same size replicas over time. For example on September 11, 1998, we saw many distinct queries such as “Starr,” “Starr Report,” “Bill Clinton,” and “Monica Lewinsky,” all presumably trying to access the Starr Report.

For topic match, we build a replica with the top documents for the top topics. For exact query match, we build a replica with the top documents for the top unique queries. For example, when we build replicas using top 1000 topics or unique queries of the week of July 14 to July 20, topic match on July 23 increases the overlap from 28.6% (query exact match) to 35.9%, which means a replica can satisfy 7.3% more queries from the original server. Replicating more topics further widens this difference.

3.3 Estimating the Size of Replicas

Based on query locality, we may estimate the replica size, which is a function of average document size, query locality, and number of top documents per query we chose to store, as shown in Table 3. The average document size varies from source to source. For example, the average document sizes of the USENET News, Wall Street Journal, and the websites operated by 10 Australia Universities are 2 KB, 3 KB, and 9

	Overlap with								
	the previous day			7/14			the week of 7/14-7/20		
	Topic match			Topic match			Topic match		
date	all	top 500	top 1000	all	top 500	top 1000	all	top 500	top 1000
7/15	43.3%	24.8%	30.1%	43.3%	24.8%	30.1%	n/a	n/a	n/a
7/16	44.4%	24.4%	30.4%	42.6%	24.0%	29.2%	n/a	n/a	n/a
7/23	45.0%	27.3%	31.5%	41.4%	23.4%	28.7%	60.8%	29.0%	35.9%
7/31	n/a	n/a	n/a	38.5%	21.9%	26.4%	58.0%	26.0%	32.3%
8/14	36.6%	21.9%	26.1%	38.1%	21.3%	26.0%	54.9%	25.6%	31.0%
8/28	32.9%	19.1%	23.0%	34.3%	18.3%	23.4%	51.9%	22.8%	28.4%
9/11	78.1%	69.2%	71.7%	44.0%	8.7%	22.2%	58.6%	11.2%	27.0%
	Exact query match			Exact query match			Exact query match		
date	all	top 500	top 1000	all	top 500	top 1000	all	top 500	top 1000
7/15	33.1%	18.9%	23.3%	33.1%	18.9%	23.3%	n/a	n/a	n/a
7/16	34.5%	19.3%	23.0%	32.9%	18.1%	22.4%	n/a	n/a	n/a
7/23	36.4%	21.1%	24.6%	32.3%	17.9%	22.3%	49.4%	23.7%	28.6%
7/31	n/a	n/a	n/a	29.4%	16.9%	20.3%	46.5%	20.8%	25.1%
8/14	28.2%	16.3%	20.0%	29.0%	16.4%	20.0%	43.4%	20.2%	24.1%
8/28	25.4%	14.5%	17.6%	25.9%	14.0%	17.5%	41.2%	18.2%	22.2%
9/11	71.8%	63.6%	65.2%	24.9%	6.6%	18.7%	43.2%	8.2%	19.3%

Table 2: Overlap over time in the THOMAS log: Topics vs. exact query match

Top	% of	Replica Size (top 200 documents per query)		
topics	queries	(2 KB per doc)	(3 KB per doc)	(9 KB per doc)
1000	16.0%	400 MB	600 MB	1.8 GB
5000	27.9%	2 GB	3 GB	9 GB
10000	34.4%	4 GB	6 GB	18 GB
20000	42.0%	8 GB	12 GB	36 GB

Table 3: The Replica Size Based on the Excite log

KB, respectively (Harman, 1997). The average document size of the 20 GB TREC VLC collection is 2.8 KB (Harman, 1997). The TREC VLC text collection consists of data from 18 sources, such as news, patents, and Web sites. Our estimation uses three different numbers: 2 KB, 3 KB, and 9 KB. For query locality, we use the statistics obtained from the Excite log, since its workloads are at the level of the system we investigate. We obtain the top 200 documents for each query. The size is simply a linear function (e.g., 100 MB for 20,000 documents). Table 3 contains overestimates because we assume there is no overlap among the documents, although as shown above, distinct queries often result in overlapping documents. In Table 3, columns 1 and 2 show the query locality from the Excite log; columns 3 through 5 show the estimated replica size when we vary the average document size. For example, a 4 GB, 6 GB, and 18 GB replica satisfies at least 34.4% of queries with an average document size of 2 KB, 3 KB, and 9 KB, respectively.

3.4 When to Build or Update a Replica

Query overlap tends to decrease very gradually as time elapses. For example, 35.9% of queries on July 23 and 32.3% of queries on July 31, 1998 matched a topic in the replica covering top documents for top 1000 topics of the week of July 14 to July 20, respectively. These statistics suggest that we do not need to update the replica daily on a typical day, since significant numbers of queries match a top query that appeared several days ago. However we do need some mechanism to deal with a bursty event like the Starr Report, as shown by the sharp decrease in locality on September 11, 1998. Regular, daily updating would catch this event, but it may be too costly, react too slowly, or unnecessarily degrade performance when the system experiences the expected gradual degradation of locality. We propose two on-demand updating strategies as follows:

- Event triggered updating: watch for bursty events, and trigger the updating procedure when some special events happen.
- Performance triggered updating: watch the percentage of workloads the replica selector sends to the replicas, and trigger the updating procedure when the percentage falls below some threshold.

For event triggered updating strategy, we can simply use human intervention. When the system manager anticipates or observes a special event, and increasingly many users issue queries on it, she initiates the updating procedure. Automatic event detection is an on-going research topic. When it becomes effective, we suggest using it to trigger the updating procedure automatically. Instead of rebuilding a replica, we could add documents into replicas without deleting others for quicker updates, which means we need to save some extra space for bursty events.

Performance triggered updating is very easy to implement in the current system. We let the replica selector record the percentage of queries that it sends to each replica, when the percentage falls below a threshold, the system informs the system manager. Performance triggered updating also works for bursty events, if a lot of users search for an event that does not exist in the replicas.

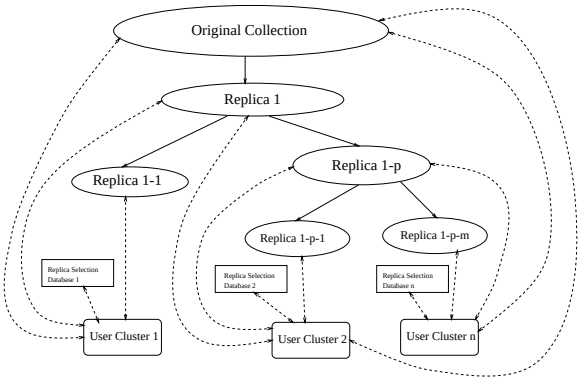


Figure 1: The replication hierarchy

4 Replication Architecture

We now propose a logical hierarchy of replicas to exploit the potential of enhanced query locality in a system with small partial replicas of a larger collection.

For our experiments, we will replicate the top documents and their index in a partial replica which helps queries about the same and similar topics but that use different terms to find their relevant documents in these partial replicas. We determine which documents to replicate as follows: for a given query, we tag all top n documents that query processing returns as "accessed" and increment their access frequencies, regardless of whether the user requests the text of these documents. We keep the access frequency for each document within a time period, e.g., a week, and then replicate the most frequently accessed documents the most.

We organize replicas as a hierarchy, illustrated in Figure 1. The top node represents an original collection that could be a single collection residing on a network node or a virtual collection consisting of several collections distributed over a network. The bottom nodes represent users. We may divide users into different clusters, each of which reside within the same domain, such as an institution, or geographical area. The inner nodes represent partial replicas. The replica in a lower layer is a subset of the replicas in upper layers, i.e., Replica 1-1 $\subset$ Replica 1 $\subset$ Original Collection. The replica that is closest to a user cluster contains the set of documents that are most frequently used by the cluster. An upper layer replica may contain frequently used documents for more than one cluster of users. The solid lines illustrate data is disseminated from the original collection to replicas. Along the arcs from the original collection, the most frequently used documents are replicated many times.

The replica selection database directs queries to a relevant partial replica or to the original collection along the arcs from the top node depending on relevance and other criteria, such as server load. The dotted lines illustrate the interaction between users and data. If we do not divide the users into different groups, the hierarchy is simply a linear hierarchy in which each layer has only one partial replica. Replica selection is a two-step process in this architecture: it ranks replicas based on relevance, and then selects one of the most relevant replicas based on load.

In the next section, we will show that the inference network model is very effective at selecting a relevant

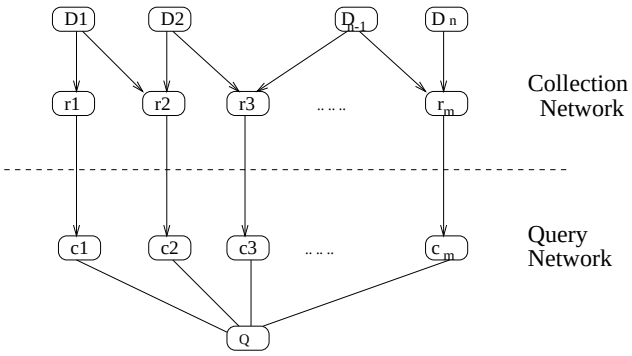


Figure 2: The collection retrieval inference network.

replica. We implement the replica selection inference network as a pseudo InQuery database, where each pseudo document corresponds to a replica or collection, and its index stores the document frequency and term frequency for each term in any of the replicas. Since the replica selection database stores document frequency and collection term frequency for each term that occurs in any of replicas, its size is determined by the number of unique terms in the largest replica. Based on our observations, the size of the replica selection database is approximately 6 MB for every 100,000 unique terms. We know the 20 GB TREC VLC collection has 13,880,064 unique terms. If our largest replica is 20 GB, the estimated size of the replica selection database is around 1.2 GB. Based on these statistics, we estimate the replica selection database for 1 terabyte of text is between 1 and 2 GB.

5 Partial Replica Selection Based on Relevance

The first step of replica selection is how to find a partial replica that contains enough relevant documents for a given query. In this section, we investigate how to do this task with inference networks, and evaluate the effectiveness of our replica selection approach using the InQuery retrieval system (Callan et al., 1992), and the 2 GB TREC Volumes 2+3 collection and the 20 GB TREC VLC collection. We use queries developed for TREC topics 51-350 in our experiments. We compare our proposed replica selection function with the collection ranking function. We measure the system's ability to pick the expected partial replica, and the precision of the resulting response as compared with searching the original collection.

The rest of the section is organized as follows: Section 5.1 investigates how to rank partial replicas and the original collection using the inference network model, Section 5.2 describes the experimental settings, Section 5.3 compares our proposed replica selection function with the collection selection function, Section 5.4 and Section 5.5 further demonstrate the effectiveness of our approach for both replicated queries and unreplicated queries, and Section 5.6 summarizes the results of this section.

5.1 Ranking Partial Replicas with the Inference Network Model

We adapt the collection retrieval inference networks (Callan et al., 1995) to rank partial replicas and the original collection. The collection retrieval inference network model consists of two component networks:

$$T = \frac{df_{ij}}{df_{ij} + k \cdot ((1-b) + b \cdot \frac{cw_i}{ave\_cw})}$$

$$I = \frac{\log(\frac{|N|}{cf} + 0.5)}{\log(|N| + 1.0)}$$

$$P(r_j|D_i) = \alpha + (1-\alpha) \cdot T \cdot I$$

where

df_{ij}	is the number of documents that contain term r_j in collection D_i,
cw_i	is the number of words in collection D_i,
ave_cw	is the average number of words,
N	is the number of collections,
cf	is the number of collections that contain r_j.
k	is a constant that controls the magnitude of df (the default is 200),
b	is a constant varying from 0 to 1 used to control the sensitivity of the function to cw (the default is 0.75), and
α	is a default belief (set to 0.4).

Figure 3: The collection ranking function in InQuery.

a collection network and a query network, illustrated in Figure 2. The D_i nodes correspond to collections, and the r_j nodes correspond to concepts in the collections. The Q node represents a query, and the c_i nodes correspond to query concepts in the query. By using the collection retrieval inference network, collection ranking becomes an estimate of $P(I|D_i)$ from combining the conditional probabilities through the network. When we adapt the collection retrieval inference network model to rank replicas, we use D_i nodes to represent the original collection and partial replicas, where D_n represents the original collection, $D_i, i = 1, 2, ...n-1$ represent partial replicas, and $D_1 \subset D_2 \subset ... \subset D_n$. (In the collection retrieval inference network, D_i nodes do not have a subset relationship.) The purpose of ranking partial replicas is to find a *single* replica that satisfies a given query instead of a subset of collections in the collection retrieval inference network. We refer to this inference network as to the replica selection inference network. As in the collection retrieval inference network model, $P(c_k|r_j)$ is set to 1.0. The central work of applying this inference network to replica selection is to develop an effective replica ranking function to estimate $P(r_j|D_i)$.

Since we adapt the collection retrieval inference network, we first examine whether the InQuery collection ranking function works well with ranking partial replicas. The InQuery collection ranking function uses *df* (the document frequency of each term) as the basic metric, and favors collections with larger *df*, as shown in Figure 3 (Callan et al., 1995). In our experiment settings in Section 5.2, the default InQuery collection ranking function directs more than 70% of the replicated queries to the original collection, however, since we use these replicated queries to build replicas, the replica selector should direct them to the replicas instead of the original collection. Although we can tune the parameters of the InQuery collection ranking function to direct more queries to the replicas, the precision drops too much, for example, the precision drops approximately 25% when the function directs 80% of replicated queries to the replicas (see Section 5.3 for the details). The

$$ave\_tf = \frac{ctf_{ij}}{df_{ij}}$$

$$cutoff_i = cutoff_1^{\frac{\log(DN_i)}{\log(DN_1)}}$$

$$AT = \begin{cases} ave\_tf & \text{if } df_{ij} > cutoff_i \\ ave\_tf \cdot \frac{df_{ij}}{cutoff_i} & \text{otherwise} \end{cases}$$

$$T = \frac{AT}{AT + k \cdot ((1-b) + b \cdot \frac{ave\_doclen_i}{ave\_ave\_doclen})}$$

$$I = \frac{\log(\frac{|N|}{rf} + 0.5)}{\log(|N| + 1.0)}$$

$$P(r_j|D_i) = \alpha + (1-\alpha) \cdot T \cdot I$$

where

ctf_{ij}	number of occurrences of term r_j in replica/collection D_i,
df_{ij}	number of documents that contain term r_j in D_i,
DN_i	number of documents in D_i,
$cutoff_1$	cutoff value for the smallest replica D_1, which we set as the number of top documents for each query,
$cutoff_i$	cutoff number of documents in D_i,
N	number of replicas plus the original collection,
rf	number of replicas and the collection that contain r_j,
ave_doclen_i	average document length in D_i,
ave_ave_doclen	average ave_doclen_i,
k	constant that controls the magnitude of AT,
b	constant varying from 0 to 1 used to control the sensitivity of the function to *ave_doclen*, and
α	default belief (set to 0.4).

Figure 4: The replica selection function.

InQuery collection ranking function does not work well with replica selection, because it favors collections with larger df, but partial replicas typically have smaller df than the original collection.

Since a partial replica contains the top documents of the most frequently used queries, by examining the document ranking function, we know that the top documents are ranked as the top, just because query terms occur more often in these documents than the others. Therefore if a replica contains the top documents for a query, the average term frequency of each query term in the replica should be higher than in the original collection. Based on this heuristic, we construct a replica selection function based on the average term frequency. In addition, we find a term is important in selecting replicas if it occurs often (with middle or high term frequency) in that replica/collection and it also occurs in a certain number of documents (above a cutoff for document frequency). A term occurring in too few documents does not help even though it has high term frequency. We need to ignore these terms. Figure 4 illustrates our replica selection function which uses the average term frequency and penalizes the terms that appear less than a given cutoff number in the corresponding replica/collection. We compare this function with the InQuery collection ranking function in Section 5.3, and demonstrate its effectiveness using 350 TREC queries on a 2 GB collection and a 20 GB

collection in Section 5.4 and Section 5.5.

We implement the replica selection inference network as a pseudo InQuery database, where each pseudo document corresponds to a replica or collection, its index stores the *df* (document frequency) and *ctf* (replica/collection term frequency) for each term. We do not store any proximity information in order to minimize the space requirements of the replica selection database. As in the collection retrieval inference network, all proximity operators are replaced with Boolean AND operators.

5.2 Experimental Settings

We evaluate the effectiveness of our replica selection approach using InQuery (Callan et al., 1992) against a 2 GB TREC collection that contains collections from TREC Volume 2 and TREC Volume 3, and a 20 GB collection that contains all TREC-6 VLC collections. We use queries developed for TREC topics 51-350 in our experiments. We measure the system's ability to pick the relevant partial replica, and the precision of the resulting response as compared with searching the original collection. We use TREC queries instead of the queries from the logs, because some of TREC queries have relevance judgments that enable us to produce precision and recall figures for evaluating the effectiveness.

By using the 2 GB collection, we compare the effectiveness of our replica selection function with the InQuery collection ranking function using short queries, and demonstrate the effectiveness of our replica selection function using both short queries and long queries. A short query is simply a sum of the terms in the corresponding description field of the topic. Long queries are automatically created from TREC topics using InQuery query generation techniques (Callan et al., 1992), which consist of terms, phrases and proximity operators. Generally, a long query for a topic is more effective than the short query (Callan et al., 1992). The average number of terms per query for the set of short queries is 8 after removing the stopwords, and the average number of terms per query for the set of long queries is 120. For each set of queries, we divide queries into two categories: replicated queries and unreplicated queries, where *the replicated queries are those whose top documents are used to build the replicas*. Since only topics 51-150 and topics 202-250 have relevance judgment files for the 2 GB TREC collection, a single trial contains 50 random unreplicated queries from these 149 topics, and we use them report the effectiveness for these topics.

We conduct our experiments by repeating the following procedure 5 times, each trial uses a different number as the seed to produce random numbers, and thus picks different queries for a query set. In each trial, we randomly choose 50 queries from queries {51-150, 202-250} as our *unreplicated* query set T, and randomly divide the remaining 250 queries in queries 51-350 into 5 sets: $\{Q_i, i = 1, 2, 3, 4, 5\}$, each set containing 50 queries. We then build a 6-layer replication hierarchy by using the 2 GB TREC collection or the 20 GB collection as the original collection C, and collecting the top n documents resulting from searching the original collection for each query in $\{Q_i, i = 1, 2, 3, 4, 5\}$ to build 5 partial replicas $\{D_i, i = 1, 2, 3, 4, 5\}$, where D_i contains at most $n * i$ documents, consisting of the top n documents for each query in query sets $\{Q_j, j = 1, ..., i\}$. Clearly, $D_1 \subset D_2 \subset D_3 \subset D_5 \subset C$. This structure mimics 5 replicas that increase in size and thus includes more of the top queries. We build a replica selection inference network to rank these

five replicas and the original collection. The queries in query sets $\{Q_i, i = 1, 2, 3, 4, 5\}$ are called *replicated* queries.

By using the 20 GB collection, we examine how the size of collection affects the effectiveness of our replica ranking function. Since we do not have relevance judgment files for topics 51-150 and topics 202-250 against the 20 GB collection, and the 2 GB collection is a subset of the 20 GB collection, we use the relevance judgment files for the 2 GB collection to produce the precision figures. We also conduct another set of experiments in order to make up for insufficient relevance judgments for topics {51-150, 202-250}. We use queries 301-350 as our unreplicated query set T, since these 50 topics are more thoroughly judged against the 20 GB VLC collection than topics {51-150, 202-250}. We use queries 51-100 as Q_1, 101-150 as Q_2, 151-200 as Q_3, 202-250 as Q_4, and 251-300 as Q_5.

When using the 2 GB collection as the original collection, the size of replicas ranges from 0.3% to 1.5%, 1% to 5%, 2% to 10%, and 5% to 20% of the original collection when replicating the top 30, 100, 200, and 500 documents, respectively. When using the 20 GB collection as the original collection, the size of replicas ranges from 0.1% to 0.5%, 0.2% to 1%, and 0.5% to 2% of the original collection when replicating the top 100, 200, and 500 documents, respectively.

When we evaluate a document or collection ranking function, we say a function is better than others if and only if it can produce higher precision at selected numbers of documents or at all standard levels of recall. In the case of replica selection, we need to add another criterion for the ranking function: directing as many queries as possible to the relevant replicas in order to improve system response time. We can tune the parameters of our functions to control the percentage of replicated queries to the replicas (as shown in Section 5.3). The range varies from 0% to 90%. None of the function we tested can direct 100% of replicated queries to the replicas. However when we direct more queries to the replicas, we have to tolerate a larger precision loss. In our experiments, we compare the precision of each function when it directs more than 80% of replicated queries to the replicas.

For a replicated query, since we know which replica contains its top documents, we define its *expected replica* as the smallest replica that is built with the top documents for the query. For an unreplicated query, since replicas may contain some relevant documents, we expect our replica selector will direct some of these queries to a relevant replica. We define the *expected replica* for an unreplicated query as the smallest replica that causes a precision drop less than 5%. For both kinds of queries, especially unreplicated queries, we expect we will have to tolerate some loss in precision in order to avoid searching the entire collection. We choose a drop in precision between 0 and 10% for a query as our acceptable range, i.e., searching the selected replica retrieves at most one less relevant document for every 10 documents as compared with searching the entire original collection.

We define *collection precise queries* as those queries that can achieve the precision above 10% when searching the original collection for the top n documents, i.e., the query finds at least one relevant document for every ten documents. We exclude collection imprecise queries when we present the ability of a replica

selector to pick the relevant replicas for unreplicated queries, because a replica with zero relevant documents is probably an acceptable choice for a query whose precision is below 10% in the original collection. We define *replica precise queries* as those for which searching the selected replica causes a precision loss less than 5% of the precision attained by searching the original collection.

5.3 Comparing Ranking Functions

In this section, we compare the effectiveness of the InQuery collection ranking function illustrated in Figure 3 and our replica selection function illustrated in Figure 4 by varying k and b for short queries in test trial 1 when we replicate the top 200 documents for each query. (We also performed experiments replicating the top 100 and 500 documents with similar results.) We will show that our replica selection function is comparable to the collection ranking function in ability to pick the expected replica for replicated queries, but that it significantly improves precision and finds the expected replica much more consistently for unreplicated queries.

Table 4 lists the results of replica selection by counting the number of queries to which replica or collection each function directs the queries, when the parameters, k and b, vary. Table 4(a) lists the results for 99 replicated queries for which we have relevance judgments. Table 4(b) lists the results for 37 unreplicated collection precise queries, 18 of which are replica precise queries. In both tables, columns 1 through 3 list the name of functions, the values of parameters k and b, and the function abbreviations. In both tables, columns 4 through 9 contains the number of queries that the replica selector sends to each of the replicas (D_i) as well as the original collection (C). (Table 4(b) only includes collection precise queries.)

For replicated queries in Table 4(a), columns 10 through 13 contain the percentages of queries that it directs to the expected replica (right), smaller replica, larger replica, and the original collection. The "expected" (E) row lists the number of judged queries that a perfect replica selector would direct to each replica and to the original collection. For unreplicated queries in Table 4(b) column 10 contains the percentages of collection precise queries that are directed to the original collection and the replicas that cause a precision loss less than 5%; columns 11 through 12 contain the percentages of collection precise queries that are directed to replicas that cause a precision loss from 5% to 10%, and more than 10%. Column 13 contains the percentage of 18 replica precise queries that are directed to the original collection. The "expected" (E) row for the unreplicated queries contains the number of queries that we expect to go to each of replicas and the original collection with less than a 5% drop in precision.

First lets consider replicated queries in Table 4(a). For the InQuery collection ranking function, varying k from 100 to 400 does not significantly change effectiveness (compare I3-I5). When we set k to 200 (the default of the InQuery collection ranking function) and increase the value of b, the replica selector directs more queries to the replicas. The default InQuery collection ranking function (k=200,b=0.75) directs only 30% of queries to the replicas, which is not our choice. When we tune the parameters to k=200 and b=1, the function directs 89% of queries to the replicas.

For the replica selection function, $k = 2$ gets better results than $k = 1$ and $k = 4$ (compare the functions R3-R5). When we decrease the value of b, the replica selector directs more queries to the replicas. For k=2

Ranking Function	Parameters k, b	Func. code	Replica D_1	D_2	D_3	D_4	D_5	C	% to replicas right	smaller	larger	C
Expected		E	18	16	25	21	19	0	100%	0%	0%	0%
Random		Ran	17	17	17	16	16	16	16%	35%	33%	16%
InQuery	200, 0.25	I1	0	0	0	0	0	99	0%	0%	0%	100%
Collection	200, 0.75	I2	0	1	4	5	20	69	14%	0%	16%	70%
Ranking	**200**, **1**	**I3**	**28**	**14**	**22**	**14**	**10**	**11**	**65%**	**16%**	**8%**	**11%**
Function	**100**, **1**	**I4**	**28**	**15**	**22**	**14**	**10**	**10**	**65%**	**17%**	**8%**	**10%**
	400, **1**	**I5**	**29**	**14**	**23**	**14**	**8**	**11**	**64%**	**17%**	**8%**	**11%**
	2, **0**	**R1**	**22**	**12**	**10**	**12**	**32**	**11**	**59%**	**7%**	**23%**	**1%**
Replica	**2**, **0.2**	**R2**	**20**	**15**	**11**	**17**	**24**	**12**	**57%**	**9%**	**22%**	**12%**
Selection	2, 0.8	R3	6	3	3	5	1	81	15%	1%	2%	82%
Function	1, 0.2	R4	17	6	7	14	28	27	47%	5%	20%	27%
	4, 0.2	R5	21	10	10	11	26	21	54%	7%	18%	21%

(a) Replicated queries (99 queries)

Ranking Function	Parameters k, b	Func. code	Replica D_1	D_2	D_3	D_4	D_5	C	Precision loss C + $< 5\%$	$5\% -$ 10%	$> 10\%$	% of replica precise queries to C
Expected		E	1	6	3	6	2	19				
Random		Ran	7	8	5	7	4	6	35%	19%	46%	22%
InQuery	200, 0.25	I1	0	0	0	0	0	37	100%	0%	0%	100%
Collection	200, 0.75	I2	0	0	0	0	6	31	89%	3%	8%	89%
Ranking	200, 1	I3	6	5	11	7	2	6	40%	30%	30%	11%
Function	100, 1	I4	5	6	12	7	2	5	38%	30%	32%	11%
	400, 1	I5	6	6	11	11	6	2	40%	30%	30%	11%
	2, 0	R1	8	7	1	2	9	10	51%	19%	30%	6%
Replica	2, 0.2	R2	7	6	2	2	8	12	68%	16%	16%	11%
Selection	2, 0.8	R3	0	0	0	0	1	36	100%	0%	0%	94%
Function	1, 0.2	R4	4	2	1	1	13	16	73%	14%	14%	22%
	4, 0.2	R5	7	2	2	1	10	15	64%	14%	22%	22%

(b) Unreplicated queries (37 collection precise queries)

Table 4: Comparing ranking functions using short queries on the 2GB TREC Volumes 2+3 collection (replicas built with top 200 documents)

and b=0.2, the function directs 88% of queries to the replicas. For unreplicated queries in Figure 5(b), we see that ranking the collection functions degrade precision by over 10% for around 30% of the queries when they avoid searching the entire collection (I3, I4, I5), and is thus not a good choice here either.

Among the functions listed in Table 4(a), six functions *random*, I3, I4, I5, R1, and R2 direct more than 80% of replicated queries to the replicas. We compare the precision of these six functions in Table 5.

The first column lists the number of documents at which we present the precision. Column 2 lists the precision when all queries go to the original collection, i.e., what percent of the top m documents is relevant when searching the original collection. Columns 3 through 8 list the results using random selection and each ranking function. The numbers in parentheses show the precision percentage difference as compared with searching the original collection. Table 5(a) lists the results for replicated queries, and Table 5(b) lists the

at m docs	Precision of Replicated Queries (%)						
	C	random	I3	I4	I5	R1	R2
10	48.7	40.4 (-17.2)	47.7 (-2.3)	48.2 (-1.2)	47.2 (-3.3)	48.4 (-0.8)	48.3 (-1.6)
20	44.8	36.1 (-19.6)	43.2 (-3.7)	43.3 (-3.4)	42.9 (-4.4)	44.4 (-1.0)	44.2 (-1.4)
30	40.7	32.4 (-20.4)	39.3 (-3.4)	39.4 (-3.0)	39.1 (-4.0)	40.2 (-1.1)	40.2 (-1.2)
100	31.1	23.9 (-23.1)	29.4 (-5.6)	29.5 (-5.5)	29.3 (-6.1)	30.5 (-2.2)	30.5 (-2.1)
200	25.1	18.7 (-25.3)	23.0 (-8.4)	23.0 (-8.4)	22.9 **(-8.7)**	24.8 (-2.8)	24.3 (-3.2)

(a) 99 Replicated queries

at m docs	Precision of Unreplicated Queries (%)						
	C	random	I3	I4	I5	R1	R2
10	39.8	24.6 (-38.2)	30.0 (-24.6)	29.4 (-26.1)	30.0 (-24.6)	33.6 (-15.6)	35.9 **(-9.6)**
20	36.8	23.7 (-35.6)	27.2 (-26.1)	26.6 (-27.7)	27.3 (-25.8)	32.3 (-12.2)	34.4 **(-7.9)**
30	33.4	22.3 (-33.1)	24.9 (-25.4)	24.3 (-27.2)	24.9 (-25.4)	30.8 (-7.8)	31.9 **(-4.6)**
100	26.4	15.0 (-43.1)	16.9 (-35.9)	16.2 (-38.7)	16.9 (-35.9)	22.8 (-13.6)	23.5 **(-10.8)**
200	21.1	10.4 (-50.5)	11.7 (-44.7)	11.1 (-47.3)	11.7 (-44.7)	17.1 (-19.2)	18.1 **(-14.5)**

(b) 50 Unreplicated queries

Table 5: Effectiveness of different ranking functions using short queries on the 2 GB TREC Volumes 2+3 collection (replicas built with top 200 documents)

results for unreplicated queries. Replicated queries produce much better results than unreplicated queries, because their top documents are stored in at least one of the replicas.

It is not surprising that random selection performs poorly, because it has high probability of picking a replica with few relevant documents. For replicated queries, it causes a precision loss ranging from 17% to 25%. For unreplicated queries, it causes a precision percentage loss ranging from 38% to 50% as compared with searching the original collection, C.

For the other five functions in Table 5, when we examine the precision for replicated queries (Table 5(a)), all these functions are acceptable, since the precision drops less than 8.7%. However, when we examine the precision for unreplicated queries (Table 5(b)), the precision difference is significant. Using InQuery collection ranking function **I3** where we set $k = 200$ and $b = 1$, the precision loss of unreplicated queries range from 24.6% to 44.7%. We get our best result using our replica selection function **R2** with $k = 2$ and $b = 0.2$. The precision of the replicated queries drops less than 3.2% of the original collection, and is better when fewer documents are returned. The precision loss of the unreplicated queries range from 4.8% to 14.5%. For the top 30 documents, the precision loss of unreplicated queries range from 4.8% to 9.6%.

In the remaining experiments, the replica selector uses the replica ranking function with $k = 2$ and $b = 0.2$, because it sends appropriate queries to replicas with an acceptable precision loss of at most 9.6% for the top 30 documents in this test suite.

5.4 Effectiveness with Replicated Queries

This section evaluates our proposed replica selection function for replicated queries on a wider range of queries and collections. For replicated queries, we want to test whether the replica selector directs most of

Size	Query Type	Top n	Average Num. of Queries to Replica					C	% to Replica			C
			D_1	D_2	D_3	D_4	D_5		right	smaller	larger	
	Expected		21.6	16.6	21.4	20.6	18.8	0				
2 GB	short	30	16.0	11.6	13.2	13.2	24.6	20.4	52.7%	4.2%	22.6%	20.6%
		100	17.2	13.0	15.8	15.2	24.0	13.8	58.4%	5.9%	21.8%	13.9%
		200	20.0	13.2	14.2	17.0	21.0	13.6	59.8%	8.1%	18.4%	13.7%
		500	25.0	14.4	15.8	18.6	15.6	9.6	65.1%	12.7%	12.5%	9.7%
		Ave.	19.5	13.0	14.8	16.0	21.3	14.3	59.0%	7.7%	18.9%	14.4%
2 GB	long	100	15.8	13.4	13.2	13.2	25.8	17.6	52.7%	5.8%	23.6%	17.8%
		200	18.0	17.4	12.6	14.4	28.2	8.4	57.0%	8.5%	26.0%	8.5%
		500	21.8	17.6	14.0	15.0	24.0	6.6	62.4%	10.7%	20.2%	6.7%
		Ave.	18.5	16.1	13.3	14.2	26.0	10.9	57.4%	8.3%	23.3%	11.0%
20 GB	short	100	15.6	14.2	12.8	15.8	20.2	20.4	54.3%	4.2%	21.0%	20.6%
		200	15.4	13.2	12.0	15.4	24.6	18.4	56.5%	4.2%	20.6%	18.6%
		500	18.2	14.4	12.2	16.0	25.8	12.4	64.2%	4.2%	19.0%	12.5%
		Ave.	16.4	13.9	12.3	15.7	23.5	17.1	58.3%	4.2%	20.2%	17.3%

Table 6: Replica selection for replicated queries

them to an expected replica. Note it is possible for a replica smaller than the expected one to contain all top documents for a given query, since the top documents of other queries could include the top documents for this query. Although we use 250 queries to build replicas, we only present the results for 99 replicated queries which have relevance judgment files in this section.

Finding the Expected Relevant Replica

This section measures the ability of the replica selector to pick the expected replica by counting the number of queries that are directed to different replicas and the original collection, as shown in Table 6. In Table 6, columns 1 and 2 indicate the size of collection and the type of queries we use in our experiments. Column 3 indicates the number of top documents for each query. The remaining columns are the same as Table 4(a).

For short queries on the 2 GB collection, on average, our replica selector directs 85.6% ($59.0\% + 7.7\% + 18.9\%$) of replicated queries to the replicas, and 66.7% of queries to the expected replica or a replica smaller than we expect. Increasing the number of replicated documents increases the accuracy of replica selection, because the replicas contain more relevant documents for replicated queries. For example, when using the top 500 documents for each query to build replicas, the replica selector directs 90.3% of queries to the replicas on the average. When using the top 100 documents, it directs 86.1% of queries to the replicas on average.

For long queries on the 2 GB collection, on average, our replica selector directs 89.0% ($57.4\% + 8.3\% + 23.3\%$) of replicated queries to the replicas, and 65.7% of queries to the expected replica or a replica smaller than expected. Increasing the number of replicated documents also increases the accuracy of replica selection, similar to the results for the short queries.

For short queries on the 20 GB collection, on average, our replica selector directs 82.7% ($58.3\% + 4.2\% + 20.2\%$) of replicated queries to the replicas, and 62.5% of queries to the expected replica or a replica smaller than we expect. Increasing the number of replicated documents increases the accuracy of replica selection,

at m docs	Precision				
	orig.	Top 30	Top 100	Top 200	Top 500
10	47.3	46.9 (-0.8)	47.0 (-0.6)	47.4 (+0.3)	46.9 (-0.8)
20	43.5	43.0 (-1.2)	43.0 (-1.1)	43.3 (-0.4)	42.9 (-1.3)
30	39.6	39.0 (-1.5)	39.1 (-1.3)	39.4 (-0.7)	39.2 (-0.9)
100	30.8		29.9 (-2.7)	30.1 (-2.0)	30.1 (-2.1)
200	24.7			24.0 (-3.0)	24.0 (-3.0)
500	16.5				16.0 (-3.1)

(a) short queries on the 2 GB collection

at m docs	Precision			
	orig.	Top 100	Top 200	Top 500
10	56.3	56.1 (-0.4)	56.4 (+0.1)	56.2 (-0.2)
20	54.6	54.2 (-0.7)	54.3 (-0.6)	54.1 (-0.9)
30	51.7	51.3 (-0.8)	51.1 (-1.1)	51.2 (-1.0)
100	41.5	41.1 (-1.1)	41.0 (-1.2)	41.0 (-1.1)
200	34.1		33.4 (-2.1)	33.6 (-1.6)
500	22.9			22.4 (-2.4)

(b) long queries on the 2 GB collection

at m docs	Precision			
	orig.	Top 100	Top 200	Top 500
10	15.5	15.5 (-1.2)	15.4 (-1.2)	15.5 (-0.3)
20	15.0	15.0 (-0.3)	15.0 (-0.5)	15.0 (+0.0)
30	14.0	14.0 (+0.0)	14.0 (-0.1)	14.0 (+0.2)
100	11.5	11.4 (-0.9)	11.5 (-0.6)	11.5 (-0.3)
200	9.8		9.7 (-0.9)	9.7 (-0.1)
500	7.5			7.4 (-0.8)

(c) short queries on the 20 GB collection

Table 7: Effectiveness of replica selection for replicated queries (each trial has 99 judged queries)

similar to the results for the 2 GB collection.

Precision of Replica Selection versus the Original Collection

Since the replica selector directs a few queries to a replica that is smaller than expected, we compare the effectiveness of executing queries against replicas or the original collection selected by the replica selector with against the original collection. Table 7 compares the average precision of replica selection over 5 test trials with searching the original collection for short queries on the 2 GB collection, long queries on the 2 GB collection, and short queries on the 20 GB collection. In these tables, column 1 lists the number of documents at which we present the precision figures. Column 2 lists the precision figures when all queries go to the original collection. Columns 3 through 6 list the precision figures when building replicas using different numbers of top documents. The numbers in the parentheses show the precision percentage difference.

For short queries on the 2 GB collection, replica selection results in a precision percentage loss less than 3.1% of searching the original collection for the same number of responses or fewer. For long queries on the

2 GB collection, replica selection results in a precision percentage loss less than 2.4%.

For short queries on the 20 GB collection, replica selection results in a precision percentage loss less than 1.2% as compared to searching the original collection, and sometimes the precision improves a little, because the replica does not contain some top-ranked unrelevant documents. In other words, selecting a smaller replica occasionally does no harm.

5.5 Effectiveness with Unreplicated Queries

This section evaluates our proposed replica selection function on a wider range of queries and collections for unreplicated queries. See Section 5.2 for detailed experimental setting.

Finding the Relevant Replica

Table 8 lists the average expected number of collection precise queries in each replica over five test trials and shows the results of replica selection by collecting the average number of collection precise queries that are directed to different replicas as well as the original collection. We list results for short queries on the 2 GB TREC Volumes 2+3 collection, long queries on the 2 GB TREC Volumes 2+3 collection, and short queries on the 20 GB TREC VLC collection. In Table 8, columns 1 and 2 indicate the size of collection and the type of the query sets. Column 3 indicates the number of documents stored for each query. The remaining columns are the same as Table 4(b).

For short queries on the 2 GB collection, on the average, our replica selector directs 83.6% ($70.2\% + 13.4\%$) of collection precise queries to the replicas that cause a precision loss less than 10% (our acceptable level) as well as the original collection, and only directs 18.1% of queries which are replica precise to the original collection. For long queries on the 2 GB collection, on the average, our replica selector directs 91.5% ($75.5\% + 16.0\%$) of collection precise queries to the replicas that cause a precision loss less than 10% as well as the original collection, and only directs 14.9% of replica precise queries to the original collection.

For short queries on the 20 GB collection, when we experiment with the same setting as the 2 GB collection, on the average, our replica selector directs 90.8% (85.1%+5.7%) of collection precise queries to the replicas that cause a precision loss less than 10% as well as the original collection. When we experiment with queries 301-350 as our unreplicated queries, our replica selector directs 87.7% (85.8%+1.9%) of collection precise queries to the replicas that cause a precision loss less than 10% as well as the original collection.

Precision of Replica Selection versus the Original Collection

This section compares the retrieval precision of executing unreplicated queries against replicas or the original collection selected by our replica selector with only searching the original collection. Table 9 lists average precision over 5 test trials for short queries on the 2 GB TREC Volumes 2+3 collection, long queries on the 2 GB TREC Volumes 2+3 collection, and short queries on the 20 GB TREC VLC collection.

For the 2 GB collection using short queries, the precision losses range from 6.8% to 17.1%. Increasing the number of replicated documents for each query improves the precision, because the replicas contain more relevant documents for each replicated query, which helps determining the similarity between unreplicated

Size	Top Type	Query n	Coll. Precise queries	Precision loss less than 5%					
				Ave. Queries Expected					
				D_1	D_2	D_3	D_4	D_5	C
2 GB	short	30	38.2	2.6	1.0	2.2	1.2	1.4	29.8
		100	37.8	3.8	2.8	3.2	2.2	1.0	24.8
		200	37.8	4.6	3.8	2.6	3.2	1.2	22.4
		500	37.8	6.2	3.8	4.4	3.4	1.4	18.6
		Ave.	37.9	4.3	2.9	3.1	2.5	1.3	23.9
2 GB	long	100	42.4	3.6	3.4	3.4	2.2	1.2	28.6
		200	42.4	5.2	3.8	3.4	2.2	2.0	25.8
		500	42.4	8.2	5.4	4.0	2.8	2.0	20.0
		Ave	42.4	5.7	4.2	3.6	2.4	1.7	24.8
20 GB	short	100	18.4	0.8	0.6	0.8	1.0	0.0	15.2
		200	19.0	0.4	1.0	1.2	1.2	0.6	14.6
		500	19.2	2.2	2.0	2.0	1.0	1.0	11.0
		Ave.	18.9	1.1	1.2	1.3	1.1	0.5	13.7
20 GB	301-350 short	100	36	0	1	1	0	3	31
		200	35	0	0	1	0	4	30
		500	35	0	1	0	1	4	29
		Ave.	35.3	0	0.6	0.6	0.3	3.7	30.0

(a) Expected number of collection precise queries in each replica

Size	Query Type	Top n	Ave. Queries to Replica					C	Precision Loss			% of Repl.Prec. queries to C
			D_1	D_2	D_3	D_4	D_5		C+$<5\%$	$5\%-10\%$	$>10\%$	
2 GB	short	30	2.6	2.0	2.0	2.8	5.6	23.2	72.4%	7.4%	16.2%	27.9%
		100	2.6	3.0	3.2	3.4	8.2	17.4	70.5%	12.2%	17.4%	13.8%
		200	4.0	3.4	2.4	4.8	6.8	16.4	71.5%	11.2%	17.3%	15.3%
		500	4.8	3.0	4.6	6.4	5.8	13.2	66.2%	22.7%	11.1%	15.4%
		Ave.	3.5	2.8	3.1	4.4	6.6	17.5	70.2%	13.4%	15.5%	18.1%
2 GB	long	100	3.6	2.4	1.8	2.8	6.6	25.2	77.5%	13.1%	9.4%	25.9%
		200	4.8	4.0	1.8	3.0	11.2	17.6	70.6%	17.6%	11.8%	13.1%
		500	5.4	4.8	3.2	5.0	10.2	13.8	78.5%	17.4%	4.2%	5.7%
		Ave.	4.6	3.7	2.3	3.6	9.3	18.9	75.5%	16.0%	8.5%	14.9%
20 GB	short	100	0.4	0.4	0.6	0.8	2.2	14.0	84.7%	4.5%	10.8%	37.0%
		200	0.6	0.6	1.2	1.4	1.8	13.4	84.2%	6.3%	9.5%	31.6%
		500	0.4	0.8	1.8	1.2	2.6	12.4	86.4%	6.2%	7.3%	29.0%
		Ave	0.5	0.6	1.2	1.1	2.2	13.3	85.1%	5.7%	9.2%	32.5%
20 GB	301-350 short	100	2	1	0	1	1	31	86.1%	0.0%	13.8%	50.0%
		200	2	1	0	1	1	30	85.7%	0.0%	14.3%	40.0%
		500	1	1	0	1	2	30	85.7%	5.8%	8.5%	40.0%
		Ave	1.7	1.0	0.0	1.0	1.3	30.3	85.8%	1.9%	12.2%	43.3%

(b) Results of replica selection for collection precise queries

Table 8: Replica selection for unreplicated queries

at m docs	Precision				
	orig.	Top 30	Top 100	Top 200	Top 500
10	42.8	37.8 (-11.9)	38.9 (-9.2)	39.4 (-8.0)	39.9 (-6.8)
20	39.4	34.0 (-13.8)	35.8 (-9.1)	36.1 (-8.4)	35.6 (-9.7)
30	35.5	30.3 (-14.6)	32.6 (-8.2)	32.7 (-7.8)	32.7 (-7.9)
100	27.2		23.3 (-14.0)	24.0 (-11.6)	24.2 (-10.8)
200	21.8			18.3 (-16.5)	18.8 (-13.9)
500	14.3				11.8 (-17.1)

(a) short queries on the 2 GB collection

at m docs	Precision			
	orig.	Top 100	Top 200	Top 500
10	55.3	52.9 (-4.3)	52.0 (-6.0)	54.8 (-0.9)
20	52.7	49.4 (-6.2)	48.6 (-7.8)	50.9 (-3.4)
30	50.0	46.2 (-7.6)	45.7 (-8.7)	47.9 (-4.3)
100	40.4	35.3 (-12.6)	35.1 (-13.2)	36.8 (-8.8)
200	33.1		27.3 (-17.4)	29.1 (-12.0)
500	21.8			18.2 (-16.6)

(b) long queries on the 2 GB collection

at m docs	Precision			
	orig.	Top 100	Top 200	Top 500
10	12.8	12.4 (-3.1)	12.6 (-1.6)	12.4 (-3.4)
20	12.3	11.6 (-5.5)	11.9 (-3.1)	11.8 (-3.4)
30	11.8	11.4 (-3.1)	11.9 (+0.8)	11.8 (+0.3)
100	10.0	9.1 (-9.6)	9.6 (-4.2)	10.1 (+0.2)
200	8.4		7.7 (-7.9)	8.3 (-1.0)
500	6.4			5.9 (-7.7)

(c) short queries on the 20 GB collection

at m docs	Precision			
	orig.	Top 100	Top 200	Top 500
10	40.4	36.2 (-10.4)	36.2 (-10.4)	37.8 (-6.4)
20	35.4	30.7 (-13.3)	30.7 (-13.3)	31.3 (-11.6)
30	31.3	26.9 (-14.2)	26.9 (-14.2)	27.0 (-14.0)
100	20.2	17.8 (-11.9)	17.8 (-11.9)	17.2 (-15.0)
200	14.4		12.8 (-10.9)	12.2 (-14.0)
500	7.8			6.7 (-14.0)

(d) short queries (topics 301-350) on the 20 GB collection

Table 9: Effectiveness of unreplicated queries (each trial has 50 queries)

and replicated queries. When the number of top retrieved documents is less than 30 documents, which are the retrieval levels that concern online users most, our replica selector causes an average precision percentage loss within 14.6% and 10% of searching the original collection, when we only replicate the top 30 documents and the top 100 documents for each replicated query, respectively. For the 2 GB collection using long queries, the precision losses range from 0.9% to 17.4%. For the top 30 retrieved documents, on the average, the precision drops less than 8.7% when we replicate more than 100 documents for each replicated queries, which is slightly better than short queries.

For the 20 GB collection using short queries, when we experiment with the same setting as the 2 GB collection and use the relevance files for the 2 GB collection, the precision ranges from losing 9.6% to improving 0.8%. For the top 30 retrieved documents, the precision loss is less than 5.5%. When we use short queries 301-350 as our unreplicated queries, the precision loss for the top 30 documents is less than 14.2%. Since topics 301-350 were much more thoroughly judged than topics $\{$51-150, 202-250$\}$ for the 20 GB VLC collection, although still only the top 30 documents of each query were judged, we think the results using topics 301-350 are more accurate, which means our replica selection performs slightly worse on the 20 GB collection than on the 2 GB collection. However, the precision percentage loss of 14.2% in our context only means we retrieve one less relevant document for the top 30 documents.

5.6 Summary

This section showed a function that selects a relevant partial replica using the inference network model. Our approach enables a system to efficiently rank partial replicas, and select one when appropriate based on relevance for a given query. We illustrate using the TREC collection that our replica selection function is more effective than previous work on collection ranking function. Our replica selection function directs at least 82% of replicated queries to a relevant partial replica rather than the original collection. When we use replicas with the 100 top documents for each query, our function achieves a precision percentage loss less than 10% for the 2 GB collection and 14.2% for the 20 GB collection, i.e., it returns one less relevant document out of the top 30 for a given query.

6 Performance of Partial Replication for Searching a Terabyte of Text

This section explores the performance benefits of partial replicas. We first briefly describe the server architectures we explore, measure a few configurations of an actual system, and compare those results to our simulator. The remaining results use a simulator which lets us easily control and vary our experiment. The bulk of this section compares partial replication with partitioning, caching, and a combination of replication and caching. It demonstrates that the additional locality benefits of replicas have performance benefits, and these benefits can be substantial.

6.1 Our Distributed Information System

We illustrate our distributed IR system in Figure 5. Clients, InQuery servers, and the connection broker reside on different machines. Clients are typically light-weight user interfaces to the retrieval system. InQuery

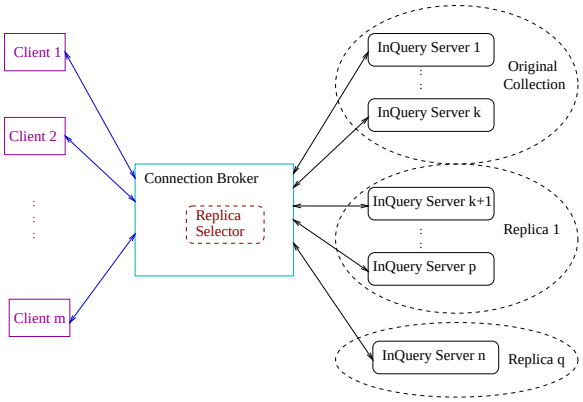


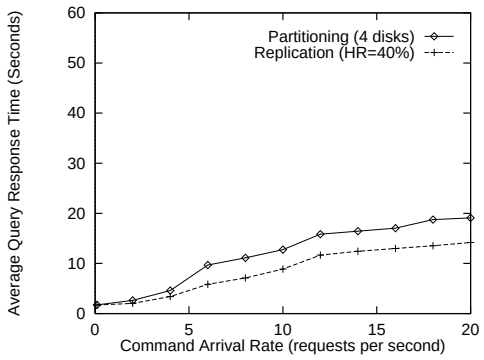
Figure 5: Our Distributed Information Retrieval System

servers store the original collection and partial replicas, and perform IR service such as query evaluation, obtaining summaries, and document retrieval. A collection or a replica may be distributed over several InQuery servers. The connection broker keeps track of all the InQuery servers for replicas or otherwise, outstanding client requests, and organizes response from InQuery servers. For partial replication, the connection broker also performs replica selection based on both relevance and load.

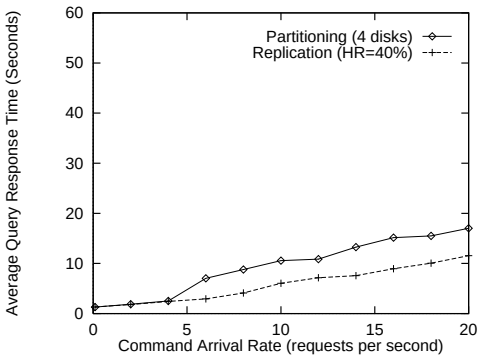
In addition to queries, our clients issue *summary*, and *document* commands to provide a more realistic command mix. For each query, a client obtains one or more summaries on relevant documents. The summary information of a document typically consists of the title and the most relevant passages in the document. A client may also retrieve complete documents.

When a client sends a query to the connection broker, the connection broker first uses a replica selector to determine whether there is a partial replica that is not only relevant to the query, but is not overloaded. If there is one, the connection broker sends the query to the InQuery server(s) that maintain the relevant replica, otherwise it sends the query to the InQuery servers that maintain the original collection. After each involved InQuery server returns the query results, the connection broker merges results and returns them to the client. For a summary command, the connection broker sends the command which contains server and document identifiers returned in a query result to the corresponding InQuery servers. The connection broker merges the summary information responses and sends a single message back to the client. For a document command, the connection broker sends the command to the InQuery server that contains the document, and then forwards the document to the client as soon as it receives the document from the InQuery server.

In this system, if query locality is high, the replica selector may send too many queries to a replica which results in load imbalance. We load balance by predicting the response time of each replica and the original collection using the average response time and the number of the outstanding queries. When the replica selector chooses a replica based on relevance, we calculate the predicted response time p_resp_j of the replica, any larger replica , and the original collection using $ave\_resp_j \cdot (1 + num\_wait\_mes_j)$, where ave_resp_j is the average response time for last 200 responses for either the replica or the original collection,



(a) the real system



(a) the simulator

Figure 6: Performance validation of simulator with partial replication.

and $num\_wait\_mes_j$ is the number of the outstanding queries to which neither the original collection or the replica have responded. We send the query to the one with the least p_resp_j. The connection broker obtains information on the response time as it receives queries responses and tracks the number of outstanding messages.

We evaluate the performance of our distributed information retrieval system using a simulator with a performance model that is driven by measurements obtained using InQuery running on DEC Alpha Server 2100 5/250 with 3 CPUs (clocked at 250 MHZ) and 1024 MB main memory, running Digital Unix V3.2D-1 (Rev 41). Servers are connected by a 10 Mbps Ethernet. In previous work, we showed the simulator closely matches a multithreaded implementation of InQuery (Cahoon et al., 2000; Lu, McKinley, & Cahoon, 1998). In addition, we report on the validation of some of our simulation results below, comparing partitioning and replication with varying degrees of locality for a 16GB collection on a single server, and again our measured times closely match our simulator. Of course, simulation enables us to explore in a controlled environment high loads and very large configurations.

6.2 Validation of Partial Replication Performance

This section compares the simulator and an implementation of partial replication for searching a 16 GB collection on a multi-tasking server using InQuery 3.1 as the query arrival rate increases on a 3-CPU Alpha Server 2100 5/250 running Digital UNIX V3.2D-1 (Rev 41). We used a multi-tasking server instead of a multithreaded server just to save us time from implementing replica selection in our legacy system, which uses too many global variables.

In this experiment, we distribute a 16 GB collection over 4 disks and used an extra disk to store a 4 GB replica. We assume queries arrive as a Poisson process, and use 50 short queries with average of 2 terms per query. Figure 6 compares the performance of using the real system and the simulator when the replica satisfies 40% of queries, and shows that two systems present the same trends and expected improvements from partial replication.

Our earlier work showed that the multitasking server performs similarly to the multithreaded server,

Parameters	Abbre.	Values
Num. of Commands	N_{com}	1000
Command Arrival Rate Poisson dist. (avg. commands/sec)	 λ	0.1 2 4 6 8 10 12 14 16 18 20
Command Mixture Ratio query:summary:document	 R_{cm}	 1:1.5:2
Terms per Query (average) shifted neg. binomial dist.	 N_{tpq}	 2
Query Term Frequency dist. from queries	 D_{qtf}	Obs. Dist.
Data per Server	S_{size}	32 GB
Size of Collection	C_{size}	1 TB
Replication Percentage	P_{repl}	3% (32 GB)
Hit Rate	HR	10% - 100% by 10

Table 10: Configuration Parameters for Terabyte Experiments

although the multithreaded server is always slightly faster (90% of measured response times fall within 10% of each other) (Lu et al., 1998). Our previous work also used and validated this simulator using parameters that matched a slower MIPS processor. Taken together these results show that our simulator is robust with respect to target architecture and could be used to model the latest, fastest processor.

6.3 Searching a Terabyte of Text

In this section, we compare the simulated performance of partial replication with collection partitioning using one and a hierarchy of replicas. We model command arrival as a Poisson process. We use short queries with an average of 2 terms per query, and set the ratio of query commands, summary commands, and document commands to 1:1.5:2, as we observed in the THOMAS log. We assume each server stores a 32 GB collection. As our baseline, we use 32 servers to store 1 terabyte of text. These experiments use a 32 GB replica, which is sufficient to satisfy more than 40% of queries in the Excite log. We vary the **hit rate** which represents the percentage of queries that the replica selector directs to partial replicas. The hit rate is also the percentage of mixed commands sent to partial replicas, since if a query is directed to a partial replica, and its corresponding summary and document commands will go to that replica too. Table 10 presents the experimental parameters, their abbreviations, and values.

Partial Replication versus Collection Partitioning

In this section, we compare the performance of the following configurations with the baseline (partitioning over 32 servers):

- Partitioning over additional servers: partitioning 1 TB of text over 33, and 64 servers, each of which stores 31 GB and 16 GB of data.
- Partial Replication: building a 32 GB replica on one additional server (33 total servers).

Figure 7 illustrates the average query response time when we partition 1 TB of text over 32, 33, and 64 servers, and over 32 servers plus one server that contains a 32 GB replica. We vary the hit rate which

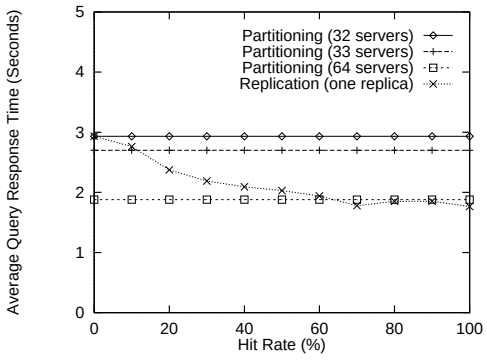
indicates the percent of queries sent to the replica. Figure 7(a)-(c) illustrate when commands arrive at 4, 10, and 20 commands per second. The graphs plot query response time versus the hit rate. When we have one additional server, using it to store a replica performs significantly better than further partitioning over this server, especially when the commands arrive at a high rate, as shown in Figure 7(c). The improvement occurs when the replica satisfies only 3% of commands for more highly loaded systems, e.g., 10 and 20 commands per second, and the improvement increases with increases in query locality. Using one partial replica also performs similar or better than partitioning over twice as many as servers when the replica satisfies at least 20% of commands. For example, when the arrival rate is 20 commands per second, partitioning over 64 servers reduces the average query response time by a factor of 1.6, while one partial replica reduces it by a factor of 2.3 when the replica satisfies 40% of commands. When the hit rate becomes high, the replica selector load balances between the replica and the partitioned original collection which maintains retrieval effectiveness and quick response times.

There are two major reasons that partial replication outperforms partitioning. (1) A server takes around 3/5 the time to search half the data according to our measurements, and thus when we partition a terabyte of text over 64 servers instead of 32 servers, each server can not process twice as many commands as using 32 servers. (2) Searching a replica results in less network traffic and needs less coordination of the results from each partition in the connection broker. For example, the utilization of the network and the connection broker for partitioning over 64 servers are 28% and 70%, while the corresponding utilization for using 32 servers and one partial replica is 12% and 58%. For highly loaded systems, replication significantly improves performance over partitioning and uses only about half of the resources!

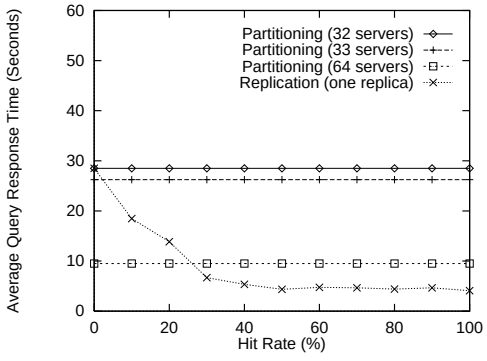
6.4 Partial Replication as a Hierarchy

In this section, we assume 1, 2, and 4 additional servers and organize them as a hierarchy of replicas. We examine how much improvement a hierarchy of replicas will produce. We assume the first, second, third, and fourth additional server stores 32 GB, 16 GB, 8 GB and 4 GB of data. where $R_1 \supset R_2 \supset R_3 \supset R_4$ and R_i represents the data on the i-th additional server. The replicas satisfy accurately a total of p% of commands, i.e., the hit rate is p%. Of these p% of commands the replica selector sends to replicas, all are satisfied by the largest replica, 10% less, i.e., (p% - 10%) are satisfied by the second largest replica, another 10% less, i.e., (p% - 20%) by the third largest replica, and (p% - 30%) by the fourth replica, where all the commands sent to a i-th largest replica may also be satisfied at the $(i-1)$-th largest replica.

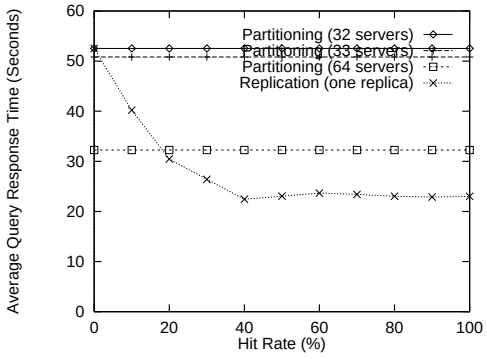
Figure 8 illustrates the average query response time when we build one and two replicas, where the replicas' hit rate (HR) is 20%, 40%, and 60%, as well as four replicas, where the hit rate is 40% and 60%. The results show that 2 replicas are sufficient to achieve large performance improvements beyond partitioning when the replicas satisfy 40% and 60% of commands. In our baseline, partitioning over 32 servers achieves an average query response time below 10 seconds at 7 commands per second. Using one replica to satisfy 20% of commands and using two replicas to satisfy 40% and 60% of commands achieve average query response time below 10 seconds at 9, 16, and more than 20 commands per second, respectively, while partitioning over



(a) $\lambda = 4$

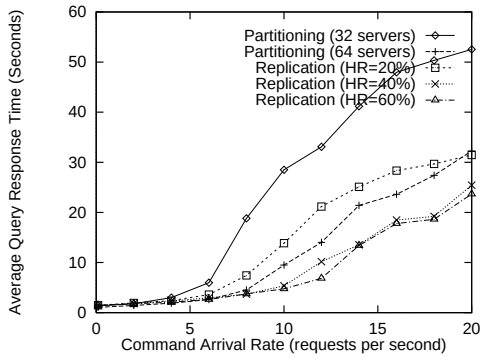


(b) $\lambda = 10$

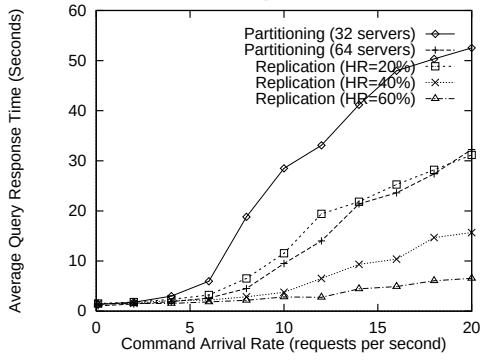


(c) $\lambda = 20$

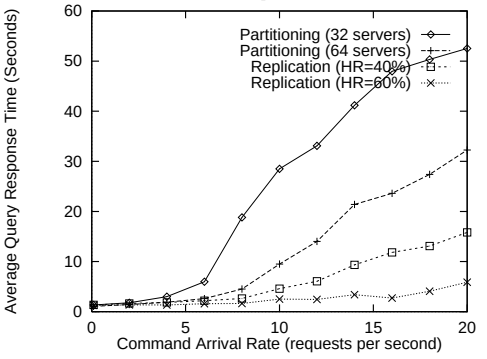
Figure 7: Partial replication versus partitioning as a function of hit rate, for three command arrival rates (λ = 4, 10, 20 requests per second)



(a) one replica



(b) two replicas



(c) four replicas

Figure 8: Partial replication versus a hierarchy of replicas as a function of command arrival rate

64 servers (using 32 additional servers) only achieves average query response time below 10 seconds at 10 commands per second.

Thus, for our system (slower than the current state of the art), we achieve query response times under 10 seconds for a relatively highly loaded system with 20 requests per second using 4 replicas and query locality of about 50%. With a faster base system, replication is still preferable to partitioning given even modest query locality, however fewer replicas are necessary to maintain fast response times. We show this result elsewhere (Lu, 1999).

6.5 Partial Replication versus Caching

In this section, we compare the performance of partial replication to caching which is the most widely used mechanism to improve performance for IR systems. We still use partitioning 1 TB of text over 32 servers as the baseline, and compare it with the following configurations:

Partitioning: partition 1 TB of text over one additional server (33 servers in total), each of which stores 31 GB of data.

Connection broker caching: partition 1 TB of text over 33 servers and build a cache in the main memory of the connection broker.

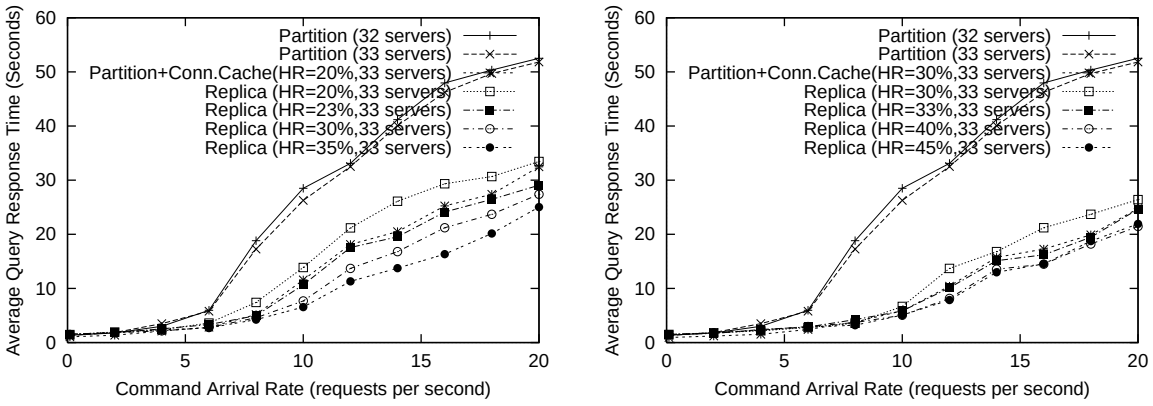
Server caching: partition 1 TB of text over 33 servers and build a cache in the main memory of each InQuery server.

Partial Replication: partition 1TB of text over 32 servers and use one additional server to build a partial replica.

Partial replication and connection broker caching: partition 1TB of text over 32 servers and use one additional server to build a partial replica, and also build a cache in the main memory of the connection broker. When a command comes in, first check the cache, if it is not in the cache, then use the replica selector to select the relevant replica. We assume that the connection broker cache satisfies 10% of commands, and the replica satisfies $HR - 10\%$ of commands.

For caching, we present an upper bound of its performance; we only count the time for cache lookup and assume cache replacement takes no time. We assume the result lists for most frequently used queries are in cache. We also assume the documents and query summaries are in memory, although not many machines have several to several tens of GB worth of memory. For the partial replica, we assume the summaries and documents must be fetched from disk. We thus give the cache a large advantage. We explore server caching to show the benefits of same server caching for independent collections which may not permit connection broker caching.

Figures 9(a) and (b) compare connection broker caching and partial replication. They illustrate the average query response time versus the command arrival rate when the connection broker cache satisfies 20% and 30% of commands, as we found in our logs. The results show that if the partial replica and the connection broker cache satisfy the same amount of commands, partial replication results in slightly worse performance



(a) a cache hit rate of 20%

(b) a cache hit rate of 30%

Figure 9: Partial replication versus caching as a function of command arrival rate

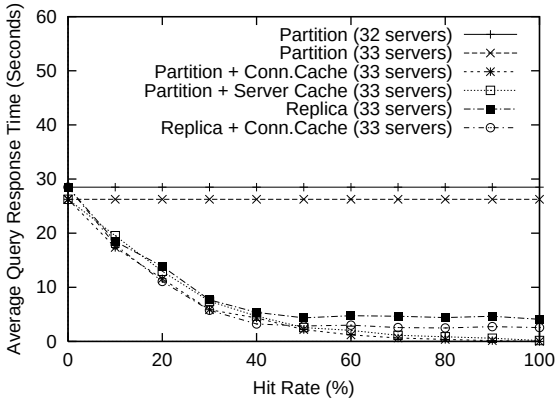


Figure 10: Partial replication versus caching as a function of hit rate at $\lambda = 10$

as compared with connection broker caching. But when the replica hit rate increases by just 3%, partial replication performs better than connection broker caching; it performs almost a factor of 2 better when the replica hit rate increases by 15% for high command arrival rates. As we showed in Section 3.2, replicas can improve locality from 7% to over 20% over time as compared with exact match caching. Searching the replica is so much faster than searching the entire collection, even small amounts of locality have a significant impact on performance.

Figure 10 demonstrates the effect of the hit rate more clearly. It plots the average query response time versus the hit rate when commands arrive at 10 commands per second. Figure 10 shows that the performance of server caching is slightly worse than connection broker caching, since the connection broker cache eliminates the coordination time of multiple servers and reduces network traffic between the connection broker and InQuery servers. Partial replication outperforms connection broker caching when the replica satisfies 3% or more of commands until partial replication needs load balancing (which occurs when the hit rate is around 40%). After this point, partial replication performs significantly worse than connection broker caching, since it redirects significant amount of commands to the original servers, while caching does not. The log analysis in Section 3 shows that the cache achieves a hit rate between 20% - 30%, or less in most cases. However,

combining the connection broker cache and partial replication further improves performance; an unsurprising result at this point.

7 Conclusions

In this paper, we investigated how to search a terabyte of text using partial replication. We built a hierarchy of replicas based on query frequency and available resources, and used the InQuery retrieval system for the replicas and the original collection. We examined queries from THOMAS (THOMAS, 1998) and Excite (Excite, 1997) to find locality patterns in real systems. We find there is sufficient query locality that remains high over long periods of time which will enable partial replication to maintain effectiveness and significantly improve performance as compared to caching. For THOMAS, updating replicas hourly or even daily is unnecessary. However, we need to some mechanism to deal with bursty events. We propose two simple updating strategies that trigger updates based on events and performance, instead of regular updating. In our traces, requiring exact match for a query misses locality between queries with different terms that in fact return the same top documents, whereas partial replication with an effective replica selection function will find the similarities. We believe this trend will hold for other query sets against text collections and for web queries.

We investigate how to select a relevant partial replica using the inference network, and demonstrate the effectiveness of our approach using the InQuery retrieval system and TREC collections. The results show that the inference network model is a very promising approach for ranking partial replicas. By using our new replica selection function, our replica selector can direct at least 82% of replicated queries to a relevant partial replica rather than the original collection, and it achieves a precision percentage loss within 10% and 14.2% for the top 30 retrieved documents for those unreplicated queries, when sizes of replicas range from 2% to 10% for the 2 GB collection, and 0.2% to 1% for the 20 GB collection, respectively.

We demonstrate the performance of our system searching a terabyte of text using a validated simulator. We compare the performance of partial replication with partitioning over additional servers. Our results show that partial replication is more effective at reducing execution times than partitioning on significantly fewer resources. For example, using 1 or 2 additional servers for replica(s) achieves similar or better performance than partitioning over 32 additional servers, even when the largest replica satisfies only 20% of commands. Higher query locality further widens the performance differences. We also compare partial replication with caching under a variety of workloads. The performance of partial replication with a connection broker exceeds that of connection broker caching as well as server caching under a variety of configurations when the partial replica increases the hit rate by at least 3%. Our work porting and validating InQuery and the simulator from a slower MIPS processor to the Alpha, as well as experiments with faster querying times which are reported elsewhere (Lu, 1999; Lu & McKinley, 2000), lead us to believe the performance trends will hold for faster systems using fewer resources. Although the simplicity of caching is appealing, a combined approach that incorporates partial replication will yield both an effective and better performing system.

Acknowledgments

The authors performed this research at the University of Massachusetts at Amherst. This material was supported in part by the National Science Foundation, Library of Congress and Department of Commerce under cooperative agreement number EEC-9209623, supported in part by United States Patent and Trademark Office and Defense Advanced Research Projects Agency/ITO under ARPA order number D468, issued by ESC/AXS contract number F19628-95-C-0235, and also supported in part by grants from Digital Equipment, NSF grant EIA-9726401, and an NSF Infrastructure grant CDA-9502639. Kathryn S. McKinley was supported by an NSF CAREER award CCR-9624209, NSF ITR grant CCR-0085792, and F33615-01-C-1892. Any opinions, findings and conclusions or recommendations expressed in this material are the authors and do not necessarily reflect those of the sponsors.

We would like especially to thank Bruce Croft for his enthusiastic and continued support of this work. We thank Ben Mealey and Library of Congress for providing the THOMAS log. We thank Doug Cutting and Excite for providing the Excite log. We also appreciate the detailed comments of one of our reviewers which helped us to improve our presentation.

References

Baentsch, M., Molter, G., & Sturm, P. (1996). Introducing application-level replication and naming into today's Web. In *Proceedings of fifth international world wide web conference.* Paris, France; Available at http://www5conf.inria.fr/fich_html/papers/P3/Overview.html.

Bell, D., & Grimson, J. (1992). *Distributed database systems.* Addison-Wesley Publishers.

Bestavros, A. (1995). Demand-based document dissemination to reduce traffic and balance load in distributed information systems. In *Proceedings of spdp'95: The 7th ieee symposium on parallel and distributed processing* (pp. 338–345). San Anotonio, Texas.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. In *Proceedings of www7* (pp. 107–117). Brisbane, Australia. (`www7.scu.edu.au/programme/fullpapers/1921/com1921.htm`)

Brown, E. W., & Chong, H. A. (1997). The GURU system in TREC-6. In *Proceedings of the sixth text REtrieval conference (trec-6)* (pp. 535–540). Gaithersburg, MD.

Burkowski, F., Cormack, G., Clarke, C., & Good, R. (1995). *A global search architecture* (Tech. Rep. No. CS-95-12). Waterloo, Canada: Department of Computer Science, University of Waterloo.

Burkowski, F. J. (1990). Retrieval performance of a distributed text database utilizing a parallel process document server. In *1990 international symposium on databases in parallel and distributed systems* (pp. 71–79). Trinity College, Dublin, Ireland.

Cahoon, B., & McKinley, K. S. (1996). Performance evaluation of a distributed architecture for information retrieval. In *Proceedings of the nineteenth annual international acm sigir conference on research and development in information retrieval* (pp. 110–118). Zurich, Switzerland.

Cahoon, B., McKinley, K. S., & Lu, Z. (2000). Evaluating the performance of distributed architectures for information

retrieval using a variety of workloads. *ACM Transaction on Information Syetems*, *18*(1), 1–43.

Callan, J. P., Croft, W. B., & Broglio, J. (1995). TREC and TIPSTER experiments with INQUERY. *Information Processing & Management*, *31*(3), 327–343.

Callan, J. P., Croft, W. B., & Harding, S. M. (1992). The INQUERY retrieval system. In *Proceedings of the 3rd international conference on database and expert system applications* (p. 78-93). Valencia, Spain.

Callan, J. P., Lu, Z., & Croft, W. B. (1995). Searching distributed collections with inference networks. In *Proceedings of the eighteenth annual international acm sigir conference on research and development in information retrieval* (pp. 21–29). Seattle, WA.

Chakravarthy, A., & Haase, K. (1995). Netserf: Using semantic knowledge to find internet information archives. In *Proceedings of the eighteenth annual international acm sigir conference on research and development in information retrieval* (pp. 4–11). Seattle, WA.

Couvreur, T. R., Benzel, R. N., Miller, S. F., Zeitler, D. N., Lee, D. L., Singhai, M., Shivaratri, N., & Wong, W. Y. P. (1994). An analysis of performance and cost factors in searching large text databases using parallel search systems. *Journal of the American Society for Information Science*, *7*(45), 443–464.

Croft, W. B., Cook, R., & Wilder, D. (1995). Providing government information on the Internet: Experiences with THOMAS. In *The second international conference on the theory and practice of digital libraries.* Austin, TX.

Danzig, P. B., Ahn, J., Noll, J., & Obraczka, K. (1991). Distributed indexing: A scalable mechanism for distributed information retrieval. In *Proceedings of the fourteenth annual international acm sigir conference on research and development in information retrieval* (pp. 221–229). Chicago, IL.

DeWitt, D., Graefe, G., Kumar, K. B., Gerber, R. H., Heytens, M. L., & Muralikrishna, M. (1986). GAMMA – a high performance dataflow database machine. In *Proceedings of the Twelfth International Conference on Very Large Data Bases* (pp. 228–237). Kyoto, Japan.

DeWitt, D., & Gray, J. (1992). Parallel database systems: The future of high performance database systems. *Communications of the ACM*, *35*(6), 85–98.

Excite. (1997). *http://www.excite.com*.

French, J. C., Powell, A. L., C. L. Viles, J. C. anc, Emmeitt, T., Prey, K. J., & Mou, Y. (1999). Comparing the performance of database selection algorithms. In *Proceedings of the twenty-second annual international acm sigir conference on research and development in information retrieval* (pp. 238–245). Berkeley, CA.

Fuhr, N. (1999). A decision-theoretic approach to database selection in networked ir. *ACM Transactions on Information Systems*, *17*(3), 229-249.

Gravano, L., & Garcia-Molina, H. (1995). Generalizing GLOSS to vector-space databases and broker hierarchies. In *Proceedings of the Twenty First International Conference on Very Large Data Bases* (pp. 78–89). Zurich, Switchland.

Gravano, L., Garcia-Molina, H., & Tomasic, A. (1994). The effectiveness of GLOSS for the text database discovery problem. In *Proceedings of the 1994 acm sigmod international conference on management of data* (pp. 126–137).

Minneapolis, MN.

Hagmann, R. B., & Ferrari, D. (1986). Performance analysis of several backend database architectures. *ACM Transactions on Database Systems, 11*(1), 1–26.

Harman, D. (Ed.). (1997). *The sixth text retrieval conference (TREC-6).* Gaithersburg, MD: National Institute of Standards and Technology Special Publication.

Harman, D., McCoy, W., Toense, R., & Candela, G. (1991). Prototyping a distributed information retrieval system that uses statistical ranking. *Information Processing & Management, 27*(5), 449–460.

Hawking, D. (1997). Scalable text retrieval for large digital libraries. In *First european conference on research and advanced technology for digital libraries* (pp. 127–145). Pisa, Italy: Springer.

Hawking, D., Craswell, N., & Thistlewaite, P. (1998). Overview of TREC-7 very large collection track. In *Proceedings of the seventh text REtrieval conference (trec-7)* (pp. 91–104). Gaithersburg, MD.

Hawking, D., & Thistlewaite, P. (1997). Overview of the TREC-6 very large collection track. In *Proceedings of the sixth text REtrieval conference (trec-6)* (pp. 93–106). Gaithersburg, MD.

Holmedahl, V., Smaith, B., & Yu, T. (1998). Cooperative caching of dynamic content on a distributed web server. In *Proceedings of HPDC-7* (p. 243-250). Chicago, IL.

Katz, E., Butler, M., & McGrath, R. (1994). A scalable HTTP server: the NCSA prototype. *Computer Networks and ISDN Systems, 27*(2), 155–164.

Lu, Z. (1999). *Scalable distributed architectures for information retrieval.* Unpublished doctoral dissertation, University of Massachusetts at Amherst.

Lu, Z., & McKinley, K. S. (1999). Partial replica selection based on relevance for information retrieval. In *Proceedings of the 22th annual international acm sigir conference on research and development in information retrieval* (pp. 97–104). Berkeley, CA.

Lu, Z., & McKinley, K. S. (2000). Partial replica selection versus caching for information retrieval. In *Proceedings of the 23rd annual international acm sigir conference on research and development in information retrieval* (p. 248-255). Athens, Greek.

Lu, Z., McKinley, K. S., & Cahoon, B. (1998). The hardware/software balancing act for information retrieval on symmetric multiprocessors. In *Proceedings of europar'98.* Southhampton, U.K.

Mackert, L. F., & Lohman, G. M. (1986). R^* optimizer validation and performance evaluation for distributed queries. In *Proceedings of the Twelfth International Conference on Very Large Data Bases* (pp. 149–159). Kyoto, Japan.

Markatos, E. P. (1999). *On caching search engine results* (Tech. Rep. No. 241). Institute of Computer Science (ICS) Foundation for Research & Technology - Hellas (FORTH), Greece.

Martin, T. P., Macleod, I. A., Russell, J. I., Lesse, K., & Foster, B. (1990). A case study of caching strategies for a distributed full text retrieval system. *Information Processing & Management, 26*(2), 227–247.

Martin, T. P., & Russell, J. I. (1991). Data caching strategies for distributed full text retrieval systems. *Information Systems, 16*(1), 1–11.

Saraiva, P. C., Moura, E. S. de, Ziviani, N., Meira, W., Fonseca, R., & Ribeiro-Neto, B. (2001). Rank-preserving two-level caching for scalable search engines. In *Proceedings of the 24th annual international acm sigir conference on research and development in information retrieval* (pp. 51–58). New Orleans, LA.

Simpson, P., & Alonso, R. (1987). Data caching in information retrieval systems. In *Proceedings of the tenth annual international acm sigir conference on research and development in information retrieval* (pp. 296–305). New Orleans, LA.

Stonebraker, M., Woodfill, J., Ranstrom, J., Kalash, J., Arnold, K., & Anderson, E. (1983). Performance analysis of distributed data base systems. In *Proceedings of the Third Symposium on Reliability in Distributed Software and Database Systems* (pp. 135–138). Clearwater Beach, FL.

THOMAS. (1998). legislative information on the internet. *http://thomas.loc.gov*.

Tomasic, A., & Garcia-Molina, H. (1992). *Caching and database scaling in distributed shared-nothing information retrieval systems* (Tech. Rep. No. STAN-CS-92-1456). Stanford University.

Turtle, H. R. (1991). *Inference networks for document retrieval.* Unpublished doctoral dissertation, University of Massachusetts.

Voorhees, E. M., Gupta, N. K., & Johnson-Laird, B. (1995). Learning collection fusion strategies. In *Proceedings of the eighteenth annual international acm sigir conference on research and development in information retrieval* (pp. 172–179). Seattle, WA.

Wang, J. (1999). A survey of web caching schemes for the internet. *Computer Communication Review*, *29*(5), 36–46.

Xu, J., & Croft, W. (1999). Cluster-based language models for distributed retrieval. In *Proceedings of the twenty-second annual international acm sigir conference on research and development in information retrieval* (p. 254-261). Berkeley, California.