Searching a Terabyte of Text Using Partial Replication

Zhihong Lu Kathryn S. McKinley

Department of Computer Science, University of Massachusetts, Amherst, MA 01003

{zlu, mckinley}@cs.umass.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. In this paper, we investigate using partial replication to search a terabyte of text in our distributed IR system. We use a replica selection database to direct queries to relevant replicas that maintain query effectiveness, but at the same time restricts some searches to a small percentage of data to improve performance and scalability, and to reduce network latency. Using a validated simulator, we compare database partitioning to partial replication with load balancing, and find partial replication is much more effective at decreasing query response time, even with *fewer* resources, and it requires only modest query locality. We also demonstrate the average query response time under 10 seconds for a variety of work loads with partial replication on a terabyte text database. We further investigate query locality with respect to time, replica size, and replica updating costs using real logs from THOMAS and Excite, and discuss the sensitivity of our results to these sample points.

1 Introduction

As information and users proliferate through the Internet and intranets, distributed information retrieval (IR) systems must cope with the challenges of scale. Distributing excessive workloads and searching as little data as possible while maintaining acceptable retrieval accuracy are two ways to improve performance and scalability of a distributed IR system. Partial replication of a text database serves these two purposes. Replicating a text databases on multiple servers can distribute excessive workloads posed on a single server. Partial replication on servers that are closer to their users can improve performance

by reducing network traffic and minimizing network latency. Replicating a small percentage of a text database and directing queries to a relevant partial replica further improves performance by searching less data.

Partial replication is truly useful for information retrieval only when it satisfies the following four conditions: (a) query and document accesses localize in a small portion of a text database for some period of time; (b) there is an effective tool to select a relevant partial replica; (c) replica selection incurs reasonable time and space overheads; and (d) updating costs are reasonable with respect to their frequency. In previous work, we describe and evaluate an effective replica selection model that satisfies condition (b) [24]. Here, we investigate the issues related to conditions (a), (c), and (d).

We describe how to build a hierarchy of replicas based on query frequency and available resources. Our system uses InQuery for our basic IR functionality for all text databases: original and replicated [9]. Using a validated simulator, we compare the performance of searching a terabyte of text using partial replication to partitioning, and 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. We investigate query locality using real logs from THOMAS [21] and Excite [13] and find query locality is sufficient to justify partial replication (as others have also found [12, 18]). In addition, we find that caching will miss many overlaps between queries with different terms that in fact return the same top documents. We measure overheads and updating costs for replicas and the replica selection database in our system, and find that partial replication for information retrieval can be implemented at a reasonable cost when updates are not necessary on a daily basis. For our traces, updating at most weekly or on demand for important events will maintain query locality and thus the performance benefits of replication.

Although distributed databases and the Web all use replication to distribute workloads and improve system performance [1, 2, 3, 4, 5, 10, 14, 20, 28, 30, 31], there are several distinctions that make this work unique. First, to our knowledge, nobody reports performance for a terabyte of text, or even half. Second, text databases are different from the focus of previous database work. Text is unstructured and it is generally read-only, while the databases of previous work use structured data such as records and

objects, and the data is read-write. In addition, structured databases can simply use set membership to find a replica. In a text database, a selection function needs to determine whether a replica contains all, some, or none of the relevant documents in order to maintain retrieval effectiveness. Third, previous work on the Web replicates documents only for document access, while our work includes the search procedure, and thus improve system performance for both query processing and document access.

The remainder of this paper is organized as follows. The next section further compares our work to related work. Section 3 describes the replication architecture. Section 4 shows the performance of searching a terabyte of text using partial replication. In Section 5, we characterize the locality and access patterns of our test logs from THOMAS and Excite, and relate the logs to our results. Section 6 and 7 discuss space overheads and updating costs for replicas and the replica selection databases. Section 8 summarizes our results and concludes.

2 Related Work

A number of studies have investigated the performance of distributed IR systems [6, 7, 11, 16, 25, 26, 29]. Most of the previous work experiments with a text database less than 1 GB and focuses on speedup when a text database is distributed over more servers [6, 16, 22, 25, 26, 27]. Only Couvreur *et al.* [11], and Cahoon and McKinley [7] use simulation to experiment with more than 100 GB data. To our knowledge, no one has reported the performance of partial replication for searching a terabyte of text in distributed information retrieval systems. None of these previous studies include partial replication, and of course they do not compare database partitioning with replication.

Both research and commercial systems (e.g. Oracle, Informix, and Sybase) in the area of distributed databases have used replication for a long time to improve system performance and availability [1, 2, 19, 28, 30]. This work uses structured data, such as objects, and presents algorithms for updating read-write data to ensure consistency of different copies of data. In a structured database, a query response is set membership, a straightforward test. Text IR systems instead return a user parameterized range of responses with varying belief values, and thus the system cannot summarize queries into set membership tests. Basically, there is no single correct response in text searches. (Although, this feature may be at odds with what users expect!)

Recently, researchers have used replication in the Web for document access [3, 4, 31]. Generally, they cache popular documents in a hierarchy of proxy servers which are located between clients and Web servers to reduce Internet traffic. Although we also adopt a replication hierarchy to store the documents of the most frequently used queries, our replicas are searchable and thus can speed up both query and document processing.

InQuery, our base system, is not the fastest text retrieval system available today [17]. For the experiments in this paper, we model it and validate against a 3 processor 250MHz Alpha which can maintain response times of under 10 seconds with 4 to 5 disks on a collection size of up to 16 GB for a heavily loaded system. Other multiprocessor systems[17] have recently reported results for a single query (rather than a loaded system) that are less than a second on 100 GB collection. We believe our results on replication versus partitioning are directly applicable to these systems as well, although they store more data on a single machine. Localizing significantly more data in a single machine coupled with faster CPU and disk processing correspondingly decreases the number of CPUs and disks per CPU we would need to achieve similar performance, and serves to decrease communication over the local network, but it will not change general trends. In fact, we first modeled a slower version of InQuery on a slower processor [7]. After an Alpha port, we again measured CPU and disk processing times for query processing and other IR commands, fed them into our system, and revalidated. We found all our trends were the same, and still our simulator closely matched the system. Thus, although the amount of hardware we need in this paper is beyond what most organizations would be willing to use, even for a terabyte of text, we believe that the trends are correct and will follow increases in retrieval, CPU, and disk speed.

3 Replication Architecture

In this section, we describe how to build a hierarchy of replicas in our system. As suggested by our log analyses in Section 5, we need to speed up both query processing and document access to maximize performance improvements. One way to achieve this goal is to cache the most frequently used queries along with their top documents, and do string comparison to determine whether a query is cached. However using this approach can not match a cached query when the new query is about the same topic, but uses different query terms. In our logs, 35% to 62% of the topics that occur more than once contain

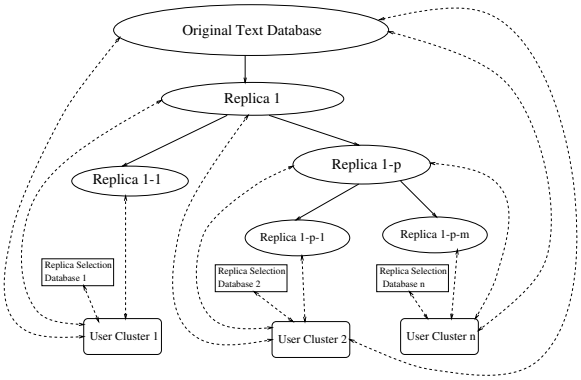


Figure 1: The replication hierarchy

more than one unique query that returns the same top 20 documents. We solve this problem by storing the top documents instead of both queries and documents, and making these documents searchable, which helps queries about the same and similar topics but using different terms to find their relevant documents in these partial replicas. In our system, each partial replica is a text database that contains the top documents of the most frequently used queries and their index. 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 Fig 1. The top node represents an original text database that could be an actual text database residing on a network node or a virtual text database consisting of several text databases 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 Text Database. 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 text database to replicas. Along the arcs from the original text database, the most frequently used documents are replicated many times.

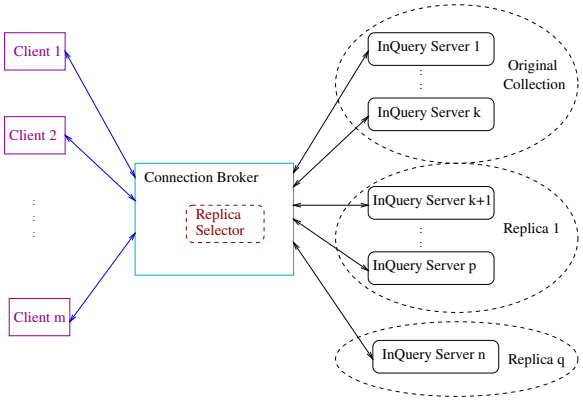


Figure 2: Our Distributed Information Retrieval System

The replica selection database directs queries to a relevant partial replica. It may send user requests to any replica or the original text database along the arcs from the top node depending on relevance and other criteria, such as server loads. 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. 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 our previous work, we showed that the inference network model is very effective at selecting a relevant replica [24]. We implement the replica selection inference network as a pseudo InQuery database, where each pseudo document corresponds to a replica or text database, and its index stores the document frequency and term frequency for each term in any of the replicas. We compared the function we use here with several others, and found it maintained the highest effectiveness while searching the least data. For the queries used to build the replicas, our replica selector directs 85% of queries to a replica and retrieves only one less relevant document for the top 200 documents on average. For unreplicated queries, typically only some or none of their top documents are in the replicas, and on average, the replica selector retrieves one less relevant document out of the top 30 documents.

4 Performance of Partial Replication for Searching a Terabyte of Text

In this section, we further describe the architecture of our distributed IR system, and compare the performance of partial replication with database partitioning when searching a terabyte of text.

4.1 Our Distributed Information System

In our distributed IR system illustrated in Figure 2, clients, InQuery servers, and the connection broker reside in different machines. Clients are user interfaces to the retrieval system. InQuery servers store the original text database and partial replicas, and perform IR service such as query evaluation, obtaining summaries, and document retrieval. A text database 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 includes a component to do replica selection based on both loads and relevance.

In our system, if query locality is high, the replica selector may send too many queries to a replica which results in load imbalance. We balance loads by predicting the response time of each replica and the original text database using the average response time and the number of the outstanding commands. When the replica selector chooses a replica based on relevance, we calculate the predicted response time p_resp_j of the replica, the replicas larger than this, and the original text database 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 text database, and $num\_wait\_mes_j$ is the number of the outstanding commands to which either the original database or the replica have not yet responded. We send the command to the one with the least p_resp_j. The connection broker obtains information on the response time as queries responses are returned and tracks the number of outstanding messages.

We evaluate the performance of our distributed information retrieval system using a simulator that uses 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), and servers are connected by a 10 Mbps Ethernet. In previous work, we showed the simulator closely matches a multithreaded implementation of InQuery [8, 23]. In addition, we validated some of our simulation results below comparing partitioning and replication with varying degrees of locality for a 16GB text database on a single server, and again our measured times closely match our simulator [23]. Of course, simulation enables us to explore in a controlled environment high loads and very large configurations.

Parameters	Abbre.	Values
Num. of Commands	NC	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	CM	1:1.5:2
Terms per Query (average) shifted neg. binomial dist.	TPQ	2
Query Term Frequency dist. from queries	QTF	Obs. Dist.
Number of CPUs	CPU	4
Number of Disks	DISK	8
Number of Threads	TH	32
Size of Text Database	CSIZE	1 TB
Replication Percentage	RP	3% (32 GB)
Distracting Percentage	DP	10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

Table 1: Configuration Parameters for Terabyte Experiments

4.2 Searching a Terabyte of Text

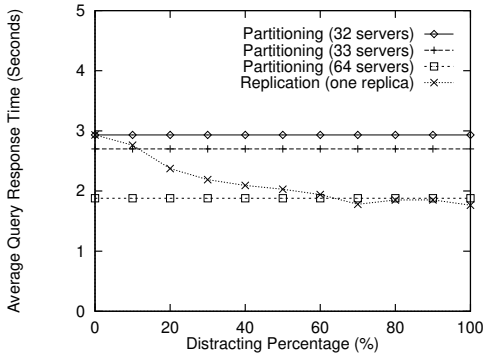
In this section, we compare the simulated performance of partial replication with database partitioning. We also demonstrate the performance using a hierarchy of replicas. We model command arrival as a Possion process. We use short queries with an average of 2 terms per query, and set the ratio of query commands, summary commands, and document commands is 1:1.5:2, as we found in the THOMAS log described below. We assume each server has 4 CPUs, 8 disks, 32 threads, and each disk stores up to 4 GB of data; each server thus stores a 32 GB database. As our baseline, we use 32 servers to store 1 terabyte of text. Table 1 presents the parameters, their abbreviations, and values in the experiments.

Partial Replication versus Database Partitioning

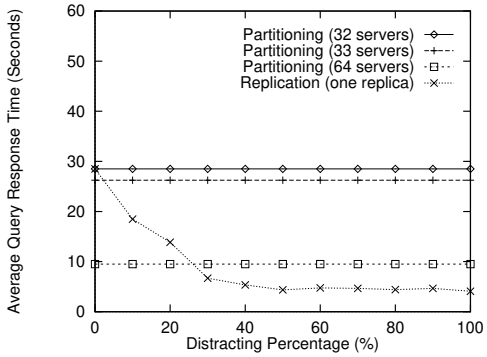
In this section, we compare the performance of the following configurations with the baseline (partitioning over 32 servers):

- 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

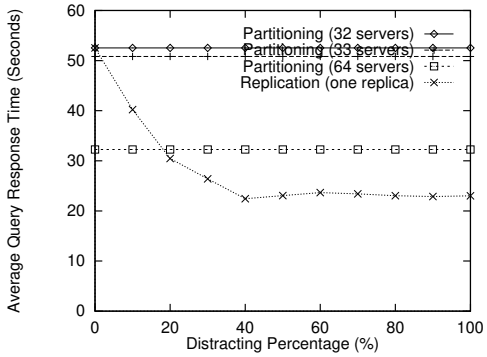
Figure 3 illustrates the average query response time when we partition 1 TB of text over 32, 33, and 64 servers, and over 32 servers with one additional server that contains a 32 GB replica. We vary the distracting percentage which indicates the percentage of commands satisfied by the replica, i.e., our measure of locality is the percent of queries sent to the replica. Figure 3(a)-(c) illustrate when



(a) $\lambda = 4$

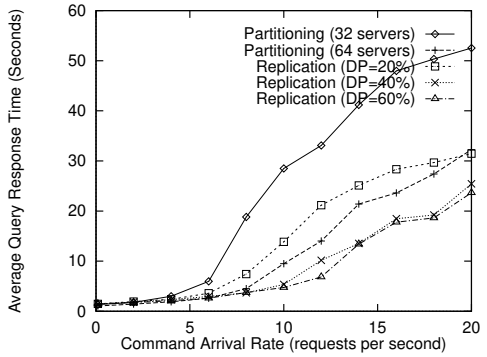


(b) $\lambda = 10$

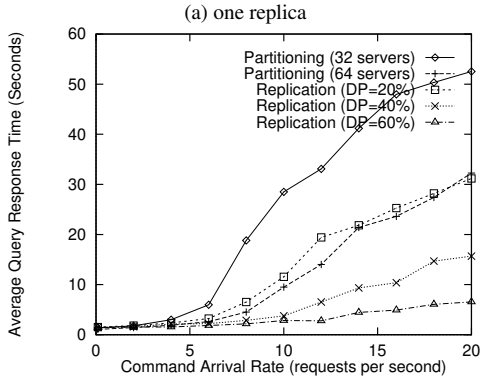


(c) $\lambda = 20$

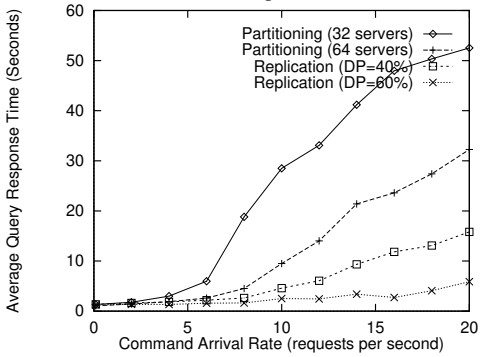
Figure 3: Comparing partial replication with partitioning



(a) one replica



(b) two replicas



(c) four replicas

Figure 4: Performance using a hierarchy of replicas

commands arrive at 4, 10, and 20 commands per second. The graphs plot query response time versus the distracting percentage. 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. 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 distracting percentage becomes high, the replica selector is load balancing between the replica and the partitioned original database which maintains retrieval effectiveness and quick response times.

There are two major reasons that partial replication outperforms partitioning: First, a server takes around 3/4 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; Second, searching a replica results in less network traffic and needs less coordination in the connection broker. For example, 33 versus 64 messages for each query coordinated in the connection broker. The utilizations of the network and the connection broker for partitioning over 64 servers are 28% and 70%, while the corresponding utilizations 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!

4.3 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 $D_1 \supset D_2 \supset D_3 \supset D_4$ and D_i represents the data on the i-th server. The replicas satisfy a total of p% of commands. Of these p% of commands going to replicas, all can go to the largest replica, 10% less, i.e., (p% - 10%) goes

to the second largest replica, another 10% less, i.e., (p% - 20%) goes to the third largest replica, and (p% - 30%) goes to the fourth replica, where the commands going to a i-th largest replica contains the commands going to the $(i + 1)$-th largest replica.

Figure 4 illustrates the average query response time when we build one, two, and four replicas, and the replicas distract 20%, 40%, and 60% of commands. The results show that 2 replicas are sufficient to achieve largest performance improvement when the replicas satisfy 40% and 60% of commands. In our baseline, partitioning over 32 servers achieves 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 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, unfortunately!), 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 probably necessary to maintain fast response times.

5 Access Characteristics in Real Systems

In the previous section, we show partial replication significantly improves performance when there is query locality in the system. In this section, we examine the query locality in two real systems. Since currently there exists no widely available or standard set of queries with locality properties, we obtained our own sets of server logs from THOMAS [21] and Excite [13]. The THOMAS system is a legislative information service of the U.S. Congress through the Library of Congress. THOMAS contains the full text the 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 transcripts of Monica Lewinski's Grand Jury testimony 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

Num. queries	Num. unique queries	Topics			
		total	occurring 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%

(a) Query locality in the THOMAS log

Num. queries	Num. unique queries	Topics			
		total	occurring 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%

(b) Query locality in the Excite log

Table 2: Query locality in the logs

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 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).

Query Locality

Table 2 shows query locality statistics for our THOMAS and Excite logs. We collect the average number of queries, unique (singleton) queries, topics, topics occurring once, and topics that contain more than one unique query. We also present the percentages of queries on the top topics. Table 2(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, 71% of topics occur once, and the remaining 29% of topics account for 63% of queries; Among the topics occurring more than once, 35% (412) contain more than one unique query. The top 2.5% of topics (100 topics) and the top 12% of

	Overlap with					
	7/14			the week on 7/14-7/20		
date	all	top 500	top 1000	all	top 500	top 1000
7/15	43.3%	24.8%	30.1%	n/a	n/a	n/a
7/16	42.6%	24.0%	29.2%	n/a	n/a	n/a
7/23	41.4%	23.4%	28.7%	60.8%	29.0%	35.9%
7/31	38.5%	21.9%	26.4%	58.0%	26.0%	32.3%
8/14	38.1%	21.3%	26.0%	54.9%	25.6%	31.0%
8/28	34.3%	18.3%	23.4%	51.9%	22.8%	28.4%
9/11	44.0%	8.7%	22.2%	58.6%	11.2%	27.0%

Table 3: Query overlap over time in the THOMAS log

topics (500 topics) account for 21.2% and 41.5%.of queries.

The Excite log on September 16, 1997, shown in Table 2(b), demonstrates that the Excite queries also have high query locality: 79% of topics occur once, and the remaining 21% of topics account for 56% of queries; Among the topics occurring more than once, 62% (32750) contain more than one unique query.

Overlap of Queries Over Time

We also examine the THOMAS logs to see how many queries on a given day match a topic that appears on a previous day or week, in order to examine query overlap as a function of time. Table 3 shows for a number of days between 7/15 and 9/11 the average percentage of queries that match all topics, or one of the top 500, or one of the top 1000 topics on July 14, 1998 and in the week from July 14 to July 20, respectively. The statistics show that query overlap tends to decrease as time elapses. On typical subsequent days when query overlap decreases, the decrease is very gradual. For example, 35.9% of queries on 7/23 and 32.3% of queries on 7/31 matched a topic in the replica covering top documents of the week of 7/14-7/20, respectively.

This degree of locality is sufficient to prefer a single replication server to partitioning over an additional 32 servers for a wide range of command arrival rates when searching a terabyte of text. Achieving average response times under 10 seconds for 20 commands per second however requires a hierarchy of replicas. These statistics also suggest that we do not need to update the replica daily, since significant numbers of queries match a top topic that appeared several days ago. Replicas may actually satisfy more queries than the 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

Top topics	% of queries	Replica Size (top 200 documents per query)		
		(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 4: The Replica Size Based on the Excite log

Original		top n	largest replica		Replica Selection Database Size (MB)
Size	unique terms		Size (MB)	unique terms	
20 GB	13,088,064	100	95	162,279	10
		200	191	258,107	15
		500	459	460,044	28

Table 5: Space Overhead of the Replica Selection Database for the 20 GB TREC VLC Text Database

document identifiers and our test database is pretty small, we can not obtain accurate figures about this situation.

Ratio of Query Processing and Document Access

We further examine the THOMAS logs for the ratio of query processing and document access. For each query command, the user views on average 1.9 documents. Since we do not have system measurements of the THOMAS system, we estimate the percentage of time used for query and document processing by simulating the ratio of queries to documents we obtained from the logs, and rerunning the top 1000 unique queries issued on July 14, 1998 against the test database. The system measurements show that the query processing accounts for 44% of total processing time, and document access (retrieving summaries and documents) accounts for 56%. These statistics suggest that we need to use partial replicas to speed up both query processing and document access.

6 Size of Replicas and the Replication Section Database

In this section, we estimate the size of replicas and the replica selection database for searching a terabyte of text as a function of average document size, query locality, and number of top documents per query.

6.1 Replica Size

Table 4 estimates the replica size for a terabyte of text. The average document size varies from source to source. For examples. 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 KB, respectively [15]. The average document size of the 20 GB TREC VLC text database is 2.8 KB [15]. The TREC VLC text

database 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. We overestimate these sizes because we assume there is no overlap among the documents, although documents really do overlap. In Table 4, 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 given an average document size of 2 KB, 3 KB, and 9 KB, respectively. The replica size requirements to achieve a 34% distracting percentage for these traces is thus smaller than we used in Section 4. Larger replicas of course further increase the distracting percentage and the performance benefits of replication.

6.2 Size of the Replica Selection Database

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. Table 5 lists the space overhead of the replica selection database in the 6-layer replication hierarchy that we used to evaluate the effectiveness of our replica selection approach [24]. We list the size and the number of unique terms in the original text database, the size and the number of unique terms in the largest replica, and the size of the replica selection database when we replicate the top 100, 200, and 500 documents, respectively. For the 20 GB VLC text database, the size of the replica selection database is approximately 6 MB for every 100,000 unique terms. We know the 20 GB TREC VLC database 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.

7 Updating

This section reports updating costs for replicas and the replica selection database in our system, suggests updating strategies based on these costs, and relates the frequency of updates to the costs.

Replica Size	Num. of Doc.	Delete-and-Add Updating Percentage				Rebuild
		20%	40%	60%	80%	
500 MB	141,882	0.50	0.86	1.35	1.97	0.92
1 GB	283,764	1.23	2.35	4.05	6.25	1.89
2 GB	567,529	2.96	6.93	15.46	25.59	3.73

Table 6: Updating time for replicas with different sizes (hours)

7.1 Costs for Updating Replicas

We compare two approaches of updating a replica: (a) *delete-and-add* – delete the old documents from the data files and indexes, and add new ones; and (b) *rebuild* – build the replica from the scratch.

Table 6 lists the updating time for replicas with different sizes when we use single CPU for processing on a DEC Alpha Server 2100 5/250 with 3 CPUs (clocked at 250 MHZ) and 1024 MB main memory. The numbers in the table are the average times over three runs. For each data point on the replica size, we collect the times when we update 20%, 40%, 60%, and 80% of a replica. Since *rebuild* builds the replica from the scratch, the time for updating a replica is only related to the size of replica. The results show that when we update 20% of documents in a replica, *delete-and-add* performs significantly better than *rebuild*. The smaller the replica is, the better *delete-and-add* performs. When we update more than 40% of documents in a replica, *rebuild* is our choice, especially for large replicas. For example, for updating 60% of a 1 GB replica, *rebuild* is 2 times faster than *delete-and-add*; for updating 60% of a 2 GB replica, *rebuild* is 4 times faster than *delete-and-add*.

The updating procedure can be parallelized when we have a symmetric multiprocessor with multiple disks. Assume we need to build a 32 GB replica distributed over 8 disks using a symmetric multiprocessor with four 250 MHZ CPUs, as described in Section 4, we need to build eight 4 GB subdatabases. Based on the statistics in Table 6, it takes 7.5 hours to build a 4 GB database. By using 4 CPUs, we can build 4 subdatabases at a time, and finish in 15 hours.

7.2 Costs for Updating the Replica Selection Database

We assume all replicas and the text database export their unique term lists and corresponding collection term frequencies and document frequencies for each term, we merge these term lists and their related information into a replica selection database.

On a DEC Alpha Server 2100 5/250 with 3 CPUs (clocked at 250 MHZ) and 1024 MB main memory, it takes 34 minutes, 39 minutes, and 45 minutes for one CPU to merge two term lists, four term lists, and eight term lists, each of which is from the 20 GB TREC VLC text database, respectively. Based on this observation, it takes approximately one to one and a half hours for one CPU to merge two term lists, four term lists, and eight term lists, each of which comes from a 32 GB database.

This updating procedure can also be parallelized. Assume the largest replica is 32 GB, and we distribute a terabyte text database over 32 servers, and each server exports a list of unique terms that is a partial list of unique terms in the terabyte text database and contains terms occurring on that server. We use a symmetric multiprocessors with four 250 MHZ CPUs. In the first pass, we produce 32 intermediate files by merging the term list of the largest replica with each of 32 partial lists of the original text database in parallel and deleting the terms that do not exist in the largest replica, where each CPU takes care of 8 merges. This pass takes 8 hours. In the second pass, we merge 32 intermediate files into a replica selection database in the following way: We execute four processes in parallel, each of which merges 8 intermediate files and then a final process merges 4 resulting intermediate files. This pass takes approximately 3 hours to finish. It takes around 11 hours to finish updating a replica selection database.

7.3 Updating Strategies

Although updating a large replica is time-consuming, fortunately, the statistics in Section 5 shows that a replica built using topics from a day ago, or even a month ago may satisfy significant number of queries. We do not need to update replicas hourly, or even daily, which amortizes the high cost of updating replicas. One strategy for updating replicas is to update replicas at regular intervals, such as every week since query locality is typically still high. The disadvantages of this strategy are that it may react too slowly to a bursty event such as the Starr report, and it could waste time when the topics the users are interested in change very slowly. In order to overcome these problems, we propose additional strategies:

- 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, an easy way to use it is through human intervention. When the system manager watches a special event, and increasingly many users issue queries to search it, the manager could start the updating procedure before the scheduled time. The automatic event detection is an on-going research topic. When it becomes effective, we suggest using it to trigger the updating procedure automatically.

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.

For bursty events, we could instead just add documents into replicas without deleting others for even quicker updates, which means we need to save some extra space for bursty events.

8 Conclusions

In this paper, we investigate how to search a terabyte of text using partial replication. We build a hierarchy of replicas based on query frequency and available resources, and use the InQuery retrieval system for the replicas and the original text database. We demonstrate the performance of 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 time 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. We believe these trends will also hold for fewer resources and faster query processing due to our previous experiences porting Inquery and validating our simulator. Higher query locality further widens the performance differences. We examine queries from THOMAS [21] and Excite [13] and 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. In our traces, caching will miss many overlaps between queries with unique terms that in fact return the same top documents whereas partial replication will find the similarities. We also investigate overheads and updating costs for the replica selection database and replicas, and

demonstrate that partial replication for information retrieval can be implemented at a reasonable cost. For THOMAS, updating replicas hourly or even daily is unnecessary. We propose two simple strategies to update replicas to insure their effectiveness. These mechanisms combine to provide a practical IR system architecture using replication that searches a terabyte of text while maintaining effectiveness and quick response times.

Acknowledgments

This material is based on work supported in part by the National Science Foundation, Library of Congress and Department of Commerce under cooperative agreement number EEC-9209623,and also 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. Kathryn S. McKinley is supported by an NSF CAREER award CCR-9624209. We thank Ben Mealey and Library of Congress for providing the THOMAS log. We thank Doug Cutting and Excite for providing the Excite log. Any opinions, findings and conclusions or recommendations expressed in this material are the authors and do not necessarily reflect those of the sponsors.

References

[1] S. Acharya and S.B. Zdonik. An efficient scheme for dynamic data replication. Technical Report CS-93-43, Department of Computer Science, Brown University, September 1993.

[2] M. Ahamad and M.H. Ammar. Performance characterization of quorum-consensus algorithms for replicated data. *IEEE Transaction of Software Engineering*, 15(4), April 1989.

[3] M. Baentsch, G. Molter, and P. Sturm. Introducing application-level replication and naming into today's Web. In *Proceedings of Fifth International World Wide Web Conference*, Paris, France, May 1996.

[4] A. Bestavros. 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*, San Anotonio, Texas, October 1995.

[5] S. Bhattacharjee, M.H. Ammar, E.W. Zegura, V. Shah, and Z. Fei. Application-layer anycasting. In *Proceedings of INFOCOM 97*, 1997.

[6] F. J. Burkowski. Retrieval performance of a distributed text database utilizing a parallel process document server. In *1990 International Symposium On Databases in Parallel and Distributed Systems*, pages 71–79, Trinity College, Dublin, Ireland, July 1990.

[7] B. Cahoon and K. S. McKinley. 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*, pages 110–118, Zurich, Switzerland, August 1996.

[8] B. Cahoon, K. S. McKinley, and Zhihong Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Transaction on Information Syetems (submitted)*, 1998.

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

[10] R.L. Carter and M.E. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical Report BU-CS-96-007, Boston University, March 1996.

[11] T. R. Couvreur, R. N. Benzel, S. F. Miller, D. N. Zeitler, D. L. Lee, M. Singhai, N. Shivaratri, and W. Y. P. Wong. 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, 1994.

[12] W. B. Croft, R. Cook, and D. Wilder. Providing government information on the Internet: Experiences with THOMAS. In *The Second International Conference on the Theory and Practice of Digital Libraries*, Austin, TX, June 1995.

[13] Excite. *http://www.excite.com*.

[14] J. Guyton and M. Schwarz. Locating nearby copies of replicated internet servers. Technical Report CU-CS-762-95, University of Colorado at Boulder, February 1995.

[15] D. Harman, editor. *The Sixth Text REtrieval Conference (TREC-6)*. National Institute of Standards and Technology Special Publication, Gaithersburg, MD, 1997.

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

[17] David Hawking, Nick Craswell, and Paul Thistlewaite. Overview of trec-7 very large collection track. In *Proceedings of the 7th Text Retrieval Conference*, 1998.

[18] Vegard Holmedahl, Ben Smaith, and Tao Yu. Aperative caching of dynamic content on a distributed web server. In *IEEE Proceedings of 7th International Symposium on High Performance istributed Computing (HPDC-7)*, pages 235–242, Chicago, IL, 1998.

[19] Y. Huang and O. Wolfson. A competitive dynamic data replication algorithm. In *IEEE Proceedings of 9th International Conference on Data Engineering*, pages 310–337, Vienna, Austria, 1993.

[20] E.D Katz, M. Butler, and R. McGrath. A scalable HTTP server: the NCSA prototype. In *Proceedings of First International World Wide Web Conference*, Geneva, Switzerland, May 1994.

[21] THOMAS legislative Information on the Internet. *http://thomas.loc.gov*.

[22] Z. Lin and S. Zhou. Parallelizing I/O intensive applications for a workstation cluster: a case study. *Computer Architecture News*, 21(5):15–22, December 1993.

[23] Zhihong Lu. *Searching a Terabyte of Text*. PhD thesis, University of Massachusetts at Amherst, February 1999.

[24] Zhihong Lu and Kathryn S. McKinley. Partial replica selection based on relevance for information retrieval. In *summited to SIGIR99*.

[25] I. A. Macleod, T. P. Martin, B. Nordin, and J. R. Phillips. Strategies for building distributed information retrieval systems. *Information Processing & Management*, 23(6):511–528, 1987.

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

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

[28] Oracle Company. Strategies and techniques for using Oracle7 replication. *http://www.oracle.com/products/servers/replication/html/collateral.html*, May 1995.

[29] A. Tomasic. *Distributed Queries and Incremental Updates In Information Retrieval Systems*. PhD thesis, Princeton University, June 1994.

[30] O. Wolfson and S. Jajodia. An algorithm for dynamic replication of data. In *Proceedings of 11th ACM Symposium on the Principles of Database Systems*, San Diego, California, June 1992.

[31] Philip S. Yu and Edward A. MacNair. Performance study of a collabortive method for hiearchical caching in proxy servers. In *Proceedings of 7th International World Wide Web Conference*, Brisbane, Australia, April 1998.