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. We first investigate query locality with respect to time and replica size using real logs from THOMAS and Excite. Our evidence indicates that there is sufficient query locality to justify partial replication for information retrieval and partial replication can achieve better performance than caching queries, because the replica selection algorithm finds similarity between non-identical queries, and thus increases observed locality. We then use a validated simulator to compare database partitioning to partial replication with load balancing, and find partial replication is much more effective at decreasing query response time than partitioning, 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.

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. *A partial replica* of a text database, which includes query logic as well as a subset of the documents, serves these two purposes. Replicating 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.

Distributed databases and the Web use *caching and replication* to distribute workloads and improve system performance [1, 2, 3, 4, 5, 10, 14, 20, 29, 31, 32]. Both caching and replication, of course, require query and document *locality*, repeated access to the same part of the database. Our investigation of partial replication is unique from previous work on caching and replication for several reasons. First, structured databases and web caching use a simple, exact test of set membership to determine if the requested record or document matches the query. We could use the same approach; but because users do not all think the same, many distinct queries return the same documents and thus exact match unnecessarily degrades locality and thus performance. We are the first to demonstrate this degradation: up to 15% on our traces, even with a very restrictive definition of query similarity.

We use server logs from THOMAS [21] and Excite [13] for our experiments. Although others report on query locality [12, 18], there exits no widely available, shared, or standard query sets with locality properties. We report the locality properties of our traces, and find locality that remains high (above 20%) over time (weeks). These results suggest we may update the replicas infrequently and/or use events to trigger updates, and they will continue to satisfy many queries and improve performance.

To use replicas rather than caches and maintain retrieval effectiveness, a selection function must determine whether replicas contain all, some, or none of the relevant documents for a query. We describe such a function elsewhere [24]. In this paper using a hierarchy of replicas, we report on performance as a function of locality using a validated simulator [23]. The simulator closely matches our prototype system that uses InQuery for the basic IR functionality on all text databases: original and replicated [9]. 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 are the first to report performance for a terabyte of text; whereas previous work is on the scale of gigabytes.

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 reports on the performance of searching a terabyte of text using partial replication and compares performance with collection partitioning. Section 6 summarizes our results and concludes.

2 Related Work

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, 29, 31]. They use structured data, such as objects, and present algorithms for updating read-write data to ensure consistency of different copies of data. In a structured database, query logic 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, 32]. 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. In addition, our replica selection function finds locality among non-identical queries, which increases observed locality. Since current Web caches and database replicas use exact match, they cannot find this locality.

A number of studies have investigated the performance of distributed IR systems [6, 7, 11, 16, 26, 27, 30]. 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, 26, 27, 28]. Only Couvreur *et al.* [11], and Cahoon and McKinley [7] use simulation to experiment with more than 100 GB of 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.

InQuery, our base system, is not the fastest text retrieval system available today [17]. For the experiments in this paper, we model 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 have simulated such response times, and found the same trends we report here. We thus believe our results on replication versus partitioning are directly applicable to these systems, 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 the general performance trends we report here.

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 [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 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.

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%

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 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

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. This definition of query similarity is arbitrary and restrictive; a looser definition would further improve the locality we observe. 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 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 7/15 and 9/11 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 9/11, we saw many distinct queries such as "Starr," "Starr Report," "Bill Clinton," and "Monica Lewinsky," all trying to access the Starr Report.

For topic match, we build a replica with the top documents for the top topics. For exact query

	Overlap with								
	the previous day			7/14			the week on 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

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 7/14-7/20, topic match on 7/23 increases the overlap from 28.6% (query exact match) to 35.9%, which means a replica can distract 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 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 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 7/23 and 32.3% of queries on 7/31 matched a topic in the replica covering top documents for top 1000 topics of the week of 7/14-7/20, 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 9/11. 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.

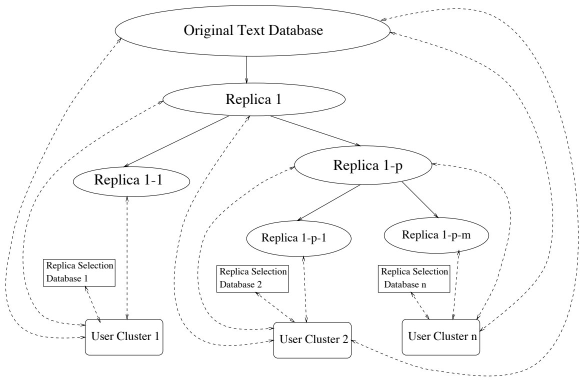


Figure 1: The replication hierarchy

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.

4 Replication Architecture

In this section, we describe how to build a hierarchy of replicas in our system. We replicate the top documents and their index in a database 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 text database that could be a single 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.

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 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. 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, where typically only some or none of their top documents are in the replicas, we define *replica precise* queries as those for which searching selected replica causes a precision loss less than 5% of the precision attained by searching the original collection. Our replica selector directs more than 80% of replica precise queries to a replica, and retrieves on average one less relevant document out of the top 30 documents.

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 observation in our system, 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

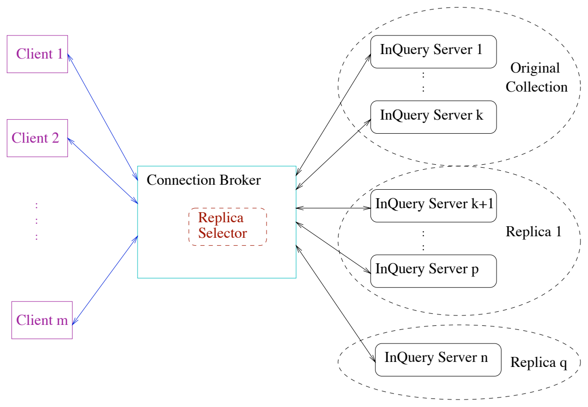


Figure 2: Our Distributed Information Retrieval System

terabyte of text is between 1 and 2 GB.

5 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.

5.1 Our Distributed Information System

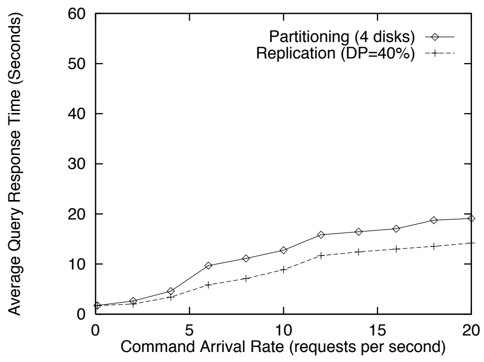
In our distributed IR system illustrated in Figure 2, clients, InQuery servers, and the connection broker reside on different machines. Clients are user interfaces to the retrieval system. InQuery servers store the original text database and partial replicas, and perform IR service such as query evaluation, 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 also performs replica selection based on both relevance and load.

In this paper, we focus on three basic IR commands: *query*, *summary*, and *document* commands. A **query command** consists of a set of words or phrases (terms), such as "information retrieval", or "distributed system." Query responses consist of a list of document identifiers ranked by belief values which estimate the probability that the document satisfies the information need. For each query, a client may obtain one or several summaries on relevant documents by sending **summary commands**. A summary command consists of a set of document identifiers and their database identifiers. The summary information of a document typically consists of the title and the most relevant passages in the document. It may also include information such as source and organization. A client may also retrieve complete documents by sending a **document command**. The command consists of a document identifier and a database identifier. In response, the system returns the complete text of the document from the database.

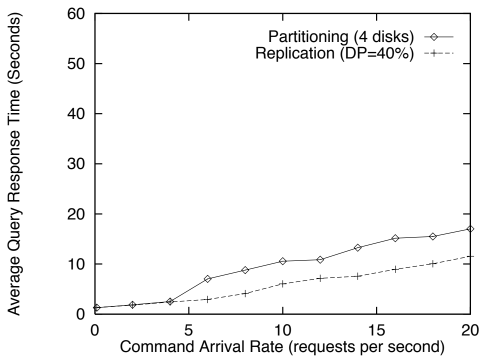
When a client sends a query to the connection broker, the connection broker first uses a replica selector to determine whether there is a partial replica that is not only relevant to the query, but also not overloaded. If there is one, the connection broker sends the query to the InQuery server(s) that maintain the relevant replica, otherwise it sends the query to the InQuery servers that maintain the original database. After each involved InQuery servers returns the results, the connection broker merges results and returns them to the client. For a summary command, the connection broker sends the command to the InQuery servers whose identifiers are described in the command. 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 our 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 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 neither the original database or the replica have responded. We send the command 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 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



(a) using the real system



(a) using the simulator

Figure 3: Performance validation of simulator with partial replication.

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 report on the validation of 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. Of course, simulation enables us to explore in a controlled environment high loads and very large configurations.

5.2 Validation of Partial Replication Performance

This section compares the simulator and the implementation on partial replication for searching a 16 GB text database 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, InQuery which uses too many global variables. Our earlier work showed that the multitasking server performs similarly to the multithreaded server, although the multithreaded server is always slightly faster (90% of measured response times fall within 10% of each other) [25].

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 3 compares the performance of using the real system and the simulator when the replica distracts 40% of commands, and shows that two systems present the same trends and expected

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 Text Database	C_{size}	1 TB
Replication Percentage	P_{repl}	3% (32 GB)
Distracting Percentage	P_{disp}	10% - 100% by 10

Table 4: Configuration Parameters for Terabyte Experiments

improvements from partial replication.

5.3 Searching a Terabyte of Text

In this section, we compare the simulated performance of partial replication with database partitioning using one and 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. We assume each server stores a 32 GB database. 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 **distracting percentage** which represents the percentage of queries that the replica selector sends to a partial replica. Table 4 presents the experimental parameters, their abbreviations, and values.

Partial Replication versus Database Partitioning

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

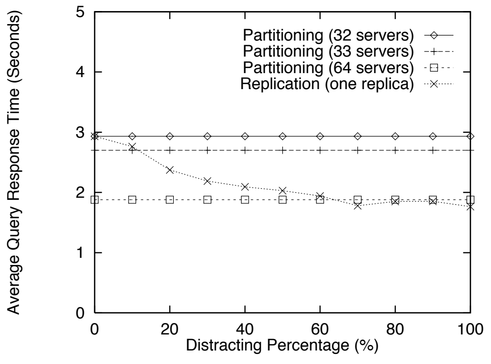
- Partitioning over additional servers: partitioning 1 TB of text over 33, and 64 servers, each of which stores 31 GB and 16 GB of data.
- Partial Replication: building a 32 GB replica on one additional server

Figure 4 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 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 4(a)-(c) illustrate when 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, as shown in Figure 4(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 distracting percentage becomes high, the replica selector load balances between the replica and the partitioned original database which maintains retrieval effectiveness and quick response times.

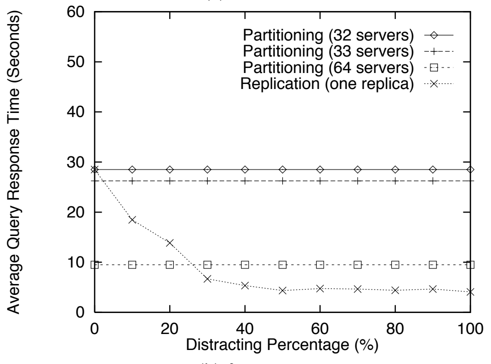
There are two major reasons that partial replication outperforms partitioning. (1) A server takes around 3/5 the time to search half the data according to our measurements, and thus when we partition a terabyte of text over 64 servers instead of 32 servers, each server can not process twice as many commands 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, 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!

5.4 Partial Replication as a Hierarchy

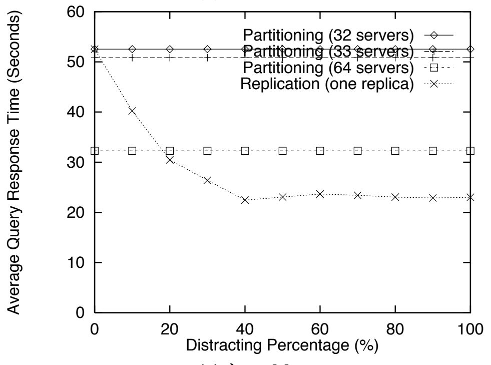
In this section, we assume 1, 2, and 4 additional servers and organize them as a hierarchy of replicas. We examine how much improvement a hierarchy of replicas will produce. We assume the first, second, third, and fourth additional server stores 32 GB, 16 GB, 8 GB and 4 GB of data. where $D_1 \supset D_2 \supset D_3 \supset D_4$



(a) $\lambda = 4$

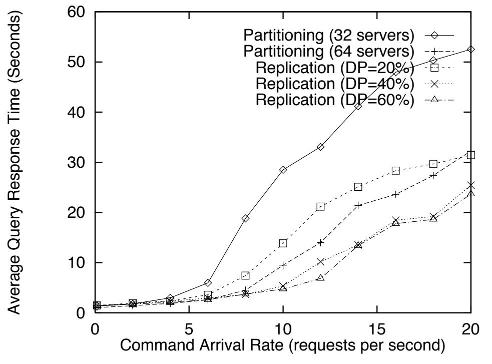


(b) $\lambda = 10$

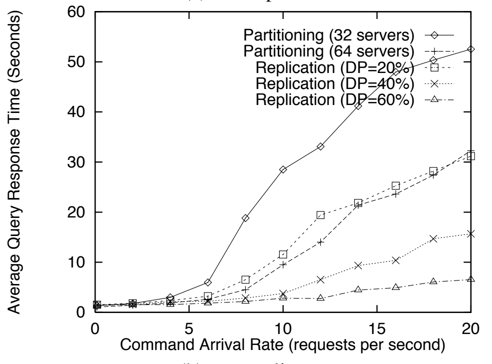


(c) $\lambda = 20$

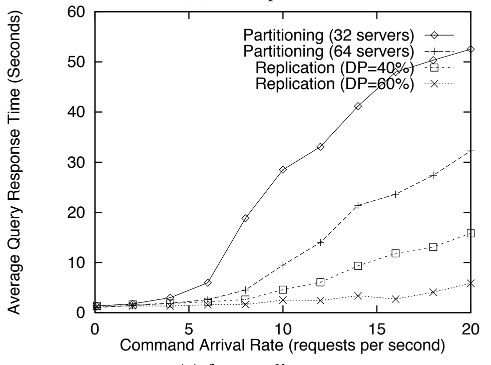
Figure 4: Comparing partial replication with partitioning



(a) one replica



(b) two replicas



(c) four replicas

Figure 5: Performance using a hierarchy of replicas

and D_i represents the data on the i-th server. The replicas satisfy accurately a total of p% of commands. 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 the commands sent to a i-th largest replica may also be satisfied at the $(i + 1)$-th largest replica.

Figure 5 illustrates the average query response time when we build one and two replicas, where the replicas distract 20%, 40%, and 60% of commands, as well as four replicas, where the replicas distract 40% and 60% of commands. 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 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.

6 Conclusions

In this paper, we investigated 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 examine queries from THOMAS [21] and Excite [13] 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. 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, query exact match misses many overlaps between queries with different terms that in fact return the same top documents, whereas partial replication with an effective replica selection function [24] will find the similarities. We believe this trend will hold for other query sets against text collections and for web queries.

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. Our work porting and validating InQuery and the simulator from a slower processor to the Alpha, as well as experiments with faster querying times which are reported elsewhere [23], lead us to believe the performance trends will hold for faster systems using fewer resources.

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, 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 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 Transactions on Information Systems (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 Distributed 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. *Scalable Distributed Architectures For Information Retrieval*. PhD thesis, University of Mas-

sachusetts at Amherst, May 1999.

[24] Zhihong Lu and Kathryn S. McKinley. 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*, Berkeley, CA, August 1999.

[25] Zhihong Lu, Kathryn S. McKinley, and Brendon Cahoon. The hardware/software balancing act for information retrieval on symmetric multiprocessors. In *Proceedings of Europar'98*, Southhampton, U.K., September 1998.

[26] 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.

[27] 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.

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

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

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

[31] 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.

[32] 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.