Partial Collection Replication versus Caching for Information Retrieval Systems

Zhihong Lu
Village Networks, Inc.
Hazlet, NJ 07730
zlu@vill.com

Kathryn S. McKinley
Department of Computer Science
University of Massachusetts, Amhest MA 01003
mckinley@cs.umass.edu

Abstract The explosion of content in distributed information retrieval (IR) systems requires new mechanisms to attain timely and accurate retrieval of unstructured text. In this paper, we compare two mechanisms to improve IR system performance: partial collection replication and caching. When queries have locality, both mechanisms return results more quickly than sending queries to the original collection(s). Caches return results when queries exactly match a previous one. Partial replicas are a form of caching that return results when the IR technology determines the query is a good match. Caches are simpler and faster, but replicas can increase locality by detecting *similarity* between queries that are not exactly the same. We use real traces from THOMAS and Excite to measure query locality and similarity. With a very restrictive definition of query similarity, similarity improves query locality up to 15% over exact match. We use a validated simulator to compare their performance, and find that even if the partial replica hit rate increases only 3 to 6%, it will outperform simple caching under a variety of configurations. A combined approach will probably yield the best performance.

Keywords: Distributed IR architectures, partial replica, cache

1 Introduction

The rapidly increasing content in distributed information retrieval (IR) systems for unstructured text has motivated performance improving techniques such as *partial replication with replica selection* [19], and confirmed the importance of techniques, such as *caching* [23, 21]. Both techniques seek to decrease query response time by searching less text, while maintaining the same *effectiveness* (i.e., returning the same number of relevant documents to each query). When queries repeat or relate to the same set of documents, they have *locality*. By separately storing the queries with the most locality, caches and partial replicas seek to improve performance by limiting the search to the cache or partial replica, rather than searching the entire collection. We can place either technique at a variety of locations to improve availability, and to reduce network traffic and latency.

Caches have a simple organization and membership test. They may store a set of queries, their responses, and the corresponding documents. The membership test is simply: is this query in the cache? If an IR system sees the same exact query or document request repeatedly, it responds from the cache rather than repeating query processing or fetching from the original source.

Partial replicas are a form of caching, but are searchable subcollections consisting of documents returned from the most frequent queries [19]. Their membership test is: is this query relevant to the partial replica? The replica selection function chooses between replicas and the original collection based on content and load. Our previous work developed a replica selection function that maintains accuracy [19]. A replica selector can increase observed query locality over simple caching, because it does not depend on exact match. For example, given distinct queries such as these from our logs, "Starr," "Starr Report," "Bill Clinton," and "Monica Lewinsky," trying to access the Starr Report, the replica selector will direct them to the same replica based on content, while caches must see exactly the same query.

Although others report on query locality [10, 14], there exits no widely available or standard query sets with locality properties. We use real server traces from THOMAS [16] for 40 days and Excite [11] for one day to analyze the locality properties, and find that locality remains high (above 20%) over time (weeks) which suggests infrequent or event triggered updates will yield caches and replicas with sufficient locality to satisfy many queries.

Most importantly, we are the first to show that exact query match misses significant amounts of query locality as compared with query similarity; between 3 and 15%, depending on the replica size. We use a conservative definition of similarity: queries form a *topic* if their top 20 documents completely overlap. Our previous results [19] presented a replica selector that achieves accurate results when it sends to the replica the queries that were used to build it, and additional queries that are a good match. The combination of these and the new trace results presented here demonstrates that replicas can satisfy more queries than caches.

We demonstrate that partial replicas can significantly outperform caches using a validated simulator [7, 18] which closely matches our working prototype system with replica selection. The prototype uses InQuery for the basic IR functionality [8]. We compare performance for searching a terabyte of data and find that partial replication begins to outperform caching when its hit rate increases by 3 to 6%. In a heavily loaded system when the replica hit rate increases 15% over the cache, it attains almost a factor of 2 decrease in query response time. A combined approach also achieves good results, and is probably the system configuration of choice. The additional complexity of partial replicas will pay off in higher hit rates and consequent performance, es-

pecially when load is high and performance is most crucial for distributed IR systems.

The remainder of this paper is organized as follows. The next section further compares our work to related work. Section 3 describes our distributed IR system. Section 4 characterizes the locality and access patterns of the THOMAS and Excite traces, our notion of query similarity, and the increased hit rates due to similarity. Section 5 compares the performance of searching a terabyte of text using partial replication and caching. Section 6 summarizes our results and concludes.

2 Related Work

Both research and commercial database systems (e.g., Oracle, Informix, and Sybase) have used replication and caching for a long time to improve performance and availability [1, 2, 15, 28]. 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, 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 since no single response is correct.

2.1 Caching

Caching in distributed IR systems also has a long research history [24, 22, 23, 26]. The client caches data and performs operations locally. The use of caching is most beneficial for systems that are distributed over networks, that evaluate queries slowly, or in which query locality is high. Clearly, the commercial web search engines have server side caching, but in order to maintain their market advantage, they do not publish their techniques.

Markatos [21] reports on caching search engine results using exact match. He caches for a short period of time to match the dynamic nature of the web. Because search engines update their databases as infrequently as a month, caching for a day or less will not degrade precision much. Markatos analyzes a trace from Excite and uses trace-driven simulations to compare several cache replacement policies. Medium-sized caches (up to 300 MB) can achieve a hit rate of around 20%. Effective cache replacement policies take into account both recency and frequency of access in their replacement decisions. As far as we know, no other papers on caching queries for web search engines exist, although many systems cache documents and respond only to document requests [3, 4, 27]. Generally, they store popular documents in a hierarchy of proxy servers placed between clients and Web servers.

2.2 Partial Collection Replication

Searchable replicas speed up both query processing and document access. We use a replica selector to select a partial replica based on content and load, rather than exact match [19, 18]. In previous work, we showed that the inference network model is very effective at selecting a relevant replica [19]. We implemented the replica selection inference network as a pseudo InQuery database. Each pseudo document corresponds to a replica or text database, and its index stores the document and term frequency for terms in 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

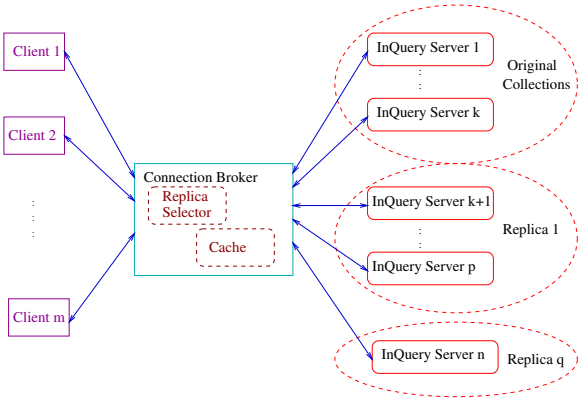


Figure 1: Our Distributed Information Retrieval System

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 the top documents match the replica, 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. Combining these results with the trace results we report here demonstrates that selection based on content increases locality, but that a combined cache and replica approach is probably best. We further show that the increase in locality corresponds to a magnified increase in performance.

2.3 Scalable IR Architectures

A number of studies have investigated the performance of distributed IR systems [5, 6, 9, 12, 20, 25]. 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 [5, 12, 17, 20]. Only Couvreur et al. [9], and Cahoon et al. [6, 7] use simulation to experiment with more than 100 GB of data. None of these previous studies include partial replication or caching.

InQuery, our base system, is not the fastest text retrieval system available today [13]. 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 [13] have recently reported results for a single query (rather than a loaded system) that are less than a second on a 100 GB collection. We simulate such response times, and find similar trends. We report some of these results in Section 5. We thus believe our results on replication, caching, and the combination are applicable to these systems.

3 Our Distributed IR System Organization

In our distributed IR system, illustrated in Figure 1, 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, outstand-

ing client requests, and organizes responses from InQuery servers. For partial replication, the connection broker maintains a replica selection database and performs replica selection based on both relevance and load. For caching, the connection server maintains a cache for the hottest queries, their results, and the documents. We also investigate the case when each InQuery server maintains a cache.

We focus on the *query*, *summary*, and *document* commands. A *query command* consists of a set of stemmed words or phrases (terms), such as "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 by sending *summary commands*. A summary response typically consists of the titles and the most relevant passages of the related documents. It may also include information such as source and organization. A client may also retrieve complete documents by sending a *document command* which consists of a document and database identifier. In response, the system returns the complete text of the document.

When the connection broker maintains a cache, for any command, it checks the cache first. If the cache does not contain the response, the connection broker directs the command to the InQuery servers. When the system has replicas, for a query command, the connection broker first uses a replica selector to determine whether there is a partial replica that is not only relevant to the query, and is not overloaded. If there is one, the connection broker sends the query to the InQuery server(s) that maintain the relevant replica, otherwise it sends the query to the InQuery servers that maintain the original database. After each involved server returns results, the connection broker merges results and returns them to the client. For a summary command, the connection broker sends the command to the specified InQuery servers. The connection broker merges 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 it when the server responds. We also configure the system to check the cache first, and then use the replica selector as described above.

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 larger replicas, 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 the 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 use a very optimistic model of the cache; we assume the match takes a couple cycles, and the summary response or document is always resident in memory. However, for partial replication, we assume the replica selection database resides in the disk space of the connection broker. Searching the replica selection database involves disk access.

We evaluate the performance of our distributed information retrieval system with a simulator that uses a performance model driven by measurements of 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). The servers are connected by a 10 Mbps Ethernet. In previous work, we showed the simulator closely matches a multithreaded implementation of InQuery [18].

4 Access Characteristics in Real Systems

We examine query locality from two real system logs in this section. We measure query similarity versus exact match, how locality changes over time, and its effect on replica size. We also suggest mechanisms for keeping replicas up to date. Since no widely available, shared, or standard set of queries with locality properties exists, we obtained query logs from THOMAS [16] and Excite [11]. 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 40 full day logs, and 22 partial day logs 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 contains queries for one day, 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. Each of our test databases uses a subset of data that THOMAS or Excite searches. 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 intended to represent the web at large (725 MB, 81334 documents). To facilitate discussion, we introduce the following definitions.

distinct queries – queries that are not identical.

exact query match – a query is identical to a previous query.

topic – a set of distinct queries whose resulting top 20 documents completely overlap.

topic match – a query's resulting top 20 documents completely overlaps with the top 20 documents of a topic.

the occurrence of a distinct query – the total times the distinct query occurs in the log.

the occurrence of a topic – the total times that all distinct queries of the topic occur in the log. A "singleton topic" has one distinct query. If "a topic occurs multiple times," it contains one distinct query and the query occurs multiple times, or it contains multiple distinct query and each occurs one or more times.

4.1 Query Locality

Table 1 shows query locality statistics for the THOMAS and Excite logs. We collect the average number of queries, distinct queries, topics, singleton topics, topics occurring multiple times, and topics that contain multiple distinct queries. We present the percent of the top n topics and distinct queries among all observed queries in the log as

Avg. Num. queries	Avg. Num. distinct queries	Avg. Num. Topics			
		total	singleton	distinct query repeats	multiple distinct queries
8143 (7703)	4876 (4651)	4069	2888 (71%)	1181 (29%)	412
percent occurrence of the top n topics					
n	100	200	500	1000	2000
%	21.2%	28.7%	41.5%	54.1%	73.0%
percent occurrence of the top n distinct queries					
n	100	200	500	1000	2000
%	18.1%	24.5%	36.4%	49.4%	64.5%

(a) Average Daily Query Locality in THOMAS log

Num. queries	Num. distinct queries	Num. Topics			
		total	singleton	distinct query repeats	multiple distinct queries
499836 (444899)	365276 (320987)	249405	196672 (79%)	52733 (21%)	32750
percent occurrence of the top n topics					
n	500	1000	5000	10000	20000
%	12.3%	16.0%	27.9%	34.4%	42.0%
percent occurrence of the top n distinct queries					
n	500	1000	5000	10000	20000
%	7.9%	10.4%	18.4%	23.0%	28.2%

(b) Query and Topic Locality in Excite log

Table 1: Query and Topic locality in THOMAS and Excite

	Topic match with the top n topics								
	the previous day			7/14			the week on 7/14-7/20		
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 with the top n distinct queries								
	the previous day			7/14			the week on 7/14-7/20		
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: Query and Topic Locality over time in THOMAS

a function of the number of top topics or distinct queries, respectively. Table 1(a) shows the average for THOMAS over the full 40 day logs. The average number of matching queries from our test database is in parentheses in columns 1 and 2. Some queries do not find any matches, due to misspelling, or because the query terms do not exist in the test database. On the average, 29% of topics occur multiple times, and they account for 63% ((7703-2888)/7703) of queries. Among the topics occurring multiple times, 35% contain multiple distinct queries. The top 500 topics (12%) and 1000 topics (25%) are 41.5% and 54.1% of all queries, while the top 500 and 1000 distinct queries drop to 36.4% and 49.4% of all queries.

The Excite log on September 16, 1997 in Table 1(b) also demonstrates high query locality: 21% of topics occur multiple times, and they account for 56% ((444899-196672)/444899) of queries. Since the Excite log is one day, the numbers are not averages as in the Thomas results. Among the topics occurring multiple times, 62% contain multiple distinct queries. For example, the top 5000 and 10000 topics are 27.9% and 34.4% of all queries. Exact match drops locality between 3 and 14%.

Our definition of *topic match* is arbitrary and restrictive, but it simplifies our analysis and gives a lower bound of performance improvement due to replica selection based on relevance. A looser definition would further improve the topic locality we observe. In our previous work [19], we instead use the InQuery inference network to determine relevance and the top 20 documents need not overlap. This previous work shows that the replica selector gets accurate results when it sends the replica both the queries that were used to build it, and additional queries that are good matches. That replica selector thus results in further increases to locality over exact match than those we report here.

4.2 Locality as a Function of Time

We examine locality as a function of time (days and weeks) in the THOMAS logs. Table 2 shows that for days between 7/15 and 9/11, the percent of queries that match a top topic or distinct query on a previous day or week. Columns 2 through 4, columns 5 through 7, and columns 8 through 10 list the percentage of queries that match one of the top n topics or top n distinct queries on the previous day with daily updates; on July 14, 1998, without an update; and from July

Top topics	% of queries	Replica Size (data file plus index) 2 KB per doc	3 KB per doc	9 KB per doc	Top distinct queries	% of queries	Cache size (all commands) 2 KB per doc	3 KB per doc	9 KB per doc	Cache Size (only query command)
1000	16.0%	520 MB	780 MB	2.3 GB	1000	10.4%	420 MB	620 MB	1.8 GB	20 MB
5000	27.9%	2.6 GB	3.9 GB	11.7 GB	5000	18.4%	2.1 GB	3.1 GB	9.1 GB	120 MB
10000	34.4%	5.2 GB	7.8 GB	23.4 GB	10000	23.0%	4.2 GB	6.2 GB	18.2 GB	240 MB
20000	42.0%	10.4 GB	15.6 GB	46.8 GB	20000	28.2%	8.5 GB	12.5 GB	36.5 GB	480 MB

Table 3: The Replica/Cache Size Based on the Excite log (top 200 documents per query)

14 to July 20, 1998, without an update. Topic match increases the hit rate up to 15% over exact match. For example, for the top 1000 topics or distinct queries between 7/14 and 7/20, topic match on 7/23 increases the hit rate to 35.9% from 28.6% for exact match. Replicating more topics further widens this difference.

4.3 Estimating the Replica and Cache Size

Since our replicas serve both query processing and document access, they include the documents and their index. The replica size is a function of average document size, query locality, number of top documents per query we chose to return, and the corresponding index. 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, 3, and 9 KB, respectively [13]. The average document size of the 20 GB TREC VLC text database is 2.8 KB [13]. We assume an average document size of 2, 3, and 9 KB. For query locality, we use the statistics obtained from the Excite log, since its workloads are at the level of the systems we investigate. We obtain the top 200 documents for each query. We overestimate because we assume there is no overlap among the documents, although as we showed above, the results of distinct queries often overlap. We assume the index size adds 30% to the replica size based on our experience with InQuery index files. In our system, the average size of a summary is 120 bytes.

In Table 3, columns 1 and 2, and columns 6 and 7 show the query locality from the Excite log for topic and distinct queries, respectively; columns 3 through 5, and columns 8 through 10 show the estimated replica and cache size when we vary the average document size. Column 11 shows the cache size when we only cache queries and summaries. When the average document size is 2, 3, and 9 KB, a replica needs 5.2, 7.8, and 23.4 GB of disk space, respectively, to satisfy 34.4% of Excite queries and document requests; a cache needs less space 4.2, 6.2, and 18.2 GB, respectively, to satisfy 23.0% of Excite queries and document requests. Replicas also need a replica selection database which stores term frequency for the entire collection and each individual collection. Its size is determined by the number of unique terms in the largest replica. Based on our observations, the size of the replica selection database is approximately 6 MB for every 100,000 unique terms. 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. We estimate the replica selection database for 1 terabyte of text is between 1 and 2 GB.

4.4 When to Build or Update a Replica

Query locality decreases very gradually with time in our logs, as illustrated in Table 2. Even in the dynamic environment of the Web, search engines routinely return URLs a few month old [21] which implies daily updates are not necessary. Nor are they practical because rebuilding a replica database can take hours. Some mechanism must react to bursty events like the Starr Report, as shown by the sharp decrease in locality on 9/11. Regular daily updating would catch this event, but may react too slowly, or unnecessarily degrade performance when the system experiences the expected gradual degradation of locality. We propose the following two updating strategies:

Event triggered: trigger updates for bursty events.
Automatic: trigger updates when the cache or replica hit rate falls below a given threshold.

The system manager can anticipate or observe a special event and perform an update. (Automatic event detection would be better, but is an on-going research topic.) Automated performance triggered updates also work for bursty events, if the hit rate falls enough. We could also add documents into replicas without deleting others, and thus respond quickly to bursty events as well as gradual changes in locality.

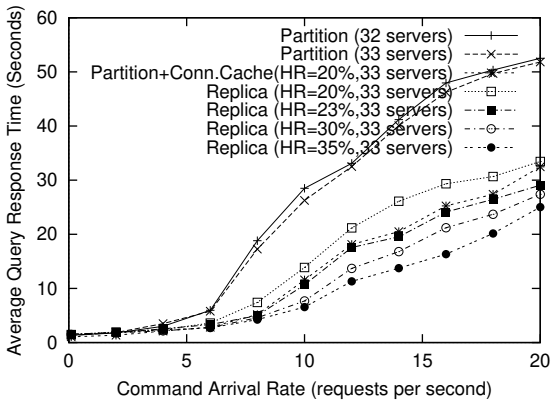
5 Performance of Partial Replication versus Caching

In this section, we compare the performance of partial replication to caching when searching a terabyte of text. We model command arrival as a Poisson process. We use short queries with an average of 2 terms per query, and set the ratio of query commands, summary commands, and document commands to 1:1.5:2, as we found in the THOMAS log. As our baseline, we use 32 servers to store 1 terabyte of text, and each server handles 32 GB of text using 8 disks. This configuration supports an average response time of under 10 seconds when commands arrive at approximately 6 per second on our working prototype. We then add one additional server. We either use it for a cache or replica, which store 32 GB of text and is sufficient to satisfy more than 30% of queries in the Excite log. We build a cache either in the connection broker or in each InQuery server. We vary the **hit rate** (HR) which represents the percent of queries and the corresponding summary and document commands that the replica and cache satisfy. We also compare further partition data over the additional server. We perform two sets of sensitivity experiments: placing 4 times as much data on a single server using bigger fast disks, and doubling the server speed.

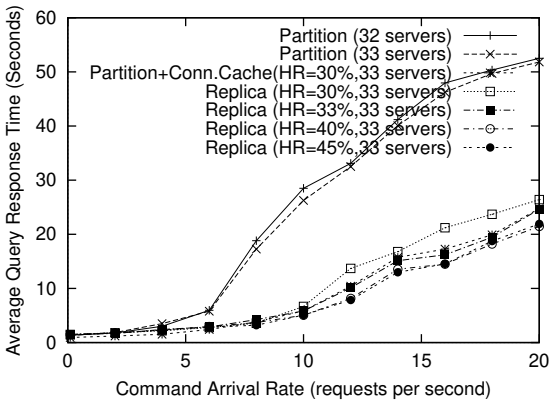
5.1 Partial Replication versus Caching

In this set of experiments, we compare the performance of partial replication to caching in the following configurations with the baseline (partitioning 1 TB of text over 32 servers):

Partitioning: partition 1 TB of text over one additional server (33 servers in total), each of which stores 31 GB of data.
Connection broker caching: partition 1 TB of text over 33 servers and build a cache in the main memory of the connection broker.
Server caching: partition 1 TB of text over 33 servers and build a cache in the main memory of each InQuery server.



(a) a cache hit rate of 20%



(b) a cache hit rate of 30%

Figure 2: Partial replication versus caching as a function of command arrival rate

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

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

For caching, we present an upper bound of its performance; we only count the time for cache lookup and assume cache replacement takes no time. We assume the documents and query summaries are in memory, although not many machines have several to several tens of GB worth of memory. For the partial replica, we assume the summaries and documents must be fetched from disk. We thus give the cache a large advantage.

Figures 2(a) and (b) compare connection broker caching and partial replication. They illustrate the average query response time versus the command arrival rate when the connection broker cache satisfies 20% and 30% of commands, as we found in our logs. The results show that if the partial replica and the connection broker cache satisfy the same amount of commands, partial replication results in slightly worse performance as compared with connection broker caching. But when the replica hit rate increases by just 3%, partial replication performs better than connection broker caching; it performs almost a factor of 2 better when the replica hit rate increases by 15% for high command arrival rates. Searching the replica is so much faster than searching the entire collection, even small amounts of locality have a significant impact on performance.

Figure 3 demonstrates the effect of the hit rate more clearly. It plots the average query response time versus the hit rate when commands arrive at 10 commands per second. Figure 3 shows that the performance of server caching is slightly worse than connection broker caching, since the connection broker cache eliminates the coordination time of multiple servers and reduces network traffic between the connection broker and InQuery servers. Partial replication outperforms connection broker caching when the replica satisfies 3% or more of commands until partial replication

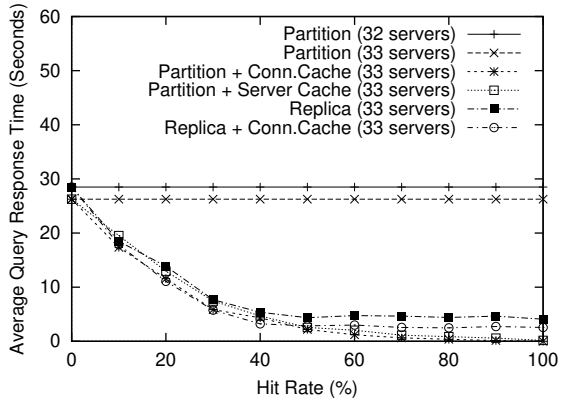


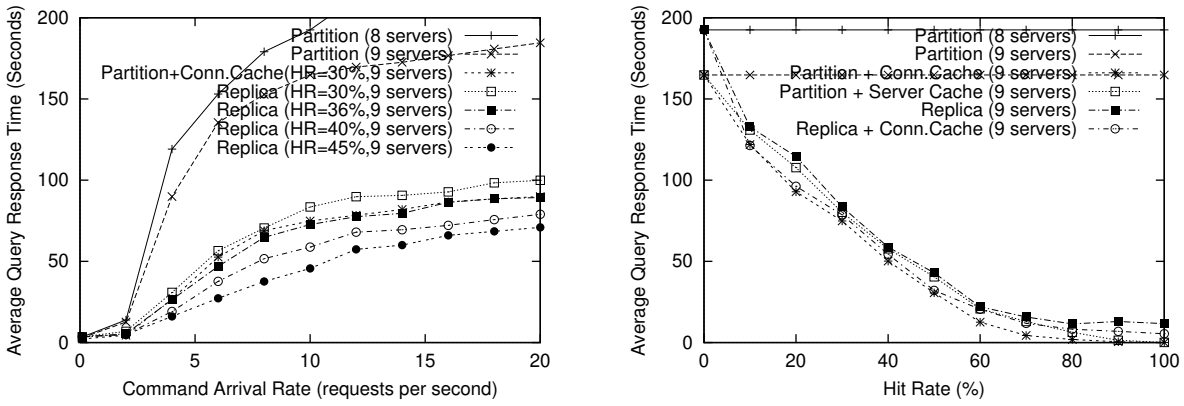
Figure 3: Partial replication versus caching as a function of hit rate at $\lambda = 10$

needs load balancing (which occurs when the hit rate is around 40%). After this point, partial replication performs significantly worse than connection broker caching, since it redirects significant amount of commands to the original servers, while caching does not. The log analysis in Section 4 shows that the cache achieves a hit rate between 20% - 30%, or less in most cases. However, combining the connection broker cache and partial replication further improves performance; an unsurprising result at this point.

In the remaining two sections, we present sensitivity results that indicate the above results will most likely hold on servers with larger, but just as fast disks, and faster CPUs.

5.2 Using larger disks

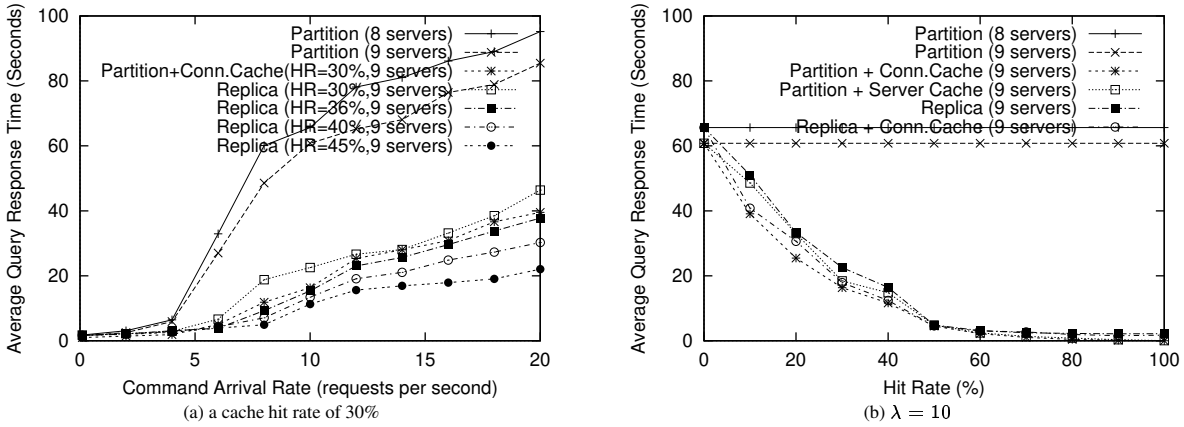
In this set of experiments, we assume disks that are 4 times bigger and just as fast as our measured results. Each server thus handles 128 GB of text on 8 disks. We repeat the comparison experiments in Section 5.1, but with 9 servers, each of which stores up to 128 GB of text. For caching, we still present the upper bound of its performance, where we only count the time for cache lookup. For partial replication, since a server can hold 128 GB in total and we build a replica that stores 32 GB of text, we put four copies of the replica on this server. Figure 4 compares the average query response time. Figure 4(a) illustrates average query response time versus the command arrival rate when the



(a) a cache hit rate of 30%

(b) $\lambda = 10$

Figure 4: Partial replication versus server caching with larger disks (each server holds up to 128 GB of text)



(a) a cache hit rate of 30%

(b) $\lambda = 10$

Figure 5: Partial replication versus caching with faster servers and 128 GB collections per server

server cache satisfies 30% of commands. Figure 4(b) illustrates average query response time versus the hit rate when commands arrive at 10 commands per second.

Compared with results in Section 5.1, these results follow the same trend except that the load balancing point for partial replication shifts to the hit rate of 70%, and partial replication outperforms caching when the replica hit rate increases by 6% or more. The load balancing point shifts because we have four copies, and thus the load balancer sends the replicas more commands than it does when a server has only one copy. The effect of a connection broker cache is larger for fewer servers because each server is busier now. Thus, the partial replicas need to satisfy more commands to match the performance of caching.

5.3 Using faster servers

In this set of experiments, we assume servers are two times faster than the configurations in Section 5.2. Figure 5(a) illustrates average query response time versus the hit rate when commands arrive at 10 commands per second. Figure 5(b) illustrates average query response time versus the command arrival rate when the server cache satisfies 30% of commands.

Compared with results in Section 5.2, these results follow the same trend except that faster servers obviously result in quicker response time. The gap between caching and partial replication is actually reduced. The performance advantage of the faster server has a larger effect on partial replication because the performance of the cache is already very optimistic due to the assumption that all its documents and summaries are found in memory.

6 Conclusions

This paper investigated how to improve IR system performance using partial replication and caching. We examine queries from THOMAS [16] and Excite [11] to find locality patterns in real systems. These traces have sufficient query locality that enables partial replication and caching to maintain effectiveness and significantly improve performance. Query exact match misses significant amounts of query locality in these traces, whereas partial replication with an effective replica selection function [19] can increase hit rates by up to 15%. We believe these trends will hold for other query sets, including web queries, but this point should be investigated further. We demonstrate the performance of our system searching a terabyte of text using a validated simulator. The performance of partial replication with a connection broker exceeds that of client side caching as well as server caching under a variety of configurations when the partial replica increases the hit rate by at least 3 to 6%. Although the simplicity of caching is appealing, a combined approach that incorporates partial replication will

yield both an effective and better performing system.

Acknowledgments

We owe a special thank you to Bruce Croft; without his support and wisdom, this work would not have been possible. We also thank Brendon Cahoon for his contributions. 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; Available at http://www5conf.inria.fr/fich_html/papers/P3/Overview.html, 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*, pages 338–345, San Anotonio, Texas, October 1995.

[5] 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.

[6] 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.

[7] B. Cahoon, K. S. McKinley, and Z. Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Transaction on Information Systems (accepted)*, 1999.

[8] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings of the 3rd International Conference on Database and Expert System Applications*, pages 78–93, Valencia, Spain, September 1992.

[9] 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.

[10] 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.

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

[12] 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.

[13] D. Hawking, N. Craswell, and P. Thistlewaite. Overview of TREC-7 very large collection track. In *Proceedings of the Seventh Text REtrieval Conference (TREC-7)*, pages 91–104, Gaithersburg, MD, 1998.

[14] Vegard Holmedahl, Ben Smaith, and Tao Yu. Cooperative caching of dynamic content on a distributed web server. In *Proceedings of HPDC-7*, pages 243–250, Chicago, IL, 1998.

[15] 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.

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

[17] 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.

[18] Z. Lu. *Scalable Distributed Architectures For Information Retrieval*. PhD thesis, University of Massachusetts at Amherst, May 1999.

[19] Z. Lu and K. 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*, pages 97–104, Berkeley, CA, August 1999.

[20] 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.

[21] E. P. Markatos. On caching search engine results. Technical Report 241, Institute of Computer Science (ICS) Foundation for Research & Technology - Hellas (FORTH), Greece, January 1999.

[22] 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.

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

[24] P. Simpson and R. Alonso. Data caching in information retrieval systems. In *Proceedings of the Tenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 296–305, New Orleans, LA, June 1987.

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

[26] A. Tomasic and H. Garcia-Molina. Caching and database scaling in distributed shared-nothing information retrieval systems. Technical Report STAN-CS-92-1456, Stanford University, December 1992.

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

[28] 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.