Partial Replica Selection Based on Relevance for Information Retrieval

Zhihong Lu Kathryn S. McKinley
Department of Computer Science, University of Massachusetts, Amherst, MA 01003
{zlu, mckinley}@cs.umass.edu

Abstract Partial collection replication improves performance and scalability of a large-scale distributed information retrieval system by distributing excessive workloads, reducing network latency, and restricting some searches to a small percentage of data. In this paper, we first examine queries from real system logs and show that there is sufficient query locality in real systems to justify partial collection replication. We then present a method for constructing a hierarchy of partial replicas from a collection where each replica is a subset of all larger replicas, and extend the inference network model to rank and select partial replicas. We compare our new selection algorithm to previous work on collection selection over a range of tuning parameters. For a given query, our replica selection algorithm correctly determines the most relevant of the replicas or original collection, and thus maintains the highest retrieval effectiveness while searching the least data as compared with the other ranking functions. Simulation results show that with load balancing, partial replication consistently improves performance over collection partitioning on multiple disks of a shared-memory multiprocessor and it requires only modest query locality.

1 Introduction

As information and users proliferate through the Internet and intranets, distributed information retrieval systems must cope with the challenge of scale. Distributing excessive workloads and searching as little data as possible while maintaining acceptable retrieval accuracy are two ways to improve performance and scalability of a distributed IR system. Partial collection replication serves these two purposes. Replicating collections on multiple servers can distribute excessive workloads; replicating collections on servers that are closer to their users can improve performance by reducing network traffic and minimizing network latency. When we replicate a small percentage of a collection and direct queries to a relevant partial replica, we can further improve performance by searching less data. However, since a partial replica in a distributed IR system may contain all, some, or none of the relevant documents for a given query, selecting a partial replica just based on load will not maintain acceptable retrieval accuracy.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGIR '99 8/99 Berkley, CA USA
Copyright 1999 ACM 1-58113-096-1/99/0007 . . . $5.00

In this paper, we use the inference network model to rank partial replicas and the original collection, and then select a replica or the original collection to process the query based on both relevance and load. 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 collection. We examine queries from THOMAS [18] and Excite [12] and find query locality sufficient to justify partial replication. The evidence also indicates that partial replication will achieve better performance than caching queries, because the replica selection algorithm finds similarity between non-identical queries, and thus increases observed locality. We modify the collection selection mechanism developed by Callan et al. [6] to select relevant partial replicas for a given query. We evaluate this new strategy using 300 TREC queries on the 2 GB TREC volumes 2+3 and 20 GB VLC collections. We compare our algorithm to Callan et al.'s collection ranking function and vary the parameters of each. The experiments demonstrate that our strategy is better at maintaining retrieval effectiveness while searching significantly less data. We also present a new and surprising performance result: partial replication is more effective at reducing execution time than collection partitioning on the same resources, even when only 12% of queries have locality with the replica(s).

The remainder of this paper is organized as follows. The next section describes related work. In Section 3, we characterize the locality and access patterns of the THOMAS and Excite logs. Section 4 and 5 describe the replication architecture and how to use the inference network model to rank replicas. Section 6 presents experiments that show our replica selection algorithm is significantly more effective than previous work on collection selection for replicas. Section 7 shows the performance of partial replication significantly improves performance over partitioning the collection over the available disks with modest query locality. Section 8 summarizes our results and concludes.

2 Related Work

Distributed database systems (e.g. Oracle, Informix, and Sybase) have used replication for a long time to improve system performance and availability [1, 11, 20, 22]. Recently, researchers have also used replication in the Web for document access [2, 3, 4, 7]. Since records and documents have well-defined names in all these systems, it is simply a name membership test to determine if a partial replica can satisfy a user request. Since partial replicas in IR systems may contain all, some, or none of the relevant documents for a given query, how to select a partial replica based on relevance for a *query* where the answer is not well defined is a problem that does not exist in these systems. As far as we know, the work presented in this paper is the first that attacks this problem.

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

(a) Query locality in the THOMAS log.

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

(b) Query locality in the Excite log.

Table 1: Query locality in THOMAS and Excite logs

The closest work to selecting a relevant partial replica which is a subset of the original collection is *collection selection*, i.e., locating the most relevant collections [6, 8, 10, 13, 15, 21], where collections are disjointed. Replica selection differs also because it directs as many queries as possible to relevant replicas in order to obtain performance improvements.

Danzig et al. [10] use a hierarchy of brokers to maintain indices for document abstracts as a representation of the contents of primary collections, and support Boolean keyword matching to locate the primary collections. If users' queries do not use keywords in the brokers, they have difficulty finding the right primary collections.

Voorhees et al. exploit similarity between a new query and relevance judgments for previous queries to compute the number of documents to retrieve from each collection [21]. Netserf extracts structured, disambiguated representations from the queries and matches these query representations to hand-coded representations [8]. Voorhees et al. and Netserf require manual intervention which limits them to relatively static and small collections.

Fuhr proposes a decision-theoretic approach to solve collection selection problem [13]. He makes decisions by using the expected recall-precision curve, expected number of relevant documents, and cost factors for query processing and document delivery. He does not report on effectiveness.

GLOSS uses document frequency information for each collection to estimate whether, and how many, potentially relevant documents are in a collection [14, 15]. The approach is easily applied to large numbers of collections, since it stores only document frequency and total weight information for each term in each collection. However its effectiveness remains unknown due to limited evaluation.

Callan *et al.* adapt the document inference network to ranking collections by replacing the document node with the collection node [6]. Similar to GLOSS, the information stored in the collection ranking inference network is document frequencies and term frequencies for each term in each collection. Experiments using the InQuery retrieval system and the 3 GB TREC 1+2+3 collection show that using this method to select the top 50% of subcollections attains similar effectiveness to searching all subcollections.

None of the above collection selection algorithms consider replicas and partial replicas. Among the approaches for collection selection, the collection inference network model [6] is the most thoroughly tested and effective. In this paper, we modify this technique to rank partial replicas and the original collection, propose a new algorithm for replica selection, and show that it is effective and improves performance.

3 Access Characteristics in Real Systems

The most often used queries for evaluating IR systems are the TREC queries [16]. However, each query in TREC is unique. To evaluate partial collection replication, we instead need typical sets of query locality patterns. Other researchers have examined query locality in the context of the Web [9, 17], but currently there exists no widely available or standard set of queries with locality properties. We therefore obtained our own set of server logs from THOMAS [18], and Excite [12]. The THOMAS system is a legislative information service of the U.S. Congress through the Library of Congress. THOMAS contains the full text congressional records and bills introduced from the 101st Congress to 105th Congress. We analyze the THOMAS logs between July 14 and September 13, 1998. We obtained full day logs for 40 days, and partial logs for remaining 22 days due to insufficient disk space in the mailer at the library of Congress. The Excite system provides online search for more than 50 million Web pages, and we obtained one day of log information for September 16, 1997.

Since the logs do not contain document identifiers returned from query evaluation, we built our own test databases. We define a *topic* as all queries whose top 20 documents completely overlap. For queries from the THOMAS log, we reran all queries against a test database that uses the Congressional Records for the 103rd Congress (235 MB, 27992 documents, a subset of the real database). 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).

Table 1 shows the statistics on query locality in the THOMAS and Excite logs. We collect the average number of queries, unique queries, topics, topics occurring once, topics occurring more than once, and topics that contain more than one unique query. We also present the percentages of queries that correspond to the top topics. 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 documents from our test database are in parentheses in columns 1 and 2. Some queries do not find any documents, due to misspelling, or because query terms do not exist in the test database. The statistics show that on the average, 71% of topics occur once, and the remaining 29% of topics account for 63% of queries. Among the topics that occur more than once, 35% (412) contains more than one

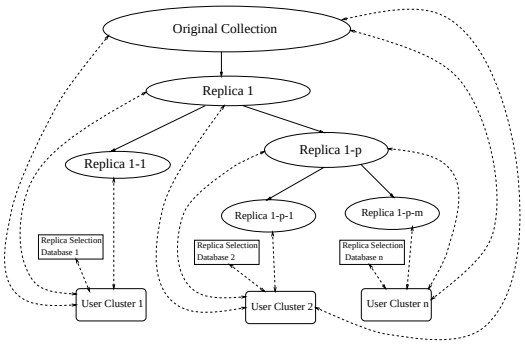


Figure 1: The replication hierarchy

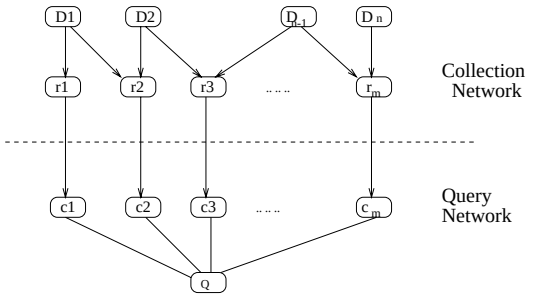


Figure 2: The collection retrieval inference network

unique query, which indicates that exactly matching queries would not find this overlap, and caching queries may miss queries that use different terms but in fact return the same top documents.

The Excite log on September 16, 1997, shown in Table 1(b), demonstrates that the Excite queries also have high query locality: 79% of topics occur once, and the remaining 21% of topics account for 56% of queries; among the topics that occur more than once, 62% (32750) contain more than one unique query, which is the overlap missed by exactly matching queries.

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

These statistics and other studies [9, 17] suggest that there is sufficient locality to justify partial collection replication for information retrieval. Our partial collection replica should be a searchable partial collection with replicated documents and their indices to speed up both query processing and document access.

4 Replication Architecture

We determine which documents to replicate as follows: for a given query, we tag all top n documents returned by query processing as "accessed" and count their access frequencies, regardless of whether the user requests the text of these documents. We keep the access frequency of each document within a period of time, such as a week or a month, and then replicate documents based on their access frequencies.

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

The solid lines illustrate data flows disseminated from the original collection to replicas. Along the arcs from the original collection, the most frequently used documents are replicated many times. The replica selection database directs queries to a relevant partial replica. User requests may go to any replica or the original collection along the arcs from the top node depending on relevance and other criteria, such as server load. The dotted lines illustrate the interaction between users and data. If we do not divide the users into different groups, the hierarchy is simply a linear hierarchy. In this architecture, replica selection is a two-step process: ranking replicas and the original collection based on relevance, and then selecting one of the most relevant replicas or the original collection based on load.

5 Ranking Replicas with the Inference Network Model

We adapt the collection retrieval inference networks that Callan et al. proposed to rank collections [6] to rank partial replicas and the original collection. The collection retrieval inference network model consists of two component networks: a collection and query network, illustrated in Figure 2. The D_i nodes are collections, and the r_j nodes are concepts in the collections. The query node Q consists of c_i, the concept nodes from the query. By using the collection retrieval inference network, collection ranking becomes an estimate of $P(Q|D_i)$ from combining the conditional probabilities through the network. When ranking replicas, we use D_n to represent the original collection, and $D_i, i = 1, 2, ...n-1$ to represent partial replicas. We call this inference network the replica selection inference network. As in the collection retrieval inference network model, $P(c_k|r_j)$ is set to 1.0. The central work of replica selection is to develop an effective replica selection function to estimate $P(r_j|D_i)$.

A document/collection ranking function is better than others if and only if it can produce higher precision at the selected numbers of documents/collections or at all levels of recall. The replica selection function also needs to direct as many queries as possible to relevant replicas in order to obtain the largest performance improvement, but this may degrade precision. We compare precision of functions when they direct at least 80% of replicated queries to the replicas.

We first consider the InQuery collection ranking function, illustrate in Figure 3 [6], but it directs too many replicated queries to the original collection ($> 70\%$), because it uses *df* (the document frequency of each term) as the basic metric, and thus favors the original collection with large *df*. Since a partial replica contains the top documents of the most frequently used queries, by examining the document ranking function, we know that the top documents are ranked as the

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

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

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

where

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

Figure 3: The Collection Ranking Function in InQuery

$$ave_tf = \frac{ctf_{ij}}{df_{ij}}$$

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

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

$$T = \frac{AT}{AT + k \cdot ((1-b) + b \cdot \frac{ave_doclen_i}{ave_ave_doclen}}$$

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

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

where

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

Figure 4: The replica selection function

top, just because they contain these query terms more often than the others. If a replica contains the top documents for a query, the average term frequency of each query term in the replica should be higher than in the original collection. Based on this heuristic, we construct a replica selection function that uses the average term frequency and penalizes the terms that occur in too few documents, as shown in Figure 4.

We implement the replica selection inference network as a pseudo InQuery database, where each pseudo document corresponds to a replica or collection, its index stores the *df* (document frequency) and *ctf* (replica/collection term frequency) for each term.

6 Effectiveness of Partial Replica Selection

In this section, we evaluate the effectiveness of our replica selection approach using the InQuery retrieval system [5], and the 2 GB TREC volumes 2+3 collection and the 20 GB TREC VLC collection. We use queries developed for TREC topics 51-350 in our experiments. We compare our proposed replica selection function with the collection ranking function. We measure their ability to pick the relevant partial replica, and the precision of the resulting response as compared with searching the original collection.

6.1 Experimental Settings

We use the 2 GB collection to compare the effectiveness of our replica selection function with the InQuery collection ranking function using short queries, and demonstrate the effectiveness of our replica selection function using both short queries and long queries. A short query is simply a sum of the terms in the corresponding description field of the topic. Long queries are automatically created from TREC topics using InQuery query generation techniques [5], which consist of terms, phrases and proximity operators. Generally, a long query for a topic is more effective than the short query [5]. The average number of terms per query is 8 for short queries and 120 for long queries after removing the stopwords. We further divide queries into replicated and unreplicated queries, where the replicated queries are those whose top documents are used to build the replicas. Since only topics 51-150 and 202-250 have relevance judgment files for the 2 GB TREC collection, we use them to test precision.

We use the 20 GB collection to examine how the collection size affects the effectiveness of our replica ranking function. Since we do not have relevance judgments for topics {51-150,202-250} against the 20 GB collection, but the 2 GB collection is a subset of the 20 GB collection, we use the relevance judgments for the 2 GB collection to produce the precision figures.

Our experiments repeat the following procedure 5 times: each trial uses a different number as the seed to produce random numbers, and thus picks different queries for a query set, in order to break the possible correlations between queries. In each trial, we randomly choose 50 queries from queries {51-150, 202-250} as our *unreplicated* query set T, and randomly divide the remaining 250 queries into 5 sets such that each set contains 50 queries, $\{Q_i, i = 1,2,3,4,5, |Q_i| = 50\}$. These queries are the *replicated queries*. We then build a 6-layer replication hierarchy from the original collection. For each query in Q_i, we collect the resulting top n documents from the original collection to build 5 partial replicas $\{D_i, i = 1,2,3,4,5\}$, where D_i contains at most $n * i$ documents, consisting of the top n documents for each query in query sets $\{Q_j, j = 1, ..., i\}$. Clearly, the $D_1 \subset D_2 \subset D_3 \subset D_4 \subset D_5 \subset$ the original collection.

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

(a) Number of replicated queries replica selection sends to replicas (99 queries).

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

(b) Precision of 50 unreplicated queries

Table 2: Comparing ranking functions using short queries on the 2GB TREC volumes 2+3 collection

When we replicate the top 200 documents for each query, the replica size ranges from 2% to 10%, and from 0.2% to 1% of the original collection for the 2 GB and 20 GB collections, respectively. We build a replica selection inference network to rank these five replicas and the original collection.

For the 20 GB collection, we also use queries 301-350 as our unreplicated query set, since these 50 topics are more thoroughly judged against the 20 GB collection than topics {51-150, 202-250}. We use queries 51-100 as Q_1, 101-150 as Q_2, 151-200 as Q_3, 202-250 as Q_4, and 251-300 as Q_5.

For both replicated and unreplicated queries, we expect we will have to tolerate some loss in precision in order to avoid searching the entire collection. We choose a drop in precision between 0% and 10% for a query as our acceptable range, i.e., searching the selected replica retrieves at most one less relevant document for every 10 documents as compared with searching the original collection.

6.2 Comparing Ranking Functions

In this section, we compare our replica selection function (Figure 4) and the InQuery collection ranking function (Figure 3) by varying k and b for short queries in test trial 1 when we replicate the top 200 documents for each query. We show that our replica selection function is comparable in precision and ability to pick the expected replica with some configurations of the collection ranking function for replicated queries, but that it significantly improves precision for unreplicated queries.

Table 2 lists the results of replica selection using different ranking functions. Table 2(a) lists the the number of replicated queries and to which replica or collection each function directs the queries, when the parameters, k and b, vary. Columns 1 through 3 list the name of functions, the values of parameters k and b, and the function abbreviations. Columns 4 through 9 contain the number of queries that the replica selector sends to each of the replicas (D_i) as well as the original collection (C); columns 10 through 13 contain the percentages of queries that are directed to the expected replica (right), smaller replica, larger replica, and the original collection. The *expected replica* for a replicated query is the smallest replica that is built with the top documents of the query. Note it is possible for a replica smaller than the expected to contain all top documents for a given query, since the top documents of other queries could include the top documents for this query. The "expected" (E) row lists the number of judged queries that we would expect the replica selector to direct to each replica and to the original collection, if it were perfect with respect to the queries used to build the replicas.

For the InQuery collection ranking function, varying k from 100 to 400 does not significantly change performance (compare I3-I5). When we set k to 200 (the default of the InQuery collection ranking function) and increase the value of b, the replica selector directs more queries to the replicas. For the replicated queries, the default InQuery collection ranking function (k=200,b=0.75) directs 70% of queries to the original collection, which is not our choice.

For the replica selection function, $k = 2$ gets better results than $k = 1$ and $k = 4$ (compare the functions R3-R5). When we decrease the value of b, the replica selector directs more queries to the replicas. Among the functions listed in Table 2(a), six functions *random*, I3, I4, I5, R1, and R2 direct more than 80% of replicated queries to the replicas. We need to compare their precision to determine which one is best for replica selection.

We compare the precision of the functions for both replicated queries and unreplicated queries. When we examine the precision for replicated queries, all these functions except random selection are acceptable, since the precision resulting from these functions drops within 8.7%. However, when we examine the precision for unreplicated queries listed in Table 2(b), the precision difference is significant. In Table 2(b), the first column lists the average number of documents at which we present the precision. Column 2 lists the precision when all queries go to the original collection, i.e., what percent is relevant to the top m documents when searching the original collection. Columns 3 through 8 list the results using random selection, and each ranking function. The numbers in parentheses show the precision percentage difference as compared with searching the original collection.

It is not surprising that random selection performs poorly, because it has high probability of picking a replica with few

Size	Query Type	Avg. Queries to Replica					C	% to Replica			C
		D_1	D_2	D_3	D_4	D_5		right	smaller	larger	
	Expected	21.6	16.6	21.4	20.6	18.8	0	100%	0%	0%	0%
2 GB	short	20.0	13.2	14.2	17.0	21.0	13.6	59.8%	8.1%	18.4%	13.7%
2 GB	long	18.0	17.4	12.6	14.4	28.2	8.4	57.0%	8.7%	25.8%	8.5%
20 GB	short	15.4	13.2	12.0	15.4	24.6	18.4	56.5%	4.2%	20.4%	18.6%

Table 3: Average number of replicated queries replica selection sends to replicas (replicas built with the top 200 documents)

at m	2 GB + short		2 GB + long		20 GB + short		20 GB + 301-350	
docs	C	Replica	C	Replica	C	Replica	C	Replica
10	42.8	39.4 (-8.0)	55.3	52.0 (-6.0)	12.8	12.6 (-1.6)	40.4	36.2 (-10.4)
20	39.4	36.1 (-8.4)	52.7	48.6 (-7.8)	12.3	11.9 (-3.1)	35.4	30.7 (-13.3)
30	35.5	32.7 (-7.8)	50.0	45.7 (-8.7)	11.8	11.9 (+0.8)	31.3	26.9 (-14.2)
100	27.2	24.0 (-11.6)	40.4	35.1 (-13.2)	10.0	9.6 (-4.2)	20.2	17.8 (-11.9)
200	21.8	18.3 (-16.5)	33.1	27.3 (-17.4)	8.4	7.7 (-7.9)	14.4	12.8 (-10.9)

Table 4: Average precision of unreplicated queries (each trial has 50 queries and replicas built with the top 200 documents)

relevant documents. For unreplicated queries, it causes precision percentage losses ranging from 38.0% to 50% compared with searching the original collection, C.

Using InQuery collection ranking function **I3** where we set $k = 200$ and $b = 1$, the precision losses of unreplicated queries range from 24.6% to 44.7%. We get our best result using our replica selection function **R2** with $k = 2$ and $b = 0.2$. The precision losses of unreplicated queries range from 4.8% to 14.5%. For the top 30 documents, the precision losses of unreplicated queries range from 4.8% to 9.6%. In the remaining experiments, we set $k = 2$ and $b = 0.2$.

6.3 Effectiveness with Replicated Queries

This section evaluates our proposed replica selection function for replicated queries on a wider range of queries and collections. We want to test whether the replica selector directs most of replicated queries to an expected replica. Although we use 250 queries to build replicas, we only present the results for the 99 replicated queries which have relevance judgment files in this section.

We count the average number of replicated queries that the replica selector directs to each of the replicas and the original collection over 5 trails, as shown in Table 3. In Table 3, columns 1 and 2 indicate the original collection size and the query type. Columns 3 through 8 on the row of "Expected" list the average number of judged queries that are used to build a replica, but not used in a smaller one over 5 trials. Columns 3 through 8 on other rows list the average number of queries the replica selector sends to each replica and the original collection. Columns 9 through 12 contain the average number of queries that the selector directs to the expected replicas (right), smaller replicas, larger replicas, and the original collection (C).

For short queries on the 2 GB collection, on the average, our replica selector directs 86.3% (59.8%+8.1%+18.4%) of replicated queries to the replicas, and 67.9% of queries to the expected replica or a replica smaller than we expect. For long queries on the 2 GB collection, on the average, our replica selector directs 91.5% (57.0%+8.7%+25.8%) of replicated queries to the replicas, and 65.7% of queries to the expected or a smaller replica. For short queries on the 20 GB collection, on the average, our replica selector directs 81.4% (56.5%+4.2%+20.4%) of replicated queries to the replicas, and 60.7% of queries to the expected or a smaller replica.

We also compare the precision of executing queries resulting from replica selection with searching the original collection. Even though some queries go to smaller replicas than expected, precision is not compromised. Replica selection results in precision percentage loss within 3.0% as compared with searching the original collection when we use short and long queries on 2 and 20 GB collections.

6.4 Effectiveness with Unreplicated Queries

This section evaluates the replica selection function for unreplicated queries. For the 2 GB collection, on the average over 5 trials (each trial has 50 queries), 15.4 short queries and 16.6 long queries have a replica that causes the precision loss within 5% as compared to searching the original collection, and the replica selector directs 84.7% and 86.9% of these queries to the replicas, respectively. Table 4 compares the retrieval precision of executing unreplicated queries against replicas or the original collection selected by our replica selector with only searching the original collection. Column 1 lists the number of documents at which we present the precision figures. Columns 2, 4, 6, and 8 list the precision figures when all queries go to the original collection. Columns 3, 5, 7, and 9 list the average precision of replica selection over 5 test trials. The numbers in the parentheses show the precision percentage difference.

For the 2 GB collection, using short and long queries cause a precision loss ranging from 7.8% to 16.5%, and from 6.0% to 17.4%, respectively. For the top 30 retrieved documents, which is a retrieval level that concerns many online users, our replica selector causes average precision percentage loss within 8.7% of searching the original collection.

For the 20 GB collection using short queries, when we use the relevance files for the 2 GB collection, the precision ranges from improving by 0.8% to a loss of 7.9%. When we use short queries 301-350 as our unreplicated queries, the precision drop for is within 14.2%. Topics 301-350 were much more thoroughly judged than topics {51-150, 202-250} for the 20 GB VLC collection. Although additional judgments are still under way, we think the results using topics 301-350 are more accurate, which means our replica selection performs slightly worse on the 20 GB collection than on the 2 GB collection. However, the precision percentage loss of 14.2% in our context only means that we retrieve one less relevant document for the top 30 documents.

7 Performance of Partial Collection Replication

In this section, we compare the performance of partial replication to collection partitioning on the same server, and show that replication is superior at improving performance even given small percentages of query locality. We do a more thorough evaluation elsewhere [19]. These results demonstrate the potential for a lot of performance improvement.

We simulate performance of a parallel IR server using a symmetric multiprocessor. In previous work, we validated this simulator against an implementation of InQuery running on a multiprocessor Alpha, and demonstrated that the simulator is accurate with respect to the implementation [19].

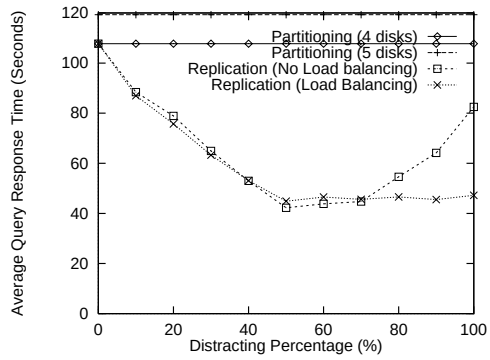
For this experiment, we model the command arrival as a Poisson process, use short queries with an average of 2 terms per query, and issue query, summary, and document commands with a ratio of 1:1.5:2, as we found in the THOMAS logs. We vary the number of CPUs from 1 to 4. The base configuration has 4 disks to which all CPUs have access. Each disk stores up to a 4 GB collection and its associated indices. The original collection in this case is 16 GB, we therefore need at least 4 disks. In the base configuration, for a query command, the server assigns 4 threads, each of which executes the query against a collection partition on a disk in parallel, and then merges the results returned by each thread; for a summary command, the server assigns $n(\leq 4)$ threads to execute and then merges the results; for a document command, the server assigns one thread to fetch the the document. We study the effects of an additional disk, asking the following question: *should we partition the collection on the 5 disks, or should we partially replicate the top query results on the 5th disk?*

If we build a partial replica using the additional disk, the replica can be as large as 4 GB, which is 25% of the original collection. If query locality is high, the replica selector may send more queries to the replica than the original collection, which may result in load imbalance. We load-balance by predicting the response time of the replica (R) and the original collection (C) using the average response time and the number of the outstanding commands. When the replica selector chooses the replica, we calculate the predicted response time $p_resp_i, i = \{R, C\}$ using $ave_resp_i \cdot num_wait_mes_i$, where ave_resp_i is the average response time for last 200 responses for either the replica or the original collection, and $num_wait_mes_i$ is the number of the outstanding commands to which either the collection or the replica has not yet responded. When p_resp_C is less than p_resp_R, the command is sent to the original collection.

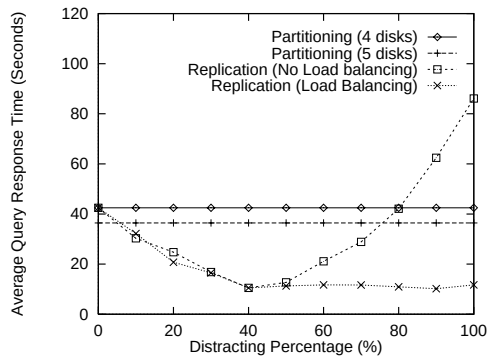
Figure 5 shows the average response time versus the distracting percentage that represents the percentage of the queries the replica selector sends to the replica, when commands arrive at 10 per second for 1, 2, and 4 CPUs, each with different performance bottlenecks. In Figure 5(d), the rows of "CPU" list the average utilization of the CPUs, the rows of "DISK" list the average utilization over the disks that store the original collection; the rows of "D_repl" lists the utilization of the disk that stores the partial replica.

In Figure 5(a), where one CPU is already overutilized for 4 disks when the commands arrive at 10 per second, partitioning the collection over 5 disks is actually worse than over 4 disks, because it exacerbates the CPU bottleneck; i.e., partitioning adds a little more CPU overhead to merge the results and disk delay of the slowest disk is exaggerated when the CPU is busy. The performance of partial replication is much better (usually a factor of 2) than partitioning at all data points of the distracting percentage, because searching the replica that uses one disk needs less CPU time than searching the original collection which is distributed over 4 disks, and thus shortens the waiting time for the CPU. For the same reason, the improvement increases as the replica distracts increasingly many commands until the disk load gets too high. At that point, load balancing is necessary.

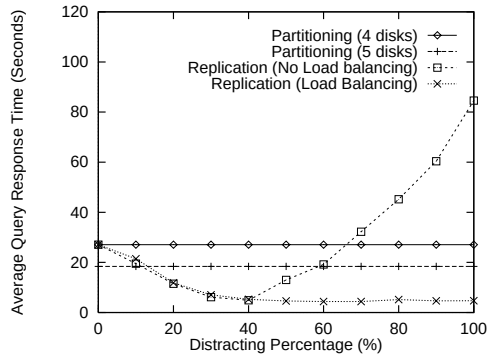
For 2 and 4 CPUs as shown in Figure 5(b) and (c), the



(a) $N_{CPU} = 1$



(b) $N_{CPU} = 2$



(c) $N_{CPU} = 4$

Num. CPUs	Res.	partitioning 4 disks	partitioning 5 disks	partial replication no bal.	partial replication with bal.
1	CPU	99.3%	99.5%	99.0%	99.4%
	DISK	48.1%	36.0%	35.6%	38.2%
	D-repl			78.5%	53.3%
2	CPU	85.3%	95.1%	81.0%	88.9%
	DISK	84.1%	68.7%	64.4%	70.8%
	D-repl			97.0%	85.1%
4	CPU	46.9%	62.6%	40.0%	48.0%
	DISK	90.6%	89.5%	50.1%	71.5%
	D-repl			99.2%	86.6%

(d) hardware utilization when $dp = 50\%$

Figure 5: Partitioning versus Partial Replication for a 16 GB collection when commands arrive at 10 per seconds

CPU and disk utilizations are better balanced. However, the CPUs are a bottleneck for 2 CPUs and 5 disks, and the disks are a bottleneck for 4 CPUs. For partitioning over 5 disks, using 2 CPUs limits its improvement due to the CPU bottleneck, and using 4 CPUs reaches the largest improvement for partitioning over one more disk. Partial replication performs better than collection partitioning when the replica distracts more than 5% and 12% of commands using 2 CPUs and 4 CPUs, respectively. System improvements due to partial collection replication are determined by query locality and increase as the query locality increases until the replica is overloaded. Replication with load balancing improves the query response time by a factor ranging from 2 to 4 as compared to partitioning over 5 disks, when the replica distracts more than 20% of commands.

8 Conclusion

In this paper, we investigate how to select a relevant partial replica using the inference network, which is the first step towards to developing replication and partial replication strategies to improve IR system performance while maintaining the retrieval accuracy. Our approach enables a system to rank partial replicas for relevance to a query, and automatically selects a replica based on both relevance and load. We examine real IR system logs and find that there is sufficient query locality in real IR systems to justify partial collection replication for information retrieval. We demonstrate the effectiveness of our approach using the InQuery retrieval system and TREC collections and show that the inference network model is a very promising approach for ranking partial replicas. By using our new replica selection function, our replica selector can direct more than 85% of replicated queries to a relevant partial replica rather than the original collection, and it achieves a precision percentage loss within 8.7% and 14.2% for the top 30 retrieved documents for those unreplicated queries, when sizes of replicas range from 2% to 10% for the 2 GB collection, and 0.2% to 1% for the 20 GB collection, respectively. Our performance experiments show that partial collection replication significantly improves performance over collection partitioning for the same number of disks when the replica satisfies a modest number of requests.

Acknowledgments

This material is based on work supported in part by the National Science Foundation, Library of Congress and Department of Commerce under cooperative agreement number EEC-9209623, and in part by Defense Advanced Research Projects Agency/ITO under ARPA order number D468, issued by ESC/AXS contract number F19628-95-C-0235. Kathryn S. McKinley is supported by an NSF CAREER award CCR-9624209. We thank Ben Mealey and Library of Congress for providing the THOMAS log. We thank Doug Cutting and Excite for providing the Excite log. Any opinions, findings and conclusions or recommendations expressed in this material are the authors and do not necessarily reflect those of the sponsors.

References

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

[2] M. Baentsch, G. Molter, and P. Sturm. Introducing application-level replication and naming into today's Web. In *Proceedings of 5th WWW Conference*, Paris, France, May 1996.

[3] A. Bestavros. Demand-based document dissemination to reduce traffic and balance load in distributed information systems. In *Proceedings of SPDP'95*, San Anotonio, Texas, October 1995.

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

[5] 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*, Valencia, Spain, September 1992.

[6] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th ACM SIGIR*, Seattle, WA, July 1995.

[7] 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.

[8] A.S. Chakravarthy and K.B. Haase. Netserf: Using semantic knowledge to find internet information archives. In *Proceedings of the 18th ACM SIGIR*, pages 4–11, Seattle, WA, July 1995.

[9] 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.

[10] P. B. Danzig, J. Ahn, J. Noll, and K. Obraczka. Distributed indexing: A scalable mechanism for distributed information retrieval. In *Proceedings of the 14th ACM SIGIR*, pages 221–229, Chicago, IL, 1991.

[11] D. Dowdy and D. Foster. Comparative models of the file assignment problem. *Computing Surveys*, 14(2), June 1982.

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

[13] N. Fuhr. A decision-theoretic approach to database selection in networked ir. In *Workshop on Distributed IR*, Germany, 1996.

[14] L. Gravano and H. Garcia-Molina. Generalizing gloss to vector-space databases and broker hierarchies. In *Proceedings of the 21st VLDB Conference*, Zurich, Switchland, 1995.

[15] L. Gravano, H. Garcia-Molina, and A. Tomasic. The effectiveness of gloss for the text database discovery problem. In *Proceedings of the SIGMOD 94*, pages 126–137, September 1994.

[16] D. Harman, editor. *The Fourth Text REtrieval Conference (TREC-4)*. National Institute of Standards and Technology Special Publication, Gaithersburg, MD, 1995.

[17] Vegard Holmedahl, Ben Smaith, and Tao Yu. Cooperative caching of dynamic content on a distributed web server. In *Proceedings of HPDC-7*, pages 235–242, Chicago, IL, 1998.

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

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

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

[21] E. M. Voorhees, N. K. Gupta, and B. Johnson-Laird. Learning collection fusion strategies. In *Proceedings of the 18th ACM SIGIR*, Seattle, WA, 1995.

[22] 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.