Server Selection Techniques for Distributed Information Retrieval

Yoshiya Kinuta Brian Neil Levine R. Manmatha
Department of Computer Science
University of Massachusetts, Amherst, MA 01003
{ykinuta, brian, manmatha}@cs.umass.edu

Abstract

Server selection is typically defined as maximizing *network performance* under the assumption that each server holds an exact replica of all data. We propose and evaluate methods of server selection when servers are not exact replicas such that we maximize both network performance and information retrieval (IR) precision (i.e., the relevance of retrieved data). We show that naive composition of previously proposed techniques from networking and IR perform poorly. We propose improving the performance of current IR selection techniques by using language model-based selection to construct local replicas of databases that network selection predicts are likely to be poor network performers and that IR selection predicts will have relevant results. In our experiments our technique is capable of selecting servers and retrieving information that is as accurate as techniques that focus on IR performance; moreover, our techniques reduce network latency significantly (30–67% over those IR selection techniques), at the cost of local storage (16–33% of all data).

1 Introduction

Distributed information retrieval (IR) systems allow clients to query multiple information databases at once. This is usually done in a three stage process [1]: *resource description*, where the information sources are modeled [6, 1, 16]; *collection selection*, where resources and collections appropriate to the query are selected and searched [2, 7, 5]; and *results merging*, where ranked results from each search engine are combined [1, 4, 13, 15, 12]. Collection selection methods have assumed that network performance is immaterial and ignored such costs. However, when servers are distributed over the Internet, the network latency and TCP performance from a client to each server will be heterogeneous. Indeed, some servers may not be available at all because of network conditions (e.g., routing partitions or persistent loss) or system problems (e.g., denial of service attacks).

Related previous work from the networking literature has studied methods of server selection where network latency is minimized (which we call *network selection* to avoid confusion) — however, such studies have considered only scenarios where each server in a group holds an exact replica of all data; those methods are not designed nor able to choose servers with any notion of IR accuracy.

In this paper, *we propose and evaluate methods of distributed information retrieval where selection is affected by both heterogeneous collections stored by servers and the heterogeneous network latencies among clients and servers.* We know of no previous work that considers both networking and IR performance. To evaluate our work, we used extensive network traces to simulate realistic behavior of servers on the Web; we mapped this performance to TREC information databases and user query logs.

Our results show that methods that naively combine collection selection (specifically, CORI [1]) and network selection methods performed poorly. This is because choosing databases according to network performance harms the accuracy of retrieval — and vice versa. There is no reason for servers with good network performance to contain the most relevant documents, nor for the databases most relevant to a query to have good network performance. In order to overcome this conflict between network performance and relevance of data, we introduce a novel technique that improves network performance while maintaining reasonable retrieval accuracy. Our techniques are based on *language models* [14], which are statistical characterizations of the content of databases.

This paper was supported in part by National Science Foundation awards ANI-033055, EIA-9983215, and EIA-0080199 and in part by the Center for Intelligent Information Retrieval. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsor.

In sum, our proposed technique involves two steps: (1) Given a set of distributed databases, the client constructs a concise *language model* description of the most relevant databases based on training queries; (2) then the client caches part of or replicates all information from those databases that are similar to the constructed language model. Our results show:

- Methods for network and IR server selection are inherently in conflict.
- Our language modeling technique is a good method of constructing local replicas that are relevant to future queries.
- Our method of selection is a more accurate approach to server selection as compared to methods that ignore IR metrics, as well as naively composed methods.
- Our method of selection has network performance better than server selection methods that do not consider network characteristics such as CORI and naively composed methods.

Our work is based primarily on investigations of two previous works. First, it is based on our network selection methods reported in Hanna, *et al.* [8], which did not consider heterogeneous collections of data at servers. We proposed a technique for selecting the best servers using ping values and observed file transfer times. Second, it is based on Callan's proposed the *CORI* database selection algorithm [10, 1], which is an IR-prioritized server selection algorithm. Callan did not consider cases when the network performance of servers is heterogeneous.

The remainder of this paper is organized as follows. Section 2 reviews related previous work in IR and network server selection. Section 3 describes our experimental methodology. Section 4 discusses the inherent conflict between network selection and IR selection. Section 5 presents our proposed method of improving the network performance of IR selection techniques. Section 6 offers our concluding remarks.

2 Background

Our investigation focuses on enhancing the performance of server selection when servers do not store replicas of one data set. We are unaware of previous work that addressed both network performance and IR accuracy. In this section, we review previous work that has considered these questions separately.

2.1 Distributed Information Retrieval

A distributed retrieval system needs to tackle three issues [1]:

1. *resource description:* the characterization of the resources stored by a server [6, 1, 16].
2. *collection selection:* the selection of servers to search that are most appropriate for a query [2, 7, 5].
3. *results merging:* the combination of results from different search engines into a a unified ranking [1, 4, 13, 15, 12].

Reviews of the work in distributed retrieval are covered more closely elsewhere; e.g., [1, 12, 2]. In this paper, we focus specifically on the *collection retrieval inference network* (CORI) [1] in a distributed IR system utilizing the INQUERY [3] search engine.

CORI assumes that external access to each database is through a limited interface, i.e., a search engine. CORI pre-computes resource descriptions for each database by probing them with a set of test queries. When a user sends a query, CORI compares the query to the resource descriptions to decide which databases are the most appropriate. CORI then sends the query to that subset of databases, who then use their own search engines to return results to CORI. CORI then merges all returned results into a single ranked list using a heuristic algorithm.

Although there has been much effort to improve the selection of relevant databases, network performance of databases chosen by such a selection algorithm is often ignored. It is very important to address this issue because in a distributed environment servers may be located far away from each other topologically, and network traffic and load can vary from server to server. In the worst case, routing layer failures or denial-of-service attacks may prevent a server from even being reachable. In general, distributed retrieval systems like INQUERY are unaware of the availability of a server when performing selection.

The work that is most related to ours is by Lu, et al. [11]. Their approach to improving IR system performance was by hierarchical replication of databases. While their focus is on processor overhead and query response time in a local area network, our experiments focus more on network latency over the Internet.

2.2 Distributed Systems and Networking

Network performance between a client and server on the Internet is dictated by the server load and the congestion on the path between them. Determining which server of many offers the best network performance is difficult because many tests that are simple to perform (e.g., hop count or ping measurements) are poor predictors of TCP performance [8]. Our work in Hanna, et al. [8] offers a review of previous work and a comparison of many popular methods. In summary, we found that network-based methods — such as selecting a server based on minimal hop count or autonomous system count — perform as poorly as a random selection of servers. Minimizing round-trip time measured by ICMP ping messages performs adequately as a selection metric.

We also showed that best performing mirror servers remain a small fraction of an increasing population of servers over time, and they have greater performance stability than average servers. Our proposed selection algorithm choses the best servers by analyzing their ping values and file transfer times as predictors of stability. Servers with the best download time for a 250K file tend to maintain good performance relative to the other servers over a period as long as a month. We build on that method in this paper.

As part of that paper, we have made publicly available a large set of measurements recorded from actual Internet servers [9]. Specifically, we continually measured network characteristics between 193 servers and six clients in North America. During a 41-day period, we continually cycled through all servers from each client and measured the following: five ping times; one trace-route; file transfer times for file size of 10k, 30k, 100k, 250k, 500k, 750k, and 1M bytes. For each client, the 193 servers were visited sequentially, each cycle termed a *round*.

The methods we propose in Hanna, et al. [8] work under the limitation that all servers consist of identical data. However, it is often the case that servers may contain different data. This may be because they are under different administrative control, or because it is difficult to synchronize the content of so many servers.

Performance of distributed systems can be measured in various ways. However, in our experiments, we focus on network download latency. We define this as the time from when a client initiates a TCP connection to the server until the time at which the client receives the requested entire object.

3 Methodology

Our evaluations of different server selection methods are based on a careful combination of real measurements that have been collected by ourselves and others for evaluating network and IR server selection, respectively. Using these two sets of data, we perform several off-line experiments.

To test IR precision, we used volumes 1, 2, and 3 of the Text Retrieval Conferences (TREC) run by the National Institute of Standards and Technology (NIST). TREC 1-2-3 consists of several collections of newspaper archives, federal document archives, and similar content totaling 3.2 GB. NIST has also created TREC Query Topics, which are collections of queries and answers to the database; real users have determined the relevance of documents in the collection. These Query Topics allow researchers to perform offline analysis of IR accuracy since the relevance of all documents is known ahead of time. We used Query Topics 51–100 in our experiments.

To perform IR server selection, we used the INQUERY [3] implementation of Callan's CORI algorithm.

We simulated network latency of the databases selected using the same experimental logs collected in the Hanna, *et al.* study [8] as described in Section 2.

In our experiments we divided the TREC databases into a distributed set of 100 (and in a later section, 18 servers). We mapped these servers to a randomly-chosen but consistent subset of servers from the network logs.

Figure 1 illustrates our experimental process. A typical run of the experiment consists of the following example: A query is chosen from the TREC set and given to INQUERY, which selects 20 of the 100 databases that it believes to contain the most collection of data. Then, the set of 20 is reduced to 5 based on, for example, the five-best ping times of the 20 servers according to the network logs. The resulting download time of the query results is taken from the network logs, and the IR accuracy of the results is determined by INQUERY based on the TREC relevance data. After all queries are run, the next round of network results from the logs is considered and the queries are re-run. The figure illustrates just one selection strategy; however, we investigate many more in this paper.

4 Naive Strategies

Our first experiments were to examine naive strategies of composing IR and network methods for select-

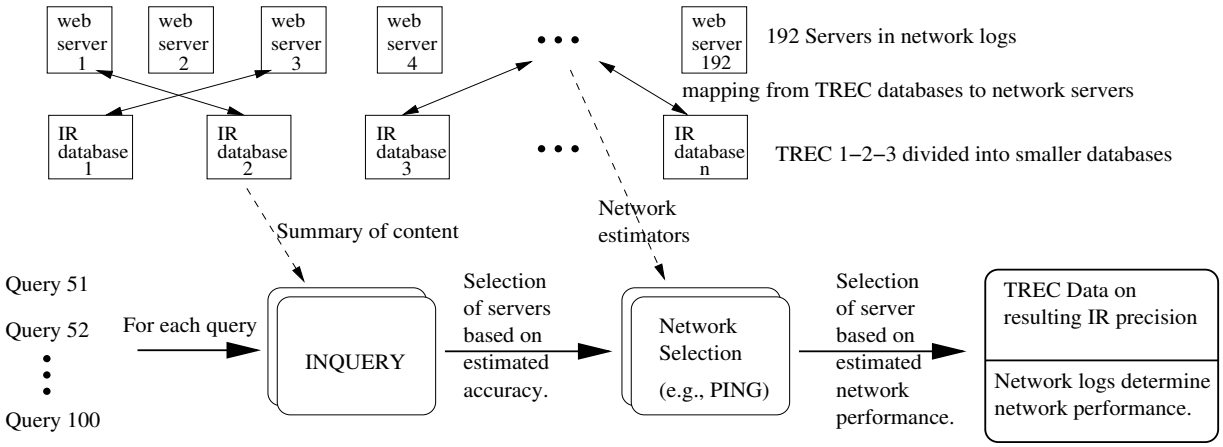

Figure 1: An overview of the mapping between Network logs and IR TREC logs. The CORI-PING strategy is diagrammed in particular.

	Avg. Precision	**Std. Dev.**	**95% c.i.**
CORI-20	13.6 %	0	0
CORI-5	5.1	0	0
Ping-20	1.2	0.7%	0.1%
Ping-CORI	1.3	0.6	0.1
CORI-Ping	0.9	0.2	0.1

Table 1: Precision of composed methods.

Average 10k File Transfer Time of Top 5 Databases (ms)					
	1st	**2nd**	**3rd**	**4th**	**5th**
CORI-20	346 (±35)	397 (±38)	432 (±38)	459 (±38)	485 (±38)
CORI-5	413 (±40)	519 (±40)	678 (±53)	1,178 (±101)	3,458 (±232)
Ping-20	217 (±32)	259 (±32)	281 (±34)	300 (±37)	317 (±39)
Ping-CORI	319 (±45)	372 (±48)	416 (±49)	665 (±116)	2,097 (±269)

Average 250k File Transfer of Top 5 Databases (ms)					
	1st	**2nd**	**3rd**	**4th**	**5th**
CORI-20	1,201 (±74)	1,516 (±66)	1,683 (±61)	1,805 (±62)	1,908 (±65)
CORI-5	1,594 (±66)	2,068 (±78)	2,719 (±103)	4,495 (±171)	10,202 (±362)
Ping-20	742 (±76)	836 (±80)	963 (±78)	1,079 (±67)	1,153 (±60)
Ping-CORI	949 (±88)	1,386 (±108)	1,839 (±124)	2,587 (±166)	4,586 (±251)

Table 2: Network performance of composed methods for 10k and 250k downloads (UCSC Client). Standard deviations appear in parentheses.

ing the best 5 out of 100 servers such that precision and download time is maximized. In each round of our simulation, we kept track of the following measurements:

- Average precision;
- Average file transfer time for 10K and 250K byte downloads.

First, we needed a base-line of how well selection methods perform on their own.

- **CORI-20** and **CORI-5:** CORI-20 specifies that the CORI selection algorithm selects 20 servers out of 100 to query. To see how CORI degrades the accuracy of the search when restricted to a small set we also show results for CORI with

5 servers.

- **PING-20:** 20 databases were chosen according to their average ping time and were used to query.

Next we evaluated two composed methods:

- **PING-CORI:** 20 servers are chosen based on the fastest average ping time, then CORI narrows the selection to the most relevant five.
- **CORI-PING:** The 20 most relevant servers are selected by CORI and then the five with the best average ping time are queried.

4.1 Evaluation

We evaluated several strategies by splitting TREC 1-2-3 into 100 databases, which we then mapped to 100 servers in the network logs. For each round of the network logs, we evaluated the performance using TREC queries 51–100. Tables 1 and 2 show the performance of the five methods. (Note that CORI has zero variation because the servers it picks does not change with network performance: all servers are available each round, and the queries do not change from round-to-round.)

The results show a clear incompatibility between network and IR server selection. There are several items of interest in the results.

The difference between CORI-20 and CORI-5 shows the effects of limiting server selection to a small number of nodes. Since CORI-5 is forced to ignore 95% of all servers (and therefore all data), precision is kept to a low 5.1%. Not surprisingly, the results for PING-20 show network selection is a poor method of selecting relevant servers.

The results for PING-CORI and CORI-PING show about the same performance as PING since any selection by network performance may remove most of the relevant servers. CORI-PING has the worst average precision of all methods.

Because CORI-20 and PING-20 select four times the number of servers, Table 2 shows the transfer time of the five fastest servers selected by four strategies. CORI-PING had such poor IR performance, we did not evaluate its network performance. Two different files sizes are shown, 10K and 250K downloads, as we observed more distinct differences between servers for larger downloads. Here we see that PING-20 can select servers that take almost half the download time of CORI-20 for the first five results that return to the client. When IR selection is restricted to querying five servers, it is difficult to maintain fast download times as a comparison between PING-CORI and CORI-5 shows; they also show an important trade-off: doubling of network time for an almost four fold increase in precision.

We believe that the results demonstrate a general result that a simple approach to combining IR and networking approaches does not work. Our conclusion is that IR precision is damaged more from network selection than the converse. Therefore, in the next section, we evaluate methods of improving the network performance of IR selection with a different strategy: locally replicating or caching information at the clients.

5 Local Replication based on LM Training

As we have discussed, there is no reason for databases most relevant to a query to also have good network performance for a particular client; this places network and IR selection methods at odds with each other. In this section, we examine the benefits of locally replicating at clients some data from the distributed database in order to improve network performance. Our results show that this strategy of trading local storage for improved network and IR performance is attractive and favorable.

There are several options for deciding what data should be replicated locally and we compared a number of them.

First, we investigated a *complement* strategy, which attempts to maximize coverage of the local partial replica of the distributed database. We first find the fastest set of servers (based on network selection); we then find another set of servers which are most different in content and then replicate these locally.

This strategy, and the others we study in this section, require a method of determining which servers are slow. Fortunately, our results published in Hanna, *et al.* already show how to pick servers that consistently provide good download speeds at clients [8]. We have shown that downloading a small 250k file from all servers is a very good predictor for up to 30 days of downloads. Therefore, this operation was performed by comparing the download speeds of the 18 servers in the first *round* of data downloads in the network logs. We considered any server that had a download time that was not more than twice the download speed of the fastest server to be "fast". This was typically 3–5 of the servers. We did not use ping because we required a metric that can select servers that remain consistently fast over time so that

we don't have to constantly switch which databases we locally replicated.

Figure 2 shows the IR performance of this technique (labeled "LM Aggregating Complementary"). The graph shows the precision of this technique in answering 50 TREC queries. In the experiment, the client starts the five fastest servers (as picked by network selection) and the precision of using these five servers is computed. The complement strategy then adds a server that is most different in content from the existing set. Our goal is to find the strategy that is best able to choose databases to replicate locally; and what is important about this experiment is that the order by which databases are added is determined by the strategy. Therefore, the graph shows the ability of each strategy to choose servers in a way that increases precision fastest.

For comparison, the graph also shows the performance of CORI (labeled "Collection Selection"), which represents an upper bound of sorts. This is because CORI has the advantage of being able to choose a different set of databases from the 18 for *each* query while the others must keep the same databases for all queries. (Though in the end, CORI will suffer in network performance because of the lack of local replication.)

Figure 2 shows that locally storing data complementary to fast servers performs much worse than CORI. We found that the poor performance arises because most of the relevant documents for most of the queries were obtained from a few database servers; i.e., some databases are completely irrelevant in the collection and it is of no help to store them locally. Thus, it seems important to find similar databases rather than different ones.

A second approach we explored was to start with the fastest servers and then add databases which are most similar to the original set of servers. As Figure 2 shows this approach (labeled "LM Aggregating Similar") is better but still may not work well. The problem is that while the initial set of servers is fast, there is no guarantee that any of them contains any relevant information. Thus finding databases similar to the initial set of fast servers does not mean that databases with a lot of relevant documents will be selected. The performance increase is probably due to luck: more databases in the set are relevant than not; and so finding relevant databases in the fast set is somewhat likely. This increases the chances that what we replicate locally is relevant, but does not guarantee anything. In fact, It is well known in IR research that one needs to look for databases which are most relevant to the query.

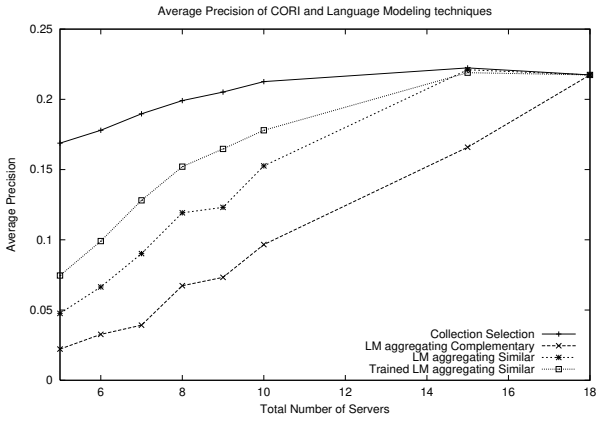

Figure 2: A comparison of strategies for locally replicating portions of the distributed database against the precision of CORI.

The approach we advocate, therefore, involves selecting the initial relevant set by training on a set of queries. We then find the set of servers that are most similar to this set and replicate or cache them. Our basic insight is that servers that are likely to be relevant but have poor network performance should be cached or replicated. In other words, no effort should be made to improve servers that are fast or are likely to be irrelevant. Experiments show that this technique can reduce network download speeds in half while not significantly reducing IR accuracy.

5.1 Language Model Based Replication

The approach behind selecting the most relevant databases for replication is the following. We need to find the statistical characteristics of the database that is most relevant to a training set of queries. We can then compare these statistical characteristics with the larger universe of all databases and select an additional set of databases that are most similar (but on which we have not trained). In our experiments we show that this training procedure generalizes.

More specifically, the procedure is as follows. First, we characterize every database statistically using *language models* (which we formally describe later). Then, we train using a small set of queries and a small set of randomly selected databases and find the subset of databases that are most relevant to our need. For example, in our case we have 18 database servers[1].

[1]We switched from 100 servers as INQUERY took excessive processing time for 100 databases. We also used TREC-2 only to decrease processing time, which we split up to have an even number of documents at each node; the collection most

We train using 20 queries (TREC queries 51–70) on five servers of these and pick the two servers that were most relevant to our training queries. We use the language model characterization of these two databases to find other servers that are the most similar from the remaining servers; in our case, we evaluated picking one and four other servers. This procedure avoids having to train on all the databases, which may not be practical. We now discuss the details of this approach.

A *Language Model* of a collection is a statistical description of how words are distributed in a collection. Language models have been used in a number of different areas like speech recognition, machine translation, optical character recognition, and information retrieval. In this paper, we use the language model of a database to provide a concise statistical description of what the particular database contains. The simplest kind of language model is a *unigram language model*, which assumes that every word occurs independently of each other and that the probability of a word occurring is given by the frequency of occurrence of the word in the corpus. Strictly speaking, words do not occur independently of each other. However, unigram language models are good approximations and have been successfully applied to many different tasks.

In many situations, the complete database may or may not be accessible directly (documents may only be accessible through a search engine). Callan, *et al.* [2] developed a procedure called query-based sampling to get around this problem. The sampling procedure involves probing a small number of documents (300–500) from the documents obtained by using a set of queries. A language model is then generated using these documents. Callan, *et al.* [2] have shown that the language models created by probing are good approximations of the language models obtained from the entire database. We used query-based sampling to create language models for each of the 18 databases in our experiments.

We can determine how similar two language models are by computing the Kullback-Leibler (KL) divergence. Given two probability mass functions, $p(x)$ and $q(x)$, their KL divergence is:

$$D(p \parallel q) = p(x) \log \frac{p(x)}{q(x)}$$

The KL divergence shows how likely it is that $q(x)$ is derived from $p(x)$. The KL distance is always greater than zero. Low KL distances denote a greater similarity between databases.

naturally broke up in to 18 parts.

	IR Avg. Precision
CORI 10	15.3%
CORI 18	16.0%
LM-replication-3	13.2%
LM-replication-6	13.2%

Table 3: The average precision that results from the four selection strategies.

	Local Storage Costs
CORI-10	0 MB
CORI-18	0 MB
LM-replication-3	143 MB (± 38MB)
LM-replication-6	242 MB (± 62 MB)

Table 4: The storage costs of the four selection strategies.

5.2 Algorithm

The specific algorithm we used to discover which databases should be cached or replicated locally is described below:

1. Training phase:
   (a) Pick a subset of databases from the set of all databases. In our case, we picked 5 databases at random from the set of 18 databases.
   (b) Query these databases using a small set of queries. We used 20 queries.
   (c) Choose the top k databases that are relevant to the most queries and create an aggregate language model by averaging the language model of the top k databases. In our case $k = 2$. This aggregate language model serves as the base relevance model.
2. We determine the C databases whose language models are closest to the base language model and replicate them locally. Closeness or similarity is determined by computing the KL distance.

Step 2 is carried out as follows. Let the language models of the two databases selected by the training procedure be T_1 and T_2. We average these two to create a language model T. We then compare T against the composite language model B_i, which is created by averaging T and the language model A_i of every other database (the remaining 16), using the KL distance and find that B_i with the smallest KL distance. Without any loss of generalization assume that this is B_1 and the corresponding database (actually its language model) is A_1. We now repeat this procedure

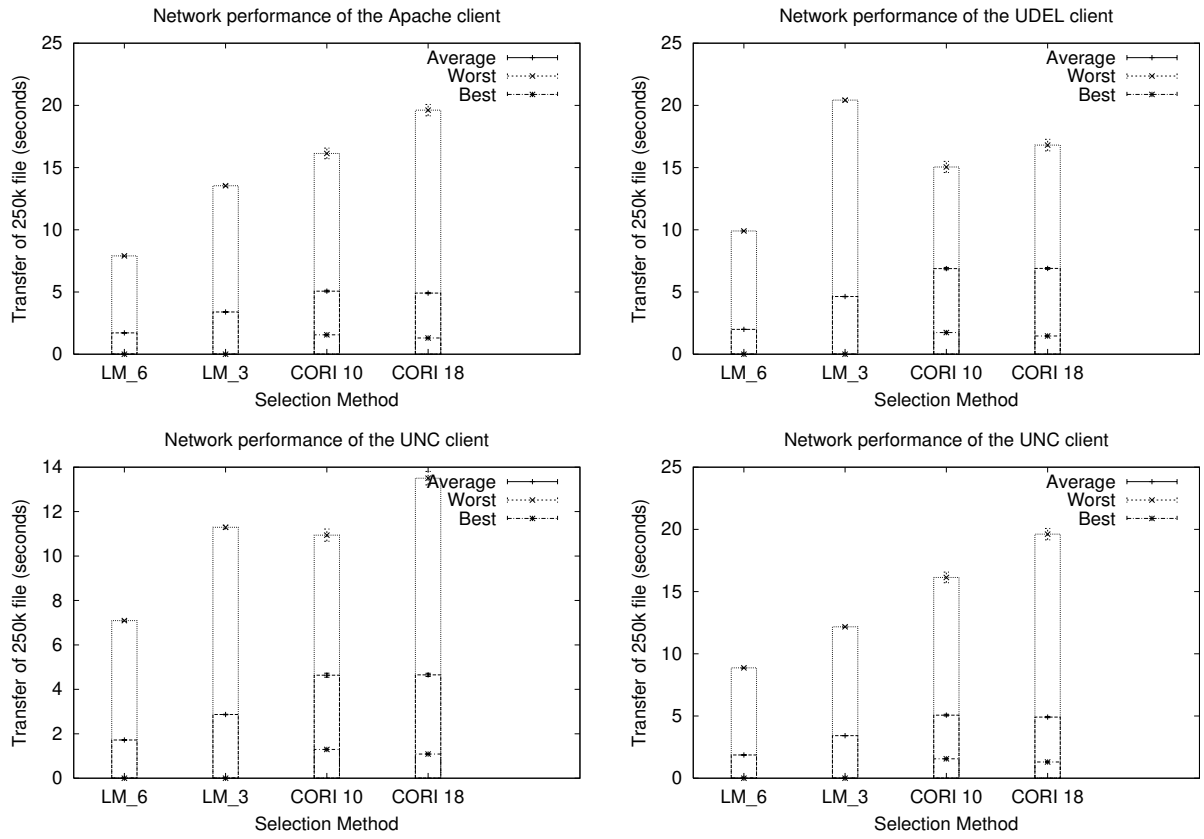

Figure 3: Network performance of CORI and LM-based replication for the four clients.

with B_1. That is, we now compare B_1 with language models C_i $(2 \leq i \leq 16)$ where C_i is the composite language model $B_1 + A_i$ $(2 \leq i \leq 16)$ and select the C_i with the lowest KL distance. We repeating this procedure until we have as many databases as we want to replicate. This approach is somewhat computationally intensive and future work would involve speeding this step up by using some approximations.

Note that this procedure of creating composite language models avoids the necessity to smooth the language models (i.e., smooth the probabilistic estimates).

5.3 Experiments

To evaluate our strategy of LM-based selection of replicated databases, we ran a number of experiments. We created ten different mappings from the 18 databases to 18 randomly selected servers; this also changed which five databases were chosen initially for training. For each mapping TREC queries 51–70 were used for training. After training, we end up with two databases (out of the five) which are most relevant to the training queries. These two servers are used to select either one more or four more databases by finding those servers whose language models are similar to theirs (two servers selected during training). These strategies we call "LM-replication-3" and "LM-replication-6", respectively, representing the number of databases replicated locally.

For each mapping, TREC queries 71–100 were run for each of round of measurements for twenty log days. We ran this for logs from four different clients in the network logs from Hanna, *et al.*: UCSC.edu, UNC.edu, UDEL.edu, AND USC.edu. The measurements closest to noon on each day for the clients was used. Therefore the experiments show the average of 200 experiments for each client.

Tables 3 and 4 shows our results for four different strategies at each client. CORI was not designed to consider network performance at all: if given the opportunity to contact all 18 servers it might contact all of them if it decides all have some relevant information to offer. However, the consequence is that CORI will pick even the slowest servers every time.

This isn't a fair comparison and so we also show the results when CORI is restricted to selecting the 10 most relevant servers. These are shown as CORI-10 and CORI-18 on the graph. Figure 3 also shows the network performance of language model-based replica construction where the size of the local replication is limited to 3 databases and 6 databases.

The results show that local storage based on language model training results in an significant increase in network performance for CORI while only a small cost in IR precision.

Replicating three databases locally (for use over a period of 20 days) reduces download times to 67%, 62%, 66%, and 67% of CORI-10 selection for clients at UDEL, UNC, USC, and UCSC for replicating three databases; and 30% , 37%, 36%, and 34% of CORI-10 selection for clients UDEL, UNC, USC, and UCSC for replicating six databases. Moreover, IR precision drops only 2–3% points from CORI performance. The cost of these improvements is storage: 143Mb for LM-replication-3 and 242Mb for LM-replication-6. It was difficult with our tools to test a caching strategy where only a portion of a database (e.g., the most popular documents in the database) are stored at the clients in this preliminary study. But we expect that future tests will show significant results can still be obtained from such an approach.

6 Conclusion

In this paper, we examined for the first time collection selection when servers have a heterogeneous set of data and heterogeneous network performance to clients. We have shown that methods that consider network selection or IR selection in isolation are not easily composed with good results. This is because servers that contain relevant data have no correlation with good network performance.

Our technique using language models overcomes this incompatibility by making local replicas of slow servers that contain relevant documents. We have also examined different strategies for finding which databases are best to replicate. In sum, servers that are most relevant but can be predicted to not be among the faster servers should be cached or replicated.

In our experiments with TREC 2, our technique is capable of providing an average precision of 13.2%, which is comparable to CORI's average precision of 15–16%. For storage costs of 143Mb (16% of the entire database), network performance was improved to 62–67% of CORI's performance; and improves to 30–37% of CORI's performance for storage of 242 Mb (33% of the entire database). Our future work will be to extend these results to consider storing only popular portions of relevant databases to reduce storage costs while maintaining networking and IR performance.

Acknowledgments

This work could not have been completed without the generous assistance of Margaret Connell and the Center for Intelligent Information Retrieval in operating and modifying the INQUERY system.

References

[1] J. Callan. Distributed information retrieval. In W. Bruce Croft, editor, *Advances in Information Retrieval: Recent Research from the CIIR*, chapter 5, pages 127–150. Kluwer Academic Publishers, 2000.

[2] J. Callan and M Connel. Query-Based Sampling of Text Databases. *ACM Transactions on Information Systems*, 19(2):97–130, 2001.

[3] J.P. Callan, W.B.Croft, and S.M.Harding. The INQUERY retrieval system. In *the 3rd International Conference on Database and Expert System Applications*, Sept 1992.

[4] N. Craswell, D. Hawking, and P. Thistlewaite. Merging results from isolated search engines. In *Proc. of the Tenth Australasian Database Conf.*, pages 189–200, 1999.

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

[6] L. Gravano, C. Chang, H. Garcia-Molina, and A. Paepcke. Starts: Stanford proposal for internet meta-searching. In *Proc. of the ACM-SIGMOD Int'l Conference on Management of Data*, 1997.

[7] L. Gravano and H. Garcia-Molina. Generalizing gloss to vector-space databases and broker hierarchies. In *Proceedings of the 21st International Conference on Very Large Databases (VLDB)*, 1995.

[8] K. M. Hanna, N. Natarajan, and B.N. Levine. Evaluation of a Novel Two-Step Server Selection Metric. In *IEEE ICNP 2001*, November 2001.

[9] K. M. Hanna, N. Natarajan, and B.N. Levine. Network trace logs. http://signl.cs.umass.edu, January 2001.

[10] J.P.Callan, Z.Lu, and W.B.Croft. Searching Distributed Collections with Inference Network. In *the 18th International ACM SIGIR Conference on Research and Development in Information Retrieval(SIGIR 95)*, pages 21–29, 1995.

[11] Z. Lu and K. McKinley. Partial Replica Selection Based on Relevance for Information Retrieval. In *22nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 99)*, pages 97–104, 1999.

[12] C. Luo and J. Callan. Using sampled data and regression to merge search engine results. In *Proc. of the 22nd ACM SIGIR conf. on Research and Developement in Information Retrieval*, 2002.

[13] R. Manmatha, T. Rath, and F. Feng. Modeling score distributions for combining the outputs of search engines. In *the Proc. of the 24th ACM SIGIR conf. on Research and Developement in Information Retrieval*, pages 267–275, Sept 2001.

[14] J. Ponte and W.B. Croft. A Language Modeling Approach to Information Retrieval. In *SIGIR 98*, pages 275–281, 1998.

[15] C. L. Viles and J. C. French. Dissemination of collection wide information in a distributed information retrieval system. In *the Proc. of the 18th ACM SIGIR conf. on Research and Developement in Information Retrieval*, 1995.

[16] B. Yuwono and D. Lee. Server ranking for distributed text retrieval systems on internet. In *Proc. of the Int. Conf. on Database Systems for Adv. Applications*, pages 41–49, 1997.