Performance Evaluation of a Distributed Architecture for Information Retrieval *

Brendon Cahoon Kathryn S. McKinley

Department of Computer Science, University of Massachusetts, Amherst, MA 01003, {cahoon,mckinley}@cs.umass.edu

Abstract

Information explosion across the Internet and elsewhere offers access to an increasing number of document collections. In order for users to effectively access these collections, information retrieval (IR) systems must provide coordinated, concurrent, and distributed access. In this paper, we describe a fully functional distributed IR system based on the Inquery unified IR system. To refine this prototype, we implement a flexible simulation model that analyzes performance issues given a wide variety of system parameters and configurations. We present a series of experiments that measure response time, system utilization, and identify bottlenecks. We vary numerous system parameters, such as the number of users, text collections, terms per query, and workload to generalize our results for other distributed IR systems. Based on our initial results, we recommend simple changes to the prototype and evaluate the changes using the simulator. Because of the significant resource demands of information retrieval, it is not difficult to generate workloads that overwhelm system resources regardless of the architecture. However under some realistic workloads, we demonstrate system organizations for which response time gracefully degrades as the workload increases and performance scales with the number of processors. This scalable architecture includes a surprisingly small number of brokers through which a large number of clients and servers communicate.

1 Introduction

The increasing numbers of large, unstructured text collections require full-text information retrieval (IR) systems in order for users to access them effectively. Current systems typically only allow users to connect to a single database either locally or perhaps on another machine. A distributed IR system should be able to provide multiple users with concurrent, efficient access to multiple text collections located on remote sites. Since the documents in unstructured text collections are independent, IR systems are ideal applications to distribute across a network of workstations. However, the high resource demands of IR systems limit their performance, especially as the number of users, as well as the size and number of text collections increase. Distributed computing offers a solution to these problems. Systems based on distributed architectures use resources more efficiently and in parallel by spreading work across a network of workstations.

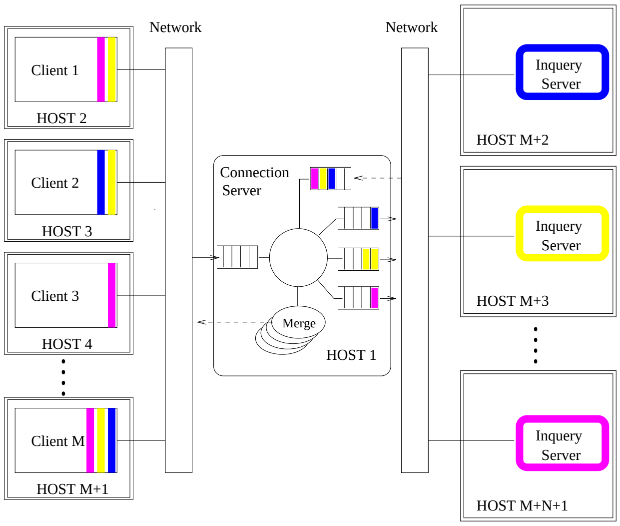

Figure 1: Our Distributed Information Retrieval System

The focus of this paper is to design appropriate distributed information retrieval architectures by analyzing the performance of potential systems under a variety of workloads. We begin with a prototype implementation of a distributed information retrieval system using Inquery; an inference network, full-text information retrieval model[4]. Our system adopts a variation of the client-server paradigm that consists of *clients* connected to *Inquery server* retrieval engines through a *connection server*, a central administration broker, as illustrated in Figure 1. In the original Inquery system (not distributed), clients specify an Inquery server, connect to it, interact with it, and finally disconnect. In the distributed system, clients search multiple databases simultaneously. To build our prototype, we made the fewest possible changes to the underlying software. We therefore began with a single connection server which maintains a list of available collections and their locations and brokers all of the clients' retrieval requests and Inquery server responses. We describe this distributed system in detail in Section 2. We measure the system and use it to drive a simulator in which we can easily move and replicate functionality to investigate alternative architectures for our distributed system. Section 3 presents this simulation model.

The simulation model is parameterized by system features such as the number of users and text collections, average query length, I/O and CPU demands, network latency, and the time to merge results from different IR servers. This model allows us to investigate systems that vary from our implementation. We measure system response time,

* This material is based on work supported by the National Science Foundation, Library of Congress, and Department of Commerce under cooperative agreement number EEC-9209623. Any opinions, findings, and conclusions or recommendations expressed in this material are the authors and do not necessarily reflect those of the sponsor.

To appear in the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Zurich, Switzerland, August, 1996

throughput, and resource utilization for a variety of configurations. During our investigation we identify potential bottlenecks and study the effects of various architectures and parameters. Our goal is to use resources efficiently by maximizing parallelism and ensuring scalability. We also maintain the effectiveness, in terms of recall and precision [3], of a stand-alone IR system.

The results show that the implemented system performs well for small configurations when the Inquery servers process queries quickly. However, as the size of the system increases, bottlenecks begin to degrade performance. We show that we can alleviate some of the bottlenecks by adding additional brokers to manage the clients and Inquery servers. Section 7 compares our work to previous work and Section 8 summarizes our results.

2 A Distributed Information Retrieval System

This section describes the implementation of our distributed IR system. It describes the functionality and interaction between the clients, the connection server, and the Inquery servers in terms of the IR commands issued by the clients.

2.1 Clients

The clients are lightweight processes that provide a user interface to the retrieval system. Clients initiate all the work in the system by sending commands to the connection server, as illustrated in Figure 1. Clients first connect to the connection server. A client can request a list of collections from the connection server or remember those it used previously. Clients specify a list of text collections to search with each IR command. The clients can issue the entire range of IR commands, but in this paper, we focus on *query*, *summary*, and *document* commands.

Query commands consists of a set of words or phrases (terms) and a set of collection identifiers on which to perform the queries. Query responses consist of a list of n document identifiers ranked by belief values that estimate the probability that the document satisfies the information need.

Summary commands consist of a set of document identifiers and their collection identifiers. Summary responses include the document title and the first few sentences of the document.

Document commands consist of a document and collection identifier. The response includes the complete text of the document.

A client issues a command and waits for the connection server to return the results before it issues another command. Users issue queries and document commands. A client automatically issues the first summary command when it receives a query response. A client issues additional summary commands at the user's request.

2.2 Connection Server

The clients and Inquery servers communicate via the connection server. The connection server is also a lightweight process that keeps track of all the Inquery servers, outstanding client requests, and organizes responses from Inquery servers. The connection server continuously checks for incoming messages from clients and Inquery servers. The connection server handles outstanding requests from multiple clients. We briefly describe the processing that the connection server performs to handle each request below.

Inquery servers add themselves to the system by sending a message to the connection server. Clients send their commands to the connection server which forwards them to the appropriate Inquery servers. The connection server maintains a queue of outstanding requests for each of the Inquery servers as illustrated in Figure 1. If an Inquery server is currently processing another command, the connection server inserts the command onto a queue. When the connection server receives an outstanding response from an Inquery server, it forwards the next command on the corresponding queue to the Inquery server.

The connection server maintains intermediate responses from the Inquery servers until it receives all the responses. It then sends the final result to the appropriate client. For a summary command, the connection server simply orders the list of responses in the same order as the request. For a query command, each Inquery server sends its top n responses back to the connection server. The connection server maintains a sorted list of the overall top n entries until all the Inquery servers respond. The connection server merges new results with the existing sorted list. We assume the relative rankings between documents in independent collections are comparable, but this assumption is clearly tenuous. For example, one collection may be irrelevant to a particular query, but if the user includes it, the overall response may still include its top ranked responses. Other research is investigating techniques to automatically select appropriate collections with respect to specific queries [5, 14].

The connection server does not maintain intermediate results for document retrieval commands; it simply forwards a document as soon as the Inquery server sends it.

2.3 Inquery Servers

The Inquery server uses the Inquery retrieval engine to provide IR services. The Inquery system is a probabilistic retrieval model that is based upon a Bayesian inference network [4]. Inquery accepts natural language or structured queries. For query operations, the system outputs a list of documents ranked by relevance. Internally, the system stores the text collections as an inverted file. Previous work demonstrates that Inquery is an effective retrieval system for large, full-text databases [3].

3 Simulation Model

In this section, we present a simulation model for exploring distributed IR system architectures. Simulation techniques provide an effective and flexible platform for analyzing large and complex distributed systems. We can quickly change the system configuration, run experiments, and analyze results without making numerous changes to large amounts of code. Furthermore, simulation models allow us to easily define very large systems and examine their performance in a controlled environment.

To implement the simulator, we use YACSIM, a process oriented discrete event simulation language [10]. YACSIM contains a set of data structures and library routines that manage user created processes and resources. Its process oriented nature enables the structure of the simulator to closely reflect the actual system.

Our simulation model is simple, yet contains enough details to accurately represent the important features of the system. We model the basic architecture and functionality described in Section 2 and illustrated in Figure 1. The model is driven by empirical measurements obtained from our prototype.

A user configures a simulation by defining the architecture of the distributed information retrieval system using a simple command language. A configuration file contains the commands that the simulator reads at start-up time.

3.1 System Measurements

To accurately model an IR system, we analyzed the distributed Inquery system and measured the resources used for each operation. We focused on CPU, disk, and network resources. The simulation does not model memory and cache effects. Empirical measurements rather than an analytical model drive the activities performed in the simulator. The simulator is driven by the following measurements: query evaluation time, document/summary retrieval time, connection server time, network latency, and time to merge results. We obtained measurements of the prototype system using Inquery version 2.1 running on a DECsystem-5000/240 (MIPS R3000 clocked at 40 MHz) workstation running Ultrix V4.2A (Rev. 47) with 64 MB of memory and 300 MB of swap space.

We examined several different text collections and query sets to obtain system measurements. We examined TIPSTER 1, a large heterogeneous collection of full-text articles and abstracts [8], a database containing the Congressional Record for the 103rd Congress [6], and a small collection of abstracts from the *Communications of the ACM* [7].

The simulator uses a simple, yet accurate model to represent query evaluation time. We found that evaluation time is very strongly related to the number of terms in the query and the frequency of each of the terms. On the TIPSTER 1 query set, the correlation between query length and query evaluation time is .96. The correlation is .95 for query term frequency. We used the TIPSTER 1 collection to measure the evaluation time for terms of different frequencies. The time to evaluate a single term ranges from 0.5 seconds for a term that appears only once to 17 seconds for a term that appears 554,658 times (the maximum term frequency in TIPSTER 1). We divide the evaluation time into CPU and disk access time. The simulator computes the evaluation time for a query by adding the evaluation times of the individual terms in the query.

The simulator represents the document retrieval time for an Inquery server as a constant value. We used the TIPSTER 1 collection to measure the the amount of time it takes the Inquery system to return documents of different sizes. The document sizes range from 0.24 KB to 12 KB. We found that the retrieval time is highly variable and does not correlate to the size of the document. The value that the simulator uses is 0.31 seconds which is the average document retrieval time for 2000 randomly selected documents from the TIPSTER 1 collection. We divide the evaluation time into CPU and disk access time. The simulator uses the document retrieval time to compute the summary information retrieval time. Inquery retrieves a complete document to obtain the summary information, but it only returns the summary part.

The connection server time consists of two values; the time to access the connection server and the time to merge results. The simulator uses a constant value to represent the connection server processing time which we obtained from measuring the actual connection server. The value is 0.1 seconds. The time to merge query results depends upon the number of answers that an Inquery server returns. A list with 1000 results takes 17.9 milliseconds.

We represent network time as the sender overhead, receiver overhead, and network latency. The sender and receiver overhead is the CPU processing time for adding and removing a message from the network. The network latency is the amount of time the message spends on the network itself. These times depend upon the size of the message and the bandwidth of the network. We obtained the sender and receiver overhead times by measuring messages sent between two DECsystem-5000 workstations connected by a 10Mbps Ethernet.

Parameter		Values		
Clients	C	1 4 8 32 64 128 256		
Inquery Servers	IS	1 4 8 32 64 128		
Terms per Query neg. binomial dist.	TPQ	2	12	27
Query Term Freq. dist. from queries	QTF	Obs. Dist.	Low Skew	High Skew
Answers Returned constant values	AR	100	1000	
Think Time/Summary normal dist.	TTS	15	30	90
Think Time/Document normal dist.	TTD	30	60	180
Documents Retrieved range of values	DR	1–5	8–12	15–20
Summary Operations range of values	SO	1–5	8–12	15–20

Table 1: Experiment Parameters

3.2 Validation

We validated the simulator against the actual implementation using a configuration consisting of single client, Inquery server, and connection server. We placed each of the components on a separate host. We used the query sets and text collections from TIPSTER 1 and the Congressional Record. We found that our simulator runs within $\pm 10\%$ of the actual system. The simulator tends to overestimate evaluation times for small queries and underestimate evaluation times for large queries. In general, the simulator follows the same trend as the actual system; larger queries take longer to evaluate.

3.3 Experiment Parameters

Based on our measurements and our system architecture, we parameterized the simulator as summarized in Table 1. Table 1 presents the parameters, their values, and abbreviations for our experiments. Below, we briefly describe each parameter.

Number of Clients/Inquery Servers (C/IS). We experiment with both small and large system configurations. Measuring the effect of increasing the number of clients and Inquery servers provides insight into identifying bottlenecks and understanding system utilization and scalability.

Terms per Query (TPQ). We use three different average query lengths in our experiments obtained from actual query sets as described in Section 3.1. We use a negative binomial distribution that matches the observed distribution of query lengths from our query sets.

Distribution of Terms in Queries (QTF). Researchers do not agree on a commonly accepted distribution for term frequencies in queries [15]. We examined our query sets to determine an appropriate distribution. The query term frequency distributions for the query sets are similar but the distributions are complex and do not closely match a mathematical function. In our experiments we use the distribution of query term frequencies from the TIPSTER 1 query set. We call this our *observed* query term frequency distribution. We also use a distribution that is skewed towards terms that occur less frequently and a distribution that is skewed towards terms that occur more frequently.

Number of Documents that Match Query (AR). The IR system returns a sorted list of matching documents to the clients. The number of documents returned affects network traffic and processing by the connection servers.

Think Time (TT). In the simulated workload, clients "think" after receiving summary information and documents. This value accounts for the time, in seconds, that users use to look at the results of their requests. Think time can be large in comparison to the time the system takes to perform requests. Since we do not have statistics that represent actual user think times we use a range of values. Further reducing think time and adding clients have similar effects on performance in this system.

Document Retrieval/Summary Information (DR/SO). We vary the number of summary and document retrieval operations after each query. The entries in Table 1 represent a range of values from which the simulator randomly chooses values. A single summary information operation retrieves entries for 15 documents. The simulator generates different document lengths from a distribution that matches the distribution of document lengths in the TIPSTER 1 collection. The summary and document size determine the time to send it across the network.

We discuss these parameters and our reasons for choosing specific values in greater depth in a technical report [2].

Unless otherwise stated, the clients, connection server, and Inquery servers operate as described in Section 2. We allocate each of the basic components in the distributed system to its own host. Each host contains its own processor, memory, and secondary storage. A local area network with a bandwidth of 10Mbps connects the machines. Each of the Inquery servers maintains a 1 Gigabyte database (except in the first experiment in Section 4.1 where a single 1 GB database is distributed).

3.4 Workload

The workload consists of the the basic retrieval operations described in Section 2: query evaluation, obtaining summary information, and document retrieval. The simulator does not model more complicated functions such as relevance feedback. In the simulator, clients repeatedly perform the following **transaction sequence**: *evaluate a query, obtain summary information of top ranking documents, think, retrieve documents, think.*

The simulator only models natural language queries and does not perform structured query operations such as phrase and proximity operators. The simulator varies the specific operations for each client and during each sequence. For example, the model generates new queries and retrieves different documents for each iteration.

3.4.1 Simulation Output

For each simulation configuration of parameters, we measure the system performance in terms of average query response time, summary response time, document response time, connection server utilization, queue lengths, network utilization, Inquery server utilization, *etc.* Due to space constraints, we only present graphs of the results for average Inquery server utilization, connection server utilization, and response time for a *transaction sequence*. For each series of graphs, we display the corresponding values of the parameters listed in Table 1. We refer the interested reader to our technical report for more results [2].

4 Experiments and Results

In this section, we present the results from four sets of experiments. Two of the experiments use the prototype architecture that we implemented. In the first experiment, we study the effect of equally distributing a single database among each of the Inquery servers. In the second, each of the Inquery servers maintains a different database and the clients broadcast queries to a subset of the available databases.

For small, realistic queries, we demonstrate several architectures that scale with the number of processors and degrade gracefully as the number clients (work) increases. Our results illustrate that the system can achieve good performance under varying conditions if we can maintain a balance between connection server and Inquery server utilization. However, we see that system performance deteriorates rapidly when either the connection server or Inquery servers become over utilized.

We then investigate several changes to the basic architecture to eliminate bottlenecks at the connection server. To introduce more parallelism, we first add connection servers. We test configurations using two and four connection servers and find this is sufficient to relieve the connection server bottleneck. We also test moving the response merging from the connection server to the clients, but this change does not improve performance because the increased number of messages the connection server must send is just as costly as merging short lists.

4.1 Distributing a Single Text Collection

In this section, we examine the performance of the system when we divide a single 1 GB text collection among all the Inquery Servers. The size of the text collection managed by each Inquery server depends upon the number of Inquery servers. For example in a system with 64 Inquery servers, each collection is 16 MB. This architecture models a distributed system that maintains a single large database, but exploits parallelism by operating independently on each portion. In this configuration, the total amount of work done by the system for each client is fixed. Each client connects to all the Inquery servers.

4.1.1 Discussion of Results

In Figures 2–7, we present and compare the average transaction time, connection server utilization, and Inquery server utilization for small queries and large queries. In all figures, we display the number of clients, 1 to 256, on the x-axis. In Figures 2 and 5, we display the number of seconds on the y-axis. In Figures 3, 4, 6, and 7, we display the percent of process utilization time on the y-axis.

Small Queries (TPQ=2)

Figure 2 illustrates that for up to 8 Inquery servers, adding Inquery servers improves the average transaction time (In this experiment, 1 Inquery server and 128 have the same performance). Going from 1 to 8 Inquery servers improves performance for 256 clients by a factor of 4.66. However, when the system contains more that 8 Inquery servers, the performance degrades because the connection server becomes over utilized.

The performance improvement is due to a couple of factors. First, as we increase Inquery servers, the size of each

Small Queries

TPQ	QTF	AR	TTS	TTD	DR	SO
2	**Obs.**	1000	30	60	1–5	1–5

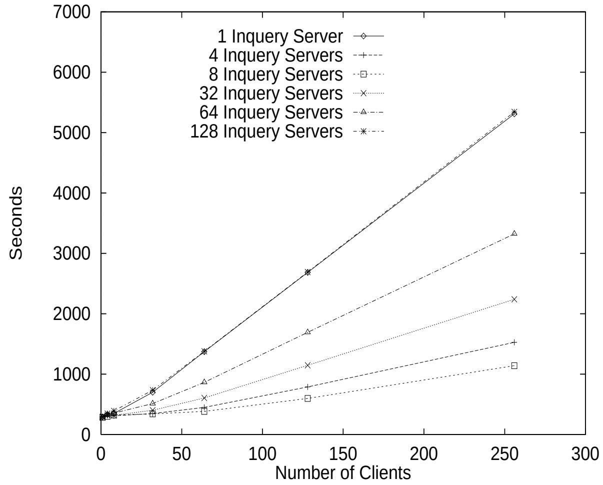

Figure 2: Average Transaction Sequence Time

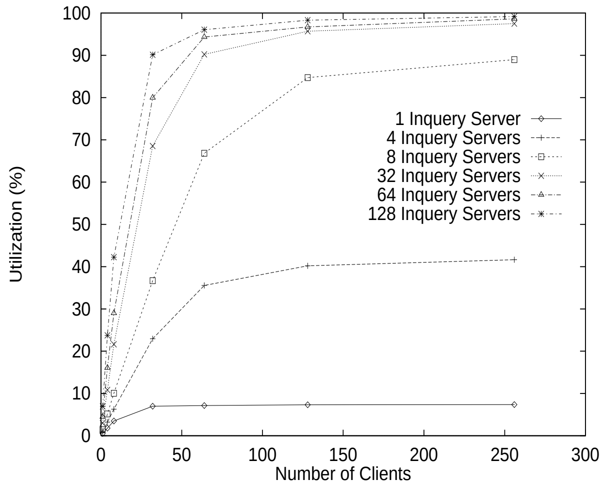

Figure 3: Connection Server Utilization

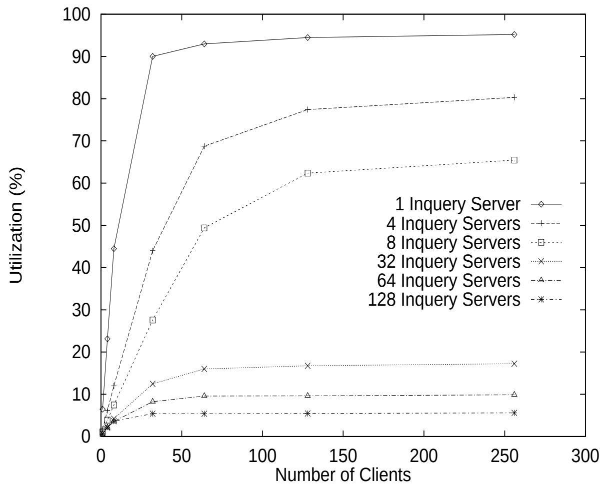

Figure 4: Average Inquery Server Utilization

Large Queries

TPQ	QTF	AR	TTS	TTD	DR	SO
27	**Obs.**	1000	30	60	1–5	1–5

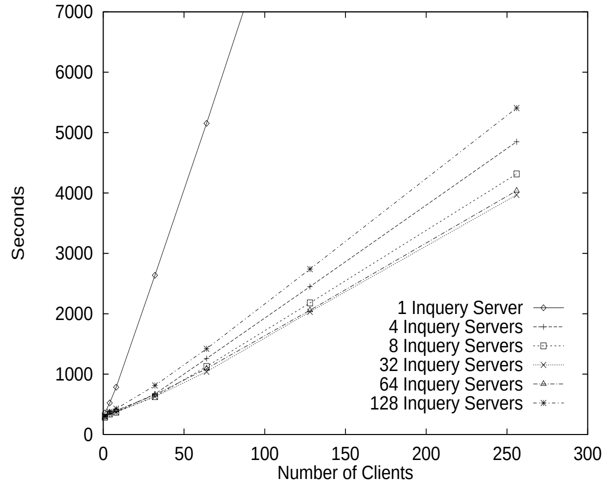

Figure 5: Average Transaction Sequence Time

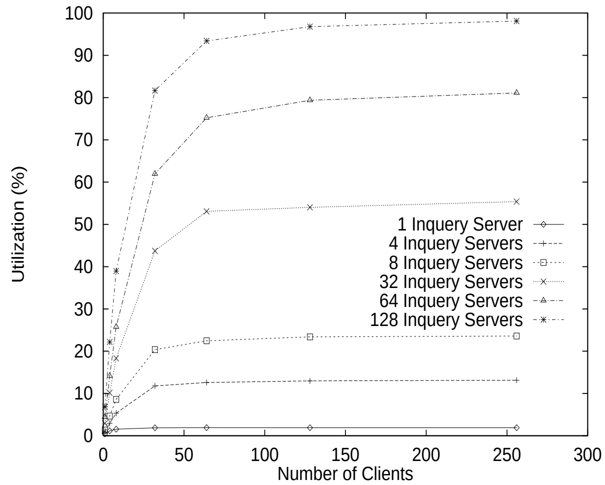

Figure 6: Connection Server Utilization

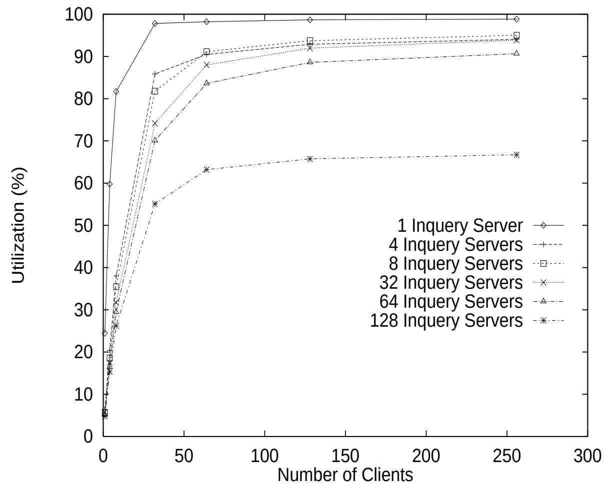

Figure 7: Average Inquery Server Utilization

database decreases which improves query evaluation time. For example, the system is able to search two 500 MB databases in parallel quicker than searching a single 1 GB database. More detailed measurements reveal that some of the improvement stems from increased parallelism during summary retrieval. Recall that a single summary information operation retrieves 15 documents. A system with one Inquery server contains all the documents on the same machine. However, in a system with multiple Inquery servers the documents are distributed among the available Inquery servers. The 15 summary entries may reside on different Inquery servers resulting in a parallel access of the summary information. In the best case, each of the 15 entries are located on different Inquery servers.

As Figure 2 shows, the system achieves the best performance with 8 Inquery servers. The average transaction sequence time degrades very slowly as we increase the number of clients. For example, for 8 Inquery servers as we increase the number of clients from by a factor of 64 (from 1 to 64), system response time degrades by a factor of 1.35. However, the jump between 1 and 256 clients degrades performance by a less acceptable factor of 4. With this configuration, the system achieves a good balance between connection server and Inquery server utilization for 8 Inquery servers.

The performance degradation for 32 or more Inquery servers occurs because the connection server becomes a bottleneck. We see in Figure 3 that the connection server utilization is very high for 32 to 128 Inquery servers. When the utilization exceeds 85% the connection server does not process messages as quickly as the clients and Inquery servers send them. For example, the connection server's incoming queue length for utilization values greater than 85% exceeds 20 messages. Our results indicate that the connection server effectively processes up to 8 requests per second. After this threshold, the connection server becomes over utilized.

The bottleneck in the connection server explains the low utilization of the Inquery servers (Figure 4). The Inquery servers remain idle when the connection server is too busy to forward outstanding requests.

Large Queries (TPQ=27)

For large queries, the performance of 4 to 128 Inquery servers is very similar and degrades rapidly as the number of clients increases. However, Figure 5 shows small improvements between 4 and 64 Inquery servers. Performance deteriorates when a single database is distributed over 128 Inquery servers. For 128 Inquery servers, extremely high utilization of the connection server and the Inquery servers causes this severe degradation.

In comparison with the results for small queries (Figure 2), the system response time does not scale well as the number of clients increases. For large queries, the Inquery servers quickly become a bottleneck. As the number of clients increases, the system places greater demands on the Inquery servers which in turn increases in the average transaction time. Contrast this result with short queries where the Inquery server is only highly utilized when the entire database resides on a single Inquery server. On a configuration with 8 Inquery servers, query evaluation using large queries takes 9 times longer than using small queries.

The system only achieves scalable performance when the utilization of the connection server and the Inquery servers remains below 80%.

4.2 Multiple Text Collections

In this section, we measure the performance of the distributed IR system that maintains multiple text collections. In this configuration, each client selects a random subset of the available collections to search for the duration of a simulation. On average, a client therefore searches half of the available collections. Thus, the workload increases both as a function of the number of Inquery servers and the number of clients. This workload mimics the scenario when the connection server is able to automatically select an appropriate subset of the available collections to search. It also is accurate when the user is given a selection of databases and then chooses some subset to search.

4.2.1 Discussion of Results

Figures 8–13 present and contrast average transaction time, connection server utilization, and Inquery server utilization for small queries (Figures 8–10) and large queries (Figures 11–13). For the scaled workload, we see that query size has an even more dramatic impact on system performance. Two different effects are evident in these graphs. In Figures 8–10, degradations occur when the connection server becomes highly utilized. In contrast, Figures 11–13 illustrate the more dramatic effect on performance when the Inquery servers are the bottleneck.

Small Queries (TPQ=2)

Figure 8 illustrates that until we reach 32 Inquery servers, the average transaction time improves as the number of Inquery servers, and therefore the workload increases. When the number of Inquery servers doubles, a client potentially searches twice as much information. However, for more than 64 Inquery servers, the average transaction time decreases. Again, our more detailed measurements reveal that the performance improvement is due to increased parallelism during the summary commands (see Small Queries in Section 4.1.1).

In Figure 9, we see a large increase in connection server utilization as the size of the distributed system grows. At the same time, Figure 10 shows the Inquery server utilization decreases as we add Inquery servers. It is apparent that as the system size increases the connection server becomes a bottleneck causing performance to degrade. We confirmed this result by measuring the size of the message queue for the connection server. We found that the queue is empty for 1, 4, and 8 Inquery servers. For 64 and 128 Inquery servers, the queue length becomes very long and approaches 90 entries when the system contains 128 clients.

Large Queries (TPQ=27)

Figure 11 illustrates that the performance of the distributed system does not scale for large queries. The average transaction time almost doubles as the number of number of Inquery servers doubles. The reason for the poor performance is that the Inquery servers cause a bottleneck in the system (see Figure 13). The time in the Inquery servers accounts for the majority of the transaction time. Note that these values represent the average utilization over all Inquery servers. Since each client connects to a subset of the available Inquery servers, it is difficult to reach 100% utilization.

In Figure 12, we see that utilization in the connection server is very low. Since query evaluation dominates processing time, the connection server remains idle most of the time.

4.3 Multiple Connection Servers

In the experiments in Sections 4.1 and 4.2, the system scales for small queries up to a certain point; if we add too many Inquery servers, the performance degrades. As we previously mentioned, the problem is that the connection server

Multiple Text Collections

Small Queries

TPQ	QTF	AR	TTS	TTD	DR	SO
2	**Obs.**	1000	30	60	1–5	1–5

Large Queries

TPQ	QTF	AR	TTS	TTD	DR	SO
27	**Obs.**	1000	30	60	1–5	1–5

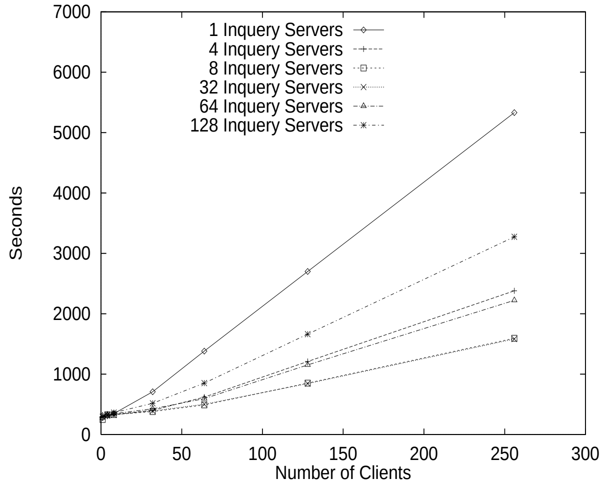

Figure 8: Average Transaction Sequence Time

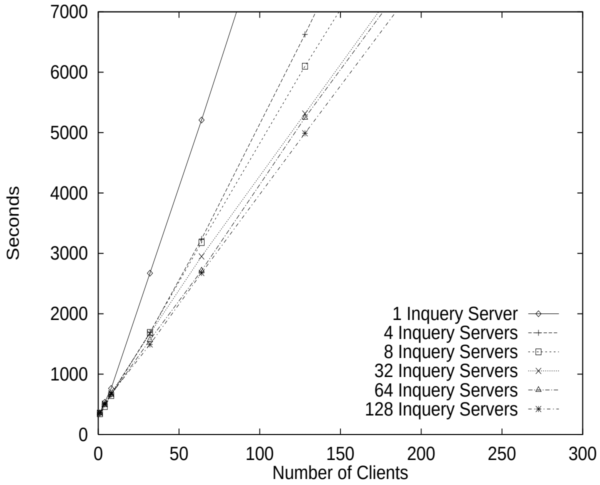

Figure 11: Average Transaction Sequence Time

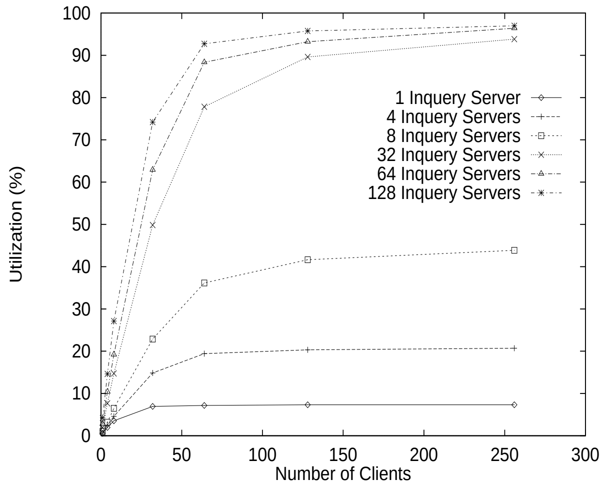

Figure 9: Connection Server Utilization

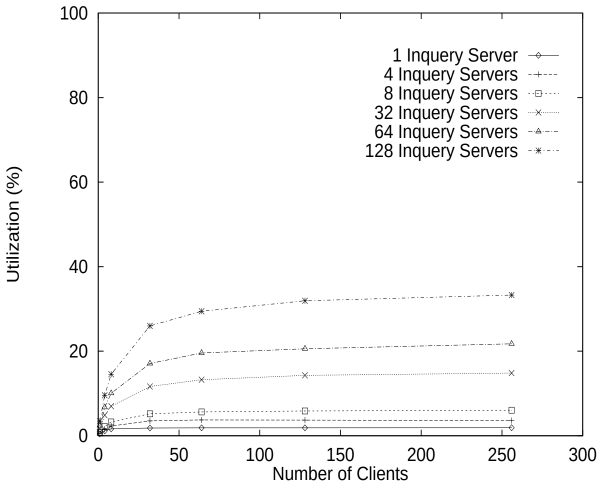

Figure 12: Connection Server Utilization

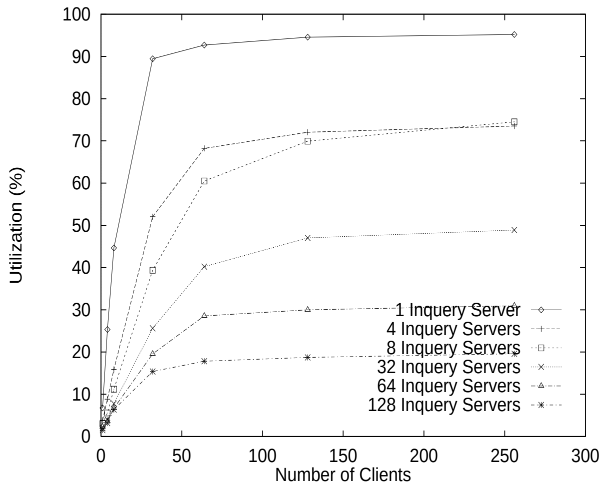

Figure 10: Average Inquery Server Utilization

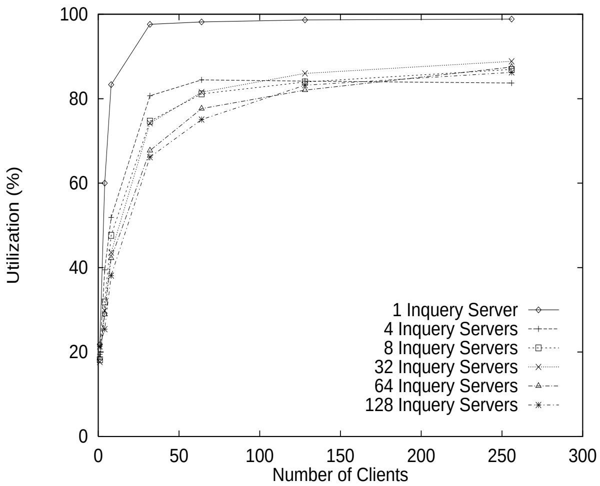

Figure 13: Average Inquery Server Utilization

becomes a bottleneck. To relieve this bottleneck we analyze the performance of a system with 2 and 4 connection servers. Adding additional connection servers reduces the average utilization of each connection server, and improves performance for small queries.

In this system, the clients divide evenly among the connection servers and each connection server maintains a link to all the Inquery servers. In the basic architecture, the connection server maintains a queue of outstanding requests for each of the Inquery servers. If an Inquery server is busy, the connection server adds the request to the queue. In the multiple connection server system, the connection server immediately forwards requests to the Inquery servers. Each of the Inquery servers instead maintains its own queue of outstanding requests. Moving the queues to the Inquery servers does not significantly effect performance.

The workload in this system is the same as in Section 4.2: each Inquery server contains a 1 GB database and the workload scales with the number of clients and Inquery servers.

4.3.1 Two Connection Servers

Figure 14 shows the average transaction sequence time for a system with two connection servers. In this test, we used small queries. Compare these results to those in Figure 8 (all of the y-axes for average transaction times are on the same scale). As with the 1 connection server architecture, the best performance occurs with 32 Inquery servers.

We see that the system performs better as the number of clients and Inquery servers increases. For combinations of 1 to 8 clients and Inquery servers, there is not a significant difference in performance. However, for all other combinations, we see that there is an improvement in performance. We get a speedup of 1.94 over the single connection server model for the configuration using 256 clients and 128 Inquery servers.

4.3.2 Four Connection Servers

Figure 15 shows the average transaction sequence time for a system with four connection servers. Again, for this test, we used small queries. The additional connection servers provide even greater improvements in performance for the larger configurations. We see that the best performance occurs for 32, 64, and 128 Inquery servers. This result is quite different from Figure 8 in which the performance begins to degrade after 32 Inquery servers. The most interesting effect of adding four connection servers is that the system scales very well for large configurations. We see this effect in Figure 15 where the average transaction response time of 32 to 128 Inquery servers remains nearly the same for all client configurations.

Discussion

Adding additional connection servers improves performance in large systems when users evaluate small queries. In the single connection server architecture, the connection server quickly becomes saturated with requests limiting performance. Adding connection servers distributes this work and improves performance. However, when the Inquery servers are the bottleneck in the system, as in Figure 11 with large queries, additional connection servers do not improve performance.

5 Moving Functionality

Another way to reduce the amount of processing that occurs in the connection server is to move the merging functionality to the clients. Currently, the connection server is responsible for collecting and merging intermediate results before sending the final answer to the client. We test this architecture using workloads that cause high contention for the connection server. We configured the simulation to match the experiments in Section 4.2 (Figures 8–10). Our results show that moving the merging functionality *does not* improve the average transaction sequence time. The reason is that the connection server sends more messages to the client. The extra processing for sending more messages is approximately the same as for merging small lists.

6 Future Work

For large queries or extremely high workloads, the Inquery servers do not provide reasonable response times. To alleviate this problem, we can hope to follow the technology curve to get some improvements in performance. Processors are getting faster and the underlying information retrieval technology is likely to get quicker as well. Other, more immediate solutions we will investigate are replicating the collections, shared-memory multiprocessing, and multithreaded servers. Replication will require additional functionality in the connection server to coordinate and load balance access. Based on our small query results, our architecture should be able to achieve good performance with this solution. Using a multiprocessor instead should provide parallel access without paying the resource costs of replication. However, the high I/O demands of information retrieval may overwhelm a shared-memory multiprocessor. We are investigating multithreading for the connection servers and Inquery servers.

7 Related Work

Our research combines and extends previous work in distributed IR since we model and analyze a complete system architecture. Although others have examined some of the issues, no one has considered the entire system under a variety of realistic conditions. We experiment with very large text collections; up to 128 GB of data. Prior work has not examined such large systems. We also base our distributed system on a proven, effective retrieval engine.

Burkowski reports on a simulation study which measures the retrieval performance of a distributed IR system [1]. The experiments explore two strategies for distributing a fixed workload across a small number number of servers. This work is the most closely related to our work, but differs in several ways. He assumes a worst case workload where each user broadcasts queries to all servers without any think time. We experiment with larger distributed configurations, we vary the number of clients, and use a more realistic user workload.

Other researchers have investigated various data partitioning schemes for distributed IR systems [12, 13, 11, 9]. We address this issue in the experiments in Section 4.1. Although we only consider one partitioning scheme, we improve upon their results in several ways. Our experiments include results for both small and large configurations. Previous research has investigated only small configurations. Also, we use an existing retrieval model that has proven to be very effective. We investigate changes to the architecture that do not involve changes to the underlying retrieval model. Several of the partitioning schemes mentioned in the previous work require changes to the retrieval model which possibly affects retrieval effectiveness.

8 Summary

To keep pace with the increasing amounts of online information, the performance of information retrieval systems must improve. In this paper, we present an implementation of a

TPQ	QTF	AR	TTS	TTD	DR	SO
2	Obs.	1000	30	60	1–5	1–5

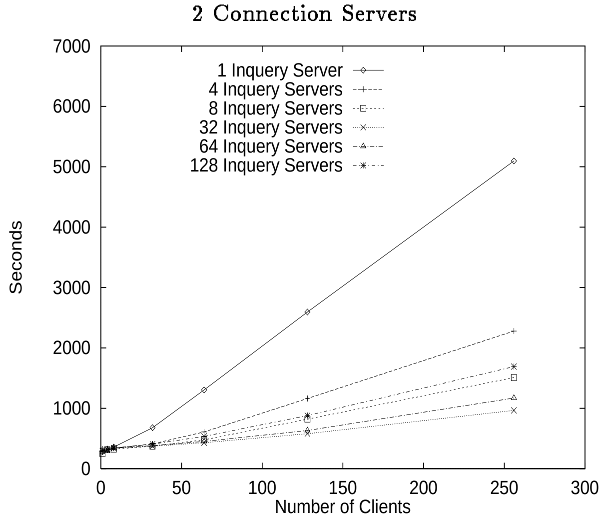

Figure 14: Average Transaction Sequence Time

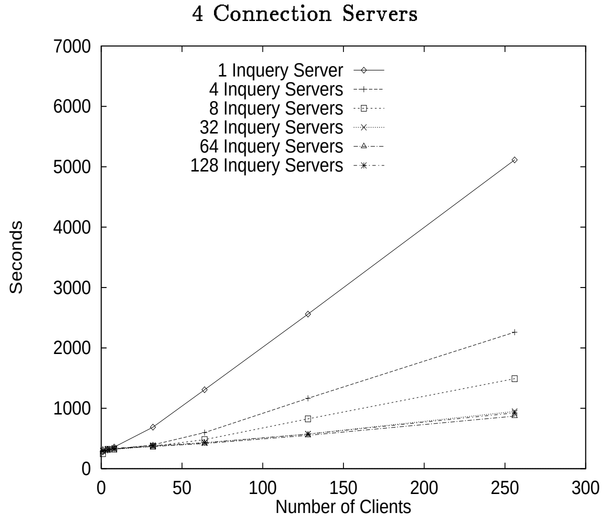

Figure 15: Average Transaction Sequence Time

distributed IR system to achieve coordinated, concurrent, and scalable access. We develop a flexible simulation model to examine the performance of the prototype using a wide variety of parameters, workloads, and configurations. We present results that measure system response time, utilization, and identify bottlenecks.

Our results show that our architecture provides scalable performance when clients enter small queries. Small queries are a realistic workload, since several studies of existing IR systems demonstrate that users tend to use small queries [6]. By adding a small number of connection servers to coordinate a large number of clients and Inquery servers the system can maintain scalable performance at higher workloads. When the system bottleneck is the Inquery server, as for large queries, it is more difficult to achieve reasonable performance. Based on the performance of the Inquery servers for short queries, we believe our future work will show that replicating text collections will mitigate much of the competition for the Inquery servers.

Acknowledgments

We would like to thank Bob Cook and Kathleen Dibella for help with the development of the prototype system. We would also like to thank Jamie Callan and Bruce Croft for their contributions to this work.

References

[1] 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.

[2] B. Cahoon and K. S. McKinley. Performance analysis of distributed information retrieval architectures. Technical Report UM-CS-1995-054, Department of Computer Science, University of Massachusetts, Amherst, June 1995.

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

[4] 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.

[5] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th International Conference on Research and Development in Information Retrieval*, Seattle, WA, July 1995.

[6] 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.

[7] E. A. Fox. Characterization of two new experimental collections in computer and information science containing textual and bibliographic concepts. Technical Report 83-561, Cornell University, Ithaca, NY, September 1983.

[8] D. Harman, editor. *The First Text REtrieval Conference (TREC1)*. National Institute of Standards and Technology Special Publication 200-217, Gaithersburg, MD, 1992.

[9] B-S. Jeong and E. Omiecinski. Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):142–153, February 1995.

[10] J. R. Jump. *YACSIM Reference Manual*. Rice University, version 2.1.1 edition, 1993.

[11] 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.

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

[13] A. Tomasic and H. Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, San Diego, CA, 1993.

[14] E. M. Voorhees, N. K. Gupta, and B. Johnson-Laird. Learning collection fusion strategies. In *Proceedings of the 18th International Conference on Research and Development in Information Retrieval*, Seattle, WA, 1995.

[15] D. Wolfram. Applying informetric characteristics of databases to IR system file design, part I: Informetric models. *Information Processing & Management*, 28(1):121–133, 1992.