Performance Analysis of Distributed Information Retrieval Architectures *

Brendon Cahoon Kathryn McKinley

Department of Computer Science
University of Massachusetts
Amherst, MA 01003, USA

June 7, 1995

Abstract

Large document collections are increasingly available over the network. In order for users to access these collections, information retrieval systems must provide coordinated, concurrent, and distributed access. Since even unified information retrieval (IR) systems place heavy demands on system resources, it is unclear how performance will be affected as user demand increases and the distributed IR systems grow in size. In this paper, we present the implementation of a simulator and a prototype system, and the design for experiments to study the performance of distributed IR systems.

The prototype distributed information retrieval system is based on Inquery, an existing, unified IR system. We have implemented a flexible simulation model to serve as a platform for analyzing performance issues given a wide variety of system parameters and configurations. We validate the accuracy of our simulation model using the prototype. We present a series of experiments that are designed to measure system utilization and identify bottlenecks. We vary numerous system parameters, such as the number of users and text collections, number of terms per query, response time, and system load to generalize our results for other distributed IR systems.

1 Introduction

Full-Text information retrieval (IR) systems are increasingly being used on larger databases and by more users. Current systems allow users to connect to a single database either locally or perhaps on another machine. The resource demands limit the performance of IR systems especially as the size of text collections and the number of users increase. Distributed computing offers a solution to these problems. Systems based on distributed architectures can use resources more efficiently by spreading work across a network of workstations and by enabling parallel computation.

An IR system is an ideal application to distribute across a network of workstations. The amount of information available and the number of people accessing data over networks is rapidly increasing. To meet future demands, a distributed IR system must provide concurrent, efficient access to multiple text collections located on remote sites. However, due to the mix of I/O and CPU intensive operations, IR systems present unique problems for distributed system designers. The disparity between I/O and processor

*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. Any opinions, findings, and conclusions or recommendations expressed in this material are the authors and do not necessarily reflect those of the sponsor.

speeds exacerbates these problems. Another concern is network traffic, since the amount of data transferred over the network by a distributed IR system fluctuates considerably.

The focus of our research is to analyze the performance of distributed information retrieval architectures. Changes to different system parameters (*e.g.,* the number of users and text collections) affects response time, throughput, and resource utilization. During our investigation we will 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. Also, we believe it is important to maintain the effectiveness, in term of recall and precision, of a standalone IR system.

In the next section, we propose a basic architecture for a distributed system and we describe the implementation a prototype based upon the design. We based the prototype on Inquery, an existing and effective standalone IR system. Using an existing system allows us to concentrate on overall architecture performance issues rather than retrieval methods. The prototype serves as a basis for a detailed simulation we implemented to perform our analysis of distributed IR architectures. The simulation, described in Section 3, accurately models the prototype system since it uses measurements we obtained from the actual system. We describe a set of experiments in Section 4 designed to measure system utilization and identify bottlenecks. We examined several diverse text collections and query sets to obtain parameters used during the experiments. We further generalize our results for other IR systems by varying other system parameters. In a future report, we will present the results of the experiments. We also briefly describe future experiments that we will base upon our initial results. Finally, in Section 5, we discuss related work.

2 A Distributed Information Retrieval System

We have implemented a prototype distributed information retrieval system that is based on Inquery; an inference network, full-text information retrieval model [CCH92]. The system adopts a variation of the client-server paradigm consisting of a set of clients connected to a set of retrieval engines through a central administration process, as illustrated in Figure 1.

The *client* is the user interface to the retrieval engine. Clients initiate the work performed in the system by sending commands to the servers. The commands include the retrieval operations such as query evaluation and document retrieval. The client also determines the set of text collections to search for query operations.

An *Inquery server* is the retrieval engine. An Inquery server represents a single open database that is selected at start-up time. All retrieval operations take place at the Inquery server. There are three types of Inquery servers: query, document, and integrated. A query server only performs query operations, a document server only performs document retrieval operations, and an integrated server performs both document retrieval and query evaluation operations.

The *connection server* is the administrator between the clients and Inquery servers. Multiple clients and Inquery servers may be connected to the same connection server. The connection server is a lightweight process that manages the work between the user interface and retrieval engine. All messages passed between the clients and Inquery servers must go through the connection server. Messages are placed in a queue until the Inquery server is available to process new operations. The connection server also maintains a limited amount of state information about current operations and available databases.

In this section, we first discuss an initial system designed to provide access to data located on remote sites. Then, we describe a new prototype system designed to address the drawbacks of the original system. We use the prototype distributed architecture to create our simulation model.

Initial System

The initial client-server version of Inquery provides a connection between a client and a single Inquery server. In this simple implementation, the clients send commands to the connection server which is responsible for

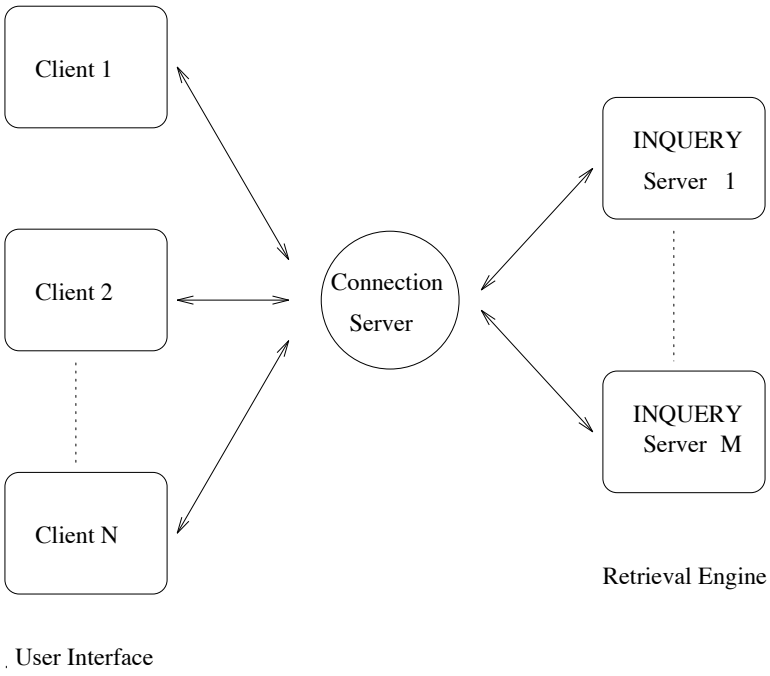


Figure 1: Client-Server Inquery

forwarding the commands to the appropriate Inquery server when it becomes available. The advantage of this system is that users can perform retrieval operations on text collections located at remote sites. Unfortunately, the system has several disadvantages.

- Inefficient allocation of resources (*i.e.*, no parallelism)
- The connection server and Inquery servers are located on same host
- Connection between a client and Inquery server remains static

The disadvantages impose severe restrictions upon the system. Only a few Inquery servers may run at the same time since the system locates the connection server and Inquery servers on the same host. The system also does not allow the connections between the clients and Inquery servers to change. That is, upon initialization, the client chooses the Inquery server to search. To search a different database, the user must create a new client process.

Prototype Distributed System

A new distributed system was designed and implemented to address the disadvantages of the initial client-server version. The new system includes the following features:

- It provides multiple, simultaneous connections between clients and Inquery servers.
- Clients, Inquery servers, and connection servers may be located on different hosts.
- Connections between clients and Inquery servers are dynamic.

The functionality of the clients and Inquery servers remains mostly unchanged. The only change to the client is that it may connect to multiple Inquery servers and the connections may change. The user may chose a different list of databases to search just before each query operation. We made several significant changes to the connection server to provide multiple connections between clients and Inquery servers.

A command from a client contains a list of Inquery server *ids*. The connection server reads the command and routes it to the appropriate Inquery servers. If the Inquery server is busy, the connection servers places the command on a queue until the Inquery server is free. The connection server does not immediately send results from the Inquery servers back to the client. Instead, the connection server temporarily maintains the intermediate results and once it collects and merges the results, it sends the final answer back to the client. The routing and merging algorithms performed by the connection server depend upon the command. We describe these algorithms in the following paragraphs.

The connection server does not change queries sent from a client; it simply forwards them to the selected Inquery servers. The answer for a query is a ranked list of matching documents. The connection server merges the lists from each of the Inquery servers before sending them back to the client. The merging process involves parsing the message and creating a list which it merges together with prior results. The connection server maintains the ranked list and sends the combined result to the client when it receives the results from the last Inquery server.

The summary information command sent by the client contains a list of documents and an associated list of Inquery servers that contain each of the documents. However, the Inquery servers expect the summary list to only contain documents that it manages. The connection server reads and changes the summary information command before passing it along to the Inquery servers. The connection server sorts the lists by Inquery server and sends the list of documents to the appropriate Inquery server. Unlike the merging process for a query answer, the merging process for the summary information is not incremental. The connection server merges the results from each of the Inquery servers in the original order before sending the the answer back to the client.

Processing a document retrieval command is very simple. A document retrieval operation sent by a client contains the document *id* and the name of Inquery server that contains the document. The connection server looks at the command and forwards it to the Inquery server indicated. When the document is returned, the connection server returns it to the requesting client. The connection server does not need to maintain intermediate results or merge answers.

Clearly, the new distributed system is more complex than the previous implementation and there are many design considerations that can affect performance. For example, the connection server appears to be a bottleneck in the system. It routes routing messages between all clients and Inquery servers, and it collects and merges intermediate results from the Inquery servers. Through experimentation we can determine the conditions that cause the connection server to noticeably degrade overall performance. We will investigate several factors that may affect performance in a series of experiments described in Section 4.

3 Simulation Model

Simulation techniques provide an effective and flexible platform for analyzing large and complex distributed systems. We can quickly change inputs, 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 results in a controlled environment.

We have written a simulation model of our distributed information retrieval system. The model is designed to be simple, yet contain enough details to accurately represent the important features of the system. The work modeled by the simulation represents the activities performed by an actual retrieval system. The model is driven by empirical measurements obtained from Inquery, an existing IR system.

Although the model is based upon Inquery, it is parameterized in order to easily model variations of our own system and other retrieval systems.

The following sections describe the simulator toolkit used to implement the simulation model, the workload model, simulation parameters, and the simulation configuration procedure.

3.1 Simulation Library Toolkit

Our simulator is built upon the YACSIM simulation library. YACSIM is a process-oriented discrete-event simulation language based on the C programming language [Jum93]. The process-oriented nature of the simulation language allows the structure of the simulator to closely reflect the actual system. YACSIM contains a set of data structures and library routines that manage user-created *simulation objects*. A simulation object defines an activity (*i.e.*, a process), a queue (*i.e.*, a resource), or a statistic record.

The simulator models the major components of our system (*i.e.*, the clients, connection server, and Inquery servers) as YACSIM processes. Processes simulate the activities of the real system by requesting services from resources. A process that requests a resource executes when the resource is available. Otherwise, the process suspends until the resource is available. The processes synchronize via a set of YACSIM message passing library routines. Processes that are waiting for messages suspend until the appropriate message is received.

YACSIM provides convenient mechanisms to model resources. Each resource is defined as a queue and a set of servers. When a resource is requested by a process, it is assigned to an available server. If the server is not available, the request is queued until a server becomes free.

Our model defines a CPU as a resource with one server. The queuing discipline used by the resource mimics an actual processor. Each request is serviced without delay, but as the number of requests increases, the remaining service time for each request increases. For example, if two processes request the CPU at the same time for one second, then it takes a total of two seconds for each process to complete.

The simulator models a disk as a resource with one server which uses a first in first out (FIFO) queuing discipline. Each request is serviced without interruption when the server becomes available. If the disk is not available, the simulator places the request at the end of the queue.

The network resource is limited by the amount of data that can be transmitted at any time rather than the number of requests that it can handle. A user parameter constrains the bandwidth of the network. The model defines the network as a resource with servers for each of the processes created. Thus, each process may send a message at any time. The simulator queues new requests when the bandwidth would be exceeded.

3.2 Workload Model

The workload model represents the work typically performed on an information retrieval system, such as Inquery. Currently, we only simulate query evaluation and document retrieval operations which are the basic retrieval operations. More complicated functions such as relevance feedback are not simulated. The clients initiate all the work performed in the system. Clients send commands to the Inquery servers and wait for the results until the next operation is initiated. In the simulator, clients repeatedly perform the following sequence of operations:

1. Evaluate a query
2. Obtain summary information of top ranked documents
3. Think
4. Retrieve documents

5. Think

The simulator varies the specific operations for each user and during each sequence. For example, the model generates new queries and retrieves different documents for each iteration. The following sections describe each of these operations in detail.

3.2.1 Query Evaluation

In Inquery, a query consists of natural language words and structured query operators [CCH92]. Our simulator only models queries consisting of words and ignores structured query operators, such as `#phrase` or `#sum`. Either the simulator generates queries or the user supplies queries in a separate file. Instead of using words, we represent queries as a list of collection term frequencies.

Users supply parameters to construct the artificial parameters generated by the simulator. The parameters include:

- Distribution of the number of terms in a query.
- Distribution of the query term frequencies.
- Maximum term frequency in the text collection.
- Number of queries to generate.

Users may also supply complete queries to the simulator. The advantage is that users may collect traces from real IR system and directly feed the traces to the simulator. The format of the queries is a simple list of term frequencies. Using Inquery, it is simple to collect the term frequencies given a trace of actual user queries and the text collection. An example of an entry in the file is `9 13583 16938 46629 25663 18943 30328 152523 33056 16084`. The first value is the number of terms and the rest are the term frequencies.

The simulator accurately reflects the operations performed by the prototype. That is, once a query is generated it is sent to the connection server over the network and the connection server is responsible for routing the message to the appropriate Inquery servers. The client receives a ranked list of matching documents and then obtains summary information about the documents.

3.2.2 Summary Information

In the Inquery system, summary information about the top ranked documents is displayed on the screen once a query is evaluated. The summary information includes the title, heading, ID, and rank score for a document. The number of document summaries retrieved depends upon the size screen. A user is able to scroll up and down the list to view more document summaries.

The model simulates the activity of browsing summary information without actually displaying any information. The simulator accounts for the resources used to obtain the summaries and the time to browse the information. The model represents the time to browse the information as "think time". A document summary command involves sending a message to the connection server that requests the summary information for a list of documents. A parameter to the simulator defines the number of documents in the list (*i.e.*, the "screen size"). The connection server routes individual summary operations to the Inquery servers that maintain the document requested. The simulator randomly chooses the Inquery servers that contain the documents. A simulation parameter indicates the number of document summary commands that a client issues after evaluating a query.

3.2.3 Document Retrieval

After browsing summary information, the client retrieves and displays a complete document from an Inquery server. The simulator models the document retrieval operation by sending a message to the connection server requesting the document. Similar to the summary information operation, the simulator accounts for disk, processor, and network resources that are needed to load the document from secondary storage and send it to the client. Time to read the document expires after retrieving a document. The simulator randomly chooses the Inquery server. The size of the document retrieved is a simulation parameter.

3.3 Simulation Parameters

To accurately model an IR system, we analyzed the Inquery system and measured the time the resources used for each operation. Empirical measurements rather than an analytical model drive the activities performed in the simulator. Creating a full analytical model requires too many simplifying assumptions to make an accurate model of a complex information retrieval system.

In this section, we first describe the text collections used to obtain resource measurements. We also describe the measurements for each of the activities performed by the prototype. The measurements include:

- Query evaluation time
- Document retrieval time
- Summary retrieval time
- Connection server time
- Time to merge results
- Network time

We obtained measurements using 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. To minimize the effects of other users we ran all tests during the evening when the system load was very light.

Text Collections Used to Obtain Measurements

We examined several different text collections and query sets to obtain measurements used in the simulation. The text collections are

- TIPSTER 1
- CACM
- Legislative Information (103rd CR)

Table 1 lists statistics about the text collections and query sets.

TIPSTER 1 is a large heterogeneous collection of full-text articles and abstracts used in the TREC-1 evaluations [Har92]. The documents in the collection come from a variety of sources including newspapers, magazines, and government announcements. The average document size is not large but the sizes of the documents vary from 100 bytes to 1 MB. The TIPSTER 1 collection includes a set of 50 queries created automatically from TIPSTER topics 51-100. The queries consist of English text and do not contain any structured operators.

Collection	Collection Statistics					Query Statistics	
	Size (MB)	No. of Documents	Avg. Doc Size (KB)	No. of Postings	Max Term Frequency	No. of Queries	Avg. No. of Terms
TIPSTER 1	1198	510887	2.3	109571494	554658	50	27.1
CACM	2.1	3204	0.5	115967	2208	50	10.7
103rd CR	500	43378	11.7	48078407	714849	5141	2

Table 1: Collection and Query Set Statistics

The legislative information collection contains the text of the Congressional Record for the 103rd Congress [CCW95]. The documents in the collection summarize of the day's events from Congress. The size of the documents range from 1K to 700K and the average document size is large. We included this collection since it is a database that is accessible over the Internet and we were able to obtain the user query traces. We examined the query logs between March 1st and April 1st, 1995 to obtain realistic query statistics. We discarded all queries that Inquery marked illegal. Interestingly, people searching the database typically enter small queries, usually only 1 or 2 words and never more than 9 words. This contrasts with the other query sets that contain long queries.

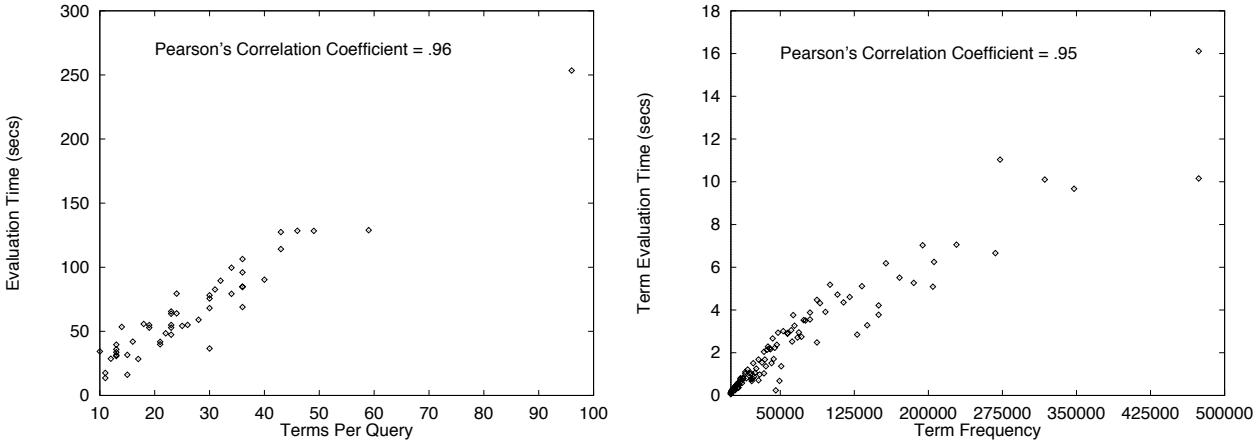
The CACM document collection consists of small abstracts from the *Communications of the ACM* [Fox83]. This collection is an older test collection consisting of a small number of homogeneous documents. The collection is only 2MB and the average size of the documents is small. The CACM collection also includes a set of 50 English text queries. Again, no structured operators appear in the queries.

3.3.1 Query Evaluation Measurements

In Inquery, a query operation consists of creating a query network, evaluating the query network on the document network, and ranking the documents that match the query. Since the process is quite complicated, we empirically measured the time required to evaluate a query using Inquery rather than creating a complex analytical model. We found that the evaluation time is very strongly related to the number of terms in the query and the frequency of each of the terms. Figure 2(a) shows a scatterplot which compares query length versus evaluation time for each query in the TIPSTER 1 query set. The correlation is very high, .96, indicating a strong linear association between query length and query evaluation time. Figure 2(b) shows the relation between term frequency and evaluation time. Again, the correlation coefficient, .95, indicates a very strong association. To collect this data, we measured evaluation times for individual terms with different term frequencies.

The simulation model contains a distribution of evaluation times based upon the term frequency. We measured the time to evaluate the individual terms with Inquery. Given a query that is internally represented as a list of term frequency values, the evaluation time is the sum of the times for evaluating the individual terms in the query. The data in Figure 2 indicates that this simple model reflects the time to perform query evaluation.

We validated the query model used by the simulation against the actual system. Figure 3 shows that the simulation model is close but does not exactly reflect the actual system, especially for large queries. However, there is a strong correlation between the simulation times and actual times. The difference between the times is due to the simple model used by the simulation. The actual retrieval process implemented by Inquery is quite complex and difficult to describe using a simple model. Although it is not perfect, the general trend of the model is accurate. (Using a prototype and a model enables us to do this type of validation.) We need to examine the actual retrieval process in greater detail to add more features to the simulation model.



(a) (b)

Figure 2: Query Length and Term Frequency vs. Evaluation Time

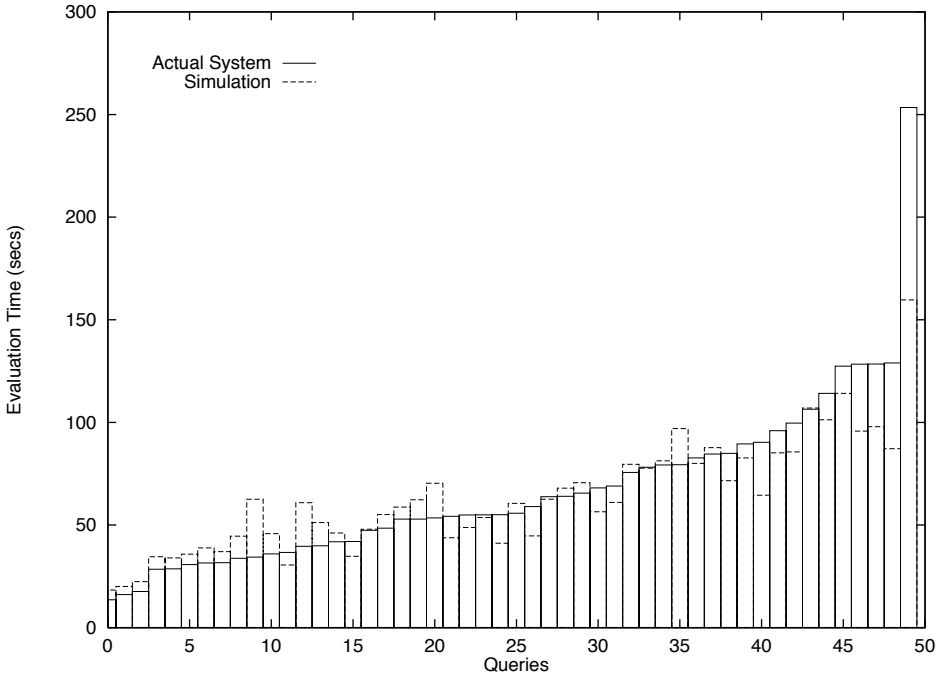


Figure 3: Query Model Validation

3.3.2 Document Retrieval and Summary Information Measurements

The time required to retrieve a document is dominated by the I/O time to read the document from secondary storage and is a function of the document size. We empirically measured the time required to retrieve documents of different sizes and use the results in the simulation.

The summary information operation is implemented as a series of document retrieval operations. The only difference is the amount of data that is transmitted back to the client. For example, if summary information is requested for 10 documents then the time for the operation is the sum of 10 document retrieval operations. User defined simulation parameters indicate the average size of the summary information for a document, since it varies with the collection.

3.3.3 Connection Server Measurements

We analyzed the connection server to determine resource usage. The connection server remains idle until either a client or an Inquery server sends a message. Once the connection server receives a message, it uses a small amount of CPU time time to perform two activities; merging intermediate results and/or routing the message. We do not account for the time necessary to create the initial connection between the connection server and client or Inquery server. During our experiments, we create the connections at the beginning of the simulation and they remain until the simulation ends.

Our measurements indicate that the connection server uses a constant amount of time to route a message. This time is 100 milliseconds. The time required to merge results is a function of the size of the data. Although the merging time is very quick, the time slightly increases as the size of the data increases.

3.3.4 Network Time Measurements

Network time consists of CPU time to process the message plus the time to transmit the data over the network. The simulation uses different models for sending and receiving messages. The model for sending a message includes both CPU and network usage. However, the model for receiving a message includes only the CPU time necessary to process the message because the sender accounts for the network time. We empirically measured the times using our local area network. It uses a 10Mbs Ethernet connection. We obtained measurements during the night to determine times when the load on the network is light. The time for transmitting messages depends upon the size of the message and the network bandwidth.

3.4 Simulation Configuration

A user defines the architecture of the distributed information retrieval system using a simple command language. A configuration file contains the commands and the simulator reads this file at start-up time. Command line options may also be used to define certain aspects of the architecture and these options override any commands in the configuration file. The command line options allow users to easy run the simulator in batch mode by using a single configuration file and changing the command line options for each simulation. For example, if an experiment tests the effect of the number of clients, then a single configuration file defines the architecture and a command line option specifies the number of clients.

The command language supports a rich and flexible set of commands for specifying different distributed architectures. Table 2 describes the set of commands that define the different processes and resources used by the simulation. Each of these commands expects a parameter which is either a constant value or an expression evaluated when the file is read.

Table 3 describes the commands that assign properties and attributes to the simulation objects. The parameters depend upon the actual command. The configuration file allows users to provide parameters as constant values or symbolic values evaluated when the simulator reads the file. A user creates a system

Category	Command	Description
Processes	`clients`	Number of clients
	`conn_servers`	Number of connection servers
	`inq_servers`	Number of Inquery servers
Resources	`cpus`	Number of processors
	`disks`	Number of disks
	`lans`	Number of local area networks
Objects	`machines`	Number of hosts
	`database`	Number of text collections

Table 2: Definition Commands

Command	Properties
`conn_server`	Machine assigned to connection server
`client`	Machine assigned to client
`inq_server`	Machine, database, and type of server
`machine`	CPU, Disk, and LAN that create a machine
`lan`	Bandwidth of network in MB/s
`database`	Size of collection, No. of documents, Avg. size of documents
`connect`	Define connections between processes
`queries`	Define attributes of queries

Table 3: Property Commands

architecture by assigning processes to resources. Also, users may assign various attributes to resources or combine several resources creating a larger object. For examine, a machine object is a combination of the CPU, disk, and network resources. A user creates a complete distributed architecture by defining connections between the different processes.

4 Experimental Methodology

In this Section, we describe the experiments we will conduct in order to analyze the performance of our system, identify potential bottlenecks, and to create a scalable system. We first describe our experiments that analyze system utilization. At the end of the section, we briefly discuss several other types of experiments we will perform.

4.1 System Utilization

We designed these initial sets of experiments to measure the utilization of the different resources in the distributed system under varying conditions.

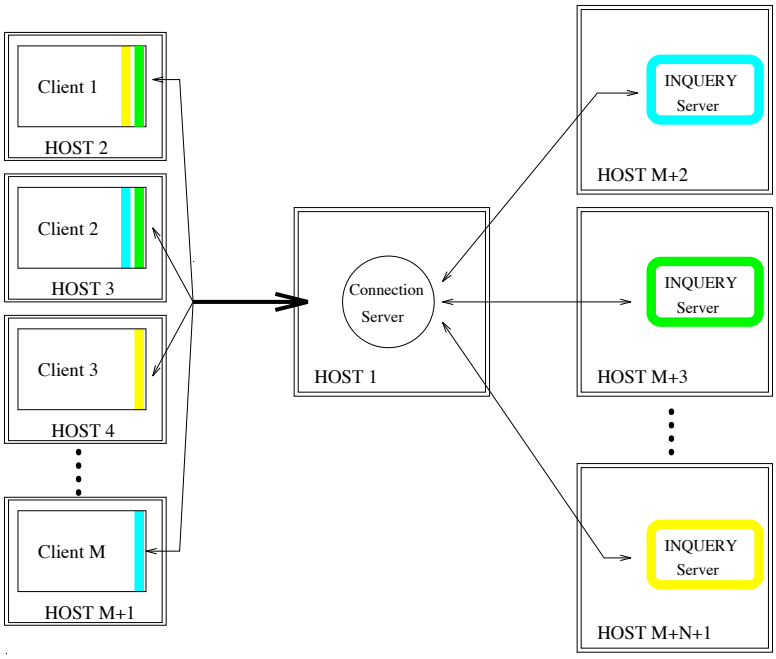


Figure 4: Basic Architecture

4.1.1 Fixed Parameters for System Utilization

In the initial experiments, the system architecture closely matches the prototype client-server version of Inquery. We must fix several parameters throughout these experiments to match the architecture of prototype system. We fix other parameters to reduce the total number of experiments performed. We will explore the effects of varying these parameters in future experiments.

The fixed parameters are:

- Functionality of processes (*e.g.*, clients, connection server, Inquery servers)
- Connectivity between clients and Inquery servers
- Size of text collections
- Network speed

The basic architecture defines a set of clients, a single connection server, and a set of Inquery servers. In Section 2, we described the basic components of the prototype client-server information retrieval system. The simulator allocates each of these components to its own host to eliminate the effects of competing processes and to measure the overall parallelism in the system. Each host contains its own processor, memory, and secondary storage. A local area network with a bandwidth of 10 Mbs connects all hosts. Figure 4 illustrates this basic architecture.

All messages sent between the clients and Inquery servers pass through the connection server. The connection server is also responsible for merging results from the Inquery servers before sending them back to the client.

Each Inquery server processes both query and document retrieval operations. The size of the text collection managed by each server is one GB. The size of the collection influences the frequency of the

terms in the queries. We analyzed the TIPSTER 1 document collection to determine the average size of a document and the average size of the summary information for each document. The document retrieval and summary information operations uses these parameters.

Upon initialization, each client connects to a random number of Inquery servers and the connections remain throughout the simulation. The process of connecting to a random number of Inquery servers simulates the actions performed by actual users. That is, a typical user will choose to search some set of available text collections, or a single collection that is distributed over a set of machines. Connecting to all the available Inquery servers or only one of the databases is probably rare.

4.1.2 Parameters for System Utilization

We analyze system utilization by running many experiments which vary several important parameters. The parameters affect system performance and are often variable in actual systems. We will determine the effects of these parameters on system performance by using the simulator. We provide a description of the experiment parameters listed below in the following sections.

- Number of clients
- Number of Inquery servers
- Terms per query
- Distribution of terms in queries
- Number of documents that match query
- Think Rate
- Documents retrieved
- Summary information operations

Number of Clients/Inquery Servers

Measuring the effect of increasing the number of clients and Inquery servers provides insight into identifying when each of the processes and resources become overburdened.

As the **number of clients** increases, the amount of total work performed in the system increases. The connection servers must handle more messages and the Inquery servers perform more retrieval operations. Also, the amount of data transmitted across the network increases. If the clients each connect to distinct Inquery servers, then the parallelism of the system improves since the components can operated concurrently. However, if several clients connect to the same Inquery servers then the effect on the system is unclear, but obviously performance will degrade somewhat due to contention of resources. Regardless of the connections, the performance of the connection server decreases as the number of clients increases. It is also unclear when the performance degradation will become intolerable.

As described in Section 2, the Inquery servers handle query evaluation and document retrieval operations. These operations are both CPU and I/O intensive. In general, the document retrieval operations occur much more quickly than query operations. Increasing the **number of Inquery servers** in the system has a slightly different effect than increasing the number of clients. Each Inquery server is located on its own processor and operates independently from the others. Any effect on performance caused by adding additional Inquery servers is due to additional connections to the server, *i.e.*, the additional connections create extra contention for resources and cause more messages to be passed between the components. In the simulation, as the

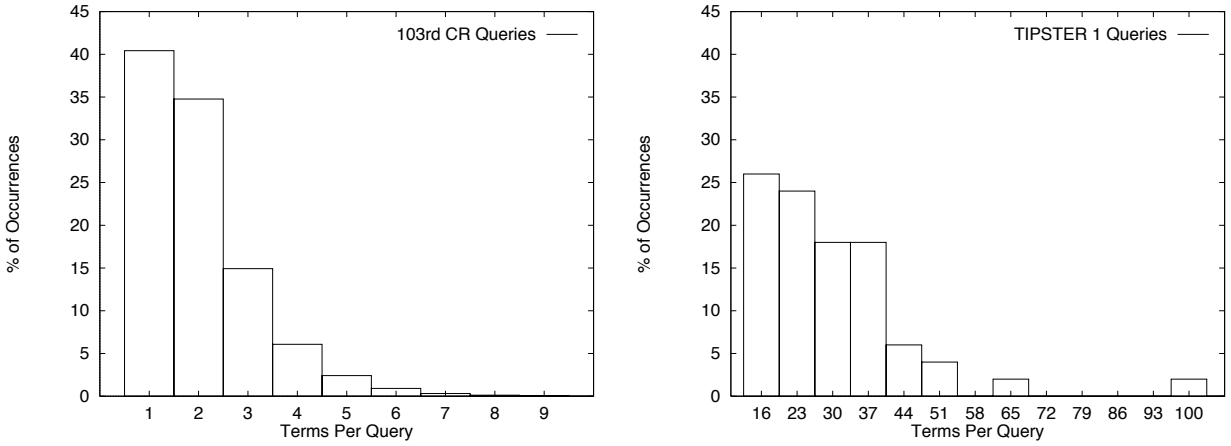


Figure 5: Query Length Distribution

number of Inquery servers increases, the number of connections between the clients and connection servers increases causing more contention for resources. Thus, increasing the number of Inquery servers increases the total amount of work performed by the system.

We will test configurations using **1**, **4**, **8**, **32**, **64**, and **128** clients and Inquery servers. A baseline comparison architecture consists of one client and one Inquery server. The other values provide meaningful comparisons between small and large configurations.

Query Evaluation

Query evaluation operations dominate the time spent in an IR system. There are a number of factors that affect the time to evaluate queries, including the number of terms per query, the query term frequency distribution, and the number of matching documents returned by the operation. In our experiments, we vary each of these parameters.

Terms Per Query There are several reasons to consider the performance effects caused by varying the number of terms per query. First, the time to evaluate a query is strongly related to the length of the query. In addition, the length of queries entered into a retrieval system varies significantly from one or two keywords to over 20.

For example, the queries used in the TREC evaluations are long and contain an average of **27.1** terms per query. Conversely, the typical query entered in the THOMAS system is very short and contains an average of **2.17** terms per query. In addition to using the values from actual data, we will perform experiments using medium length queries that contain contain **12** terms per query.

Figure 5 shows the query length distribution for the 103rd Congressional Record and TIPSTER 1 query sets. Notice that, even though the two query sets are very different, the shapes of the distributions are very similar. We use this information to create the synthetic queries.

Matching Documents Returned The IR system returns a sorted list of matching documents *ids* from the Inquery servers. To reduce the number of matching documents returned, users usually specify a maximum value. The number of documents returned has an effect on network traffic and processing by the connection server and Inquery servers. We will experiment with returning the top **1000** and top **100** documents.

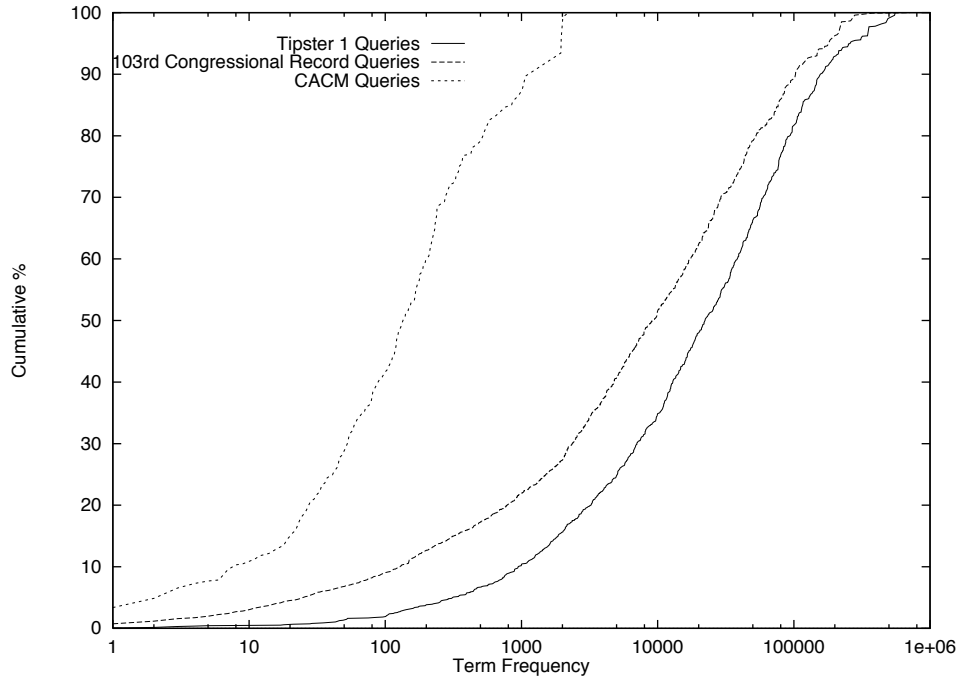


Figure 6: Query Term Frequency Distributions

Query Term Frequency Distribution Zipf documented the widely accepted distribution of term frequencies in text collections based upon empirical measurements [Zip49]. In contrast, the distribution of term frequencies in queries is more difficult to characterize and researchers do not agree on a commonly accepted distribution [Wol92]. Figure 6 shows the query term frequency distributions for our query sets. The shapes of the distributions for the three query sets are very similar. The difference between the three distributions is due to the size of the text collections. The TIPSTER text collection is the largest, so the distribution is shifted to the right indicating that the terms used in the queries appear more frequently. As the figure shows, the shape of the collection term frequency and the query term frequency distributions are very different. The frequency of most of the query terms is somewhere in the middle of the text collection distribution. Terms that occur very infrequently or very frequently in the text collection rarely occur in queries.

Since the frequency of terms in queries has an impact on evaluation time, we consider the following query term frequency distributions.

Mid-range - The frequency distribution observed from our query sets.

Low Frequency - The frequency distribution is skewed towards terms that occur less frequently.

High Frequency - The frequency distribution is skewed towards terms that occur often.

Think Rate

In the simulated workload, "thinking" occurs after receiving summary information and after retrieving a document. Thinking represents the time users need to look at the results of their requests. The amount of thinking performed after each retrieval operation can have a significant effect on overall performance. When a client is thinking, it remains idle and other processes are allowed access to the system resources. Clients

initiate operations more frequently and there is more contention for resources when the think time is short. On the other hand, system resources may remain idle if the think time is too long.

It is important to understand under what conditions system resources become saturated or under utilized. In the experiments, we will change the time spent thinking to study this effect. The think time after a summary information operation will be **30**, **90**, or **240** seconds. The think time after a document retrieval operation will be **60**, **180**, or **480** seconds.

Document Retrieval and Summary Information

During our experiments, we will vary the number of summary information operations and the number of documents received after each query is evaluated. These operations are similar since they both involve retrieving documents. Document retrieval operations are I/O intensive, but, in general, take less time than query operations. When very few documents are retrieved, query evaluation time will dominate the performance of the distributed system. When the client process retrieves many documents, the effects of the operations will be more balanced. As more documents are retrieved, more time is spent thinking. This idle time also has an effect on the system since it frees system resources.

We categorize the number of summary operations and documents retrieved that the client performs into three groups. These groups represent workloads seen in practice and each group represents a range of summary information or document operations. The number of operations to perform is randomly chosen from the following ranges: **1-5**, **8-12**, and **15-20**. For example, in the first range, the client performs between 1 and 5 summary information and document retrieval operations.

4.2 Future System Architecture Experiments

After we perform the initial system utilization experiments, we will perform a series of experiments to examine different architectures. We will base our experiments on the results obtained from the initial experiments. The experiments will focus on the Inquery server configuration, distribution of functionality, and dynamic responses to system load and we describe them below.

Inquery Server Configuration

The Inquery server receives IR commands, evaluates them, and sends results back to the connection server. In the system utilization experiments, a single Inquery server is responsible for both query evaluation and document retrieval operations. However, these operations use systems resources differently. For example, evaluating a query usually takes longer than retrieving a document and query evaluation uses more processor time.

We will explore the effects of *splitting* an Inquery server into two servers, one for document retrieval and one for query evaluation. A split configuration reduces the competition for the single server by allowing document retrieval and query evaluation operations to proceed concurrently.

There is a cost associated with splitting a server. For example, as the amount of network traffic increases, the demands on the connection server increase and the number of processes created increases. The goal of this experiment is to identify the circumstances under which a split server is better than a combined server.

Functionality Distribution

Based upon our system utilization results, we will run a series of experiments testing the effects of distributing certain functions in the system. Two areas identified as potential bottlenecks during preliminary experiments are the connection server and the Inquery servers.

The connection server is responsible for managing messages passed between the clients and the Inquery servers. The connection server also collects and combines results obtained from the Inquery servers before forwarding the final result to the client. For example, when multiple servers evaluate a query, the connection server merges results together. We would like to experiment with moving this process from the connection server to the client to determine if it improves performance. Moving this process allows the connection server to perform other activities. The cost of moving the merging process is increased network traffic. It also increases the CPU demands on the client.

Another costly process is document ranking. During query evaluation, the Inquery server ranks the matching documents according to a "belief" value. The Inquery server ranks the list as a final step before returning the results. We will experiment with moving the ranking process to the connection server or possibly the client. Preliminary experiments show that query evaluation is very time consuming and moving the ranking process might improve overall throughput by allowing the Inquery server to process more operations. However, the disadvantage is increased network utilization.

Increasing Connection Servers

In the prototype system, only one connection server manages all the messages between the clients and Inquery servers. The scalability of the prototype system is limited since it creates only one connection server. We will use the results from the system utilization experiments to determine the conditions under which the connection server becomes overloaded.

We will vary the number of connection servers to determine the best configuration. Creating more connection servers increases the complexity of the architecture. The clients and Inquery server may connect to one or more of the existing connection server. We will need to define a process to manage the connection servers, called the Distributed Services Manager. The clients and Inquery servers determine information about the configuration by talking with the Distributed Services Manager. We will also experiment with a dynamic Distributed Services Manager that knows when to replicate connection servers.

One experiment will have a connection server for each Inquery server, so the connection server will become merged with the Inquery server on the same machine. This experiment simulates the effect of implementing an Inquery server that directly manages multiple connections from the clients. The problem with this configuration is that we must move the merging process to the clients (as discussed above).

5 Related Work

Very few people have investigated issues specific to distributed IR systems. We present a brief description of previous work related to our research. We organize the descriptions into the following sections:

- Architecture Performance
- Data Partitioning
- Caching
- Multiprocessor Systems

The work on architecture performance relates most closely to this work. Our research combines and extends the previous work since we model and analyze complete system architectures. Although others have examined some of the design issues, no one has considered the entire system. Our research measures system utilization under a variety of realistic conditions. The results will provide information which identify areas where we can improve parallelism and scalability.

Architecture Performance

Prior work on analyzing complete distributed IR architectures is limited. Most work concentrates on particular aspects of system performance such as data partitioning and caching. Our research complements previous work on architecture performance analysis by extending the scope of the experiments. In our experiments, we vary more parameters and study larger systems consisting of more text collections and users. Also, our simulation more accurately reflects user workloads since we use information from our analysis of actual text collections.

Burkowski reports on a simulation study for a distributed text retrieval system which uses a local area network to connect a set of clients (*i.e.* users) to a set of servers (*i.e.* text collections) [Bur90]. The operations modeled in the system involve both query evaluation and text retrieval. The experiments performed explore two strategies for distributing the workload across the servers. The first equally distributes the text collection among all the servers. In the second implementation, the servers are split into two groups, one group for query evaluation and the other group for document retrieval. Experiments performed use 8 clients and 1, 2, 4, 8 or 16 servers. Response time, speedup, and efficiency statistics are presented. Burkowski concludes that the uniform distribution strategy is very reasonable and performance approaches linear speedup depending on the overheads. He further concludes that the split functionality strategy provides better response times than the uniform approach under certain conditions. Our work differs from this work in several ways. Burkowski assumes a worst case workload where each client broadcasts queries to all servers without any delay (*i.e.* no user think time). Also, we experiment with larger distributed configurations, vary the number of clients, and use a more accurate system model. Another difference is that our system architecture includes a middle component between the clients and servers. Our experiments will identify the advantages and disadvantages of this architecture.

The implementation and analysis of a distributed IR system is studied by Lin and Zhou [LZ93]. They implement the retrieval system on a network of DEC5000 workstations. The PVM (Parallel Virtual Machine) software manages the distributed system by coordinating work and communication over the network. The retrieval model uses a variation of the signature file scheme to encode documents and to map the signature file across the network. Experiments are performed to determine the effectiveness of a distributed architecture for IR systems. Their results show large speedup improvements due to parallelization. The motivation behind this work is to show that an implementation of an IR system on a network of workstations can improve performance. Our research extends this work by analyzing a distributed system to *increase* efficiency. Since we use a simulation, we are also able to run more experiments under different conditions. Another advantage of our distributed system is that it is based upon proven, effective retrieval model rather than a new model developed to expose parallelism.

Brumfield *et al.* also create a detailed simulation to measure the performance of different configurations and workloads [BMC88]. However, the application is a distributed object-oriented database system which is quite different than an IR system. The paper is relevant since it presents a model for using a simulation to examine the performance of a distributed system. As they show, a flexible simulation model enables different aspects of a distributed system to be tested prior to implementation. They obtain a variety of statistics such as query processing time and resource utilization. The results provide valuable insights into potential problems.

Data Partitioning

In this section, we discuss previous research which examines the performance of various data partitioning schemes. Our initial experiments do not specifically examine the performance of different data partitioning schemes. Instead, we are concerned with overall performance and scalability for large systems involving many text collections. The work on data partitioning only examines systems that distribute a single text

collection. Furthermore, no one has considered the tradeoff between performance and effectiveness when distributing a text collection. These issues have only been considered in isolation. Callen *et al*. [CLC95] and Viles and French [VF95] present recent studies on retrieval effectiveness for distributed collections.

Tomasic and Garcia-Molina investigate the performance of various inverted file organizations in a distributed shared-nothing text retrieval system [Tom94, TGM93]. A shared-nothing model is a system in which each processor has its own memory and secondary storage. Three index organizations are studied. The *system* organization evenly spreads the inverted file across all disks in the system. The *disk* and *host* organizations partition the collection into two groups, one for each disk and processor, respectively. These organizations create a separate inverted file for each partition. They use a detailed simulation model to study performance tradeoffs of each organization. The simulator defines synthetic databases and generates queries from probability distributions based on actual statistics.

A single host initiates query evaluation which broadcasts subqueries to the hosts containing the appropriate data. They conclude that the choice of index organization depends upon the network speed and secondary storage access time. In general, the host organization performs the best or close to the best. The system organization only performs well when the network speed and storage access times are quick. In this work, they only simulate query operations resulting in a workload that does not accurately reflect real user sessions. That is, the experiments ignore the effects of document retrieval operations and user think time.

Macleod *et al*. present an implementation of a prototype distributed IR system based upon Fulcrum FUL/Text, an existing unified system that uses inverted files [MMNP87]. They describe and test four data distribution strategies that partition and replicate data. They implement a distributed simulation that is physically distributed across a network to test the performance of the data distribution strategies. The hardware includes five personal computers connected by IBM's PC Network. The simulation runs on a single node and directly compared against the FUL/Text system to validate the model. They perform several experiments to evaluate the effectiveness of the distribution strategies. The experiments only create small distributed architectures consisting of only a couple users. The paper does not discuss the effect of varying simulation parameters, such as terms per query and query term frequency, on performance.

Jeong and Omiecinski study the performance of two different inverted file partitioning schemes in a shared-everything multiprocessor [JO92]. In the shared-everything model, memory and secondary storage is common to all processors. The first scheme partitions by term *id* while the second scheme partitions by document *id*. They use a simulation model to compare the performance of the two strategies. Validation occurs by comparing the simulation to an analytical model. The experiments vary several parameters including document term frequencies, query term frequencies, the number of disks, and the multiprogramming level. Neither scheme is superior under all conditions. Partitioning by term *id* is best when document term frequencies are small or when query term frequencies are uniformly distributed. Partitioning by document *id* is best when the multiprogramming level is high. The main disadvantage of the shared-everything model is scalability. It is important to ensure scalability as the size of collections and the number of users increases.

Caching

Several studies have examined the use of caching in distributed information retrieval systems. The client caches data so that operations are not repeatedly sent to the remote server. Instead, the client locally performs frequent operations. The use of caching is most beneficial for systems that are distributed over slow networks or that evaluate queries slowly. We do not study caching effects since we implement our system on a fast local area network. Furthermore, certain caching schemes (*e.g*. caching data at the client) would require a different retrieval engine. Our work focuses on using a existing and proven retrieval system. The Inquery retrieval engine actually performs a small amount of caching to improve the speed of query evaluation.

Martin *et al*. [MMR+90], and Martin and Russell [MR91] investigate several strategies for caching data in a distributed IR system. These studies are a continuation of the work performed by Maclead *et*

al. [MMNP87]. Again, they use a simulation model of an existing system, called Ful/Text, to evaluate the caching strategies. The system architecture consists of a single server, which maintains the text collection, and several clients which access the text collection over 9600 baud communication lines. The workload model consists of both search and browse transactions. They analyze the performance of two caching strategies, a *client cache* and a *server cache*. The client cache saves documents retrieved for browsing. The server cache simply retrieves data from disk and holds the data in a buffer. They experiment with different cache replacement algorithms and modify other parameters such as server load, think time, and degree of locality during the experiments. They conclude that distributed IR systems are effective and useful when intelligent data caching schemes are employed. These studies do not discuss more complex system architectures using multiple servers. Furthermore, the communication lines in the network are limited to 9600 baud lines.

Simpson and Alonso investigate the effect of adding caching to personal computers that access a single information retrieval system over a network [SA87]. For example, a sophisticated caching scheme retains any data that is necessary to reevaluate a query on the PC after it is downloaded from the central retrieval system. They create a detailed simulation to study the performance of different caching strategies. Their results show that simple caching strategies are useful but that the amount of caching performed must be carefully chosen. They present a general model of an information retrieval system but do not discuss the details (*e.g.*, the retrieval model). Also, their study only examines caching on a system connected by very slow communication lines.

Continuing the work performed in a previous study [TGM93], Tomasic and Garcia-Molina also perform experiments on caching [TGM92]. The main difference between this paper and their previous work is that they use real traces from FOLIO, an existing IR system that provides access to an abstracts database. As well as studying caching, they also report on data partitioning strategies and database scaling. The caching experiments investigate the effects of caching inverted lists in main memory. This type of caching is beneficial when similar queries are repeatedly issued. When searching abstracts people tend to enter multiple queries to reduce the number of answers. Typically, each query contains similar terms which provides an opportunity to exploit caching. They report that a small cache improves performance significantly. However, the experiments are limited small collections and workloads consisting of only query operations.

Implementations on Multiprocessors

In this section, we provide a very brief description of systems implemented on multiprocessor machines (*i.e.*, shared-everything model). Our work differs greatly from these systems since we use the shared-nothing model. Also, we use a simulation to perform our experiments. In the future, we will investigate distributed IR systems based on the shared-everything model.

Pogue and Willett use the ICL Distributed Array Processor to implement an IR system based on signature files [PW87]. They designed this system as a batch system although they suggest how an interactive system may be constructed. Stanfill and Kahle use the CM-2 system to implement and IR system which uses signature files [SK86]. They report on a batch system and an interactive system which exploits the CM-2 hardware. Later Stanfill *et al*. propose an IR system on the CM-2 based on inverted files rather than signature files [STW89]. They conclude that cost-effective interactive access to 100-1000 GB collections is possible. Salton and Buckley implement the Smart retrieval system, a vector processing model, on a Connection Machine [SB88]. They suggest that parallel methods do not provide large enough gains in effectiveness or efficiently. However, this decision is based on cost factors and the assumption that users are not primarily interested in speed. Cringean *et al*. report on the implementation of a retrieval system, based on signature files, built for a transputer based M40 Meiko Computing Surface [CEMW90]. This paper is an initial study and uses a small document collection and a small number of queries. Rather than studying a real distributed system, we have created a simulation to examine the performance of different system architectures. This

environment is more flexible for analyzing different systems. We will use this information to improve our real distributed information retrieval system.

6 Summary

As the number of online document collections increase and the number of users accessing these collections increase, it will be necessary to develop efficient distributed IR systems to access these collections. Distributed IR systems present unique problems to designers of distributed system due to the types of operations performed.

In this paper, we present the implementation of a prototype distributed information retrieval system. The prototype is based on Inquery, an existing and effective standalone IR system. We developed a detailed simulation model to test the performance of the distributed system under varying parameters and configurations. The simulator provides an easy and flexible platform for quickly performing different experiments in a controlled environment. To accurately model the actual system, the simulator uses many measurements obtained from the prototype system.

We present a series of experiments using the simulator to analyze the performance of the distributed IR architecture. We designed the experiments to test system utilization and identify potential bottlenecks under different workloads and system configurations. The experiments vary many parameters including the number of clients and Inquery servers, terms per query, distribution of terms in queries, and the number of documents retrieved. We will present the results of the experiments in a future report.

References

[BMC88] Jeffrey A. Brumfield, Janet Lynn Miller, and Hong-Tai Chou. Performance modelling of distributed object-oriented database systems. In *1988 International Symposium On Databases in Parallel and Distributed Systems*, pages 22–32, Austin, TX, December 1988.

[Bur90] Forbes 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.

[CCH92] James P. Callan, W. Bruce Croft, and Stephan M. Harding. The INQUERY retrieval system. In *Proceedings of the 3rd International Conference on Database and Expert System Applications*, Valencia, Spain, September 1992.

[CCW95] W. Bruce Croft, Robert Cook, and Dean 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.

[CEMW90] Janey K. Cringean, Roger England, Gordon A. Mason, and Peter Willett. Parallel text searching in serial files using a processor farm. In *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*, Brussels, Belgium, September 1990.

[CLC95] James P. Callan, Zhihong Lu, and W. Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th International Conference on Research and Development in Inforamtion Retrieval*, Seattle, WA, July 1995.

[Fox83] 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.

[Har92] D. Harman, editor. *The First Text REtrieval Conference (TREC1)*. National Institute of Standards and Technology Special Publication 200-217, Gaithersburg, MD, 1992.

[JO92] Byeong-Soo Jeong and Edward Omiecinski. Inverted file partitioning schemes for a shared-everything multiprocessor. Technical Report GIT-CS-92/39, College of Computing, Georgia Institute of Technology, July 1992.

[Jum93] J. Robert Jump. *YACSIM Reference Manual*. Rice University, version 2.1.1 edition, 1993.

[LZ93] Z. Lin and S. Zhou. Parallelizing I/O intensive applications for a workstation cluster: a case study. *Computer Architecture News*, 21(5):15–22, December 1993.

[MMNP87] Ian A. Macleod, T. Patrick Martin, Brent Nordin, and John R. Phillips. Strategies for building distributed information retrieval systems. *Information Processing & Management*, 23(6):511–528, 1987.

[MMR+90] T. Patrick Martin, Ian A. Macleod, Jody I. Russell, Ken Lesse, and Brett Foster. A case study of caching strategies for a distributed full text retrieval system. *Information Processing & Management*, 26(2):227–247, 1990.

[MR91] T. Patrick Martin and Jody I. Russell. Data caching strategies for distributed full text retrieval systems. *Information Systems*, 16(1):1–11, 1991.

[PW87] Cristine A. Pogue and Peter Willett. Use of text signatures for document retrieval in a highly parallel environment. *Parallel Computing*, 4:259–268, 1987.

[SA87] Patricia Simpson and Rafael Alonso. Data caching in information retrieval systems. In *Proceedings of the Tenth Annual International ACMSIGIR Conference on Research and Development in Information Retrieval*, pages 296–305, New Orleans, LA, 1987.

[SB88] Gerard Salton and Chris Buckley. Parallel text search methods. *Communications of the ACM*, 32(2):202–215, February 1988.

[SK86] Craig Stanfill and Brewster Kahle. Parallel free-text search on the connection machine system. *Communications of the ACM*, 29(12):1229–1239, December 1986.

[STW89] Craig Stanfill, Robert Thau, and David Waltz. A parallel indexed algorithm for information retrieval. In *Proceedings of the Twelfth Annual International Conference on Research and Development in Information Retrieval*, pages 88–97, Cambridge, MA, June 1989.

[TGM92] Anthony Tomasic and Hector Garcia-Molina. Caching and database scaling in distributed shared-nothing information retrieval systems. Technical Report STAN-CS-92-1456, Stanford University, December 1992.

[TGM93] Antony Tomasic and Hector 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 System*, San Diego, CA, 1993.

[Tom94] Anthony Tomasic. *Distributed Queries and Incremental Updates In Information Retrieval Systems*. PhD thesis, Princeton University, June 1994.

[VF95] Charles L. Viles and James C. French. Dissemination of collection wide information in a distributed information retrieval system. In *Proceedings of the 18th International Conference on Research and Development in Inforamtion Retrieval*, Seattle, WA, 1995.

[Wol92] Dietmar Wolfram. Applying informetric characteristics of databases to IR system file design, part i: Informetric models. *Information Processing & Management*, 28(1):121–133, 1992.

[Zip49] G.K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley Press, 1949.