Evaluating the Performance of Distributed Architectures for Information Retrieval Using a Variety of Workloads

BRENDON CAHOON and KATHRYN S. McKINLEY
University of Massachusetts
and
ZHIHONG LU
Village Networks

The 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 article, we explore how to achieve scalable performance in a distributed system for collection sizes ranging from 1GB to 128GB. We implement a fully functional distributed IR system based on a multithreaded version of the Inquery unified IR system. To explore the design space more fully, we also implement and validate a flexible simulation model. We measure performance as a function of system parameters such as client command rate, number of document collections, terms per query, query term frequency, number of answers returned, and command mixture. Our results show that it is important to model both query and document commands because the heterogeneity of commands significantly impacts performance. Based on our 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.

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. This work is supported in part by NSF Multimedia award CDA-9502639. Kathryn S. McKinley is supported by an NSF CAREER Award CCR-9624209. Any opinions, findings, and conclusions or recommendations expressed in this material are the authors and do not necessarily reflect those of the sponsors.

Authors' addresses: B. Cahoon and K. S. McKinley, Department of Computer Science, University of Massachusetts, Amherst, MA 01003; email: cahoon@cs.umass.edu; mckinley@cs.umass.edu; Z. Lu, Village Networks, 100 Village Court, Hazlet, NJ 07730; email: zlu@vill.com.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.4 [**Computer Systems Organization**]: Performance of Systems—*Performance attributes*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software

General Terms: Experimentation, Performance

Additional Key Words and Phrases: Distributed information retrieval architectures

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 disparate 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, increases. Distributed computing offers a solution to these problems.

Only recently have people published work on distributed architectures for information retrieval. The Very Large Collection track in the TREC conferences promotes the development of distributed and shared memory architectures for IR [Hawking and Thistlewaite 1997; Hawking et al. 1998]. Several researchers created distributed IR systems and demonstrated the feasibility of distributed architectures for information retrieval [Harman et al. 1991; Macleod et al. 1987]. However, it is not clear from these initial implementations how the systems will perform in practice, since, unlike the case for database systems, very little experimental investigation of distributed IR systems has been performed, and researchers have typically limited their collection sizes to at most 1GB. The focus of this article is to design large-scale distributed IR architectures by analyzing the performance of potential systems under a variety of workloads. In our work, we simulate configurations with up to 128 servers, each containing a 1GB text collection (i.e., 128GB of total text). We begin with a prototype implementation of a distributed information retrieval system using Inquery; an inference network, full-text information retrieval model [Callan et al. 1992]. Our system adopts a variation of the client-server paradigm that consists of clients connected to information retrieval engines through a central administration broker, as we illustrate in Figure 1.

In our experiments, we model a simple hardware configuration using a single CPU and disk. In our prototype, we use a multithreaded version of the Inquery server to obtain high machine utilization. We illustrate that using a single-threaded Inquery server does not effectively utilize CPU and disk resources; we achieve the best performance when we use four threads.

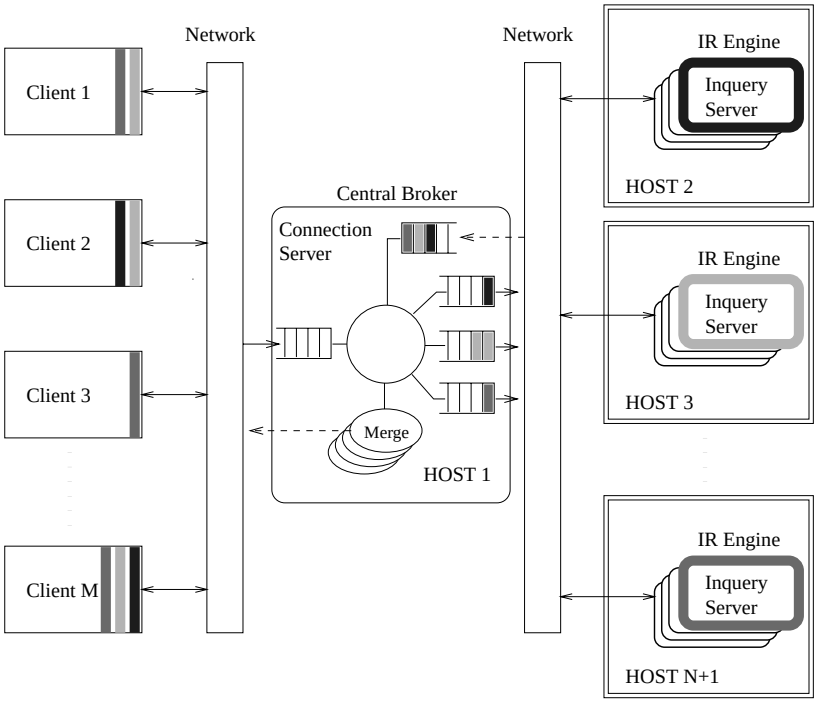

Fig. 1. Our distributed information retrieval system.

We use this configuration for our base system. A symmetric multiprocessor with multiple disks clearly improves performance as well [Lu et al. 1998], but, in this article, we focus on improvements in a system distributed over a network of workstations.

In the original Inquery system (not distributed or multithreaded), clients specify an Inquery text collection, connect to it, interact with it, and finally disconnect. In the prototype distributed system, clients search multiple text collections simultaneously. To build our prototype, we made the fewest possible changes to the underlying software. As illustrated in Figure 1, we use a single central broker, which we call the *connection server*, to administrate connections between clients and the IR engines. We refer to the IR engines as *Inquery servers*. The connection server 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 3. 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 4 presents this simulation model. We validate our model by comparing the performance of the simulator to the actual system for query operations.

We parameterize our simulation model by system features such as the number of CPUs, number of disks, and network utilization. This model allows us to investigate systems that vary from our implementation. In Section 5, we measure system performance including response time and

resource utilization for a variety of configurations. During our investigation we identify 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 [Callan et al. 1995a], of a standalone IR system.

Clients model the activities of realistic users during our experiments. Each client issues query commands, obtains summary information for query results, and retrieves documents. We simulate client think time by varying the client command rate; a slower command rate simulates a larger amount of think time. The different IR commands, i.e., query and document, have distinct effects on performance. For example, we show the architecture is able exploit parallelism in the summary information commands, which improves response times as the size of the architecture increases, up to eight or 32 Inquery servers.

In our main experiments, we explore an architecture that distributes multiple, independent text collections. As the number of machines increases, the amount of text to search also increases. We believe the multiple-text-collection architecture is similar to actual systems, especially for large architectures, and we present detailed experiments.[1] We demonstrate performance as a factor of the client command rate, number of text collections, the number of terms per query, the query term frequencies, number of answers returned, and the mixture of query, summary, and document retrieval operations. In order to generalize our results, we experiment with a range of values for these parameters and discuss their effects on system performance.

When clients issue short queries, response times are reasonable for many different numbers of clients and Inquery servers. However, the connection server is a bottleneck when (1) clients issue commands quickly and (2) there are a large number of Inquery servers. We show that adding a small number of brokers to manage the clients and Inquery servers is an effective strategy for improving performance and ensuring scalability for short queries. The architecture does not perform well when users enter long queries, due to bottlenecks in the Inquery servers. We show that adding more hardware, such as CPUs and disks, enables us to add more threads and alleviate this bottleneck.

We, thus, present a large-scale distributed IR architecture that provides efficient access to collection sizes up to 128GB by exploiting the heterogeneity among query and document operations. This article also identifies several important factors that system architects must consider when designing distributed IR systems.

We organize the rest of the article as follows. Section 2 presents the related work. We describe our distributed IR system in Section 3. In Section 4 we describe our simulation model. We list the simulator's timing values, validate the model, and describe the parameters we use in our

[1]We have examined architectures that distribute a single text collection among the Inquery servers in previous work [Cahoon and McKinley 1996].

experiments. Using the simulator, we measure the performance of the distributed IR system in Section 5. We also experiment with several changes in order to improve performance. Finally, we summarize our work in Section 6.

2. RELATED WORK

Related work on architectures for distributed IR systems include designing and evaluating architecture performance, data partitioning, caching, and multiprocessor systems. There is also a large volume of related work in database systems, so we only briefly discuss the differences between IR and database systems.

The work on architecture performance is most closely related to this 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 workloads and conditions. We base our distributed system on a proven, effective retrieval engine. We also model architectures with very large text collections—up to 128GB of data on 128 Inquery servers. Much of the prior work on distributed IR architectures has been restricted to small collections, typically less than 1GB and/or 16 servers. Participants in the TREC Very Large Collection track use collections up to 100GB, but they only provide query-processing times for a single query at a time, rather than a variety of workloads as we do here [Hawking et al. 1998]. It is clear that some industrial sites have collections even larger than ours, but they choose not to report on them in the literature to maintain their competitive edge.

Harman et al. [1991] show the feasibility of a distributed IR system by developing a prototype architecture and performing user testing to demonstrate usefulness. Unlike our research, which emphasizes performance, Harman et al.'s does not study efficiency issues, and they use a small text collection (i.e., less than 1GB).

Burkowski [1990] reports on a simulation study which measures the retrieval performance of a distributed IR system. The experiments explore two strategies for distributing a fixed workload across a small number of servers. The first equally distributes the text collection among all the servers. The second splits servers into two groups, one group for query evaluation and one group for document retrieval. This work is 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, vary the rate in which clients send commands, and use a more realistic user workload which uses both query and document retrieval operations.

Lin and Zhou [1993] implement a distributed IR system on a network of DEC5000 workstations using PVM (Parallel Virtual Machine) to coordinate work and network communication. They develop a new retrieval model that uses a variation of the signature file encoding scheme to map document

collections over the network. Their results show large speedup improvements due to parallelization. Our research extends this work by analyzing a distributed system to increase efficiency. Since we use a simulator, we are also able to run more experiments under diverse conditions. Another advantage of our work is that we base our distributed system on a proven and effective retrieval model rather than using a new model developed to expose parallelism.

Couvreur et al. [1994] analyze the performance and cost factors of searching large text collections on parallel systems They use simulation models to investigate three different hardware architectures and search algorithms including a mainframe system using an inverted-list IR system, a collection of RISC processors using a superimposed IR system, and a special-purpose machine architecture that uses a direct search. The focus of the work is on analyzing the trade-off between performance and cost. Their results show that the mainframe configuration is the most cost effective. They also suggest that using an inverted list algorithm on a network of workstations would be beneficial, but they are concerned about the complexity. Our research shows the effectiveness of using a network of workstations. We evaluate the performance of different configurations and explore improvements. Under realistic workloads, we present a scalable architecture for distributed information retrieval.

Hawking [1997] designs and implements a parallel information retrieval system, called PADRE97, on a collection of workstations. The basic architecture of PADRE97, which is similar to ours, contains a central process that checks for user commands and broadcasts them to the IR engines on each of the workstations. The central process also merges results before sending a final result back to the user. Hawking presents results for a single workstation and a cluster of workstations using a single 51–term query. A large-scale experiment evaluates query processing on a system with up to 64 workstations each containing a 10.2GB collection. The experiment uses four shorter queries of four to 16 terms. Our work varies several system parameters that Hawking does not, such as the query size and the rate at which clients issues queries, and we generate document commands and query commands.

Several studies have examined the use of caching in distributed IR systems [Martin and Russell 1991; Martin et al. 1990; Schatz 1990; Simpson and Alonso 1987; Tomasic and Garcia-Molina 1992]. 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 network. Furthermore, implementing a system that performs caching at the client requires a different retrieval engine.

Other researchers have investigated various data-partitioning schemes for distributed IR systems [Jeong and Omiecinski 1995; Macleod et al. 1987; Tomasic 1994; Tomasic and Garcia-Molina 1993]. We model a very

simple partitioning scheme in which each Inquery server maintains an independent set of documents. The documents may be part of the same larger text collection, but we evaluate the complete query on each Inquery server. Although we only consider one partitioning scheme, we extend previous results in several ways. Our experiments include results for both small and large configurations. Previous research has investigated only small configurations. 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 alters retrieval effectiveness.

The performance of our distributed system relies on the amount of parallelism we are able to exploit. Beginning in the late 1980's several researchers implemented IR systems on distributed-memory multiprocessor machines [Bailey and Hawking 1996; Cringean et al. 1990; Frieder and Siegelmann 1991; Pogue and Willett 1987; Stanfill and Kahle 1986; Stanfill et al. 1989]. We do not investigate multiprocessor systems in our experiments. Instead, we concentrate on a system in which each component of the architecture is independent. Our architecture is able to use a network of workstations instead of relying on special hardware. In a different work, we have also investigated the performance of an IR server on a shared-memory multiprocessor [Lu et al. 1998].

The TREC conference recently added the Very Large Collection track for evaluating the performance of IR systems on large text collections [Hawking and Thistlewaite 1997; Hawking et al. 1998]. To handle large collections, participants use shared-memory multiprocessors or distributed architectures. The experiments in TREC-7 use 49 long queries on a 100GB collection of Web documents. The Very Large Collection track summary presents precision and query-processing time results but does not provide significant details about each system. Two of the participants present details of their distributed systems elsewhere, but neither provides significant performance evaluations [Brown and Chong 1998; Burkowski et al. 1995].

There is also a large volume of work on architectures for distributed and parallel database systems including research on performance, e.g., see Bell and Grimson [1992], DeWitt and Gray [1992], DeWitt et al. [1986], Hagmann and Ferrari [1986], Mackert and Lohman [1986], and Stonebraker et al. [1983]. Although the fields of information retrieval and databases are similar, there are several distinctions which make studying the performance of IR systems unique. A major difference between database systems and information retrieval systems is structured versus unstructured data. The unstructured nature of the data in IR raises questions about how to create large, efficient architectures. Our work attempts to discover some of the factors that affect performance when searching and retrieving unstructured data. Another obvious contrast between the two systems is that relational database operators are highly parallelizable. It is unclear how to effectively parallelize natural-language queries. Furthermore, the types of common operations that are typical to database and IR systems are slightly

different. For example, the basic commands in an IR system—query evaluation and document retrieval—differ from those in a database system. In our IR system, we are not concerned with updates (commit protocols) and concurrency control, which are important issues in distributed database systems. We assume our IR system performs updates offline.

3. A DISTRIBUTED INFORMATION RETRIEVAL SYSTEM

This section describes the implementation of our distributed IR system. As shown in Figure 1, the distributed system consists of a set of clients, a connection server, and a set of Inquery servers. The distributed system enables multiple, simultaneous connections between clients and Inquery servers. The different components of the architecture communicate using a local-area network. Each component resides on a different host and operates independently of the others. In this section, we describe the functionality and interaction between the clients, the connection server, and the Inquery servers.

3.1 Clients

The clients are lightweight processes that provide a user interface to the retrieval system. Clients interact with the distributed IR system by connecting to the connection server, as illustrated in Figure 1. The clients initiate all work in the system, but perform very little computation. They can issue the entire range of IR commands but, in this article, we focus on *query*, *summary information*, and *document retrieval* commands.

A client sends query commands to the connection server. A **query command** consists of a set of words or phrases (terms). Either of the following occurs: the command specifies the list of Inquery servers to search, or the client allows the connection server to determine the appropriate collections to search. In our experiments, we assume the clients choose the Inquery servers. Query responses consist of a list of n document identifiers ranked by belief values which estimate the probability that the document satisfies the information need. The user specifies an appropriate value for n, or the system returns a default number of answers. We experiment using different values for n.

Summary information consists of the title and the first few sentences of a document. The **summary information command** consists of a set of document identifiers and their collection identifiers. The user may specify the number of summary entries to obtain, but, in our experiments, each client obtains summaries for 15 documents at a time. Our results show that the number of summaries has a significant effect on performance.

Clients may also retrieve complete documents by sending a **document retrieval command** to the connection server. The command consists of a document identifier and collection identifier. In response, the connection server returns the complete text of the document from the appropriate Inquery server.

In our prototype distributed IR system, each client issues a command and waits for the connection server to return the results before it issues another command. Users issue query and document commands. A client automatically issues the first summary information command when it receives a query response. A client issues additional summary information and document retrieval commands at the user's request. In Section 4.3, we describe a variation of the client processing which we use to support our simulation model.

3.2 Connection Server

The clients and Inquery servers communicate via a central broker called the connection server. The connection server is a lightweight process that keeps track of all the Inquery servers, outstanding client requests, and organizes responses from Inquery servers. The connection server continuously polls for incoming messages from clients and Inquery servers. The connection server is able to handle outstanding requests from multiple clients.

A client sends a command to the connection server, which forwards it to the appropriate Inquery servers. In our experiments, the command indicates the Inquery servers. Since each Inquery server only processes a limited number of requests at a time, the connection server maintains a queue of outstanding requests for each Inquery server, as illustrated in Figure 1. The connection server inserts commands from clients onto a queue if an Inquery server is currently processing the maximum number of commands. When the connection server receives an answer from an Inquery server, it forwards the next command on the corresponding queue to the Inquery server.

The connection server maintains intermediate results for commands specifying multiple Inquery servers. When Inquery servers return results, the connection server merges them with other results. After all the Inquery servers involved in a command return results, the connection server sends a final result to the client. Only query and summary commands may specify multiple Inquery servers. 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 from 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 and merge results (e.g., see Callan et al. [1995b], Voorhees et al. [1995], Viles and French [1995], Moffat and Zobel [1995], and Crowder and Nicholas [1995]). For a summary command, the connection server must read the message to determine which Inquery servers contain the appropriate

documents and then send a message to each Inquery server to acquire the summary information. The connection server merges the summary information responses and sends a single message back to the client with the summary information for all the documents. The connection server does not maintain intermediate results for document retrieval commands; it simply forwards documents immediately after receiving them.

There are several benefits of using a broker to manage messages between the clients and the Inquery servers. Although the broker in our distributed system is very simple, we can enhance it to provide metadata facilities, routing, security, and more precise merging. In the context of our IR system, metadata is a concise index of the text that each Inquery server maintains. For example, the system may use the metadata to perform automatic collection selection. The connection server may also use the metadata to merge the results from each Inquery server more effectively, based on the relative importance of each collection. Routing enables a broker to manage multiple versions of the same text collection. The broker routes commands to an appropriate text collection, based on availability and usage. Security becomes an issue for systems that want to protect sensitive data or allow certain clients access to certain text collections. The broker is also able to effectively manage the addition or deletion of Inquery servers. We can easily add any of these enhancements to our distributed system because a central broker manages actions between the clients and Inquery servers.

3.3 Inquery Servers

The Inquery server uses the Inquery retrieval engine to provide IR services such as query evaluation and document retrieval. Inquery is a probabilistic retrieval model that is based on a Bayesian inference network [Callan et al. 1992]. Inquery accepts natural-language queries or structured queries. Internally, the system stores text collections using an inverted file. Previous work demonstrates that Inquery is an effective retrieval system for large, full-text databases [Callan et al. 1995a].

For this work, we use a **multithreaded** Inquery server that runs on a host with one CPU and one disk. We are able to achieve high resource utilization by using a multithreaded server. We use the POSIX thread library package to implement the multithreaded server. With this configuration, we have shown in related work that the best performance occurs with four threads [Lu et al. 1998]. More than four threads does not yield any performance improvement, since the disk becomes overutilized. In Section 5.3.2, we briefly discuss improving the performance of our distributed system by adding more disks and threads.

Each Inquery server thread accepts a command from the connection server, processes the request, and returns the result back to the connection server. Each thread may only process one command at a time. Since we allow four threads, each Inquery server processes up to four commands at a time.

4. 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 create 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 [Jump 1993]. YACSIM contains a set of data structures and library routines to 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. Section 3 describes our model's basic architecture and functionality. The hardware resources we model include CPUs, disks, and the network. We model the clients, Inquery servers, and connection servers as different processes. Processes simulate the activities of the real system by requesting services from resources. The simulator is driven by empirical timing measurements obtained from our prototype. Section 4.2 shows that the simulator closely models the prototype implementation. We configure a simulation by defining the architecture of the distributed IR system using a simple command language. A configuration file contains the commands which the simulator reads at startup time.

Our technique for designing an environment for studying distributed information retrieval architectures is similar to the model that Brumfield et al. [1988] present. However, they simulate a distributed object-oriented database system, while our work focuses on IR systems. The main differences between database and IR systems are the type of commands and the structure of the data. IR commands include both query and document operations, and the data in IR systems are unstructured.

4.1 System Measurements

To accurately model an information retrieval system, we analyze the prototype distributed Inquery system and measure the resources used for each operation. We focus on CPU, disk, and network resources, and we do not measure memory and cache effects. Empirical measurements rather than an analytical model drive the activities performed in the simulator. The simulator uses the following parameters: query evaluation time, document/summary retrieval time, connection server time, and network latency. We obtain measurements from the prototype system using Inquery version 3.1 running on a DEC AlphaServer 2100 5/250 clocked at 250MHz with 1024MB of memory and 2007MB of swap space. We instrument the Inquery system to report time measurements during run time using the `ftime()`

Table I. Query Evaluation Timing Values (seconds)

$$query\_eval\_time = \sum_{i=1}^{n} eval\_term\_time(term_i) \times (1 + \sqrt{n-1} \times 0.075)$$

$$eval\_term\_time(term) = cpu\_time(term) + disk\_time(term)$$

$$cpu\_time(term) = \begin{cases} 0.009 + (3.2e-6 \times term) & : \quad term \leq 10000 \\ 0.025 + (1.08e-6 \times term) & : \quad 10000 \leq term \leq 100000 \\ 0.159 + (1.7e-7 \times term) & : \quad term \geq 100000 \end{cases}$$

$$disk\_time(term) = \begin{cases} 0.061 + (4.7e-6 \times term) & : \quad term \leq 10000 \\ 0.095 + (7.2e-7 \times term) & : \quad 10000 \leq term \leq 100000 \\ 0.011 + (8.2e-7 \times term) & : \quad term \geq 100000 \end{cases}$$

and `getrusage()` system calls. To obtain TCP network performance, we use TTCP from the US Army Research Laboratory.

We examine several different text collections and query sets to obtain system measurements. We examine TIPSTER 1, a large heterogeneous collection of full-text articles and abstracts [Harman 1992], a database containing the Congressional Record for the 103rd Congress [Croft et al. 1995], and a small collection of abstracts from *Communications of the ACM* [Fox 1983]. The number of documents in each collection is 510,887 (1GB of text), 43,378 (500MB of text), and 3204 (2MB of text), respectively.

4.1.1 *Query Evaluation Measurements*. The simulator uses a simple yet accurate model to represent query evaluation time. Based on our measurements of Inquery using our query sets, evaluation time is very strongly related to the number of terms per query and the frequency of each of the terms in the collection. Analysis of the TIPSTER 1 query set shows that there is a strong correlation of terms per query and query term frequency to evaluation time. Specifically, Pearson's correlation coefficient for both sets of data is 0.96 which indicates a strong positive relationship.[2]

Our query evaluation model is a function of the number of terms per query and the frequency of the individual query terms plus an additional overhead for the CPU and disk. Table I lists the query evaluation time values we use in our simulation. We model *eval_term_time*() as an increasing linear function of the term frequency. We also include a small CPU and disk overhead to account for the time Inquery spends combining the results of each of the terms. Without the overhead amount, we underestimate the evaluation time of long queries. We obtain the *eval_term_time*() function and the overhead value by using the TIPSTER 1 query set and text collection to measure the evaluation time for terms of different frequencies.

[2]Pearson's correlation coefficient ranges between −1 and +1. Values −1 and +1 indicate a perfect linear relationship and occur when all points lie on a downward or upward sloping line, respectively. A value of 0 indicates there is no linear relationship.

Recall that the evaluation time has a strong positive correlation with the term frequency. We create *eval_term_time*() by fitting the term evaluation time measurements with three straight lines using a least-squares method. The first line covers term frequencies up to 10,000; the second lines covers term frequencies 10,000–100,000; and the third lines covers term frequencies above 100,000. The model is more accurate using three lines rather than a single straight line for all the data. We divide *eval_term_time*() into CPU and disk access time.

Using our model, the time to evaluate a single term ranges from 0.07 seconds for a term appearing once to 0.72 seconds for a term appearing 554,568 times (the maximum term frequency in TIPSTER 1 after removing stopwords). Disk access time is typically larger than CPU processing time, due to the speed of the Alpha processor. Our measurements show that disk access time accounts for 32% to 90% of the total evaluation time with an average of 73%. For slower processors, CPU processing time may be larger than disk time. We notice the disk time is especially large for very small term frequencies or very large term frequencies (i.e., $\leq$ 10,000 or $\geq$ 450,000).

The query evaluation model is slightly different than the model we used in previous work [Cahoon and McKinley 1996]. The new *eval_term_time*() function is more accurate, and we did not include any overhead in the old model. In practice, our simulator overestimates query evaluation times, since our model assumes a query command returns the entire list of relevant answers. In our experiments, the Inquery servers only return the top n answers. For small values of n, the actual system performs slightly less processing. We do not include this effect in our model, since it is small and difficult to characterize.

4.1.2 *Document Retrieval Measurements*. We measure Inquery to determine the amount of time it takes to retrieve a document. For our text collections, the retrieval time is variable, and there is not a strong correlation between document size and retrieval time. The low correlation is due to very quick retrieval times. In general, the sizes of the documents in our text collection are not very large, which results in fast retrieval times. In our collections, the average size of a document in the TIPSTER 1, Congressional Record, and the Fox CACM collection is 2.3KB, 11.7KB, and 0.5KB, respectively.

The simulator represents the document retrieval time for an Inquery server as a constant value, 0.027 seconds, which is the average document retrieval time for 2000 randomly selected documents from the TIPSTER 1 collection. The simulator represents retrieval time as CPU processing time plus disk access time. Our measurements show that disk access time accounts for 87% of the total retrieval time on average. To ensure our simulator is accurate for our experiments, the average document size returned by an Inquery server is the same as the average size in the TIPSTER 1 collection, i.e., 2.3KB. The simulator also uses the document retrieval time to compute the summary information retrieval time. We

Table II. Connection Server Time Values

Message Encoding and Decoding Time (seconds)

$$query() = 0.075$$

$$summary(nsums) = 0.017 + (0.025 \times nsums)$$

$$document\_list(ndocs) = 0.85 + (0.0048 \times ndocs)$$

Processing Time (seconds)

$$command(nservers, cmd) = \begin{cases} 0.6 & : \ cmd = document \\ 1.3 + (0.17 \times nservers) & : \ otherwise \end{cases}$$

$$servers\_result(cmd) = \begin{cases} 0.7 & : \ cmd = document \\ 0.13 & : \ otherwise \end{cases}$$

Merging Time (microseconds)

$$query(nanswers) = 26 + (2 \times nanswers)$$

$$summary(nsums) = -75 + (24 \times nsums)$$

implement the summary information operation as a series of document retrieval operations. For each summary entry, an Inquery server reads a complete document. However, the Inquery servers only return the summary portion of the document. In our experiments, the average size of a document summary is 120 bytes.

4.1.3 *Connection Server Time*. We divide the time in the connection server into three categories; the time to encode and decode messages, the processing time for handling messages, and the time to merge results. Table II lists the formulas we use to compute the different values. The functions return seconds, except for the merge time function, which returns microseconds.

Each time a message arrives, the connection server must decode the message to determine an appropriate action. The connection server also creates new messages and must encode the data into the message format. The time to decode a query command, $query()$, is a constant value. The $summary(nsums)$ function is the time to decode and encode a summary information command; the time depends on the number of summaries, which is 15 in our experiments. The $document\_list(ndocs)$ function is the time to decode query results (from the Inquery servers) and the time to encode the final results that the connection server sends to the client; the value depends on the number of results, or documents.

The processing time for handling messages depends on who sends the message and the message type. A command from a client is either a query, summary, or document command, which the connection server places on the queue for the appropriate destination Inquery server(s). The connection server processing depends on the type of the command. A document retrieval command is only sent to one Inquery server and takes a constant

amount of processing. A query or summary command is sent to multiple Inquery servers, so the processing time is a function of the number of servers.

Similarly, messages from the Inquery servers contain different types of results. We represent the processing time using constant values. The processing time for a document response is slightly higher than a query or summary response because the document response is larger. The connection server also spends time merging query and summary results. The time to merge query and summary results depends on the number of answers or summaries an Inquery server returns. We model the merge times as linear functions of the number of results.

4.1.4 *Network Time*. We represent network time as 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 on the size of the message and the bandwidth of the network. We obtain the sender and receiver overhead times using TTCP by measuring the time to send messages between two AlphaServer 2100 5/250 workstations connected by a lightly loaded 10Mbps Ethernet. We list the equations for the sender overhead, the receiver overhead, and the network latency (each function returns its result in seconds):

$$sender\_time(size) = 0.02 + (0.00003 \times size)$$

$$receiver\_time(size) = 0.02 + (0.00005 \times size)$$

$$network\_time(size) = size \times 8/speed$$

In each function, $size$ represents the number of bytes in the message, and $speed$ is the speed of the network in bits per second.

4.2 Validation

We validate the simulator's query evaluation times against the actual implementation, using a configuration consisting of a single client, a multithreaded Inquery server, and a connection server. Each of the components runs on a separate host. We do not validate the document retrieval times, since our measurements show the actual document retrieval time is variable due to the quick access times. As we mention in Section 4.1, the time to retrieve documents in the simulator is a constant value.

We validate our simulator by creating artificial queries and comparing the results from running the queries on the actual system to the results from our simulator. We randomly generate queries which correspond to the types of queries our simulator generates during our experiments (see Section 4.3). The two parameters we use to generate queries are the number of terms per query and the query term frequencies. We generate

Table III. Percentage Difference of Average Response Times between the Implementation and Simulator

Query Type	Arrival Rate λ per Second	Number of Threads					
		1	2	3	4	8	16
Short	0.5	1.5%	0.5%	-2.5%	2.5%	-0.8%	1.5%
	1	13.7%	3.0%	1.9%	-1.5%	1.0%	-0.3%
	5	**27.2%**	17.6%	3.8%	-0.4%	2.1%	1.0%
Medium	0.5	19.8%	9.0%	4.4%	1.9%	1.2%	0.5%
	1	26.0%	12.0%	2.5%	**-3.9%**	1.9%	-0.4%
	5	21.5%	15.7%	7.2%	8.8%	5.0%	4.1%
Long	0.5	15.8%	6.6%	-3.8%	-0.4%	0.4%	-0.5%
	1	10.1%	1.8%	-2.1%	1.8%	1.0%	1.7%
	5	12.7%	10.3%	1.7%	2.4%	3.9%	3.1%
Average		16.5%	8.5%	1.5%	*1.2%*	1.7%	1.2%

100 short queries, 100 medium-length queries, and 100 long queries. We do not generate queries with multiple occurrences of the same term, since our model does not account for these types of queries.

Table III shows our validation results using the short, medium, and long generated queries as we vary the number of threads. Positive numbers indicate the simulator overestimates the actual system. On average, the simulator is 5.1% slower than the actual system, and the difference ranges from 3.9% faster to 27.2% slower. The difference between the actual system and the simulator tends to decrease as the number of threads increases and as the query arrival rate decreases, because we slightly overestimate the query evaluation time. In general, the simulator matches the actual system closely, but we are not able to accurately model every query. Due to our simple model it is difficult to reduce the amount of variation between our simulator and the actual system times. Given that our simulator evaluates most queries accurately, especially for four threads (average 1.2% error), and it usually conservatively overestimates, we believe it is not worth the added complexity to change the model.

4.3 Experimental Parameters

In this section, we describe the parameters we use in our simulation experiments. Table IV presents the parameters, their values, and abbreviations. We describe each parameter in more detail below.

4.3.1 *Command Arrival Rate (λ).* In our experiments, we model different numbers of clients by varying the rate at which we send commands to the connection server. We simulate a small or large number of clients by slowing down or speeding up the command arrival rate. The client command rate also determines the amount of "think time," i.e., the amount of time a client spends reading a document. A slow command rate simulates a

Table IV. Experimental Parameters

Parameters	Abbrev.	Values
Command Arrival Rate Poisson Dist. (avg. commands/sec.)	λ	0.1 0.5 1 2 4 6 8 10
Inquery Servers	IS	1 4 8 32 64 128
Terms per Query (avg.) Shifted Neg. Binomial Dist.	TPQ	2 12 27
Query Term Frequency Dist. from Queries	QTF	Obs. Dist. Low Skew High Skew
Answers Returned Constant Values	AR	15 100 1000
Command Mixture query:summary:document	CM	1:1:1 1:2:2 1:3:3 1:4:4

large think time. We model the command rate using a Poisson distribution. Table IV lists the different distribution averages that we use in our experiments. The slowest rate averages one command every 10 seconds, and the fastest rate averages 10 commands per second.

4.3.2 *Number of Inquery Servers (IS).* We experiment with both small and large system configurations. Measuring the effect of increasing the number of Inquery servers provides insight into distributed IR architectures by identifying bottlenecks and helping to understand system utilization and scalability. We experiment with up to 128 Inquery servers (128GB of data).

4.3.3 *Terms per Query (TPQ).* Table IV shows the three different average query lengths we use in our experiments. We obtain two of the values from the 103rd Congressional Record (two terms) and TIPSTER 1 query set (27 terms). We also use an intermediate value (12 terms), since the two query sets have very different characteristics. We observe that a shifted negative binomial distribution closely matches the distributions in the two query sets. The characteristics of a shifted negative binomial distribution are the number of trials, n, the probability of success, p, and the amount of shift, s. Table V shows the values we use in our experiments.

Wolfram [1992] also uses a shifted negative binomial distribution to model terms per query. However, we use slightly different values for n, p, and s in order to obtain better estimates of our data sets.

4.3.4 *Distribution of Terms in Queries (QTF).* Zipf [1949] documented the widely accepted distribution of term frequencies in text collections. In contrast, researchers do not agree on a commonly accepted distribution for term frequencies in queries [Wolfram 1992]. Figure 2 shows the distribution of our query sets. The query term frequency distributions for the query

Table V. Values Used in Experiments

Description	Query Set	Average Length	n	p	s
Short Queries	103rd CR	2	4	0.8	1
Medium Queries	N/A	12	2	0.77	5
Long Queries	TIPSTER 1	27	2	0.1	10

sets are similar, but the distributions do not closely match a well-known distribution. 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 toward terms occurring less frequently and a distribution that is skewed toward terms occurring more frequently. To obtain the low and high skewed distribution we multiply the values in the observed distribution by 0.15 and 2, respectively. Skewing the observed distribution has the effect of shifting the term frequency values in Figure 2 to either the left or the right.

4.3.5 *Number of Answers Returned (AR).* For each query, the IR system returns a sorted list of matching documents to the clients. The list contains document identifier numbers instead of the complete text. The number of answers returned affects network traffic and processing by the connection servers. The three different constant values we use are 15, 100, and 1000 answers.

4.3.6 *Command Mixture (CM).* We simulate user activity by issuing query, summary information, and document retrieval commands. We vary the mixture of the commands to model different user patterns. Since we do not have statistics on user behavior, we chose four sets of values that we feel are representative. Table IV shows the different ratios we use in our experiments. For example, *1:1:1* means that the ratios of commands are the same. The *1:4:4* notation means that for each query command the client issues four summary and four document commands.

4.3.7 *Simulation Output.* During each simulation execution, we measure different performance statistics such as the query, summary, and document responses times. We also measure various utilization values including CPU, disk, network, connection server, and Inquery server utilization. For each series of experiments, we display the corresponding values of the parameters and the abbreviations listed in Table IV.

5. EXPERIMENTS AND RESULTS

In this section, we evaluate the performance of our distributed IR architecture and present the results. Running experiments and measuring performance as a function of the command arrival rate, number of text collections, terms per query, query term frequency, answers returned, and the types of commands allows us to gain a better understanding of system performance under different workloads. To our knowledge, previous researchers have not examined the performance of a distributed IR system by

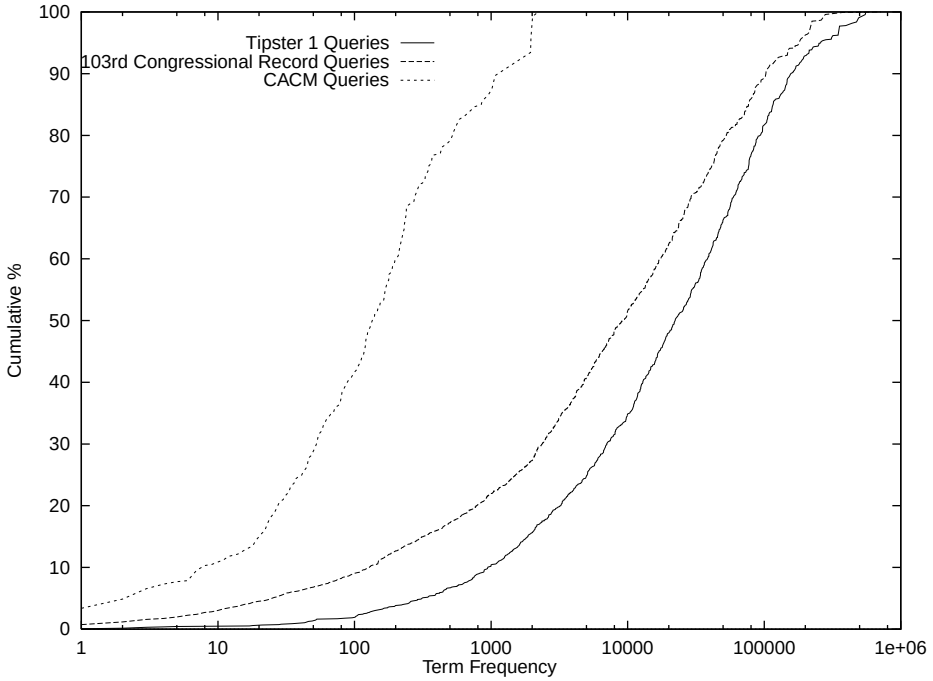

Fig. 2. Query term frequency distributions.

varying these values. We show that each feature has a significant impact on performance, and we discuss the effect of each feature. Designers of distributed architectures for IR must account for each parameter in order to make appropriate trade-offs that maximize the performance of a particular architecture.

In Section 5.1, we show the performance when a single client searches a single text collection. We present results using the multithreaded Inquery server (with four threads) and the single-threaded Inquery server. As we expect, using a multithreaded server achieves better performance than the single-threaded server, due to a higher utilization of resources. The multithreaded server is able to provide reasonable response times until the command rate exceeds four or six commands/second.

In Section 5.2, we present results on an architecture that manages multiple, distinct text collections—up to 128 collections. In this case, each multithreaded Inquery server maintains a different 1GB database, and the clients evaluate queries on a subset of the available collections. Thus, the clients potentially search up to 128GB of data. We vary the command arrival rate, the number of text collections, terms per query, query term frequency, answers returned, and the ratios of commands sent by the clients. We show that performance improves as we increase the number of Inquery servers, up to eight or 32 servers, even though clients search more

information. The improvement is due to parallelism in the summary information commands.[3]

Our results show that for short, realistic queries, several architecture configurations scale with the number of processors and degrade gracefully as the number of clients (work) increases. Our results illustrate that the system achieves good performance under varying conditions if we can maintain a balance between connection server and Inquery server utilization. Some conditions do cause the system performance to rapidly deteriorate. For example, the architecture does not perform well for very large configurations or when either the utilization of the connection server or Inquery servers is high. We often see poor utilization for long user queries.

Varying query term frequency changes the Inquery server utilization when the Inquery servers are the bottleneck. Unfortunately, the overall system performance is still poor for large configurations. We demonstrate that varying the number of answers returned affects network utilization. Our results illustrate that increasing the number of summary and document commands relative to the number of query commands improves response time, since summary and document operations are less computationally intensive than query operations.

In Section 5.3, we change the architecture in order to improve the performance of our IR system. We show how to improve the performance by reducing the bottlenecks in the connection server and Inquery server. In Section 5.3.1, we experiment with adding connection servers to introduce more parallelism. We eliminate the connection server bottleneck for all configurations with only four connection servers. We also experiment with moving the response merging from the connection server to the clients. Unfortunately, this change does not improve performance because the amount of time spent merging is small, while the cost to send more messages decreases the benefit of moving the merging functionality. In Section 5.3.2, we describe how to improve Inquery server response times by adding hardware; we are able to increase the number of threads by adding disks. Our results show that simply adding another disk significantly improves performance when the Inquery server is a bottleneck.

Finally, in Section 5.4, we simulate a large-scale architecture that achieves high performance given our experimental parameters. Based on our results in Sections 5.2 and 5.3, we choose the parameters that provide the best results. We show that this architecture is able to provide response times under 10 seconds for short, medium, and, in some cases, long queries.

Unless otherwise stated, in each of our experiments, the clients, connection server, and Inquery servers operate as we describe in Section 3. 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.

[3]In previous work we have also run experiments when partitioning a single 1GB text collection among all the Inquery servers [Cahoon and McKinley 1996]. The results are very similar to those in Section 5.2.

Table VI. Command Response Times in Base Configuration (seconds)

Command	Number of Threads	Command Rate (Average per Second) 0.1	0.5	1	2	4	6	8	10
Query	1	0.49	0.52	0.58	0.87	27.15	68.24	89.01	104.53
	4	0.49	0.50	0.53	0.65	1.53	25.60	44.20	57.20
Summary	1	0.44	0.48	0.54	0.81	28.02	68.70	89.32	105.02
	4	0.44	0.47	0.50	0.61	1.57	26.02	44.78	57.66
Document	1	0.05	0.08	0.16	0.43	27.17	68.08	88.97	104.66
	4	0.04	0.08	0.13	0.27	1.24	25.51	44.19	57.33

A local-area network with a bandwidth of 10Mbps connects the machines. Each of the Inquery servers maintains a 1GB database.

5.1 Base Configuration

In this section, we show the performance of a system containing one client connected to one Inquery server. The purpose of this experiment is to provide a baseline for comparison. We compare results of an Inquery server with one and four threads to illustrate the benefit of using a multithreaded server to improve response time and utilization. Previous work demonstrates that, for one disk, more threads do not improve performance [Lu et al. 1998]. Thus, our base configuration uses four threads.

Table VI compares the command response times for an Inquery server using 1 and 4 threads. As the command rate increases, the response times for the multithreaded server become much better than the single-threaded server. We first see a large difference between 1 and 4 threads when the command rate increases from 2 to 4 commands/second. For example, Table VI shows that the query response time increases by a factor of 31.2 when the command rate increases from 2 to 4 commands/second. When using 4 threads, we see an increase of only 2.35. The results show that the multithreaded server always achieves response times under 1 minute. However, there are large increases in response times once the command rate exceeds 4 commands/second. We see a factor of 16.7 increase in query response time when the command rate increases from from 4 to 6. The table also shows that the document response times become very similar to the query and summary command times as the command rate increases. We explain each of these effects by showing the utilization values of the Inquery servers as the command rate increases.

Figures 3 and 4 show utilization values using the single-threaded and multithreaded Inquery server, respectively. The figures show the utilization of the server process, the disk, and the CPU.[4] For a single thread, the

[4]The Inquery server utilization is equal to the disk plus the CPU utilization values. In our system with one disk and one CPU, the maximum utilization of the Inquery server process is 200%.

Table for Figures 3 and 4

IS	TPQ	QTF	AR	CM
1	2	Obs.	1000	1:1:1

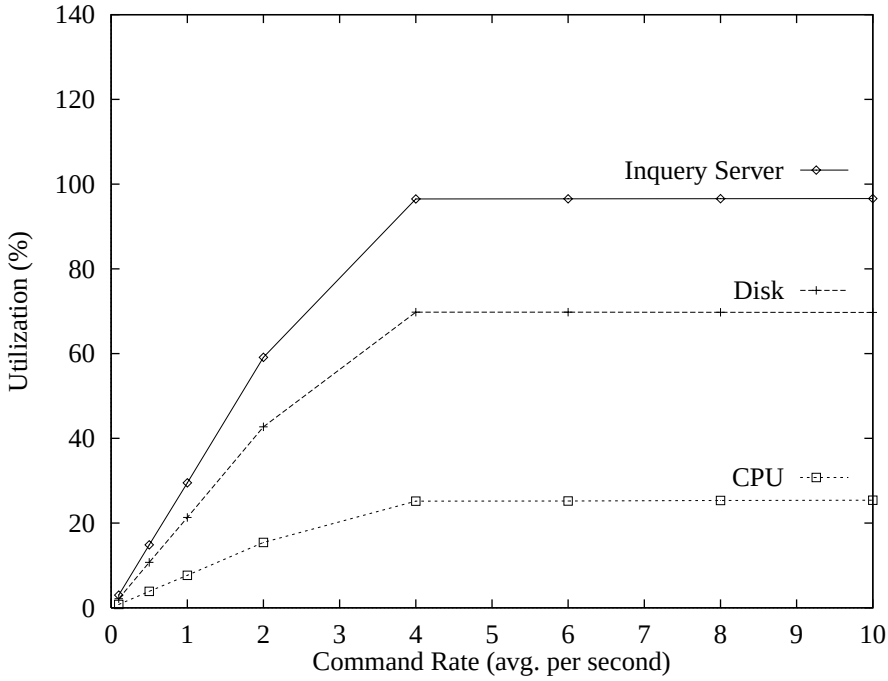

Fig. 3. Utilization of single-threaded server.

utilization of the server reaches the maximum value after four commands/second. Disk usage accounts for most of the time in these configurations. Figure 4 illustrates that disk utilization approaches 100%, which limits efficiency. For example, we see that CPU utilization reaches a maximum of 36% because the Inquery server waits for the disk and is unable to fully utilize the CPU. Results from other experiments show that the maximum CPU utilization is approximately 36% regardless of query length. When users issue longer queries, CPU and disk utilization reach their maximums more quickly.

5.2 Searching Multiple Text Collections

In this section, we measure the performance of a distributed IR system that maintains multiple 1GB text collections. In this configuration, each client selects a random subset of the available collections to search and searches half of the available collections on average. Each collection has an equal chance of being chosen. The workload increases both as a function of the number of Inquery servers and the command arrival rate (i.e., the total amount of information to search increases with the number of Inquery

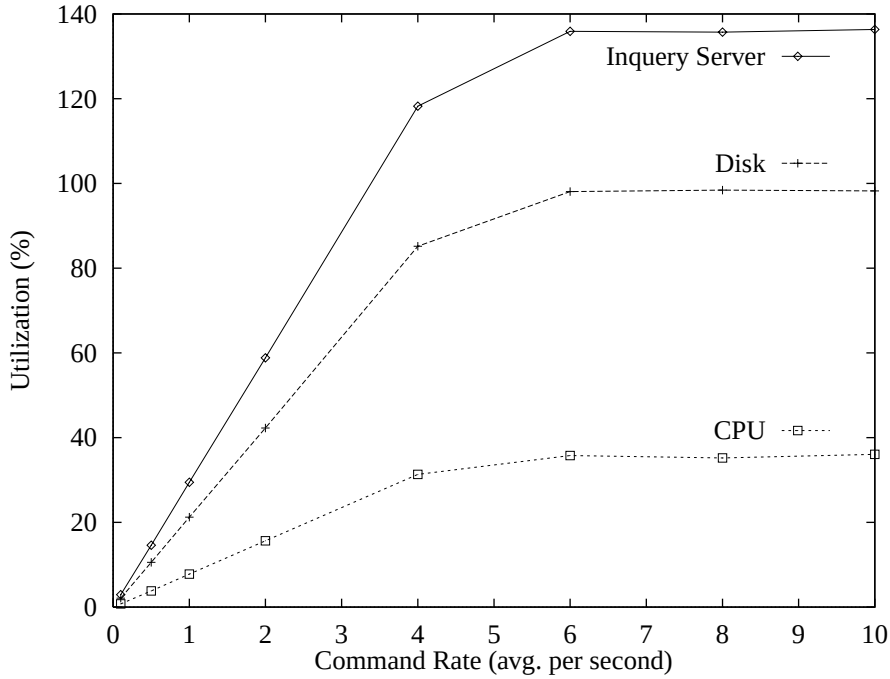

Fig. 4. Utilization of multithreaded server.

servers). This workload mimics the scenario when the connection server automatically selects an appropriate subset of the available collections to search. It also models the situation when the user manually chooses a subset of the collections to search.

In the following sections, we present results showing the effect of our experimental parameters on the performance of the architecture. These parameters include the command arrival rate, number of text collections, terms per query, query term frequency, answers returned, and number of query, summary, and document commands. We list the values for each parameter in a table accompanying each of the figures which appear in this section.

5.2.1 *Effect of Terms per Query*. In this section, we discuss the performance effect of using short, medium, and long queries. We see that query length has a dramatic impact on system performance. The results in this section show that bottlenecks sometimes occur in the connection server and the Inquery server. In general, system performance is poor when users issue medium or long queries. However, user queries tend to be very short, and our architecture is able to efficiently handle many configurations.

5.2.1.1 *Short Queries (TPQ = 2)*. Figure 5 shows the query response time when clients issue short queries. We vary the number of Inquery servers, and we vary the command rate. Figure 5 illustrates an improvement in query response time as we add Inquery servers and exploit

IS	QTF	AR	CM
2	Obs.	1000	1:1:1

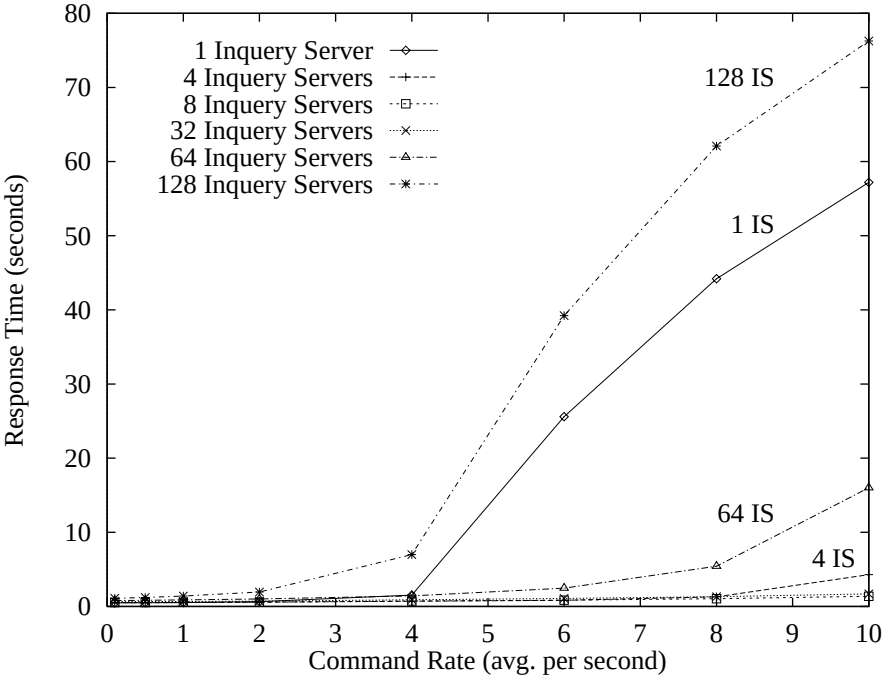

Fig. 5. Query response time (short queries).

parallelism, up to eight or 32 Inquery servers.[5] Going from one to eight Inquery servers improves performance by a factor of 42, when clients issue 10 commands/second. However, when the system contains more than eight or 32 Inquery servers, performance begins to degrade. This result is interesting because the amount of text to search increases as the number of Inquery servers increases. For example, when the number of Inquery servers doubles, a client potentially searches twice as much information.

More detailed measurements reveal the improvement stems from increased parallelism during summary information retrieval. Recall that a single summary information operation retrieves 15 summaries. A system with one Inquery server contains all the documents on the same machine. However, a system with multiple Inquery servers distributes the documents among the available Inquery servers. The 15 summary entries may reside on different Inquery servers resulting in parallel access of the summary information. In the best case, each of the 15 entries is located on different Inquery servers.

[5]The response times for 32 Inquery servers are only slightly higher than eight Inquery servers.

Table VII. Response Times for Short, Medium, and Long Queries (seconds)

Query Length	Command	Command Rate 0.1	0.5	1	2	4	6	8	10
Short (8 IS)	Query	0.5	0.5	0.6	0.6	0.7	0.8	1.0	**1.4**
	Summary	0.2	0.2	0.3	0.3	0.4	0.6	0.8	**1.1**
	Document	0.04	0.05	0.07	0.1	0.2	0.3	0.47	**0.67**
Short (128 IS)	Query	1.1	1.2	1.4	1.9	**7.0**	39.2	62.1	76.3
	Summary	0.1	0.2	0.4	0.9	**6.1**	37.6	59.4	72.5
	Document	0.0	0.1	0.3	0.7	**5.3**	34.4	54.6	66.2
Medium (128 IS)	Query	3.8	4.6	5.8	**11.1**	54.0	94.1	110.8	119.9
	Summary	0.3	1.4	3.0	**8.1**	46.7	86.4	103.7	112.1
	Document	0.2	0.8	1.7	**4.3**	31.5	68.9	85.7	94.4
Long (128 IS)	Query	**9.3**	15.6	41.9	192.1	304.7	342.6	387.9	399.9
	Summary	1.2	**7.7**	30.8	170.6	280.8	320.5	360.6	373.7
	Document	0.7	**3.7**	15.3	128.1	236.4	274.4	311.0	322.5

As Figure 5 shows, the response time degrades very slowly as we increase the command rate for eight or 32 Inquery servers. For example, as we increase the command rate by a factor of 100 (0.1 to 10 command/second), query response time degrades by a factor of 2.6 for eight or 32 Inquery servers. For the other combinations of Inquery servers, the response time rapidly increases as the command rate increases. Bottlenecks in either the connection server or the Inquery server cause this decrease in performance.

Figures 6 and 7 show the utilization of the connection server and Inquery server when clients issue short queries. The graphs show opposite effects; as the command rate increases, connection server utilization increases, and Inquery server utilization decreases. The system achieves the best performance when neither utilization values are high. It is apparent that as the system size increases the connection server becomes a bottleneck, causing performance to degrade. The problem is due to the large number of messages sent to the connection server. Although the connection server processes messages quickly, it is unable to keep up with the quantity of messages. The maximum number of results sent to the connection server from the Inquery servers at any single point in time is $|Inquery\ servers| \times |threads|$.

The first two rows of Table VII show the response times for query, summary, and document commands for short queries when the system contains eight and 128 Inquery servers. Using short queries, the architecture achieves the best response times using eight Inquery servers. For example, the largest query response time is 1.4 seconds when clients average 10 commands/second, which is quite reasonable. We also show the response times when the system contains 128 Inquery servers to compare the results for medium and long queries. Using 128 Inquery servers, performance begins to degrade after four commands/seconds. Table VII also

Table for Figures 6 and 7

TPQ	QTF	AR	CM
2	Obs.	1000	1:1:1

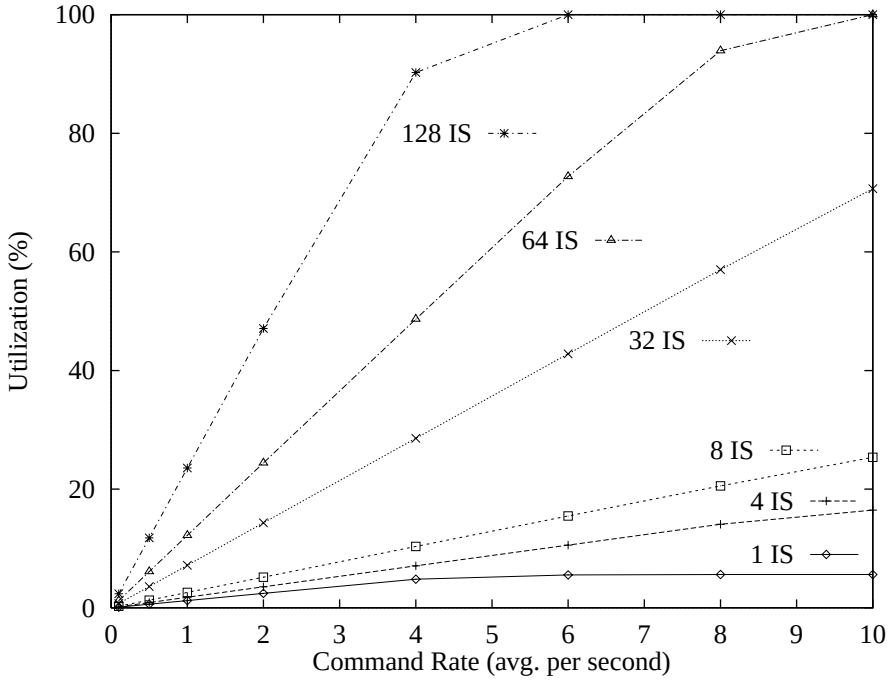

Fig. 6. Connection server utilization.

illustrates that the summary and document response times grow at a similar rate as the query commands. This trend occurs in each of the experiments we perform. Thus, we will only report query response times throughout the rest of the article.

5.2.1.2 *Medium Queries (TPQ = 12).* System performance degrades when clients issue medium-length queries. Unlike the situation when clients issue short queries, the response times do not initially improve and then degrade after eight Inquery servers. Instead, response times improve as we add Inquery servers, and the system achieves the best performance with 128 Inquery servers. The best response time when clients issue 10 commands/second is 119.9 seconds, which is larger than the best time for short queries by a factor of 86. However, the third row of Table VII shows that the system achieves a response time of 11 seconds or less with a command rate less than two commands/second. The reason for the poor performance when clients issue commands quickly is that the Inquery servers are unable to process commands fast enough. The main bottleneck in the Inquery server is the disk, which reaches 100% utilization after the command rate exceeds two commands/second.

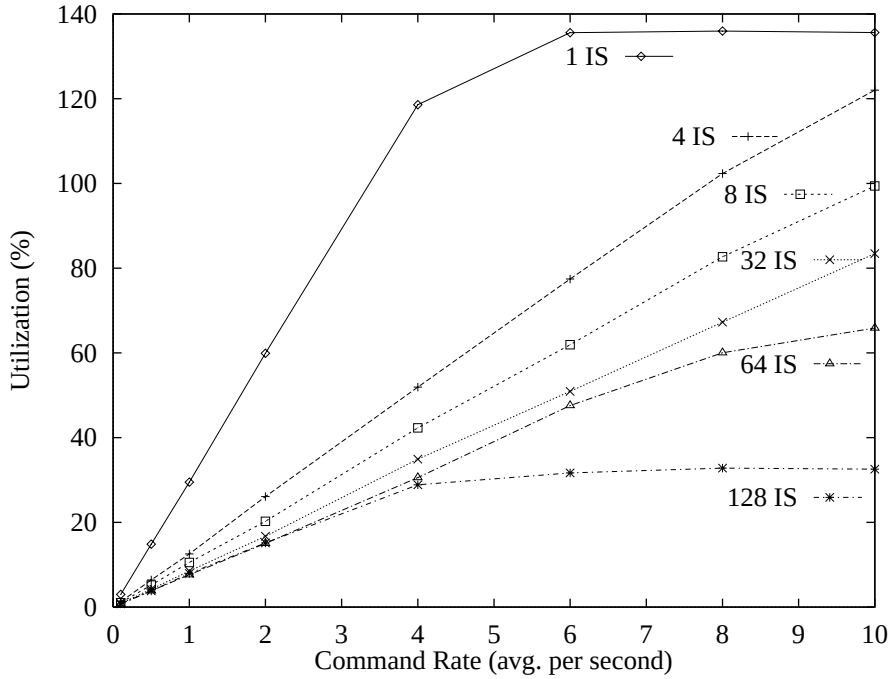

Fig. 7. Average Inquery server utilization.

5.2.1.3 *Long Queries (TPQ = 27).* When clients issue long queries, the response times become poor once clients average greater than one command every two seconds. As we increase the command rate from 0.1 to 10 on a system with 128 Inquery servers, the performance degrades by a factor of 43 (compare this with factors of 2.8 and 31.5 for short and medium queries, respectively). The poor performance is due to the high utilization of the Inquery server's disk. The bottom row of Table VII shows that performance is noticeably worse when clients issue medium-length queries. For example, when clients issue two commands/second, the response time for long queries increases by a factor of 17.3 and 320 over medium and short queries, respectively. Similar to the situation with medium queries, the system performance improves as the number of Inquery servers increases. Unfortunately, the response times are poor, except when clients issue commands at an extremely slow rate.

5.2.2 *Effect of Query Term Frequency*. Varying the query term frequency only has a noticeable effect on performance when the Inquery servers are the bottleneck. The reason is that queries containing terms which occur more frequently in the text collection evaluate more slowly. The opposite is true of queries containing terms which occur less frequently. Our results reflect this difference in evaluation time. We see little, if any, changes in performance due to the query term frequency when the connection server is the bottleneck.

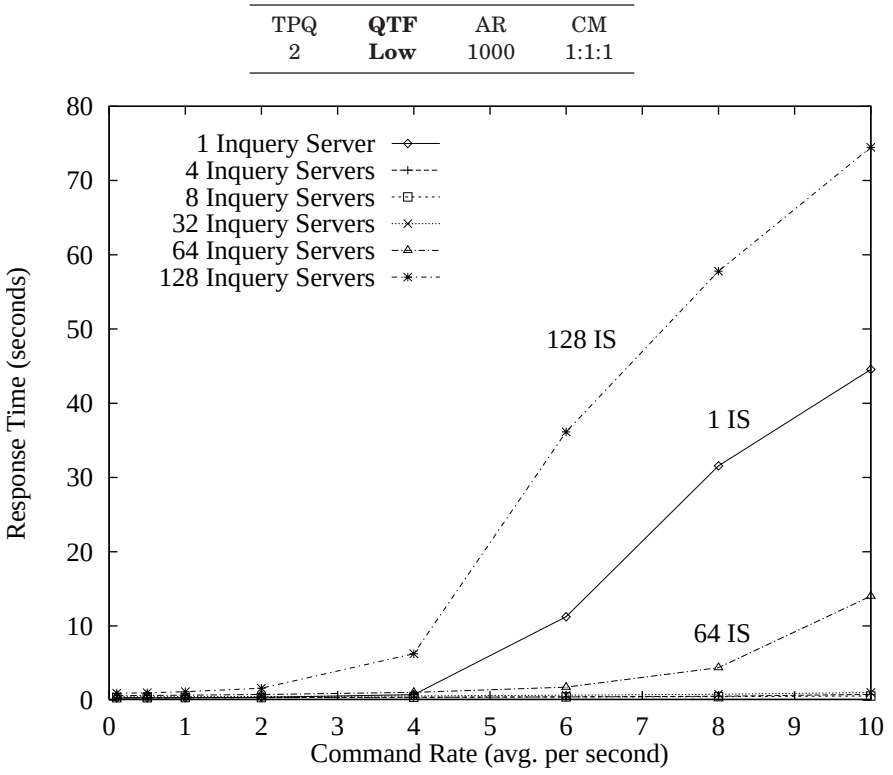

Fig. 8. Query response time (low skew).

In Figures 8 and 9, we present the query response times for low and high skewed query term frequencies when clients issue short queries. As we expect, the figures show that we obtain better performance for low skewed queries and worse performance for high skewed queries when we compare the results to the observed distribution (see Figure 5). Compared to the results in Figure 5, using low skewed terms decreases the query response time by 36% to 58% in a configuration with eight Inquery servers. In the same configuration, using high skewed terms increases query response time by 24% to 121%. The performance differences are larger as the command rate increases. Although we do not show graphs for medium and long queries, our experiments show similar performance trends as the results in Figures 8 and 9. Low skewed queries improve performance, and high skewed queries degrade performance. Unfortunately, the performance is still quite poor.

We do not see any differences in performance once the connection server becomes the bottleneck. In configurations involving 64 or 128 Inquery servers, the connection server is the bottleneck when clients issue commands at a relatively fast rate (see Figure 6). In this situation, the response times when using the low, observed, or high distribution are similar. For example, when clients issue 10 commands/second, the query response times in a configuration with 64 Inquery servers for the low,

TPQ	**QTF**	AR	CM
2	**High**	1000	1:1:1

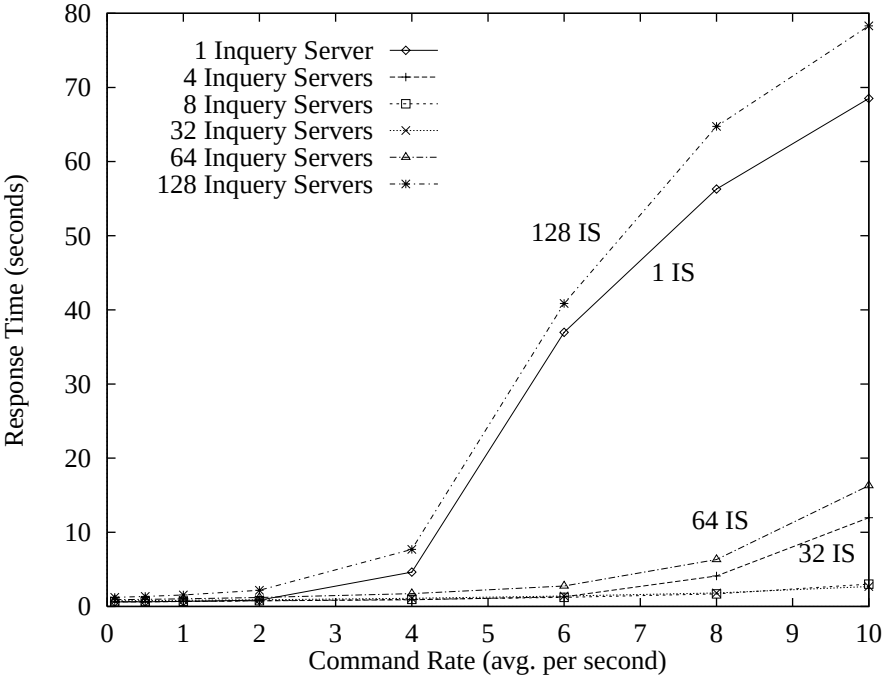

Fig. 9. Query response time (high skew).

observed, and high distributions are 14, 16, and 16.3 seconds, respectively. In a configuration with 128 Inquery servers, the query response times are 74.5 (low), 76.3 (observed), and 78 (high).

5.2.3 *Effect of Answers Returned*. Changing the number of answers returned for each query has several direct effects on system performance. As we decrease the number of answers, we see an improvement in the command response times, especially when there are a large number of Inquery servers. Another effect is that the amount of processing the connection server performs is related to the number of answers returned. Finally, we observe that the number of answers affects network usage. Our results show that reducing the number of answers returned is an effective technique for improving performance, especially when the connection server is the bottleneck.Figures 10 and 11 show the query response times when the Inquery servers compute the top 15 and 100 answers for each query, respectively. When we compare these figures to the results in Figure 5 for 1000 answers, we see the number of answers an Inquery server computes has an impact on the performance of several configurations. Notice the change in the x-axis for Figures 10 and 11 (i.e., 0 to 10 versus 0 to 80 in Figure 5). Although all cases show poor performance when there is only one Inquery server, the graphs show that the query response time

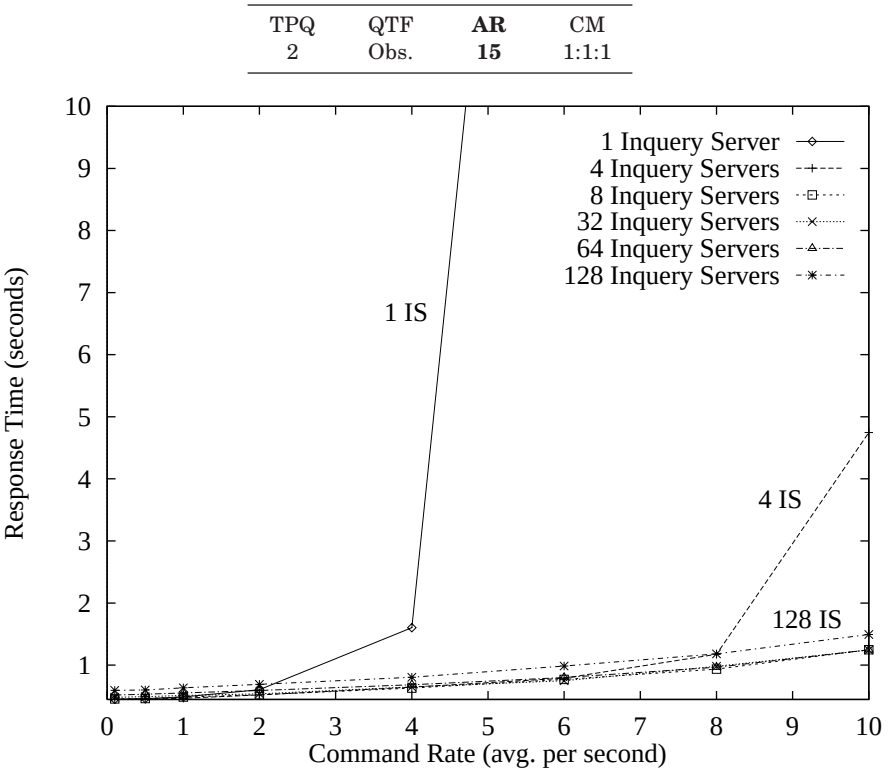

Fig. 10. Query response time (15 answers).

improves as we decrease the number of answers returned. There is a large improvement for 64 and 128 Inquery servers when we decrease the number of answers from 1000 to 100. For example, when clients issue 10 commands/second the response time improves by 92% (64 IS) and 96% (128 IS). Reducing the number of answers to 15 also improves performance although the improvement between 100 to 15 is small for most configurations.

Our results suggest that the response time improvements are due to lower utilization of the connection server and network. Figure 6 shows the connection server utilization when the Inquery servers return 1000 answers. The utilization values for 64 and 128 Inquery servers becomes extremely high when the command rate increases (e.g., the utilization reaches 100%). The maximum utilization values when Inquery servers return 100 and 15 answers is 82% and 70%, respectively.

Figure 12 shows the relative network utilization when clients request 15, 100, and 1000 answers. The relative values are the averages of all the clients. The error bars show the lowest and highest values. Figure 12 shows that as the number of clients increases, the number of answers returned has a relatively large impact on network utilization. As we reduce the number of answers, the network utilization decreases. For example, the network utilization for 15 and 100 answers is 31% and 36% of the network utilization for 1000 answers, respectively, when there is one Inquery

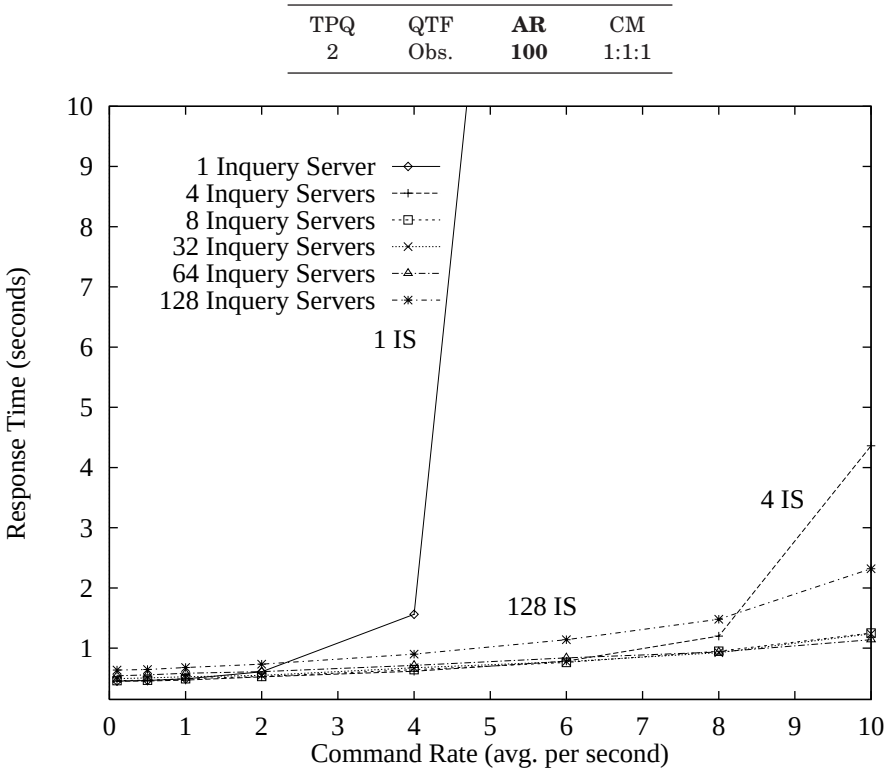

Fig. 11. Query response time (100 answers).

server. As the number of Inquery servers increases, the gap between 1000 answers and 15 or 100 answers becomes larger.

In terms of absolute values, network utilization becomes quite large as the number of Inquery servers and the command rate increases. The maximum network utilization when 1000 answers are returned is 93%, which occurs for 128 Inquery servers when the command rate exceeds six commands/second. However, for a small number of Inquery servers or a relatively slow command rate, the network utilization is very low and is usually less than 20%. As a comparison, the highest network utilization when Inquery servers return 15 answers is 7%.

5.2.4 *Effect of Changing the Command Mixture*. In this section, we vary the command mixture to include more summary and document commands. We run tests using ratios of 1:1:1, 1:2:2, 1:3:3, and 1:4:4 (the number of query:summary:document commands). The performance of the system improves as the number of summary and document commands increases. We expect an improvement, since it takes less resources to service a summary or document command compared to a query.

Figures 13 and 14 illustrate the effect of increasing the number of summary and document operations on response time for an architecture with eight and 128 Inquery servers, respectively. Our results show signifi-

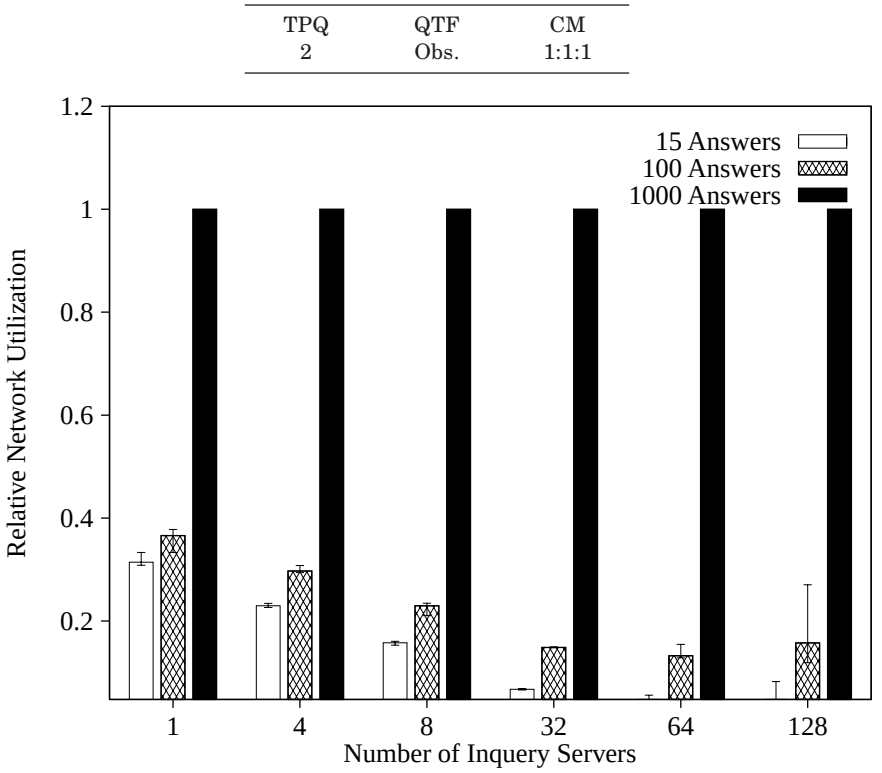

Fig. 12. Effect of answers returned on network utilization.

cant improvements in performance as the mixture of summary and document commands increases. The most impressive improvements occur when there are a large number of Inquery servers. When clients issue 10 commands/second and the command ratios change from 1:1:1 to 1:4:4, the query response time decreases by 46% for eight Inquery servers and 95% for 128 Inquery servers. For 128 Inquery servers, we also see large improvements when the command mixture changes from 1:1:1 to 1:2:2.

Our system also achieves reasonable response times when clients issue medium-length queries and the command mixture is 1:4:4. We observe response times less than 20 seconds for 64 and 128 Inquery servers (compare this with 120 seconds in Table VII). Unfortunately, increasing the number of summary and document commands does not produce reasonable response times when clients issue long queries.

5.3 Improving Performance

In this section, we discuss changes to the basic architecture to improve system performance. Our results show that the system scales for short queries up to a certain point; if we add too many Inquery servers, then performance degrades, since the connection server becomes a bottleneck. In this section, we show that adding a small number of connection servers

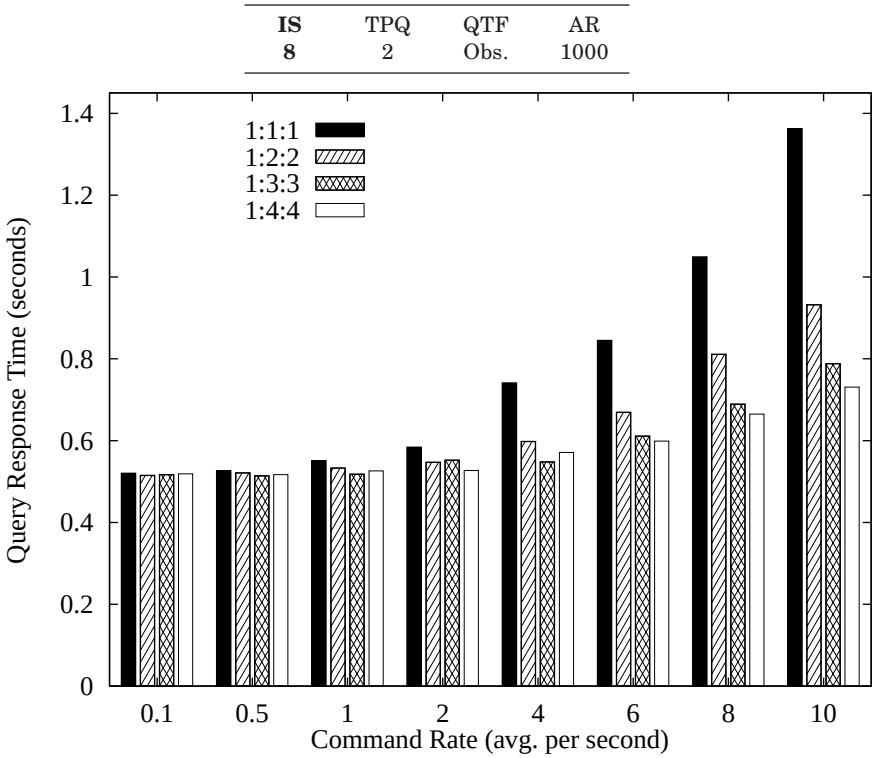

Fig. 13. Effect of varying command mixture (short queries—8 Inquery servers).

eliminates the bottleneck in the connection server. We have also shown that, in some cases, the Inquery server is the bottleneck due to high disk utilization. In this section, we improve the performance of the Inquery server by adding more disks and threads to alleviate the bottleneck.

We have also run experiments trying to improve the performance of the connection server by moving the merging functionality to the clients. Recall that the connection server is responsible for collecting and merging intermediate results before sending the final answer to the client. We do not present the results because moving merging functionality has very little effect on performance. We do not see significant changes in performance because the merge time is fast, and the time to send more messages from the connection server to the client offsets the time the connection server previously spent merging results.

5.3.1 *Improving Connection Server Performance*. The results in Section 5.2 show that the connection server is a bottleneck when clients issue short queries and the system contains a large number of Inquery servers. The cause of the bottleneck is that the connection server is unable to handle requests quickly enough. Previous results show the connection server is the bottleneck more frequently when the architecture uses a slower processor [Cahoon and McKinley 1996]. We can eliminate this bottleneck using two

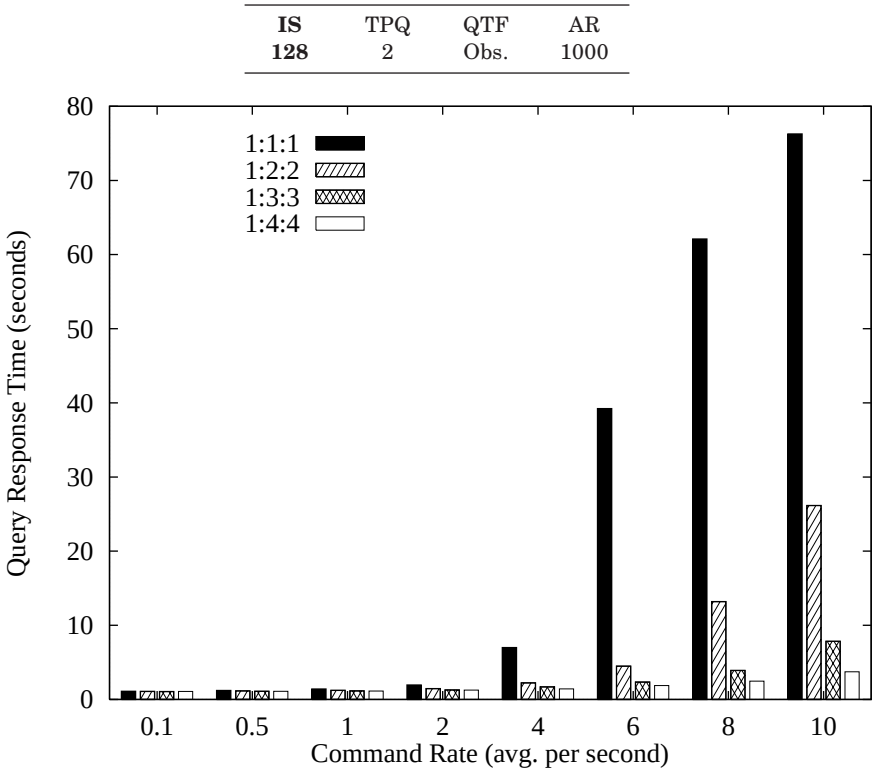

Fig. 14. Effect of varying mixture (short queries—128 Inquery servers).

or four connection servers to handle up to 128 Inquery servers. Although using a faster processor for the connection server is one solution, we show that performance significantly improves using a small number of connection servers. Simply increasing the speed of the CPU does not completely solve the problem because we anticipate that the functionality of the connection servers will increase in the future which may offset gains in processor speeds.

When the system has multiple connection servers, the clients evenly divide 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 Inquery server. If the 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 maintains its own queue of outstanding requests. We have run experiments which indicate that simply moving the queues to the Inquery servers does not significantly change performance.

In these experiments, we also increase the speed of the network from 10Mbps to 100Mbps. Initial results show that the network becomes a bottleneck when we increase the number of connection servers. The network utilization approaches 100% in configurations with 64 or 128 Inquery

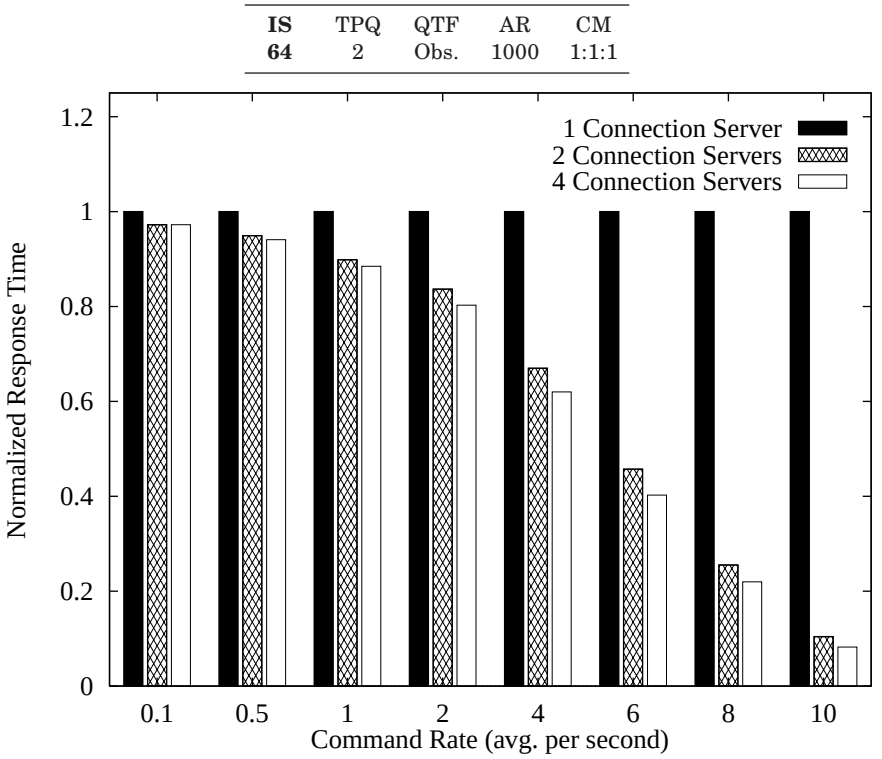

Fig. 15. Multiple-connection server query response time—64 Inquery servers.

servers. Due to the high network utilization, we see very little benefit when we add connection servers. In order to show the benefit of additional connection servers, we increase the network speed. If it is not possible to increase the network speed, then we can decrease network utilization by decreasing the number of answers returned (see Section 5.2.3). In the single-connection-server architecture, increasing the speed of the network has very little benefit. Figures 15 and 16 compare the query response times when the architecture has one, two, or four connection servers for 64 and 128 Inquery servers. We only show results when the connection server is a bottleneck. There is little, if any, benefit of increasing the number of connection servers when there is no bottleneck.

The figures show relative performance values compared to the absolute times from Figure 5 when the architecture has only one connection server. The figures show that increasing the number of connection servers significantly improves the response time. When the architecture has 64 Inquery servers, the difference in the normalized response time increases as the command rate increases. For example, Figure 15 shows that the normalized response time at 10 commands/second is 10% and 8% for two and four connection servers. These relative values translate into absolute times that are 1.67 and 1.32 seconds (compared with 16 seconds for the one-connection-server case).

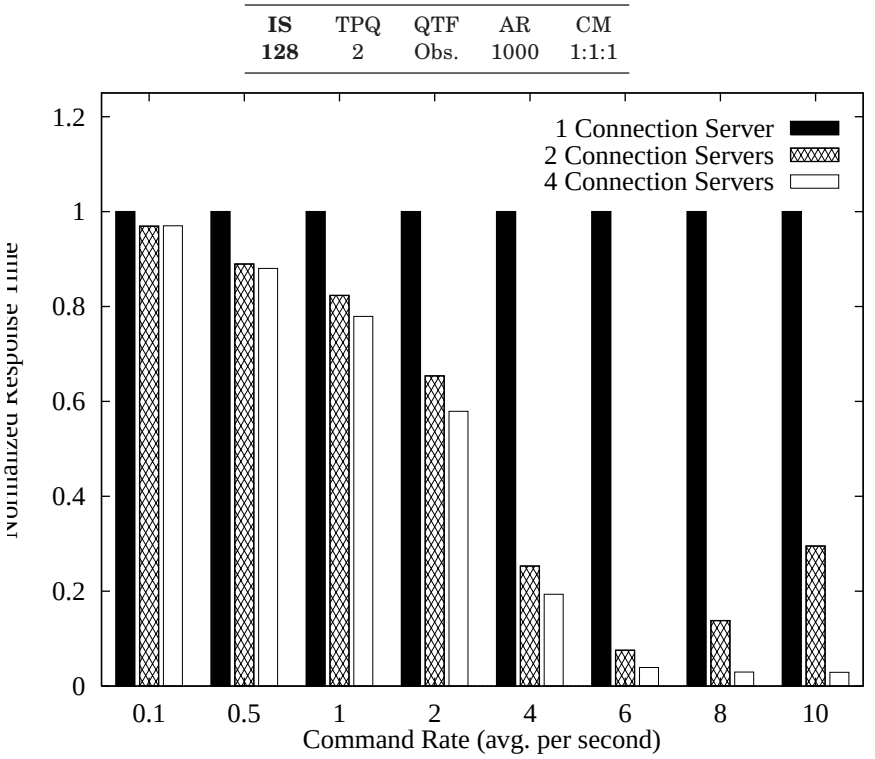

Fig. 16. Multiple-connection-server query response time—128 Inquery servers.

When the architecture has 128 Inquery servers (Figure 16), the performance using two connection servers improves until the command rate reaches six commands/second. At that point, the connection servers start to become a bottleneck and the response time increases. However, when the architecture has four connection servers, the relative response time continues to improve as the command rate increases. When clients issues 10 commands/second, the relative response times for two and four connection servers are 29.5% and 2.9%, respectively. The absolute response times are 22.5 and 2.2 seconds (compare with 76 seconds for the one-connection-server configuration).

5.3.2 *Improving Inquery Server Performance*. The Inquery server is a bottleneck when the command rate is high and when there are a small number of Inquery servers. Recall that each Inquery server allows up to four threads and runs on a host with a single CPU and disk. Our results show the disk is the primary problem. We see disk utilization values as high as 98.5%. In previous work, we have investigated the performance of an information retrieval engine on symmetric multiprocessors that contain multiple CPUs and disks [Lu et al. 1998]. The results suggest that we can improve performance by adding resources. In this section, we illustrate how adding only one more disk significantly improves performance when the

Inquery servers are a bottleneck. We also increase the number of threads from four to eight for this experiment. It is possible to improve performance even further by adding CPUs, more disks, and increasing the number of threads, but the focus of this article is the distributed architecture and not the peak performance of a multiprocessor system.

Figure 17 shows the results of adding another disk to each host on query response time. We see significant performance improvements when the Inquery servers are the bottleneck. In a configuration with 1 Inquery server, we see improvements by a factor of 15.84, 3.68, and 2.66 when clients issue 6, 8, and 10 commands/second, respectively. When the issue rate is less than 6 commands/second, the response times are very similar. We also see improvements in a configuration with 4 and 8 Inquery servers when clients issue 10 commands/second. When the number of Inquery servers is greater than 8, including more disks has no benefit because the Inquery servers are not the bottleneck in those configurations.

Figure 18 suggests that one reason for the performance improvement is due to better utilization of the Inquery server. Compare the utilization values in Figure 18 to Figure 7. The largest utilization when the Inquery servers contain two disks is 260% out of 300% (compare to 140% out of 200% when the Inquery servers contain a single disk).[6] The high utilization comes from an increase in the CPU utilization, which approaches 70% coupled with just a 4% decrease in disk utilization (per disk).

5.4 Achieving High Performance in Large Systems

Ultimately, users of IR systems want to enter queries and obtain fast and accurate results. They are not concerned with the different factors that affect performance. In contrast, system designers must account for many parameters in order to maximize performance, especially when creating large architectures. In this section, we simulate a high-performance architecture in order to determine the best possible performance in a highly utilized, large-scale system given our experimental parameters.

We simulate an architecture containing four connection servers and 128 Inquery servers (each with 1GB of data). Each Inquery server contains one CPU and two disks and allows up to eight threads. We vary the query length (TPQ) and the command arrival rate (λ). We set the other parameters to the values that produce the best results. These parameters include the query term frequency (QTF = Low Skew), number of answers returned (AR = 15), and command mixture (CM = 1:4:4).

Table VIII shows the query response times for our high-performance architecture (we italicize values greater than 10 seconds). For short queries, the query response time is very fast, less than 0.46 seconds, and does not significantly increase as the command rate increases. The architecture

[6]On a system with two disks and one CPU, the maximum potential utilization is 300%, which means that the two disks and the CPU are always working concurrently.

Table for Figures 17 and 18

TPQ	QTF	AR	CM
2	Obs.	1000	1:1:1

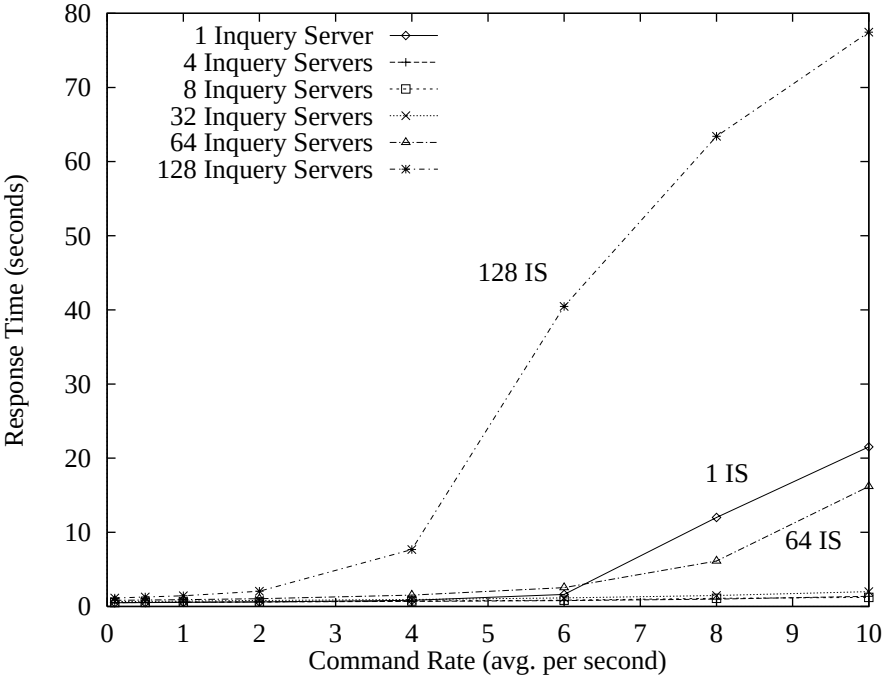

Fig. 17. Query response time (two disks).

also achieves response times under seven seconds when users issue medium-length queries. Using long queries, we see response times under 10 seconds until the command rate exceeds two commands/second. Even when the command rate is 10 commands/second, the response time is under one minute.

The response times in Table VIII occur only in best-case scenarios. Unfortunately, system designers do not have complete control over each of the experimental parameters. The system designer controls the number of Inquery servers and connection servers, the amount of hardware resources, and the number of answers returned. User behavior defines the other parameters, which include the number terms per query, the query term frequency, and the command mixture. However, these results suggest it is possible to provide high performance under many different realistic conditions. The effectiveness of the system depends on architectural decisions as well as typical user behavior.

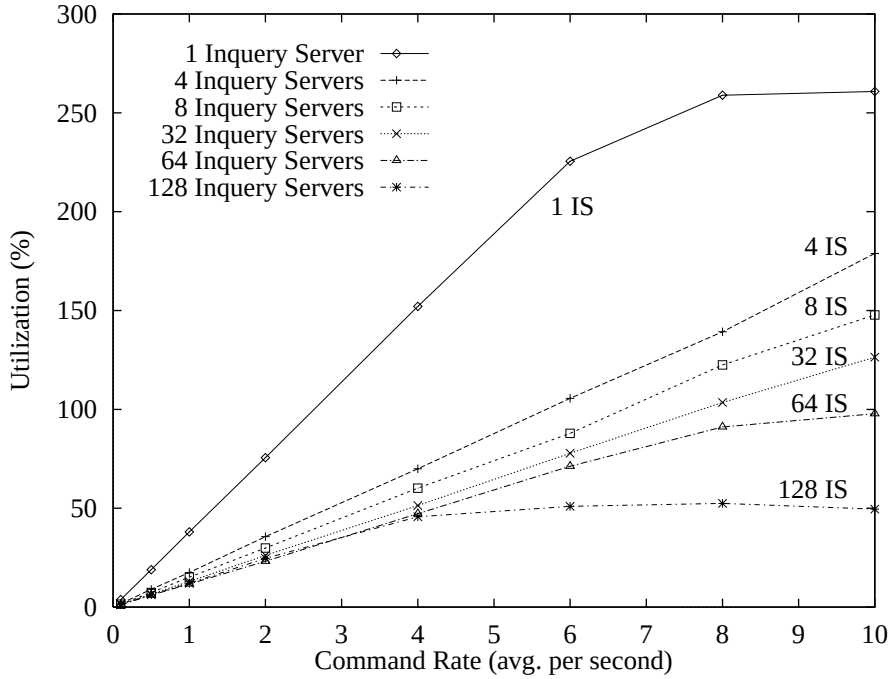

Fig. 18. Average Inquery server utilization.

Table VIII. Best Query Response Times for Large Systems (seconds)

IS	QTF	AR	CM
128	Low	15	1:4:4

Query Length	Command Rate (Average per Second)							
	0.1	0.5	1	2	4	6	8	10
Short	0.41	0.42	0.42	0.41	0.42	0.43	0.46	0.46
Medium	1.88	2.06	2.22	2.23	2.76	3.73	4.15	6.57
Long	4.74	5.38	6.31	7.51	*13.71*	*25.98*	*41.98*	*57.77*

6. SUMMARY

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

a function of several key IR characteristics: client command rate, number of text collections, number of terms per query, query term frequency, number of answers returned, and the mixture of query, summary, and document commands.

We present results for a system that distributes multiple text collections over a set of independent hosts. The best performance occurs for architectures with either eight or 32 Inquery servers when the system is able to exploit parallelism in the summary information commands. Performance degrades for more than 32 Inquery servers when the connection server becomes a bottleneck, especially for short queries. Using these results, we change the architecture to improve performance and scalability by eliminating bottlenecks that occur in either the connection server or the Inquery servers. By adding a small number of connection servers to coordinate a large number of clients and Inquery servers, the system maintains scalable performance at higher workloads. We also illustrate that including another disk significantly improves Inquery server performance in cases when the disk is a bottleneck.

Our work presents an efficient and scalable architecture for large-scale distributed IR. We model and measure a large architecture, up to 128GB of data on 128 machines, and vary key IR features in order to provide insight into designing distributed IR systems. We also illustrate that it is important to model both query and document commands because the heterogeneity of the commands has an significant impact on performance.

ACKNOWLEDGMENTS

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

REFERENCES

BAILEY, P. AND HAWKING, D. 1996. A parallel architecture for query processing over a terabyte of text. Tech. Rep. TR-CS-96-04. Department of Computer Science, Australian National Univ., Canberra, Australia.

BELL, D. AND GRIMSON, J. 1992. *Distributed Database Systems*. Addison-Wesley, Reading, MA.

BROWN, E. W. AND CHONG, H. A. 1998. The GURU system in TREC-7. In *Proceedings of the 7th Text Retrieval Conference* (TREC-7),

BRUMFIELD, J. A., MILLER, J. L., AND CHOU, H. T. 1988. Performance modeling of distributed object-oriented database systems. In *Proceedings of the International Symposium on Databases in Parallel and Distributed Systems* (Austin, TX, Dec. 5–7), J. E. Urban, Ed. IEEE Computer Society Press, Los Alamitos, CA, 22–32.

BURKOWSKI, F. J. 1990. Retrieval performance of a distributed text database utilizing a parallel process document server. In *Proceedings of the International Symposium on Databases in Parallel and Distributed Systems* (Dublin, Ireland, July), 71–79.

BURKOWSKI, F., CORMACK, G., CLARKE, C., AND GOOD, R. 1995. A global search architecture. Tech. Rep. CS-95-12. Computer Science Dept., University of Waterloo, Waterloo, Canada.

CAHOON, B. AND MCKINLEY, K. S. 1996. Performance evaluation of a distributed architecture for information retrieval. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '96, Zurich, Switzerland, Aug. 18–22), H.-P. Frei, D. Harman, P. Schaüble, and R. Wilkinson, Eds. ACM Press, New York, NY, 110–118.

CALLAN, J. P., CROFT, W. B., AND BROGLIO, J. 1995a. TREC and TIPSTER experiments with INQUERY. *Inf. Process. Manage. 31*, 3 (May–June), 327–343.

CALLAN, J. P., CROFT, W. B., AND HARDING, S. M. 1992. The INQUERY retrieval system. In *Proceedings of the 3rd International Conference on Database and Expert System Applications* (Valencia, Spain, Sept.),

CALLAN, J. P., LU, Z., AND CROFT, W. B. 1995b. Searching distributed collections with inference networks. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '95, Seattle, WA, July 9–13), E. A. Fox, P. Ingwersen, and R. Fidel, Eds. ACM Press, New York, NY, 21–28.

COUVREUR, T. R., BENZEL, R. N., MILLER, S. F., ZEITLER, D. N., LEE, D. L., SINGHAL, M., SHIVARATRI, N., AND WONG, W. Y. P. 1994. An analysis of performance and cost factors in searching large text databases using parallel search systems. *J. Am. Soc. Inf. Sci. 45*, 7 (Aug.), 443–464.

CRINGEAN, J. K., ENGLAND, R., MANSON, G. A., AND WILLETT, P. 1990. Parallel text searching in serial files using a processor farm. In *Proceedings of the 13th International Conference on Research and Development in Information Retrieval* (SIGIR '90, Brussels, Belgium, Sept. 5–7), J.-L. Vidick, Ed. ACM Press, New York, NY, 429–453.

CROFT, W. B., COOK, R., AND WILDER, D. 1995. Providing government information on the Internet: Experiences with THOMAS. In *Proceedings of the 2nd International Conference on Theory and Practice of Digital Libraries* (DL '95, Austin, TX, June), 19–24.

CROWDER, G. AND NICHOLAS, C. 1995. An approach to large scale distributed information systems using statistical properties of text to guide agent search. In *Proceedings of the CIKM Workshop on Intelligent Information Agents* (Baltimore, MD, Dec.),

DEWITT, D. AND GRAY, J. 1992. Parallel database systems: the future of high performance database systems. *Commun. ACM 35*, 6 (June), 85–98.

DEWITT, D., GRAEFE, G., KUMAR, K. B., GERBER, R. H., HEYTENS, M. L., AND MURALIKRISHNA, M. 1986. GAMMA—A high performance dataflow database machine. In *Proceedings of the 12th International Conference on Very Large Data Bases* (Kyoto, Japan, Aug.), VLDB Endowment, Berkeley, CA.

FOX, E. A. 1983. Characterization of two new experimental collections in computer and information science containing textual and bibliographic concepts. Tech. Rep. 83-561. Cornell University, Ithaca, NY.

FRIEDER, O. AND SIEGELMANN, H. T. 1991. On the allocation of documents in multiprocessor information retrieval systems. In *Procedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '91, Chicago, IL, Oct. 13-16), E. Fox, Ed. ACM Press, New York, NY, 230–239.

HAGMANN, R. B. AND FERRARI, D. 1986. Performance analysis of several back-end database architectures. *ACM Trans. Database Syst. 11*, 1 (Mar. 1986), 1–26.

HARMAN, D. K., Ed. 1992. *Proceedings of the 1st Text Retrieval Conference*. (TREC-1, Gaithersburg, MD, Nov.). National Institute of Standards and Technology, Gaithersburg, MD. NIST Special Publication 500-217.

HARMAN, D., MCCOY, W., TOENSE, R., AND CANDELA, G. 1991. Prototyping a distributed information retrieval system that uses statistical ranking. *Inf. Process. Manage. 27*, 5 (1991), 449–459.

HAWKING, D. 1997. Scalable text retrieval for large digital libraries. In *Proceedings of the 1st European Conference on Research and Advanced Technology for Digital Libraries* (Pisa, Italy, Sept.), Springer Lecture Notes in Computer Science, vol. 1324. Springer-Verlag, Vienna, Austria.

HAWKING, D. AND THISTLEWAITE, P. 1997. Overview of the TREC-6 very large collection track. In *Proceedings of the 6th Text Retreival Conference* (TREC-6, Nov.), E. Voorhees and D. Harman, Eds.

HAWKING, D., CRASWELL, N., AND THISTLEWAITE, P. 1998. Overview of TREC-7 very large collection track. In *Proceedings of the 7th Text Retrieval Conference* (TREC-7),

JEONG, B.-S. AND OMIECINSKI, E. 1995. Inverted file partitioning schemes in multiple disk systems. *IEEE Trans. Parallel Distrib. Syst. 6*, 2 (Feb.), 142–153.

JUMP, J. R. 1993. YACSIM reference manual. Version 2.1.1. Rice University, Houston, TX.

LIN, Z. AND ZHOU, S. 1993. Parallelizing I/O intensive applications for a workstation cluster: a case study. *SIGARCH Comput. Arch. News 21*, 5 (Dec. 1993), 15–22.

LU, Z., MCKINLEY, K. S., AND CAHOON, B. 1998. The hardware/software balancing act for information retrieval on symmetric multiprocessors. In *Proceedings of Euro-Par'98* (Sept., Southampton, UK),

MACKERT, L. F. AND LOHMAN, G. M. 1986. R* optimizer validation and performance evaluation for distributed queries. In *Proceedings of the 12th International Conference on Very Large Data Bases* (Kyoto, Japan, Aug.), VLDB Endowment, Berkeley, CA, 149–159.

MACLEOD, I. A., MARTIN, P., NORDIN, B., AND PHILLIPS, J. R. 1987. Strategies for building distributed information retrieval systems. *Inf. Process. Manage. 23*, 6 (July 1, 1987), 511–528.

MARTIN, T. P. AND RUSSELL, J. I. 1991. Data caching strategies for distributed full text retrieval systems. *Inf. Syst. 16*, 1 (1991), 1–11.

MARTIN, T. P., MACLEOD, I. A., RUSSELL, J. I., LEESE, K., AND FOSTER, B. 1990. A case study of caching strategies for a distributed full text retrieval system. *Inf. Process. Manage. 26*, 2 (1990), 227–247.

MOFFAT, A. AND ZOBEL, J. 1995. Information retrieval systems for large document collections. In *Proceedings of the 3rd Text Retrieval Conference* (TREC-3), D. Harman, Ed. National Institute of Standards and Technology, Gaithersburg, MD, 500–525.

POGUE, C. A. AND WILLETT, P. 1987. Use of text signatures for document retrieval in a highly parallel environment. *Parallel Comput. 4*, 3 (June), 259–268.

SCHATZ, B. R. 1990. Interactive retrieval in information spaces distributed across a wide-area network. TR 90-35. Department of Computer Science, University of Arizona, Tucson, AZ.

SIMPSON, P. AND ALONSO, R. 1987. Data caching in information retrieval systems. In *Proceedings of the 10th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '87, New Orleans, LA, June 3–5), C. T. Yu and C. J. Van Rijsbergen, Eds. ACM Press, New York, NY, 296–305.

STANFILL, C. AND KAHLE, B. 1986. Parallel free-text search on the connection machine system. *Commun. ACM 29*, 12 (Dec. 1986), 1229–1239.

STANFILL, C., THAU, R., AND WALTZ, D. 1989. A parallel indexed algorithm for information retrieval. In *Proceedings of the 12th Annual International ACM Conference on Research and Development in Information Retrieval* (SIGIR '89, Cambridge, MA, June 25–28), N. J. Belkin and C. J. van Rijsbergen, Eds. ACM Press, New York, NY, 88–97.

STONEBRAKER, M., WOODFILL, J., RANSTROM, J., KALASH, J., ARNOLD, K., AND ANDERSON, E. 1983. Performance analysis of distributed data base systems. In *Proceedings of the 3rd Symposium on Reliability in Distributed Software and Database Systems* (Clearwater Beach, FL, Oct.),

TOMASIC, A. S. 1994. Distributed queries and incremental updates in information retrieval systems. Ph.D. Dissertation. Department of Computer Science, Princeton Univ., Princeton, NJ.

TOMASIC, A. AND GARCIA-MOLINA, H. 1992. Caching and database scaling in distributed shared-nothing information retrieval systems. Tech. Rep. STAN-CS-92-1456. Stanford University, Stanford, CA.

TOMASIC, A. AND GARCIA-MOLINA, H. 1993. Performance of inverted indices in distributed text document retrieval systems. In *Proceedings of the 2nd International Conference on Parallel and Distributed Systems* (Dec.), 8–17.

VILES, C. L. AND FRENCH, J. C. 1995. Dissemination of collection wide information in a distributed information retrieval system. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '95, Seattle, WA, July 9–13), E. A. Fox, P. Ingwersen, and R. Fidel, Eds. ACM Press, New York, NY, 12–20.

VOORHEES, E. M., GUPTA, N. K., AND JOHNSON-LAIRD, B. 1995. Learning collection fusion strategies. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '95, Seattle, WA, July 9–13), E. A. Fox, P. Ingwersen, and R. Fidel, Eds. ACM Press, New York, NY, 172–179.

WOLFRAM, D. 1992. Applying informetric characteristics of databases to IR system file design, Part I: informetric models. *Inf. Process. Manage. 28*, 1 (Jan.), 121–133.

ZIPF, G. K. 1949. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley, Reading, MA.

Received: April 1997; revised: February 1998, February 1999, and June 1999; accepted: September 1999