SCALABLE DISTRIBUTED ARCHITECTURES FOR INFORMATION RETRIEVAL

A Dissertation Presented

by

ZHIHONG LU

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 1999

Department of Computer Science

SCALABLE DISTRIBUTED ARCHITECTURES FOR INFORMATION RETRIEVAL

A Dissertation Presented

by

ZHIHONG LU

Approved as to style and content by:

Kathryn S. McKinley, Chair

W. Bruce Croft, Member

Donald F. Towsley, Member

James P. Callan, Member

C. Mani Krishna, Member

James F. Kurose, Department Chair
Department of Computer Science

To my parents, Professor Hao Lu and Professor Jiangqiu Liu,

and my brother, Qinyi Lu.

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere thanks to my advisor, Kathryn McKinley for her contribution in the development of my research and her guidance in my professional growth. Without her broad vision, deep insight, and strong encouragement, this work would not have been possible.

I would like to thank Bruce Croft and Jamie Callan for teaching me sound research principles on distributed information retrieval and their valuable advice.

I would like to thank the rest of my committee, Don Towsley and Mani Krishna for their valuable comments and suggestions.

I would like to thank all present and past members of the Center for Intelligent Information Retrieval at the University of Massachusetts, Amherst. I was fortunate enough to be a member of this friendly, enjoying, and stimulating research community. In particular, I would like to thank Brendon Cahoon for his preliminary work on distributed information retrieval architectures and his kindness to tutor me the simulation techniques. I would like to thank Jay Ponte for proof-reading my proposal. I would like to thank our secretary, Kate Moruzzi, for her kindness and efficient assistance whenever I needed.

I would like to thank all my friends I have met in Amherst. Without them, the last five and a half years would have been unbearable. In particular, I would like to thank the Chinese community in the Department of Computer Science and the Zinsmeister family.

I would like to thank my husband, Jianmin Wang, for his love, understanding, and absolute faith in my abilities. I owe my son, Luhan Wang, for his four-year life without his mother around. I wish he could understand me when he grows up.

I dedicate this dissertation to my parents, Hao Lu and Jiangqiu Liu, for their love, caring, sacrifice, and support. I will always be grateful to my parents, who have always given me more than what I could ever ask for. I especially appreciate that they teach me the positive attitude to survive in this world. I also dedicate this dissertation to my brother, Qinyi Lu. The dreams we were sharing during our childhood have always been the greatest inspiration for me to move on and face whatever challenge may appear.

This material is based on work supported in part by the National Science Foundation, Library of Congress and Department of Commerce under cooperative agreement number EEC-9209623, and in part by Defense Advanced Research Projects Agency/ITO under ARPA order number D468, issued by ESC/AXS contract number F19628-95-C-0235. Any opinions, findings and conclusions or recommendations expressed in this material are the author and do not necessarily reflect those of the sponsors.

ABSTRACT

SCALABLE DISTRIBUTED ARCHITECTURES FOR INFORMATION RETRIEVAL

MAY 1999

ZHIHONG LU
B.Sc., TONGJI UNIVERSITY
M.Sc., INSTITUTE OF COMPUTING TECHNOLOGY,
CHINESE ACADEMY OF SCIENCES
Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Kathryn S. McKinley

As information explodes across the Internet and intranets, information retrieval (IR) systems must cope with the challenge of scale. How to provide scalable performance for rapidly increasing data and workloads is critical in the design of next generation information retrieval systems. This dissertation studies scalable distributed IR architectures that not only provide quick response but also maintain acceptable retrieval accuracy. Our distributed architectures exploit parallelism in information retrieval on a cluster of parallel IR servers using symmetric multiprocessors, and use partial collection replication and selection as well as collection selection to restrict the search to a small percentage of data while maintaining retrieval accuracy.

We first investigate using partial collection replication for IR systems. We examine query locality in real systems, how to select a partial replica based on relevance, how to load-balance between replicas and the original collection, as well as updating overheads and strategies. Our results show that there exists sufficient query locality to justify partial replication for information retrieval. Our proposed replica selection algorithm effectively selects relevant partial replicas, and is inexpensive to implement. Our evidence also indicates that partial replication achieves better performance than caching queries, because the replica selection algorithm finds similarity between non-identical queries, and thus increases observed locality.

We use a validated simulator to perform a detailed performance evaluation of distributed IR architectures. We explore how best to build parallel IR servers using symmetric multiprocessors, evaluate the performance of partial collection replication and collection selection, and compare the performance of partial collection replication with collection partitioning as well as collection selection. At last we present experiments for searching a terabyte of text. We also examine performance changes when we use fewer large servers, faster servers, and longer queries.

Our results show that because IR systems have heavy computational and I/O loads, the number of CPUs, disks, and threads must be carefully balanced to achieve scalable performance. Our results show that partial collection replication is much more effective at decreasing the query response time than collection partitioning for a loaded system, even with *fewer* resources, and it requires only modest query locality. Our results also show that partial collection replication performs better than collection selection when there exists enough query locality, and it performs worse when the collection access is fairly uniform after collection selection. Finally our results show that replica and collection selection can be combined to provide quick response time for a terabyte of text. Changes of system configurations do not significantly change the relative improvements due to partial collection replication and collection selection, although they affect the absolute response time.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 The Problem of Scale in Information Retrieval

Information Retrieval (IR) is concerned with locating documents relevant to the information needs of users. Distributed information retrieval provides information access to text collections distributed across a network. As information explodes everywhere, increasingly many vendors and users distribute and search their information through the Internet and intranets. The data volume a distributed IR system must cope with is nearing the terabyte level. For example, AltaVista, a web search engine, claims that it searches over 140 million web pages [4], which indicates that its search space is already several hundred gigabytes. In this dissertation, we investigate distributed architectures that scale with rapidly increasing data and workloads, and not only provide quick response but also maintain acceptable retrieval accuracy.

When we face a huge number of text collections and tremendous workloads, the first reaction is to use lots of hardware resources. Using lots of hardware resources does help solve the problem. Because information retrieval is an inherently parallel application, we can execute queries independently, and we can divide a collection into several partitions and execute a query on multiple partitions at the same time. Therefore, we may either use a multiprocessor to increase the processing capability of each single server, or use multiple machines such that each machine processes a part of data and workloads. However using lots of hardware resources does not necessarily produce high performance, since a single bottleneck can degrade performance. We need to carefully balance the hardware and software resources to ensure performance improvements. In addition, information retrieval is a resource-intensive application. A system that searches hundreds of collections or that receives millions of requests a day although it only searches a small collection could take minutes to complete a request, which is beyond online users' tolerance for waiting for a response.

Restricting the search to the most relevant collections is a way of producing quick response while maintaining acceptable retrieval accuracy, because searching fewer collections takes less time. However, selecting the most relevant collections may work ineffectively when the relevant documents about a topic are scattered over a large number of collections. In addition, excessive workloads are still able to overwhelm a server that is frequently chosen.

For excessive workloads, the only solution is to distribute workloads on a single server over several "identical" servers, which requires full or partial data replication. The huge number of documents in a large-scale IR system prohibits full data replication. We investigate replicating parts of collections by disseminating documents based on query locality, resources, and access patterns or other constraints in order to replicate as little data as possible and maintain acceptable retrieval accuracy. In a distributed IR system, partial collection replication serves two purposes: distributing excessive workloads and restricting the search to a small percentage of data.

In this dissertation, we investigate the scalable architectures that exploit parallelism by distributing data over a cluster of symmetric multiprocessors, and use partial collection replication and collection selection to search as little data as possible while maintaining acceptable retrieval accuracy. We also observe characteristics of IR workloads and IR systems in order to use them as a guideline.

1.2 Research Summary

This dissertation studies distributed architectures that scale with rapidly increasing data and workloads, and not only produce quick response time but also maintain acceptable retrieval accuracy. Our architectures incorporate parallel information retrieval, partial collection replication, and collection selection. We implement a simulator for distributed architectures, and validate it against the prototype implementation. We use the validated simulator to evaluate system performance with a variety of workloads. We also investigate how to replicate a small percentage of data and select relevant partial replicas.

The information retrieval engine used in this dissertation is InQuery [14], which is based on the inference network retrieval model [72]. There are two major reasons to choose InQuery. First, it is a proven effective retrieval engine [36, 37, 38, 39, 40] that enables our research to provide high retrieval accuracy. Second, it is used in Web searching, large libraries, companies, and government agencies such as Library of Congress, White House, West Publishing, and Lotus [46]. For the experiments in this dissertation, we implement our simulator using the measurements from InQuery version 3.1 and validate it against a prototype implementation on a 3-CPU 250 MHZ DEC Alpha for a heavily loaded system. We also discuss the sensitivity of our results with respect to our base system.

1.2.1 Parallel Information Retrieval using Symmetric Multiprocessors

Although the previous research has demonstrated information retrieval is easily parallelized, it typically investigated an IR system on a distributed memory, massively parallel processing (MMP) machine (e.g. CM-5 with 65,536 processing elements), which is very expensive and needs specialized algorithms to take advantage of hardware resources [6, 21, 26, 57, 66, 67, 68] (Section 2.4 describes the previous research). Both the high price and the need for specialized algorithms prohibit such machines from being widely used. In contrast, symmetric multiprocessors (SMP) are very popular and affordable. Easily upgrading existing software from single-CPU machines to SMPs is another advantage of using SMPs.

Currently, commercial information retrieval systems, such as Web search engines AltaVista [4] and Infoseek [45], handle tremendous loads by exploiting the parallelism implicit in their tasks and use SMPs to support their services. Although it is clear that the more CPUs and disks you have the more load the system can handle, the important questions are how much of which hardware and what software structure is needed to exploit these resources. Unfortunately, commercial systems have not published their hardware and software configurations. The previous research on SMPs investigated a part of system, such as the disk system [47] or it compared the cost factors of the SMP architecture with other architectures [20]. Recently the TREC conference reported results on a single query (rather than a loaded system) against a 100 GB collection [42], where some of its participated institutions used SMPs. None of the commercial systems and the previous research on SMPs reported how to balance hardware and software resources in order to achieve scalable performance in a SMP system.

In this dissertation, we conduct a systematic study on a proven effective system using symmetric multiprocessors under a variety of realistic workloads and hardware configurations. Unlike the previous research where the workloads only include query commands, our workloads are a mixture of three IR commands: query, summary, and document commands, which provides a more realistic investigation. Our workloads mimic the statistics from real system logs. During our investigation, we demonstrate how to balance the hardware and software with respect to number of threads, CPUs, and disks as the collection size increases. We investigate the factors that affect the necessary number of threads. We demonstrate system scalability when the system is well balanced, and when one of the hardware components is a bottleneck. Our results provide insights for building high performance IR servers using symmetric multiprocessors.

1.2.2 Partial Collection Replication

Replication has long been used in the areas of distributed file systems and distributed database systems in order to improve system performance and availability [1, 2, 3, 16, 19, 24, 25, 30, 43, 44, 51, 65, 69, 76] (see Section 2.5 for detailed discussions). Database vendors such as Oracle, Informix, and Sybase offer replication techniques in their products [60]. However, there are several distinctions that make this work unique. First, information retrieval is different from the focus of previous database and file system work. Information retrieval systems are working on text, which is unstructured and is generally read-only. The database and file systems are working on structured data such as records and objects, and the data is read-write. Their work focuses on algorithms for updating read-write data to ensure consistency of different copies of data. In addition, structured database and file systems can simply use the set membership to find a replica, while information retrieval needs a selection function to determine whether a replica contains all, some, or none of the relevant documents in order to maintain retrieval effectiveness.

Recently, researchers have used replication in the Web for document access [5, 7, 78]. Generally, they cache popular documents in a hierarchy of proxy servers which are located between clients and Web servers to reduce Internet traffic. Although we also adopt a replication hierarchy to store the documents of the most frequently used queries, our replicas are searchable and thus can speed up both query and document processing.

As suggested by our log analyses, we need to speed up both query processing and document access to maximize performance improvements. One way to achieve this goal is to cache the most frequently used queries along with their top documents, and do string comparison to determine whether a query is cached. However using this approach can not match a cached query when the new query is about the same topic, but uses different query terms or different term forms. In our logs, 35% to 62% of the topics that occur more than once contain more than one unique query that returns the same top 20 documents. We solve this problem by storing a searchable replica of the top documents for each query instead of caching query results. This structure enables our system to detect whether queries about the same topic but using different terms are in the replica, which increases query locality as compared to caching queries which requires queries to match exactly.

Before we evaluate how much performance improvement partial collection replication can yield, we investigate two problems first. The first problem is whether there exists sufficient query locality in real systems. Fortunately, the access logs of real IR systems demonstrate there is such locality (see Appendix B). The second problem is whether we can find an effective tool to select a relevant partial replica, since not all partial replicas are equally or nearly equally effective for a query. Selecting replicas just based on load does not maintain acceptable retrieval accuracy, and a fast IR system without acceptable retrieval accuracy is useless. In this dissertation, we adapt the collection retrieval inference network model for selecting a relevant partial replica. We propose a new replica selection function, compare it with the collection ranking function [15], and demonstrate its effectiveness using a 2 GB collection and a 20 GB collection.

After solving the problem of effectiveness in partial replication, we investigate load balancing, updating strategies, and overheads for building replicas and replica selection databases. We then evaluate the performance of partial replication by varying numerous parameters, such as command arrival rate, distracting percentage, replicating percentage, and collection size. We compare the performance of partitioning and replication over additional hardware resources.

1.2.3 Collection Selection

The previous research on collection selection focuses on the effectiveness of collection selection: how to rank collections (see Section 2.3), but does not include performance evaluation. In this dissertation, we present performance evaluation for collection selection using inference network. We demonstrate the sensitivity of collection selection with respect to the number of top collections searched and the collection access skew after collection selection. We compare the performance of collection selection and partial collection replication.

1.2.4 Scalable Distributed Architectures

The previous research on distributed IR systems works on a relatively small amount of data. Only two works investigate collection sizes of more than 100 GB (see Section 2.2). Our distributed IR system simulates searching over a terabyte of text, and incorporates technologies that do not exist in the previous research, such as partial collection replication and collection selection. We demonstrate the overheads of incorporating these technologies, compare implementation options, and evaluate performance with a variety of workloads .

1.3 Research Contributions

The most important problem for IR systems is to attain quick response for rapidly increasing data and workloads while maintaining retrieval accuracy. This dissertation presents a significant step towards solving this key problem. The contributions include:

- First work on partial collection replication and selection in information retrieval:
    - justifying the usefulness of partial replication for IR based on traces;
    - developing a replication architecture;

  - developing an effective replica selection function and demonstrating its performance;
  - estimating updating costs for replicas and replica selection database;
  - proposing updating strategies.

- Scalable distributed architectures and an evaluation of their performance:
  - implementing and validating a simulator for distributed IR systems;
  - performance evaluation of parallel servers using symmetric multiprocessors;
  - performance evaluation of partial collection replication;
  - comparison of partial collection replication and collection partitioning;
  - performance evaluation of collection selection;
  - comparison of collection selection and partial collection replication;
  - mechanisms for searching a terabyte of text.

1.4 Structure of the Dissertation

The remainder of the dissertation is organized as follows: Chapter 2 briefly introduces the basic concepts used in information retrieval and reviews related work. Chapter 3 describes our distributed information retrieval architectures. Chapter 4 investigates issues related to partial collection replication. Chapter 5 describes the simulation model and its validation. Chapter 6 evaluates the performance of parallel information retrieval using symmetric multiprocessors, and the performance of a distributed IR system with partial collection replication and collection selection. Chapter 7 presents experiments that use these technologies to search a terabyte of text. Chapter 8 summarizes the dissertation and indicates future research directions.

CHAPTER 2

RELATED WORK

In this chapter, we briefly introduce basic concepts used in information retrieval, and review the research related to this dissertation. We introduce retrieval models, indexing, and measures for retrieval effectiveness in Section 2.1. We discuss related work on performance in distributed information retrieval in Section 2.2, effectiveness in distributed information retrieval in Section 2.3, parallel information retrieval in Section 2.4, and data replication in Section 2.5.

2.1 Information Retrieval Background

2.1.1 Retrieval Models

Information retrieval is the process of identifying and retrieving relevant documents based on a user's query. An information retrieval system consists of three basic elements: a document representation, a query representation, and a measure of similarity between queries and documents. The document representation provides a formal description of the information contained in the documents; the query representation provides a formal description of user's information need; and the similarity measure defines the rules and procedures for matching the query and relevant documents. These three elements collectively define a retrieval model. The most common models include the Boolean model [73], the vector space model [63] as implemented in SMART [10], the probabilistic model [73], and the inference network model [72] as implemented in InQuery[14]. We briefly describe these models below. In this thesis, we use the inference network model, but our results apply to other models.

The Boolean Model

In Boolean retrieval [73], a document is represented as a set of terms $d_j = \{t_1, ..., t_k\}$, where each t_i is a term that appears in document d_j. A query is represented as a Boolean expression of terms using the standard Boolean operators: *and*, *or*, and *not*. A document matches the query if the set of terms associated with the document satisfies the Boolean expression that represents the query. The result of the query is the set of matching documents.

The Vector Space Model

The vector space model [63] enhances the document representation of the Boolean model by assigning a weight to each term that appears in a document. A document is then represented as a vector of term weights. The number of dimensions in the vector space is equal to the number of terms used in the overall document collection. The weight of a term in a document is calculated using a function of the form $tf \cdot idf$, where tf (term frequency weight) is a function of the number of occurrences of the term within the document, and *idf* (inverse document frequency weight) is an inverse function of the total number of documents that contain the term.

A query in the vector space model is treated as if it were just another document, allowing the same vector representation to be used for queries as for documents. This representation naturally leads to the use of the vector inner products as the measure of similarity between the query and a document. This measure is typically normalized for vector length, such that the actual similarity measure is the cosine of the angle between the two vectors. After all of the documents in the collection have been compared to the query, the system sorts the documents by decreasing similarity measure and returns a ranked listing of documents as the result of the query.

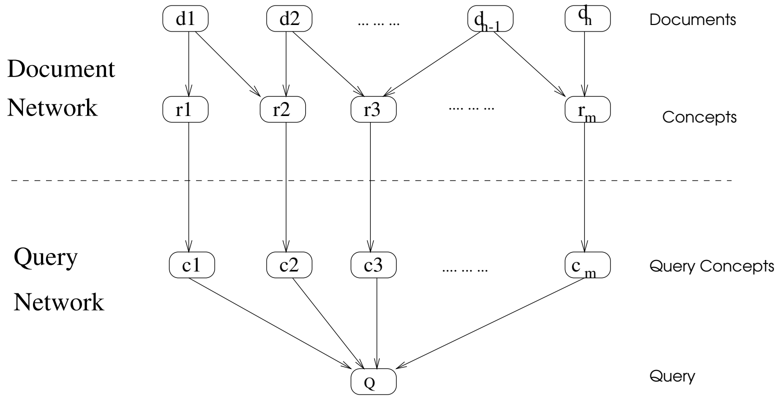


Figure 2.1. Document retrieval inference network

The Probabilistic Model

The probabilistic retrieval model [73] is based on the Probability Ranking Principle [62], which states that an information retrieval system is most effective when it responds to an expressed information need with a list of documents ranked in the decreasing order of probability of relevance, and the probabilities are estimated as accurately as possible given all the available information. In this model, the answer to a query is generated by estimating $P(relevant|d)$ (the probability of the information need being satisfied given document d) for every document, and ranking the documents according to these estimates. Using Bayes' theorem and a set of independence assumptions about the distribution of terms in documents and queries, $P(relevant|d)$ can be expressed as a function of the probabilities of the terms in d appearing in relevant and non-relevant documents. Different independence assumptions lead to different forms of the function. The most popular functions are in the category of $tf \cdot idf$ style.

The Inference Network Model

The inference network model [72] generalizes the probabilistic model by viewing information retrieval as an inference or evidential reasoning process. An inference network is a directed, acyclic dependency graph (DAG) in which nodes represent propositional variables or constants and edges represent dependence relations between propositions. If a proposition represented by node p "causes" or implies the proposition represented by node q, we draw a directed edge from p to q. The node q contains a *link matrix* that specifies $P(q|p)$ for all possible values of the two variables. When a node has multiple parents, the link matrix specifies the dependence on the set of parents and characterizes the dependence relationship between that node and all nodes representing its potential causes. Given a set of prior probabilities for the root of the DAG, the network can be used to compute the probability or degree of belief associated with all remaining nodes.

The inference network model used for InQuery [14], shown in Figure 2.1, consists of two component networks: a document network and a query network. The document network consists of document nodes(d_i), and concept representation nodes(r_i). A document node corresponds to a document. A concept node corresponds to an index unit: a term or a phrase. The link from a d_i node to a r_j node means that the document d_i is "about" the particular concept r_j. The query network consists of the query concept nodes (c_i) and the query node(Q). A query concept corresponds to a basic unit used to construct a query. The query concept nodes define the mapping between the concepts in the document collection and the concepts in the query. In InQuery, each query concept has exactly one parent in the document network. A query node represents an individual query.

The inference network is used by attaching the query network to the leaves of the document network. To produce a belief score for document d_j, we assume that $d_j = true$ and $d_k = false$ for all $k \neq j$, and condition the probabilities through the network to obtain $P(I|d_j)$. In InQuery, the belief value $belief_{ij}$ for concept r_i in document d_j is a $tf \cdot idf$ style function. The value of $belief_{ij}$ is larger when r_i occurs more frequently in the d_j and less frequently in the entire collection. The system presents the documents to users by the order of their belief scores.

2.1.2 Indexing

Indexing is a standard component in a retrieval system to improve the search efficiency. Signature files [28] and inverted index files [64] are two popular data structures for indexing. A signature file contains document signatures, one for each document in the collection. A document signature is a bit-string created by applying a hash function to each term in the document. During query evaluation, the system creates a signature for the query in the same way as for documents and matches the query to a document, if the intersection of the query signature and a document signature is equal to the query signature. However since different terms may have the same signature, a document that does not match the query in fact could be flagged as matching, which is called *false hit.* Signature files typically support the Boolean model only.
An inverted index file contains an inverted list for every term that appears in the collection. A term's inverted list identifies all of the documents that contain the corresponding term. During query evaluation, the system matches the query to documents by obtaining the inverted list for the query terms and processing the document entries in those lists. Inverted files are very flexible and support all retrieval models. Recently Zobel *et al.* [79] give both analytical and empirical results that show inverted files are superior to signature files in all aspects, regardless of the retrieval models. InQuery uses inverted files.

2.1.3 Effectiveness

The effectiveness of a retrieval system is a measure of how well the system ranks relevant documents ahead of non-relevant ones. In this thesis, we will use "effectiveness" to refer to retrieval accuracy explicitly, and "performance" to refer to execution time and system throughput. The two most widely used retrieval effectiveness measures are **Precision** (P) and **Recall** (R) defined as follows:

$$\text{Precision} = \frac{\text{Number of relevant documents returned}}{\text{Number of documents returned}}$$

$$\text{Recall} = \frac{\text{Number of relevant documents returned}}{\text{Total number of relevant documents in the collections}}$$

.

Precision is a measure of speciality – how well the system retrieves only relevant documents. Recall is a measure of completeness – how well the system retrieves all the relevant documents. In practice, effectiveness is evaluated by two methods. The first method is to present the precision figures at selected numbers of documents, such as 5, 10, 15, 20, and 50 documents. The second one is to plot precision at standard level of recalls. The 11 point R/P method defines precision at the 11 recall levels 0.0, 0.1, 0.2, ..., 1.0. Since recall increases at discrete points that do not necessarily correspond to these 11 points, a standard interpolation algorithm is used to calculate precision at 11 points [64]. In this thesis, we use *trec_eval* [38], a TREC evaluation program, to calculate the precision-recall table.

2.2 Performance of Distributed Information Retrieval

Distributed IR systems enable users to simultaneously access multiple text collections distributed over the network. A number of studies have investigated the performance of distributed IR systems [11, 12, 13, 20, 41, 53, 56, 58, 59, 70, 71]. Most of the previous work experiments with collections less than 1 GB and focuses on speedup of query processing for an unloaded system when a collection is distributed over several servers [11, 41, 53, 56, 58, 59]. Only Couvreur *et al.* [20], and Cahoon and McKinley [12, 13] use simulation to experiment with more than 100 GB of data.

Macleod *et al.*, Burkowski, and Martin *et al.* built their distributed IR systems on a network of very slow servers [11, 41, 56, 58, 59]. The machines they used are IBM PC/XT, IBM PC/AT, and Apple Macintosh II, and the link speed is 9600 baud. Macleod *et al.* explore strategies for distributing the index file and the text file [56]. Burkowski compares uniformly distributing server functionality versus splitting functionality across the servers[11]. Harman *et al.* report performance in a distributed prototype system using statistical ranking [41]. Martin *et al.* consider the effects of various data caching strategies on browsing and search response time [58, 59]. Since these early works experiment with orders of magnitude slower machines and networks, the interpretation of their results are hard to apply to today's systems.

Lin and Zhou implement a distributed IR system on a network of up to 10 DEC5000 workstations using PVM (Parallel Virtual Machine) to coordinate work and network communication [53]. They use a variation of the signature file encoding scheme to map document collections over the network. They experiment with a 100 MB collection and a 200 MB collection and show large speedups for a single query due to parallel disk access.

Tomasic *et al.* examine four possible distributed organization for the inverted indices in a boolean retrieval system: system, host, I/O bus, and system organizations [70, 71]. In the system organization, inverted lists are distributed evenly across all disks in the system and each disk holds the same number of terms; In the host and I/O bus organizations, documents are distributed evenly over hosts or I/O buses, and the inverted lists on each host and I/O bus are evenly distributed over disks attached to the host and I/O bus; In the disk organization, documents are distributed evenly over all disks and an inverted index is built for each disk. They offer empirical results for systems with up to 4 hosts using both synthetic and actual workloads. Their results using synthetic workloads that represents a legal document database and assume query terms are uniformly distributed show that the host organization is superior and the system organization performs worst due to the bottleneck of Lan bandwidth. Their results using actual workloads from a 2 GB INSPEC abstract database where inverted lists were relatively short show that the system organization performs best, because it does not pay to split the short lists across the hosts. Their results can not directly apply to our systems, because the query terms in our experiments are skewed, our retrieval system is a full-text retrieval system using the inference network model, and we experiment with up to a terabyte of text.

Ribeiro-Neto *et al.* compare the global index organization (system organization) and the local index organization (disk organization) for a retrieval system based on the vector space model using a 1 GB TREC3 collection and 50 TREC3 queries [61]. They show that the global index organization performs better than the local index organization when using a 100 Mbps network, but performs worse when using a 8 Mbps network. In our system, we do not favor the global index organization (system organization), because our system processes 3 orders of magnitude as much data, and we investigate performance improvements due to searching a small portion of data instead of all the data.

Couvreur *et al.* analyze the performance and cost factors of searching large text collections (up to 112 GB) [20]. 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 tradeoff 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.

Cahoon and McKinley report a simulation study on a large-scale distributed information retrieval system [12] built on a network of DEC5000 workstations. They model and analyze a complete system based on a state of the art effective retrieval engine, InQuery [14]. They experiment with collections up to 128 GB using a variety of workloads and investigate how different system parameters affect performance and scalability of a distributed IR system. In their investigation, they simplify each server as a single CPU system that handles a 1 GB collection. They show that in many instances a simple distributed architecture performs well under large, realistic configurations.

In this dissertation, we use a simulator to experiment with collections up to a terabyte. We distribute data over a cluster of symmetric multiprocessors, and include the components that do not exist in the previous research: collection selection and partial collection partition. Our workloads include three IR commands: query, summary, and document commands. We analyze the logs from real IR systems and use their statistics as a guideline of our workloads. We explore balancing hardware resources, identify bottlenecks, and investigate performance gains of parallelism, collection selection, and partial collection replication.

2.3 Effectiveness of Distributed Information Retrieval

To maintain effectiveness of distributed information retrieval, the system must be able to select the most relevant subset of collections in order to reduce the search space (*the collection selection problem*) and merge the results from different collections (*the result merging problem or the collection fusion problem*). The work on collection selection is directly beneficial to the execution performance of a distributed information system, since of course searching fewer collections takes less time.

2.3.1 Automatic Collection Selection

Danzig *et al.* [23] use a hierarchy of brokers to maintain indices for abstracts of primary databases (individual collections) and support Boolean keyword matching to locate the primary databases. This broker architecture is a component of the Harvest system [9] and uses the Essence system [35] to generate the meta information. Since brokers only keep partial information about each primary database, if users' queries do not use keywords in the brokers, they have difficulty finding the right primary databases.

Voorhees *et al.* exploit the similarity of a new query to previously evaluated queries [74]. They use relevance judgments for previous queries to compute the number of documents to retrieve from each collection. This technique only works for relatively static collections. In addition, relevance judgments are not always available for widely distributed collections.

GLOSS uses document frequency information for each individual collection to estimate whether, and how many, potentially relevant documents are in a collection [32, 31]. In its boolean version, it estimates the number of potentially relevant documents in collection C as $|C| \cdot \Pi_{t \epsilon Q}(df_t/|C|)$, where t is a term in the query Q, df_t is the number of documents in the collection C containing t, and $|C|$ is the number of documents in the collection C. In its vector-space version, it ranks the collections based on the sum of the average weight of each query term (w_t/df_t), where w_t is the sum of weights of the term t over all documents in the collection C. The GLOSS approach is easily applied to large numbers of collections, because GLOSS stores only document frequency and total weight information for each term in each collection. However its effectiveness remains undetermined due to limited evaluation.

The Netserf system is an example of incorporating AI technology to locate collections [18]. Netserf extracts structured, disambiguated representations from the queries and matches these query representations to hand-coded representations of the archives (individual collections) using semantic knowledge from WordNet (a semantic thesaurus) and an on-line Webster's dictionary. Manually constructing archive representations makes this approach only practical for a small number of archives.

Fuhr proposes a decision-theoretic approach to solve the collection selection problem [29]. He makes decisions by using the expected recall-precision curve, expected number of relevant documents, and cost factors for query processing and document delivery. He does not report on effectiveness.

Callan *et al.* adapt the document inference network to ranking collections by simply replacing the document node with the collection node [15, 54]. Similar to GLOSS, the collection retrieval inference network stores document frequencies and term frequencies for each term. The query processing is the same as that in the document inference network, except all proximity operators are replaced with the Boolean AND operators. Effectiveness is demonstrated with the experiments using the InQuery retrieval system and 3 TREC collections of 1 GB each. The results show that this method can select the top 50% of subcollections and get about the same level of effectiveness as searching all subcollections. The significance of this work is that it shows that techniques which are effective for document retrieval are also effective for collection selection.

Xu and Callan present further experiments for using the collection retrieval inference network model to search the top 10% of subcollections. In their baseline, the precision drop for the top 30 documents is 30% compared with searching the whole collection. They improve the accuracy of collection selection by adding phrases in the collection selection index and using query expansion [77]. By using query expansion, the precision drop of the top 30 documents reduces to around 10%. But they pay for this precision improvement with the time for query expansion (executing the original query and analyzing its top documents) and the time to process additional 20 terms or phrases added by query expansion for each query.

Among the approaches for collection selection, the collection retrieval inference network model is the one that has been tested more thoroughly than others and shown to be the most effective. In this dissertation, we use the collection inference network model to select collections, and adapt this model to select a relevant partial replica.

2.3.2 Result Merging

The problem of result merging arises from incomparable ranking scores returned by searches of the different collections in a distributed environment. Directly merging results based on the incomparable scores hurts effectiveness [15, 74].

Voorhees *et al.* propose approaches based on document rankings and previous relevance judgments [74]. The first approach is to interleave the document rankings in a round-robin fashion. The second approach is uneven interleaving, biased by the expected relevance of the collection to the query. They demonstrate effectiveness using the TREC collections. The approach is very effective when a query can find a previous resemblance. The approach can be easily adapted to search using different retrieval systems. However, relevance judgments are not always available.

Callan *et al.* propose a merging approach that assumes the scores for collection ranking are available and use the collection scores to weight document scores from different collections [15]. This approach is effective when the same retrieval system is used to search all of the collections, and works for dynamic collections. The effectiveness of this approach is demonstrated in the experiments with the InQuery retrieval system on 3 TREC collections of 1 GB each. Ranking based on this approach is about as effective as ranking based on global scores and more effective than Voorhees *et al.*'s. We use this approach in our system.

2.4 Parallel Information Retrieval

Although distributed information retrieval exploits parallelism, we use *Parallel Information Retrieval* to refer to information retrieval implemented on a tightly coupled multiprocessor in this dissertation.

There have been a number of papers regarding using multiprocessor machines for information retrieval [6, 20, 21, 26, 47, 67, 68, 66]. Most of them use a distributed memory, massively parallel processing (MPP) architecture [6, 21, 26, 57, 66, 67, 68]. In this dissertation, we investigate how to exploit symmetric multiprocessors (SMPs) to build parallel IR servers, because SMPs are the most popular and affordable multiprocessors today. The previous work using SMPs either compares the cost factors of SMP architecture with other architectures or it investigates a subset of the system such as the disk system [47]. Although commercial information retrieval systems, such as the Web search engines AltaVista and Infoseek exploit parallelism, parallel computers, and other optimizations to support their services, they have not published their hardware and software configurations.

Couvreur *et al.* analyze the tradeoff between performance and cost when searching large text collections [20]. They use simulation models to investigate three different hardware architectures: a mainframe, a collection of RISC processors connected by a network, and a special purpose machine. The experiments using a mainframe are most related to our work. They measure the response time under different query arrival rates and identify the query arrival rate the system can support within 30-40 seconds. By using a 4-CPU IBM 3090/400E mainframe, they achieve 45 searches per minute when searching a 14 GB collection.

Jeong and Omiecinski investigate two inverted file partitioning schemes in a shared-everything multiprocessor system [47]. One scheme partitions the posting file by term identifiers while the other scheme partitions the posting file by document identifiers. They focus on the effect of adding disks on system performance. They show that response time decreases as the number of disks increases up to some threshold. Partitioning based on term identifiers performs the best when the term distribution is less skewed (i.e., when the term distribution in the query is uniformly distributed).

Partitioning based on document identifiers performs the best when term distribution is highly skewed.

Recently the TREC conference [42] reported results for SMPs to process a single query (rather than in a loaded system as we do here) on a 100 GB collection. Their fastest SMP system used a 8 CPU 266 MHZ Alpha with 8 disks and achieved the response time for a single query less than 2 second. In this report, the InQuery system used a 4 CPU 167 MHZ Sun Ultra and achieved 500 seconds for a query, because it used slower processors and very long queries (more than 50 terms).

Other related studies use MPPs and focus on how to speed up single query processing. Stanfill *et al.* implement their IR system on the connection machine (CM), which is a fine-grained, massively parallel distributed-memory SIMD architecture with up to 65,536 processing elements [66, 67, 68]. Bailey and Hawking report their IR system on Fujitsu AP1000, which is a 128-node distributed-memory multicomputer and each node has a 25 MHZ CPU and 16 MB memory [6]. Cringean *et al.* and Efraimidis *et al.* implement their IR systems on a transputer network, which belongs to the MIMD class of parallel computers [21, 26].

In this dissertation, we focus on a parallel server built on symmetric multiprocessors and analyze how different system parameters such as the number of threads, the number of CPUs, and the number of disks affect the performance of a heavy loaded system (see Section 6.1). Besides measuring response time, we also measure the system utilization and investigate how to balance hardware resources. We only consider partitioning based on document identifiers in our experiments on parallel IR server, because observations on query term frequency distributions in previous work show that query term distribution is skewed and users tend to use frequently used terms in the collection [12].

2.5 Data Replication

2.5.1 Replication Strategies

Replication has been studied in areas of distributed file systems and distributed database systems in order to improve system availability and performance. Much work, especially early work, focuses on availability due to replication [2, 3, 16, 19, 25, 30, 51, 65, 69]. Relatively few papers focus on performance due to replication [1, 24, 44, 76].

Replication strategies can be classified as static and dynamic replication, based on when the system decides what to replicate. The File Allocation Problem (FAP) is an example of static replication, where data is replicated at design time based on probable access patterns, and remains unchanged during run time [24]. The static scheme works well if the access pattern is known a priori. However access patterns are not always predictable. In order to overcome this drawback, researchers have started to study approaches to dynamic data replication [1, 44, 76]. For example, Acharya and Zdonik propose a dynamic replication scheme based on the object access pattern [1] in a distributed database system. They use a finite automaton based structure to maintain statistics about reads and writes, and use it to learn access patterns. They use predicted access sequences to dynamically reorder, replicate, or delete copies of data in the network in order to reduce network message costs.

Replication strategies also can be classified as eager replication (synchronous replication) and lazy replication (asynchronous replication), based on how to deliver update information to all replicas [33, 60]. Eager replication makes changes to all replicas at the same time in order to guarantee strict data convergence and the most up-to-date data. Lazy replication immediately records information about replicated activities but delivers them later. Data will converge to consistent values over time.

The work on replication in distributed file systems and database systems can not directly apply to our work, because the objects we are working on are different from theirs. Our objects are text, generally read-only. while their objects are files or structured records, and read-write. The data consistency is thus critical in their applications. Of course we can borrow some ideas from their work, for example, we construct our replicas based on access patterns.

Recently, researchers have used replication techniques to solve the problem of scale that occurs in large-scale distributed information systems such as the Web. Katz *et al.* reported a prototype of a scalable Web server [49]. They treat several identically configured HTTP servers as a cluster, and use the DNS (Domain Name System) service to distribute HTTP requests across the cluster in a round-robin fashion.

Bestavros proposes a hierarchical demand-based replication strategy that optimally disseminates information from its producer to servers that are closer to its consumers in the environment of the Web [7]. The level of dissemination depends on the popularity of that document (relative to other documents in the system) and the expected reduction in traffic that results from its dissemination.

Baentsch *et al.* implement a replication system called CgR/WLIS (Caching goes Replication/Web Location and Information Service) [5]. As their name suggests, CgR/WLIS turns Web caches into replicated servers as needed. In addition, the primary servers forward the data to their replicated servers. A name service WLIS is used to manage and resolve different copies of data.

Although we also organize replicas as a hierarchy, our work is different from these works on Web servers, because our system is a retrieval system while their servers only contain Web documents.

2.5.2 Server Selection

When multiple replication servers exist in the system, the system needs to select a "best" server in order to minimize network traffic and distribute the workload.

The simplest strategy is random selection among a set of replicas [49]. However this approach only works when the network and server times for requests are nearly equal. Guyton and Schwartz investigate several server location techniques based on hop counts as a distance measure to locate the nearby servers [34]. They collect the information about hop counts by using router support, route probing, and hop-count probing. They show that route probing is a method with a reasonable cost and hop-count probing is the most portable method. All their methods need the network topology as the input. However knowledge of the network topology is sometimes hard to acquire.

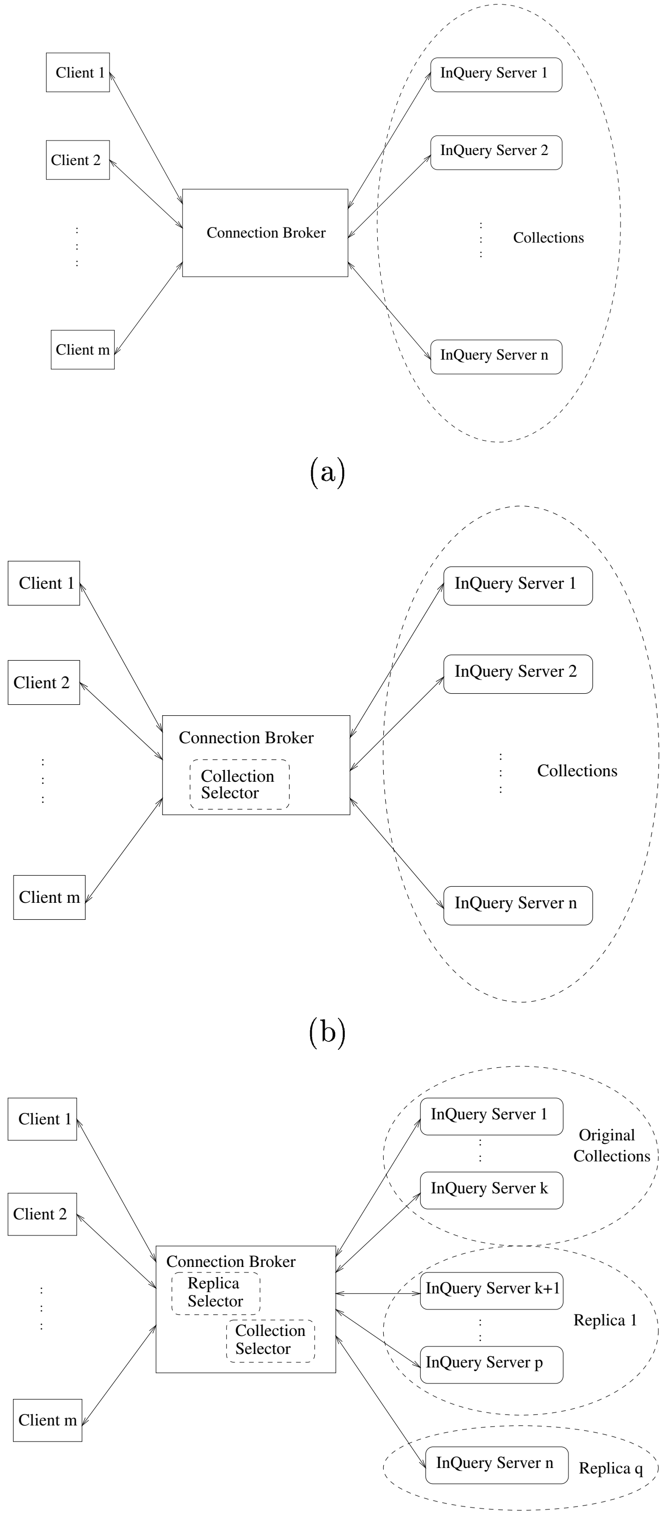
Carter and Crovella propose a dynamic server selection strategy based on round-trip measurement and available bandwidth [17]. In order to measure the available bandwidth, they develop two tools: BPROBE which uses ECHO packets to measure the bottleneck link speed of paths between hosts in the Internet, and CPROBE which estimates the congestion along a path. They show that performance of this system is much better than those of using hop counts as a measure. They also discover that the server load is an important factor to determine the best server, but their experiments do not show improvement using this factor due to the high overhead of measuring server load.

Bhattacharjee *et al.* investigate using application-layer anycasting to select the best server [8]. They propose a general architecture for server selection based on application-layer anycasting, which can support any metric. In particular, they discuss four metrics: server response time, server-to-user throughput, server load, and processor load. They use the anycast domain name to identify replicated network services. An ADN (anycast domain name) resolver is used to resolve the name. They investigate four possible approaches to maintain information in the anycast servers' database: remote server performance probing, server push, reading server logs, and user experience. Their results show that server push and reading server logs can result in high-accuracy, but they require modifications of current servers in order to collect the metrics; user experience puts the least overhead on collecting metrics, but its accuracy is low and varied.

The above technologies are directly applicable to our work to help us find a server with the least loaded or in the nearest place. However there is a problem in our system that does not exist in their systems: how to select a replica based on relevance, because we only partially replicate collections which leads to partial replicas and original collections that are not equally effective for a given query. We propose an approach to select partial replicas based on relevance and server load.

CHAPTER 3
SYSTEM ARCHITECTURES

This chapter describes the architectures of our distributed information retrieval system, as shown in Figure 3.1. The distributed system enables multiple clients to simultaneously access multiple collections over a network. As shown in Figure 3.1(a), the basic components of the system are a set of clients, a connection broker, and a set of InQuery servers for storing indexed collections, where are connected by a network. In this dissertation, we use a local area network. In order to improve system performance and eliminate bottlenecks, we add full or partial replicas of collections, collection selectors, replica selectors, and replicas of the connection broker. Figure 3.1(b) and (c) illustrate two examples of our potential architectures. In Figure 3.1(b), we add a collection selector in the connection broker to select the most relevant collections on a query-by-query basis and restrict the search to the selected collections. In Figure 3.1(c), we build partial replicas for the original collections, add a replica selector in the connection brokers to direct as many queries as possible to relevant partial replicas, and for the queries going to the original collection, we use a collection selector to restrict the search to the most relevant collections. For high performance, we build each server on a symmetric multiprocessor computer. In the rest of this chapter, we describe the functionality of each component and their interactions in details.



(a)

(b)

(c)

Figure 3.1. Architectures for distributed information retrieval.

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 broker. The clients initiate all work in the system, but perform very little computation. They issue IR commands and then wait for responses. The clients can issue the entire range of IR commands but, in this dissertation, we focus on three basic IR commands: *query*, *summary*, and *document retrieval* commands.

In InQuery, queries could be natural language queries or structured queries. We focus on natural language queries in this dissertation. A natural language **query command** consists of a set of words or phrases (terms), such as "information retrieval", or "distributed system." Query responses consist of a list of document identifiers ranked by belief values which estimate the probability that the document satisfies the information need.

For each query, a client may obtain one or several summaries on relevant documents by sending **summary commands**. A summary command consists of a set of document identifiers and their collection identifiers. The summary information of a document typically consists of the title and the first few sentences of the document. It may also include information such as source and organization.

A client may also retrieve complete documents by sending a **document retrieval command**. The command consists of a document identifier and a collection identifier. In response, the system returns the complete text of the document from the collection.

3.2 Collections and InQuery Servers

A collection is a set of documents the retrieval system is working on. A large collection could contain millions even billions of documents. For simplicity, we assume there are no overlaps between documents in any two collections. In this dissertation, we use the InQuery retrieval engine [14, 72] as our testbed to index collections and provide IR services such as evaluating queries, obtaining summary information, and retrieving documents. We refer to the server as the InQuery server. The InQuery server accepts a command from the connection broker, processes the command, and returns its result back to the connection broker.

3.3 Connection Broker

Clients and InQuery servers communicate via the connection broker. The connection broker is a process that keeps track of all registered clients and InQuery servers. A client sends a command to the connection broker which forwards it to the appropriate InQuery servers. The connection broker maintains intermediate results for commands that involve multiple InQuery servers. When an InQuery server returns a result, the connection broker merges it with other results. After all InQuery servers involved in a command return results, the connection broker sends the final result to the client. Besides keeping track of all clients and InQuery servers, we may also enhance the connection broker to perform collection selection or replica selection.

3.3.1 Collection Selector

When there are a large number of collections, we use a collection selector to automatically select the most relevant collections on a query-by-query basis. A collection selector maintains a collection selection database that contains collection-level information on data distributed over all InQuery servers. When the collection selector receives a query, it searches the collection selection database and returns a list of the most relevant collection identifiers and corresponding collection ranking scores. The connection broker uses the collection ranking scores to weight document scores from different collections in order to produce comparable overall document rankings (see [15]).

3.3.2 Replica Selector

When we only replicate a small portion of the original collection(s) for efficiency, we need a replica selector to direct as many queries as possible to partial replicas based on both relevance and loads. As opposed to collection selection which ranks disjunctive collections, replica selection ranks partial replicas and the original collection(s), which hold a subset relationship. A partial replica could be a replica of a single collection, or a replica of several collections. In Figure 3.1(c), we view all original collections as a whole, and build partial replicas for the whole original collections. In our system, we organize our partial replicas as a hierarchy: a smaller replica is a subset of larger replicas and the original collection(s) (see Section 4.2 for details). A replica selector maintains a replica selection database and the load information of each server. For each query command, the replica selector searches this database and returns either a replica identifier if there is a relevant replica, or a collection identifier. When the replica selector returns a replica identifier, it sends the query to the replica if it is not overloaded, otherwise it sends the query to a larger replica or the server(s) that store(s) the original collection(s).

3.3.3 Interactions

For a query command, the connection broker could perform different functions according to the system architecture. In the base system illustrated in Figure 3.1(a), the connection broker sends the query to all InQuery servers, and then merges the results. In the system illustrated in Figure 3.1(b), the connection broker uses a collection selector to obtain a list of top collections, sends the query to the corresponding InQuery servers maintaining these collections, and then merges the results. In the system illustrated in Figure 3.1(c), the connection broker first uses a replica selector to determine whether there is a partial replica that is not only relevant to the query, but also not overloaded; If there is one, the connection broker sends the query to the

InQuery server(s) that maintain the relevant replica, otherwise it uses a collection selector to obtain a list of top collections out of the original collections, and sends the query to the InQuery servers that maintain the selected collections.

For a summary command, the connection broker sends the command to the InQuery servers whose identifiers are described in the command. The connection broker merges the summary information responses and sends a single message back to the client.

For a document retrieval command, the connection broker sends the command to the InQuery server that contains the document, and then forwards the document to the client as soon as it receives the document from the InQuery server.

CHAPTER 4

PARTIAL COLLECTION REPLICATION

Partial collection replication is truly useful for information retrieval only when it satisfies the following four conditions:

1. Query and document accesses localize in a small portion of a collection within a period of time.

2. There is an effective tool to select a relevant partial replica.

3. The time and space overheads for replica selection are reasonable.

4. The updating costs are reasonable.

In this chapter, we discuss the issues related to these conditions. The remainder of this chapter is organized as follows: we examine the locality in real systems in Section 4.1, present our partial replication architecture in Section 4.2, explore how to select a relevant replica in Section 4.3, discuss how to select a replica based on loads in Section 4.4, investigate space and time overheads for the replica selection database in Section 4.5, investigate updating costs and strategies in Section 4.6, and summarize in Section 4.7.

4.1 Access Characteristics in Real IR Systems

We examine the locality of access in the logs of real IR systems. We analyze two server access logs, one from THOMAS [52], and the other from Excite [27]. The THOMAS system is a legislative information service of the U.S. Congress through the Library of Congress, which uses InQuery as the retrieval engine. It contains the full text of all bills introduced in the 101st - 105th Congresses as well as the text of the Congressional Records for those Congresses. We analyze the logs of THOMAS between July 14 and September 13, 1998. During this period, we obtained the full day logs for 40 days, and partial logs on the remaining 22 days. The reasons for the latter was lack of disk space in the mailing system of the Library of Congress when they sent us the logs. The Excite system provides online search for more than 50 million Web pages. We obtained the Excite for the day of September 16, 1997.

Figure 4.1 lists some excerpts from the THOMAS and the Excite logs. The THOMAS logs contain information on both query and document commands. For each command, the system records the date and time for receiving the command, the database identifier, and the response time. In addition, for each query, the system records query terms and creates a server-side temporary file to save the search results; For each document command, the system records its reference to a temporary file that saves the search results for the corresponding query. The Excite log only contains information on query commands. It contains the user identifier, the date and time for receiving the query and query terms for each query.

In examining the logs, we paid attention to the following characteristics.

1. The ratio of query processing and document access. The ratio determines which database components we need to replicate to maximize the benefit of partial collection replication.

2. Query locality within a period of time, which determines the usefulness of partial replication.

3. The overlap of queries between days and weeks, which determines how often we need to update the replicas.

```
      Date & Time         DB ID.   Queries or Document Commands
------------------------------------------------------------------------
[Tue Jul 14 00:14:54 1998] c105: bliley (1 sec.)
[Tue Jul 14 00:14:59 1998] c105: taxpayer  relief act (0 sec.)
[Tue Jul 14 00:15:02 1998] r105: GET 2:./temp/~r105lrYH9g:e8650: (0 sec.)
[Tue Jul 14 00:15:02 1998] c105: GET 1:./temp/~c1051k16ck:e13399: (0 sec.)
[Tue Jul 14 00:15:05 1998] c105: Patients (1 sec.)
[Tue Jul 14 00:15:09 1998] c105: GET ./temp/~c105xXJ6Ff (2 sec.)
[Tue Jul 14 00:15:16 1998] c105: religion, persecution (1 sec.)
[Tue Jul 14 00:15:18 1998] r105: Irian Jaya (1 sec.)
[Tue Jul 14 00:15:20 1998] r102: GET 7:./temp/~r102B83a2h:e35483: (1 sec.)
[Tue Jul 14 00:15:24 1998] c105: GET 2:./temp/~c105A4ny:: (0 sec.)
[Tue Jul 14 00:15:38 1998] c105: GET S.1882: (2 sec.)
```

(a) Excerpts from the THOMAS log

```
   User ID.             Date & Time        Queries
--------------------------------------------------------------------
54DBD92B4F227161        970916121245    adams mark hotel
54DBD92B4F227161        970916121344    adams mark hotel tulsa
B94D2253628531DC        970916154549    site maps
B94D2253628531DC        970916154551    site map
B94D2253628531DC        970916154632    site map servers
B94D2253628531DC        970916154918    site map virtual
B94D2253628531DC        970916160709    web site maps
```

(b) Excerpts from the Excite log

Figure 4.1. Excerpts from the THOMAS and Excite logs.

4.1.1 Ratio of Query Processing and Document Access

We know an IR system has three basic IR commands: query, summary, and document commands. Among them, query commands access term inverted list files to produce a ranked list of relevant document identifiers; summary commands and document commands access document inverted list files and the original data files to obtain summary information for a set of documents and the full text of a document. According to the database components each command accesses, we categorize the commands into two categories: query processing that includes the query command, and document access that includes both summary and document commands. If one

queries : documents		
average	minimum	maximum
1:1.9	1:2.4	1:0.4

percentage of documents falling within ranks									
1-10	11-20	21-30	31-40	41-50	51-60	61-80	81-100	101-200	>200
75.8%	9.5%	5.2%	3.7%	3.6%	0.3%	0.5%	0.4%	0.5%	0.6%

Table 4.1. Ratios of queries and documents in the THOMAS log.

of them accounts for the majority of system processing time, we only need to replicate a subset of database components to improve system performance.

We examine the THOMAS logs for the ratio of query processing and document accesses, where, on the average, the system receives 5928 queries and 10433 document commands per day. We do not examine the Excite log, since it does not contain information about document access. Table 4.1 lists the average, minimum, and maximum ratios of queries and viewed documents per day, and the average percentages of viewed documents that are ranked as from m to n (see Table B.6 in Appendix B for more details). On the average, for each query command, the user viewed 1.9 documents. 75.8%, 90.5% and 97.8% of the documents the users viewed were ranked in the top 10, the top 30, and the top 50, respectively. Only 0.6% of documents the users viewed were ranked higher than 200.

Since we do not have system measurements of the THOMAS system, we estimate the percentage of the total system processing time used for each type of IR commands by building a test database using the Congress Record for 103rd Congress (235 MB, 27992 documents), which is a part of the TREC 4 collection [39] and one of the collections THOMAS is using. We ran the top 1000 unique queries issued on July 14, 1998 against this test database. The average terms per query for these 1000 unique queries was 1.9. We assume each summary command returns the summary information for 10 documents, and calculate the average number of summary command based on the document ranks the users viewed in Table 4.1. For each query command, the user issued 1.5 summary commands. When we obtained the times for processing each type of commands, we set the ratio of the three basic IR commands (query:summary:document) to be 1:1.5:2, and chilled the system by reading a large file before processing each IR command. The ratio of time used for processing query command, summary command, and document commands is 1:1.2:0.07, which means the query processing accounted for 44% of total processing time, obtaining summary information accounted for 53%, and document retrieval accounted for 3%.

Our analysis suggests that our partial collection replica should be beneficial for speeding up both query processing and document access (summary and document commands), since each of them accounts almost a half of the total system processing time.

4.1.2 Query Locality

We examine the query locality in the THOMAS and the Excite logs. Since the logs do not contain document identifiers returned from each query evaluation, we built test databases. For queries from the THOMAS log, we reran all queries against a test database that uses the Congress Record for 103rd Congress (235 MB, 27992 documents). For queries from the Excite log, we reran all queries against a test database using downloads of the websites operated by ten Australian Universities (725 MB, 81334 documents), which is a part of the TREC 6 collection (see Appendix A). We use the test databases to automatically cluster queries. We group the queries whose top 20 documents completely overlap as a topic, i.e., we view these distinct queries as the same.

Table 4.2 shows the statistics on query locality from the THOMAS and Excite logs. We collect the average number of queries, unique queries, topics, topics occurring once, topics occurring more than once, and topics that contain more than one unique query. We also present the percentage of queries that correspond to the top topics.

Num. queries	Num. unique queries	Topics			
		total	occurring once	more than once	more than one unique query
8143 (7703)	4876 (4651)	4069	2888 (71%)	1181 (29%)	412

percentages of queries that top topics account for				
100	200	500	1000	2000
21.2%	28.7%	41.5%	54.1%	73.0%

(a) Query locality in the THOMAS log.

Num. queries	Num. unique queries	Topics			
		total	occurring once	more than once	more than one unique query
499836 (444899)	365276 (320987)	249405	196672 (79%)	52733 (21%)	32750

percentages of queries that top topics account for				
500	1000	5000	10000	20000
12.3%	16.0%	27.9%	34.4%	42.0%

(b) Query locality in the Excite log.

Table 4.2. Query locality in the logs.

Table 4.2(a) shows the average numbers in the THOMAS logs over the 40 days with full day logs (see Table B.1 and Table B.2 in Appendix B for more details). The numbers in the parentheses in columns 1 and 2 are the numbers of queries that actually found matching documents from our test database. Some queries did not find any matching document, due to misspelling, or simply because the query terms did not exist in the test database. The statistics show that on the average, 71% of topics occur once, and the remaining 29% of topics account for 63% of queries. Among the topics that occur more than once, 35% (412) contains more than one unique query, which means query pattern match would not find this overlap. The top 2.5% of topics (top 100 topics) account for 21.2% of queries, and the top 12% of topics (the top 500 topics) account for 41.5%.

Table 4.2(b) shows the numbers in the Excite log on September 16, 1997. The numbers in the parentheses are the numbers of queries that actually found matching documents from our test database. The statistics show that the Excite queries also have high query locality: 79% of topics occur once, and the remaining 21% of topics account for 61% of queries; Among the topics that occurred more than once, 62% (32750) contain more than one unique query. The top 2% of topics (top 500 topics) account for 12.3% of queries, and the top 10% of topics (the top 20000 topics) account for 42.0%.

These statistics suggest that it is possible for us to replicate a small portion of collection that contains documents for the top queries. In addition, since a significant amount of topics contains more than one unique query, caching queries may miss many overlaps between queries with different terms that in fact return the same top documents. Instead, we make our replicas searchable to improve both query processing and document access.

4.1.3 Overlap of Queries Over Time

In this section, we examine how many queries on a given day match a topic from the previous days in the THOMAS logs. We collect the percentages of queries that appear in the previous day, and in the previous seven days. We also collect the percentages of queries that appear on a particular day or a particular week, in order to examine query overlap over time. Table 4.3 summarizes the statistics (see Table B.3-B.5 for more details).

Table 4.3(a) shows the average percentages of queries on a given day such that the topics of these queries match a topic, or one of the top 1000 topics from the previous day or the previous week, respectively. We calculate the numbers in the table from 28 days of our logs. We exclude the first 7 days and include the second day of the remaining pairs of days which have full logs for both days. We exclude the days in the first week to enable comparisons between the numbers for the previous day and the previous week.

	Overlap with					
	the previous day			the previous week		
	ave.	min.	max	ave.	min.	max
all	43.1%	31.2%	81.1%	57.8%	49.1%	82.6%
top 1000	32.4%	21.3%	78.1%	34.6%	26.2%	77.8%

(a) overlap with the previous day(s).

	Overlap with					
	7/14/98			the week on 7/14-7/20/98		
date	all	top 500	top 1000	all	top 500	top 1000
7/15	43.3%	24.8%	30.1%	n/a	n/a	n/a
7/16	42.6%	24.0%	29.2%	n/a	n/a	n/a
7/17	42.0%	23.5%	28.7%	n/a	n/a	n/a
7/21	41.6%	23.6%	28.5%	60.2%	29.0%	36.0%
7/22	42.5%	24.0%	28.8%	60.7%	29.8%	36.8%
7/23	41.4%	23.4%	28.7%	61.0%	29.2%	36.8%
7/31	38.5%	21.9%	26.4%	59.1%	27.3%	34.2%
8/14	38.1%	21.3%	26.0%	54.6%	26.8%	32.3%
8/28	34.3%	18.3%	23.4%	54.0%	23.5%	30.7%
9/11	44.0%	8.7%	22.2%	82.7%	54.5%	64.5%

(b) overlap with a particular day or week.

Table 4.3. Query overlap over time in the THOMAS log.

When our replica contains the top documents for all the topics occurring on the previous day, 43.1% of queries match a topic in the replica on the average, with the minimum of 31.2% and the maximum of 81.1%. When our replica just contains documents for the top 1000 topics occurring on the previous day, 32.4% of queries match a topic in the replica on the average, with the minimum of 21.3% and the maximum of 78.1%. When our replica contains documents for all topics occurring on the previous seven days, 57.8% of queries matched a topic in the replica on the average, with the minimum of 47.0% and the maximum of 82.6%. When our replica contains documents for the top 1000 topics occurring on the previous seven days, the 34.6% of queries match a topic on the average, with the minimum of 26.2% and the maximum of 77.8%. As compared with the overlap with the previous day, 34.1%

and 6.7% more queries match a topic in the replica of the previous week, when we replicate all topics and top 1000 topics, respectively.

Table 4.3(b) shows the average percentages of queries on a given day such that these queries matched a topic, or one of the top 500, or one of the top 1000 topics on July 14, 1998 and in the week from July 14 to July 20, 1998, respectively. The statistics show that typically query overlap tends to decrease as time elapses, but occasionally query overlap could increase significantly, e.g, on 9/11/98. In the typical days when the query overlap decrease, the decrease is very gradual, for example, let (m,n) represents the query overlap between the day(s) of m and the day of n, the difference between (7/14, 7/15) and (7/14, 7/17) is 1.3%, 1.3%, and 1.4% when we replicate all topics, the top 500 topics, and top 1000 topics on 7/14, respectively. The difference between (7/14-7/20, 7/21) and (7/14-7/20, 7/23) is 0.8%, 0.2%, and 0.8% when we replicate all topics, the top 500, and top 1000 topics on 7/14-7/20, respectively.

The statistics on query overlap suggests that if we replicate the top topics on the previous days, significant amount of queries can match a topic in the replica. Actually the replica can satisfy more queries than the numbers we report in the tables on query overlap, because we do not count the queries whose topics do not appear on the previous days, but their top documents returned from query evaluation appear in the replica, i.e. the set of top documents of a query is a combination of top documents of another two or more topics. Since the logs do not contain document identifiers and our test database is pretty small, we can not obtain the figures about this situation.

Summary

Both the THOMAS logs and the Excite log show there exists high query locality in the real IR systems. Caching queries and query exact matching miss many overlaps between queries with different terms that in fact return the same top documents. If we replicate topics appeared on the previous day(s), the replica can satisfy 20% to 80% of queries. Since the IR systems typically perform significant amount of query processing and document access, our partial replica should support both operations.

4.2 The Partial Replication Architecture

Since our goal is to improve system performance for both query processing and document access, each replica is a searchable partial collection, i.e., each replica includes replicated documents and their corresponding indexes. We determine which portion of documents to replicate in the following way: for a given query, we tag all top n documents returned by query processing as "accessed" and count their access frequencies, regardless of whether the user requests the full text of these documents. We keep the access frequency of each document within a period of time, such as a week or a month, and then replicate documents based on their access frequencies: the more frequently "accessed" the documents, the more times they are replicated.

We organize replicas as a hierarchy, illustrated in Fig 4.2. The top node represents an *original collection* that could be *an actual collection* residing on a network node or *a virtual collection* consisting of several collections distributed over a network. The bottom nodes represent users. We may divide users into different clusters, each of which corresponds to a group of users that reside within the same domain, which could be an institution, an organization, or a geographical area such as a state or a country. The inner nodes represent partial replicas. The replica in a lower layer is a subset of the replicas in upper layers, for example, Replica 1-1 $\subset$ Replica 1 $\subset$ Original Collection. The replica that is closest to a user cluster contains the set of documents that are most frequently used by the cluster of users. The replica may contain frequently used documents for more than one cluster of users, for example, Replica 1 contains the frequently used documents for n user clusters. The solid lines illustrate data flows disseminated from the original collection to replicas. Along the

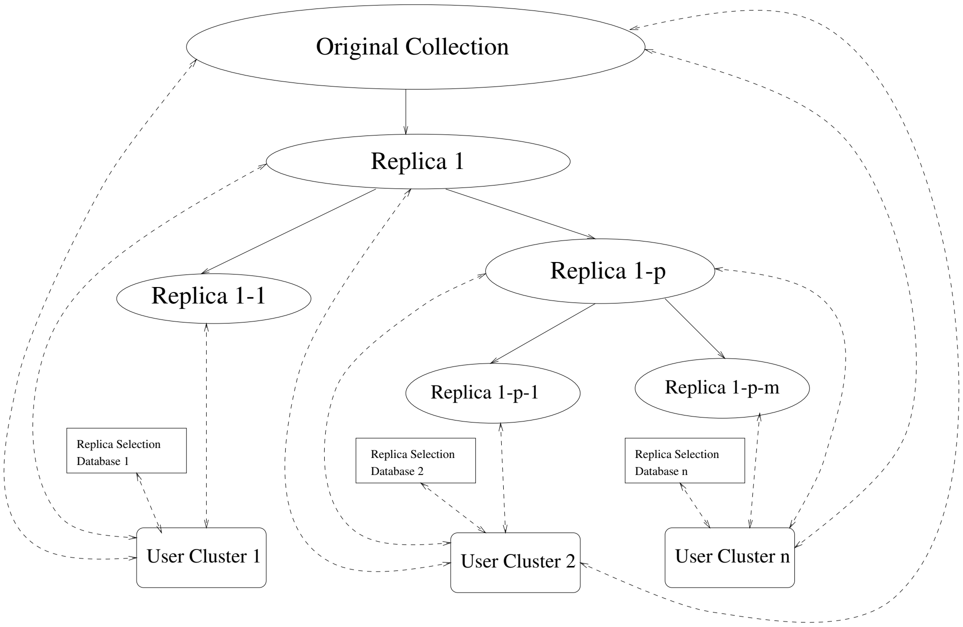


Figure 4.2. The replication hierarchy.

arcs from the original collection, the most frequently used documents are replicated many times. Requests from the users in a cluster may go to any replica or the original collection along the arcs from the top node depending on relevance and other criteria, such as server loads. We build a replica selection database to rank replicas based on relevance, which directs queries to a relevant partial replica. The dotted lines illustrate the interaction between users and data. If we do not divide the users into different groups, the hierarchy simplifies to a linear hierarchy. In this architecture, replica selection is a two-step process:

1. ranking partial replicas and the original collection based on relevance.

2. selecting one of the most relevant replicas or the original collection based on loads.

Here we give an example to show how this architecture works. In Figure 4.2, we replicate the top documents of the most frequently used queries issued by the users in user cluster 1 in both Replica 1-1 and Replica 1. Replica 1 also contains other less frequently used documents for user cluster 1, and frequently used documents for other user clusters. When a user in the user cluster 1 issues a query, the query goes to replica selection database 1 to determine relevance of Replica 1-1, Replica 1, and the original collection. If the replica selector ranks Replica 1-1 as the top one, it may send the query to any of Replica 1-1, Replica 1, or the original collection for processing, based on other criteria, such as server load, network traffic, and network distance. If the selector ranks Replica 1 as the top one, it may send the query to either Replica 1 or the original collection. Otherwise it always sends the query to the original collection.

4.3 Partial Replica Selection Based on Relevance

The first step of replica selection is how to find a partial replica that contains enough relevant documents for a given query. In this section, we investigate how to do this task with inference networks, and evaluate the effectiveness of our replica selection approach using the InQuery retrieval system [14], and the 2 GB TREC 2+3 collection and the 20 GB TREC VLC collection. We use queries developed for TREC topics 51-350 in our experiments. We compare our proposed replica selection function with the collection ranking function. We measure the system's ability to pick the expected partial replica, and the precision of the resulting response as compared with searching the original collection.

The rest of the section is organized as follows: Section 4.3.1 investigates how to rank partial replicas and the original collection using the inference network model, Section 4.3.2 describes the experimental settings, Section 4.3.3 compares our proposed replica selection function with the collection selection function, Section 4.3.4 and Section 4.3.5 further demonstrate the effectiveness of our approach for both replicated queries and unreplicated queries, and Section 4.3.6 summarizes the results of this section.

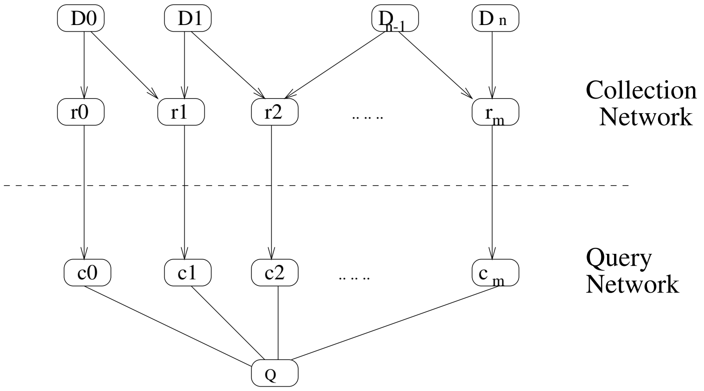


Figure 4.3. The collection retrieval inference network.

4.3.1 Ranking Partial Replicas with the Inference Network Model

We adapt the collection retrieval inference networks that Callan *et al.* propose to rank collections [15] to rank partial replicas and the original collection. The collection retrieval inference network model consists of two component networks: a collection network and a query network, illustrated in Figure 4.3. The D_i nodes correspond to collections, and the r_j nodes correspond to concepts in the collections. The Q node represents a query, and the c_i nodes correspond to query concepts in the query. By using the collection retrieval inference network, collection ranking becomes an estimate of $P(I|D_i)$ from combining the conditional probabilities through the network. When we adapt the collection retrieval inference network model to rank replicas, we use D_i nodes to represent the original collection and partial replicas, where D_n represents the original collection, $D_i, i = 1, 2, ...n - 1$ represent partial replicas, and $D_1 \subset D_2 \subset ... \subset D_n$. (In the collection retrieval inference network, the D_i nodes do not hold the subset relationship.) The purpose of ranking partial replicas is to find a *single* replica that satisfies a given query instead of a subset of collections in the collection retrieval inference network. We refer to this inference network as to the replica selection inference network. As in the collection retrieval inference network model, $P(c_k|r_j)$ is set to 1.0. The central work of applying this inference network

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

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

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

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

Figure 4.4. The collection ranking function in InQuery.

to replica selection is to develop an effective replica ranking function to estimate $P(r_j|D_i)$.

Since we adapt the collection retrieval inference network, we first examine whether the InQuery collection ranking function works well with ranking partial replicas. The InQuery collection ranking function uses *df* (the document frequency of each term) as the basic metric, as shown in Figure 4.4 [15]. In our experiment settings in Section 4.3.2, the default InQuery collection ranking function directs more than 70% of the replicated queries to the original collection, however, since we use these replicated queries to build replicas, the replica selector should direct them to the replicas instead of the original collection. Although we can tune the parameters of the InQuery collection ranking function to direct more queries to the replicas, the precision drops too much, for example, the precision drops approximately 25% when the function directs 80% of replicated queries to the replicas (see Section 4.3.3 for the details). The InQuery collection ranking function does not work well with replica selection, because it favors collections with larger *df*, but partial replicas typically have smaller *df* than the original collection.

Since a partial replica contains the top documents of the most frequently used queries, by examining the document ranking function, we know that the top documents are ranked as the top, just because query terms occur more often in these documents than the others. Therefore if a replica contains the top documents for a query, the average term frequency of each query term in the replica should be higher than in the original collection. Based on this heuristic, we construct a replica selection function based on the average term frequency. In addition, we find a term is important in selecting replicas if it occurs often (middle or high term frequency) in that replica/collection and it also occurs in a certain number of documents (above a cutoff for document frequency). A term occurring in too few documents does not help even though it has high term frequency. We need to cut off these terms. Figure 4.5 illustrates our replica selection function which uses the average term frequency and penalizes the terms that appear less than a cutoff in the corresponding replica/collection. We compare this function with the InQuery collection ranking function in Section 4.3.3, and demonstrate its effectiveness using 350 TREC queries on a 2 GB collection and a 20 GB collection in Section 4.3.4 and Section 4.3.5.

In order to automatically set the cutoff value in each replica, we examine the statistics of the document frequency in each replica/collection. We find there is an approximate relationship between average document frequency and the number of documents, and between maximum document frequency and the number of documents in replicas with different sizes, illustrated in Figure 4.6. Figure 4.6(a) shows the statistics for short queries from test trial 1 described in Section 4.3.2. Figure 4.6(b) illustrates the equations that describe the relationship between document frequencies derived from the statistics. When we replicate the top n documents for each frequently used

$$ave\_tf = \frac{ctf_{ij}}{df_{ij}}$$

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

$$T = \frac{AT}{AT + k \cdot ((1-b) + b \cdot \frac{ave\_doclen_i}{ave\_ave\_doclen}}$$

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

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

where

ctf_{ij} is the number of occurrences of term r_j in collection/replica D_i,

df_{ij} is the number of documents that contain term r_j in collection/replica D_i,

$cutoff_i$ is the cutoff number of documents in replica/collection D_i,

N is the number of replicas plus the original collection,

rf is the number of replicas and the collection that contain r_j,

ave_doclen_i is the average document length in collection/replica D_i,

ave_ave_doclen is the average ave_doclen_i,

k is a constant that controls the magnitude of AT,

b is a constant varying from 0 to 1 used to control the sensitivity of the function to ave_doclen, and

α is a default belief (set to 0.4).

Figure 4.5. The replica selection function.

query, we set the cutoff of the smallest replica as n, and set the cutoffs for other replica/collection using the equation shown in Figure 4.6(c).

We implement the replica selection inference network as a pseudo InQuery database, where each pseudo document corresponds to a replica or collection, its index stores the *df* (document frequency) and *ctf* (replica/collection term frequency) for each term. We do not store any proximity information in order to minimize the space requirements of the replica selection database. As in the collection retrieval inference network, all proximity operators are replaced with Boolean AND operators.

Top n	repl_db	Docs (DN)	ave_df	$\frac{\log(DN)}{\log(ave\_df)}$	max_df	$\frac{\log(DN)}{\log(max\_df)}$
100	$D1$	4,868	19.18	2.87	2,986	1.06
	$D2$	9,495	25.31	2.83	5,584	1.06
	$D3$	13,738	29.77	2.81	8,181	1.06
	$D4$	17,555	32.44	2.81	10,498	1.06
	$D5$	21,643	34.29	2.83	12,941	1.05
200	$D1$	9,573	23.96	2.88	5,817	1.06
	$D2$	18,336	30.65	2.87	10,868	1.06
	$D3$	26,280	36.47	2.83	15,709	1.05
	$D4$	33,133	40.32	2.82	19,941	1.05
	$D5$	40,611	43.10	2.82	24,319	1.05
500	$D1$	23,133	33.62	2.86	13,837	1.05
	$D2$	42,669	41.72	2.86	25,447	1.05
	$D3$	60,240	49.20	2.83	35,908	1.05
	$D4$	73,747	53.76	2.81	44,329	1.05
	$D5$	88,739	57.48	2.81	52,805	1.05
TREC 2+3 (2 GB)		567,529	101.13	2.87	255,529	1.06

(a) The statistics for short queries on relationship of document frequencies from test trial 1

$$\frac{\log(DN_i)}{\log(ave\_df_i)} = \frac{\log(DN_j)}{\log(ave\_df_j)}$$

$$\frac{\log(DN_i)}{\log(max\_df_i)} = \frac{\log(DN_j)}{\log(max\_df_j)}$$

where
DN_i is the number of documents in replica/collection D_i,
ave_df_i is average document frequency in replica/collection D_i,
max_df_i is maximum document frequency in replica/collection D_i.

(b) The equations for relationship between document frequencies

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

where
$cutoff_1$ is the cutoff value for the smallest replica D_1, which we set as the number of top documents for each query,
$cutoff_i$ is the cutoff value for the replica/collection $D_i, i > 2$,
DN_1 is the number of documents in the smallest replica D_1, and
DN_i is the number of documents in replica/collection $D_i, i > 2$.

(c) The equation for the cutoff

Figure 4.6. The relationship between document frequencies in different replicas with different sizes.

4.3.2 Experimental Settings

We evaluate the effectiveness of our replica selection approach using the InQuery retrieval system [14, 72], and a 2 GB TREC collection that contains TREC 2+3 collections, and a 20 GB collection that contains all TREC 6 VLC collections (see Table A.1). We use queries developed for TREC topics 51-350 in our experiments. We measure the system's ability to pick the relevant partial replica, and the precision of the resulting response as compared with searching the original collection. We use TREC queries instead of the queries from the logs, because some of TREC queries have relevance judgments that enable us to produce precision and recall figures for evaluating the effectiveness.

By using the 2 GB collection, we compare the effectiveness of our replica selection function with the InQuery collection ranking function using short queries, and demonstrate the effectiveness of our replica selection function using both short queries and long queries. A short query is simply a sum of the terms in the corresponding description field of the topic. Long queries are automatically created from TREC topics using InQuery query generation techniques [14], which consist of terms, phrases and proximity operators. Generally, a long query for a topic is more effective than the short query [14]. The average number of terms per query for the set of short queries is 8 after removing the stopwords, and the average number of terms per query for the set of long queries is 120. For each set of queries, we divide queries into two categories: replicated queries and unreplicated queries, where *the replicated queries are those whose top documents are used to build the replicas.* Since only topics 51-150 and topics 202-250 have relevance judgment files for the 2 GB TREC collection, we randomly pick 50 unreplicated queries from these 149 topics and report the effectiveness for these topics only.

By using the 20 GB collection, we examine how the size of collection affects the effectiveness of our replica ranking function. Since we do not have relevance judgment files for topics 51-150 and topics 202-250 against the 20 GB collection, and the 2 GB collection is a subset of the 20 GB collection, we use the relevance judgment files for the 2 GB collection to produce the precision figures.

We conduct our experiments by repeating the following procedure 5 times, each trial uses a different number as the seed to produce random numbers, and thus picks different queries for a query set. In each trial, we randomly choose 50 queries from queries {51-150, 202-250} as our *unreplicated* query set T, and randomly divide the remaining 250 queries in queries 51-350 into 5 sets: $\{Q_i, i = 1, 2, 3, 4, 5\}$, each set containing 50 queries. We then build a 6-layer replication hierarchy by using the 2 GB TREC collection or the 20 GB collection as the original collection C, and collecting the top n documents resulting from searching the original collection for each query in $\{Q_i, i = 1, 2, 3, 4, 5\}$ to build 5 partial replicas $\{D_i, i = 1, 2, 3, 4, 5\}$:

- D_1: contains the top n documents for each query in query set Q_1, $|Q_1| = 50$, total $n * 50$ documents.

- D_2: contains the top n documents for each query in query sets Q_1 and Q_2, $|Q_i| = 50$, total $n * 100$ documents.

- D_3: contains the top n documents for each query in query sets $\{Q_i, i = 1, 2, 3\}$, $|Q_i| = 50$, total $n * 150$ documents.

- D_4: contains the top n documents for each query in query sets $\{Q_i, i = 1, 2, 3, 4\}$, $|Q_i| = 50$, total $n * 200$ documents.

- D_5: contains the top n documents for each query in query sets $\{Q_i, i = 1, 2, 3, 4, 5\}$, $|Q_i| = 50$, total $n * 250$ documents.

Clearly, $D_1 \subset D_2 \subset D_3 \subset D_5 \subset C$. This structure mimics 5 replicas that increase in size and thus include more of the top queries for each size increase. We build a replica selection inference network to rank these five replicas and the original collection. The queries in query sets $\{Q_i, i = 1, 2, 3, 4, 5\}$ are called *replicated* queries.

Table 4.4 lists the average number of documents in each replica over the five trials after deleting duplicated documents, the average number of words, and the average size of replicas when building different replicas by using different numbers of top documents, as well as the statistics from two original collections. When using the 2 GB collection as the original collection, the size of replicas ranges from 0.3% to 1.5%, 1% to 5%, 2% to 10%, and 5% to 20% of the original collection when replicating the top 30, 100, 200, and 500 documents, respectively. When using the 20 GB collection as the original collection, the size of replicas ranges from 0.1% to 0.5%, 0.2% to 1%, and 0.5% to 2% of the original collection when replicating the top 100, 200, and 500 documents, respectively.

For the 20 GB collection, we conduct one more experiment in order to make up for insufficient relevance judgments for topics {51-150,202-250}. We use queries 301-350 as our unreplicated query set T, since these 50 topics are more thoroughly judged against the 20 GB VLC collection than topics {51-150, 202-250}. We use queries 51-100 as Q_1, 101-150 as Q_2, 151-200 as Q_3, 202-250 as Q_4, and 251-300 as Q_5.

When we evaluate a document or collection ranking function, we say a function is better than others if and only if it can produce higher precision at selected numbers of documents or at all standard levels of recall (see Section 2.1.3). In the case of replica selection, we need to add another criterion for the ranking function: directing as many queries as possible to the relevant replicas in order to improve system execution performance. We can tune the parameters of our functions to control the percentage of replicated queries to the replicas (as shown in Section 4.3.3). The range varies from 0% to 90%. None of the function we tested can direct 100% of replicated queries to the replicas. However when we direct more queries to the replicas, we have to tolerate

			Docu–		Mega–
Type	Top n	Replica	ments	Words	bytes
short queries (2 GB)	30	D_1	1,485	573,664	7
		D_2	2,937	1,130,997	14
		D_3	4,357	1,666,232	20
		D_4	5,736	2,208,214	26
		D_5	7,089	2,790,213	33
	100	D_1	4,876	1,923,076	23
		D_2	9,528	3,744,509	45
		D_3	13,968	5,498,714	66
		D_4	18,206	7,121,502	85
		D_5	22,261	8,858,102	106
	200	D_1	9,636	3,981,842	48
		D_2	18,552	7,618,586	91
		D_3	26,888	10,950,917	131
		D_4	34,649	14,050,086	168
		D_5	41,913	17,071,744	203
	500	D_1	23,301	9,932,345	118
		D_2	43,594	18,283,682	218
		D_3	61,710	25,673,460	306
		D_4	77,611	31,896,449	380
		D_5	92,063	37.595,894	447
long queries (2 GB)	100	D_1	4,832	2,185,651	26
		D_2	9,394	4,186,336	50
		D_3	13,619	5,886,587	70
		D_4	17,716	7,618,973	90
		D_5	21,568	9,365,920	111
	200	D_1	9,492	4,341,903	51
		D_2	18,167	8,246,402	97
		D_3	26,017	11,449,203	135
		D_4	33,473	14,624,078	173
		D_5	40,359	17,693,365	209
	500	D_1	22,826	9,993,915	118
		D_2	42,285	18,310,400	217
		D_3	59,149	24,894,518	295
		D_4	74,310	30,891,460	366
		D_5	87,931	36,408,245	431
short queries (20 GB)	100	D_1	4,945	1,878,596	19
		D_2	9,820	3,773,565	38
		D_3	14,594	5,545,532	56
		D_4	19,273	7,395,924	76
		D_5	23,862	9,273,290	94
	200	D_1	9,854	3,799,082	39
		D_2	19,507	7,621,580	77
		D_3	28,910	11,222,525	114
		D_4	38,040	14,883,811	151
		D_5	46,988	18,703,226	190
	500	D_1	24,372	9,558,280	96
		D_2	47,763	18,675,898	189
		D_3	70,683	27,381,547	277
		D_4	92,701	36,326,863	368
		D_5	113,664	44,966,965	455
TREC 2+3 (2 GB)			567,529	169,886,550	2,073
VLC (20 GB)			7,492,048	1,976,187,365	20,574

Table 4.4. Statistics of partial replicas

a larger precision loss. In our experiments, we compare the precision of each function when it directs more than 80% of replicated queries to the replicas.

For a replicated query, since we know which replica contains its top documents, we define its *expected replica* as the smallest replica that is built with the top documents for the query. For an unreplicated query, since replicas may contain some relevant documents, we expect our replica selector will direct some of these queries to a relevant replica. We define the *expected replica* for an unreplicated query as the smallest replica that causes a precision drop less than 5%. For both kinds of queries, especially unreplicated queries, we expect we will have to tolerate some loss in precision in order to avoid searching the entire collection. We choose a drop in precision between 0 and 10% for a query as our acceptable range, i.e., searching the selected replica retrieves at most one less relevant document for every 10 documents as compared with searching the entire original collection.

We define *collection precise queries* as those queries that can achieve the precision above 10% when searching the original collection for the top n documents, i.e., the query finds at least one relevant document for every ten documents. We exclude collection imprecise queries when we present the ability of a replica selector to pick the relevant replicas for unreplicated queries, because a replica with zero relevant documents is probably an acceptable choice for a query whose precision is below 10% in the original collection. We define *replica precise queries* as those for which searching the selected replica causes a precision loss less than 5% of the precision attained by searching the original collection.

4.3.3 Comparing Ranking Functions

In this section, we compare the effectiveness of the InQuery collection ranking function illustrated in Figure 4.4 and our replica selection function illustrated in Figure 4.5 by varying k and b for short queries in test trial 1 when we replicate the top 200 doc-

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

(a) Replicated queries (99 queries)

			Replica						Precision loss			% of replica
Ranking	Parameters	Func.							C +	5%−		precise
Function	k, b	code	D_1	D_2	D_3	D_4	D_5	C	< 5%	10%	> 10%	queries to C
Expected		E	1	6	3	6	2	19				
Random		Ran	7	8	5	7	4	6	35%	19%	46%	22%
InQuery	200, 0.25	I1	0	0	0	0	0	37	100%	0%	0%	100%
Collection	200, 0.75	I2	0	0	0	0	6	31	89%	3%	8%	89%
Ranking	200, 1	I3	6	5	11	7	2	6	40%	30%	30%	11%
Function	100, 1	I4	5	6	12	7	2	5	38%	30%	32%	11%
	400, 1	I5	6	6	11	11	6	2	40%	30%	30%	11%
	2, 0	R1	8	7	1	2	9	10	51%	19%	30%	6%
Replica	2, 0.2	R2	7	6	2	2	8	12	68%	16%	16%	11%
Selection	2, 0.8	R3	0	0	0	0	1	36	100%	0%	0%	94%
Function	1, 0.2	R4	4	2	1	1	13	16	73%	14%	14%	22%
	4, 0.2	R5	7	2	2	1	10	15	64%	14%	22%	22%

(b) Unreplicated queries (37 collection precise queries)

Table 4.5. Comparing ranking functions using short queries on the 2GB TREC 2+3 collection (replicas built with top 200 documents)

uments for each query. (We also performed experiments replicating the top 100 and 500 documents with similar results.) We will show that our replica selection function is comparable in the ability to pick the expected replica with some configurations of the collection ranking function for replicated queries, but that it significantly improves precision and finds the expected replica much more consistently as compared with the collection ranking function for unreplicated queries.

Table 4.5 lists the results of replica selection by counting the number of queries and to which replica or collection each function directs the queries, when the parameters, k and b, vary. Table 4.5(a) lists the results for 99 replicated queries for which we have relevance judgments. Table 4.5(b) lists the results for 37 unreplicated collection precise queries, 18 of which are replica precise queries. In both tables, columns 1 through 3 list the name of functions, the values of parameters k and b, and the function abbreviations. For replicated queries, columns 4 through 9 contain the number of queries that the replica selector sends to each of the replicas (D_i) as well as the original collection (C); columns 10 through 13 contain the percentages of queries that are directed to the expected replica (right), smaller replica, larger replica, and the original collection. The "expected" (E) row lists the number of judged queries that we would expect the replica selector to direct to each replica and to the original collection, if it were perfect with respect to the queries used to build the replicas. For unreplicated queries, columns 4 through 9 contain the number of collection precise queries the replica selector sends to each of the replicas as well as the original collection; column 10 contains the percentages of collection precise queries that are directed to the original collection and the replicas that cause a precision loss less than 5%; columns 11 through 12 contain the percentages of collection precise queries that are directed to replicas that cause a precision loss from 5% to 10%, and more than 10%. Column 13 contains the percentage of 18 replica precise queries that are directed to the original collection. The "expected" (E) row for the unreplicated queries contains the number of queries that we expect to go to each of replicas and the original collection because the result will cause a 5% or less drop in precision.

For the InQuery collection ranking function, varying k from 100 to 400 does not significantly change effectiveness (compare I3-I5). When we set k to 200 (the default of the InQuery collection ranking function) and increase the value of b, the replica selector directs more queries to the replicas. For the replicated queries, the default InQuery collection ranking function (k=200,b=0.75) only directs 30% of queries to the replicas, which is not our choice. When we tune the parameters as k=200 and b=1, the function directs 89% of queries to the replicas.

at m	Precision of Replicated Queries (%)						
docs	C	random	I3	I4	I5	R1	R2
10	48.7	40.4 (-17.2)	47.7 (-2.3)	48.2 (-1.2)	47.2 (-3.3)	48.4 (-0.8)	48.3 (-1.6)
20	44.8	36.1 (-19.6)	43.2 (-3.7)	43.3 (-3.4)	42.9 (-4.4)	44.4 (-1.0)	44.2 (-1.4)
30	40.7	32.4 (-20.4)	39.3 (-3.4)	39.4 (-3.0)	39.1 (-4.0)	40.2 (-1.1)	40.2 (-1.2)
100	31.1	23.9 (-23.1)	29.4 (-5.6)	29.5 (-5.5)	29.3 (-6.1)	30.5 (-2.2)	30.5 (-2.1)
200	25.1	18.7 (-25.3)	23.0 (-8.4)	23.0 (-8.4)	22.9 **(-8.7)**	24.8 (-2.8)	24.3 (-3.2)

(a) 99 Replicated queries

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

(b) 50 Unreplicated queries

Table 4.6. Effectiveness of different ranking functions using short queries on the 2 GB TREC2+3 collection (replicas built with top 200 documents)

For the replica selection function, $k = 2$ gets better results than $k = 1$ and $k = 4$ (compare the functions R3-R5). When we decrease the value of b, the replica selector directs more queries to the replicas. For k=2 and b=0.2, the function directs 88% of queries to the replicas.

Among the functions listed in Table 4.5, only six functions *random*, I3, I4, I5, R1, and R2 direct more than 80% of replicated queries to the replicas. We compare the precision of these six functions in Table 4.6. The first column lists the number of documents at which we present the precision. Column 2 lists the precision when all queries go to the original collection, i.e., what percent of the top m documents is relevant when searching the original collection. Columns 3 through 8 list the results using random selection and each ranking function. The numbers in parentheses show the precision percentage difference as compared with searching the original collection. Table 4.6(a) lists the results for replicated queries, and Table 4.6(b) lists the results for unreplicated queries. Replicated queries produce much better results than unreplicated queries, because their top documents are stored in at least one of the replicas.

It is not surprising that random selection performs poorly, because it has high probability of picking a replica with few relevant documents. For unreplicated queries, it causes a precision percentage loss ranging from 38% to 50% as compared with searching the original collection, C. For replicated queries, it causes a precision loss ranging from 17% to 25%.

For the other five functions in Table 4.6, when we examine the precision for replicated queries (Table 4.6(a)), all these functions are acceptable, since the precision drops less than 8.7%. However, when we examine the precision for unreplicated queries (Table 4.6(b)), the precision difference is significant. Using InQuery collection ranking function **I3** where we set $k = 200$ and $b = 1$, the precision losses of unreplicated queries range from 24.6% to 44.7%. We get our best result using our replica selection function **R2** with $k = 2$ and $b = 0.2$. The precision of the replicated queries drops less than 3.2% of the original collection, and is better when fewer documents are returned. The precision losses of the unreplicated queries range from 4.8% to 14.5%. For the top 30 documents, the precision losses of unreplicated queries range from 4.8% to 9.6%.

In the experiments presented in Section 4.3.4 and Section 4.3.5, the replica selector uses the replica ranking function with $k = 2$ and $b = 0.2$, because it sends appropriate queries to replicas with an acceptable precision loss of at most 9.6% for the top 30 documents in this test suit.

4.3.4 Effectiveness with Replicated Queries

This section evaluates our proposed replica selection function for replicated queries on a wider range of queries and collections. For replicated queries, we want to test whether the replica selector directs most of them to an expected replica. Note it is possible for a replica smaller than the expected one to contain all top documents for a given query, since the top documents of other queries could include the top documents

Size	Query Type	Top n	Average Num. of Queries to Replica						% to Replica			
			D_1	D_2	D_3	D_4	D_5	C	right	smaller	larger	C
	Expected		21.6	16.6	21.4	20.6	18.8	0				
2 GB	short	30	16.0	11.6	13.2	13.2	24.6	20.4	52.7%	4.2%	22.6%	20.6%
		100	17.2	13.0	15.8	15.2	24.0	13.8	58.4%	5.9%	21.8%	13.9%
		200	20.0	13.2	14.2	17.0	21.0	13.6	59.8%	8.1%	18.4%	13.7%
		500	25.0	14.4	15.8	18.6	15.6	9.6	65.1%	12.7%	12.5%	9.7%
		Ave.	19.5	13.0	14.8	16.0	21.3	14.3	59.0%	7.7%	18.9%	14.4%
2 GB	long	100	15.8	13.4	13.2	13.2	25.8	17.6	52.7%	5.8%	23.6%	17.8%
		200	18.0	17.4	12.6	14.4	28.2	8.4	57.0%	8.5%	26.0%	8.5%
		500	21.8	17.6	14.0	15.0	24.0	6.6	62.4%	10.7%	20.2%	6.7%
		Ave.	18.5	16.1	13.3	14.2	26.0	10.9	57.4%	8.3%	23.3%	11.0%
20 GB	short	100	15.6	14.2	12.8	15.8	20.2	20.4	54.3%	4.2%	21.0%	20.6%
		200	15.4	13.2	12.0	15.4	24.6	18.4	56.5%	4.2%	20.6%	18.6%
		500	18.2	14.4	12.2	16.0	25.8	12.4	64.2%	4.2%	19.0%	12.5%
		Ave.	16.4	13.9	12.3	15.7	23.5	17.1	58.3%	4.2%	20.2%	17.3%

Table 4.7. Replica selection for replicated queries

for this query. Although we use 250 queries to build replicas, we only present the results for 99 replicated queries which have relevance judgment files in this section.

Finding the Expected Relevant Replica

This section measures the ability of the replica selector to pick the expected replica by counting the number of queries that are directed to different replicas and the original collection, as shown in Table 4.7. In Table 4.7, columns 1 and 2 indicate the size of collection and the type of queries we use in our experiments. Column 3 indicates the number of top documents for each query. Columns 4 through 9 on the row of "Expected" list the number of judged queries that are used to build a replica, but not used in a smaller one. Columns 4 through 9 on other rows list the the number of queries the replica selector sends to each of the replicas as well as the original collection. Columns 10 through 12 contain the percentages of judged queries that the selector directs to the expected replicas (right), smaller replicas, and larger replicas. Column 13 contains the percentage of queries that the selector directs to the original collection (C).

For short queries on the 2 GB collection, on the average, our replica selector directs 85.6% (59.0%+7.7%+18.9%) of replicated queries to the replicas, and 66.7% of queries to the expected replica or a replica smaller than we expect. Increasing the number of replicated documents increases the accuracy of replica selection, because the replicas contain more relevant documents for replicated queries. For example, using the top 500 documents for each query to build replicas, the replica selector directs 90.3% of queries to the replicas on the average, while using top 100 documents directs 86.1% of queries to the replicas on the average.

For long queries on the 2 GB collection, on the average, our replica selector directs 89.0% (57.4%+8.3%+23.3%) of replicated queries to the replicas, and 65.7% of queries to the expected replica or a replica smaller than we expect. Increasing the number of replicated documents also increases the accuracy of replica selection, as for the short queries. For example, using the top 500 documents for each query to build replicas, the replica selector directs 93.3% of queries to the replicas on the average, while using the top 100 documents directs 82.2% of queries to the replicas on the average.

For short queries on the 20 GB collection, on the average, our replica selector directs 82.7% (58.3%+4.2%+20.2%) of replicated queries to the replicas, and 62.5% of queries to the expected replica or a replica smaller than we expect. Increasing the number of replicated documents increases the accuracy of replica selection, as against the 2 GB collection. For example, using the top 500 documents for each query to build replicas, the replica selector directs 87.5% of queries to the replicas on the average, while using top 100 documents directs 79.4% of queries to the replicas on the average.

Precision of Replica Selection versus the Original Collection

Since the replica selector directs a few queries to a replica that is smaller than expected, we compare the effectiveness of executing queries against replicas or the original collection selected by the replica selector with against the original collection. Table 4.8 compares the average precision of replica selection over 5 test trials with searching the original collection for short queries on the 2 GB collection, long queries

at	Precision				
m docs	orig.	Top 30	Top 100	Top 200	Top 500
10	47.3	46.9 (-0.8)	47.0 (-0.6)	47.4 (+0.3)	46.9 (-0.8)
20	43.5	43.0 (-1.2)	43.0 (-1.1)	43.3 (-0.4)	42.9 (-1.3)
30	39.6	39.0 (-1.5)	39.1 (-1.3)	39.4 (-0.7)	39.2 (-0.9)
100	30.8		29.9 (-2.7)	30.1 (-2.0)	30.1 (-2.1)
200	24.7			24.0 (-3.0)	24.0 (-3.0)
500	16.5				16.0 (-3.1)

(a) short queries on the 2 GB collection

at	Precision			
m docs	orig.	Top 100	Top 200	Top 500
10	56.3	56.1 (-0.4)	56.4 (+0.1)	56.2 (-0.2)
20	54.6	54.2 (-0.7)	54.3 (-0.6)	54.1 (-0.9)
30	51.7	51.3 (-0.8)	51.1 (-1.1)	51.2 (-1.0)
100	41.5	41.1 (-1.1)	41.0 (-1.2)	41.0 (-1.1)
200	34.1		33.4 (-2.1)	33.6 (-1.6)
500	22.9			22.4 (-2.4)

(b) long queries on the 2 GB collection

at	Precision			
m docs	orig.	Top 100	Top 200	Top 500
10	15.5	15.5 (-1.2)	15.4 (-1.2)	15.5 (-0.3)
20	15.0	15.0 (-0.3)	15.0 (-0.5)	15.0 (+0.0)
30	14.0	14.0 (+0.0)	14.0 (-0.1)	14.0 (+0.2)
100	11.5	11.4 (-0.9)	11.5 (-0.6)	11.5 (-0.3)
200	9.8		9.7 (-0.9)	9.7 (-0.1)
500	7.5			7.4 (-0.8)

(c) short queries on the 20 GB collection

Table 4.8. Effectiveness of replica selection for replicated queries (each trial has 99 judged queries)

on the 2 GB collection, and short queries on the 20 GB collection. In these tables, column 1 lists the number of documents at which we present the precision figures. Column 2 lists the precision figures when all queries go to the original collection. Columns 3 through 6 list the precision figures when building replicas using different numbers of top documents. The numbers in the parentheses show the precision percentage difference.

For short queries on the 2 GB collection, replica selection results in a precision percentage loss less than 3.1% of searching the original collection for the same number of responses or fewer.

For long queries on the 2 GB collection, replica selection results in a precision percentage loss less than 2.4%.

For short queries on the 20 GB collection, replica selection results in a precision percentage loss less than 1.2% as compared to searching the original collection, and sometimes the precision improves a little, because the replica does not contain some top-ranked unrelevant documents. In other words, selecting a smaller replica occasionally does no harm.

4.3.5 Effectiveness with Unreplicated Queries

This section evaluates our proposed replica selection function on a wider range of queries and collections for unreplicated queries. See Section 4.3.2 for detailed experimental setting.

Finding the Relevant Replica

Table 4.9 lists the average expected number of collection precise queries in each replica over five test trials and shows the results of replica selection by collecting the average number of collection precise queries that are directed to different replicas as well as the original collection. We list results for short queries on the 2 GB TREC 2+3

			Coll.	Precision loss less than 5%					
	Top	Query	Precise	Ave. Queries Expected					
Size	Type	n	queries	D_1	D_2	D_3	D_4	D_5	C
2 GB	short	30	38.2	2.6	1.0	2.2	1.2	1.4	29.8
		100	37.8	3.8	2.8	3.2	2.2	1.0	24.8
		200	37.8	4.6	3.8	2.6	3.2	1.2	22.4
		500	37.8	6.2	3.8	4.4	3.4	1.4	18.6
		Ave.	37.9	4.3	2.9	3.1	2.5	1.3	23.9
2 GB	long	100	42.4	3.6	3.4	3.4	2.2	1.2	28.6
		200	42.4	5.2	3.8	3.4	2.2	2.0	25.8
		500	42.4	8.2	5.4	4.0	2.8	2.0	20.0
		Ave	42.4	5.7	4.2	3.6	2.4	1.7	24.8
20 GB	short	100	18.4	0.8	0.6	0.8	1.0	0.0	15.2
		200	19.0	0.4	1.0	1.2	1.2	0.6	14.6
		500	19.2	2.2	2.0	2.0	1.0	1.0	11.0
		Ave.	18.9	1.1	1.2	1.3	1.1	0.5	13.7
20 GB	301-350 short	100	36	0	1	1	0	3	31
		200	35	0	0	1	0	4	30
		500	35	0	1	0	1	4	29
		Ave.	35.3	0	0.6	0.6	0.3	3.7	30.0

(a) Expected number of collection precise queries in each replica

	Query	Top	Ave. Queries to Replica						Precision Loss			% of Repl.Prec.
Size	Type	n	D_1	D_2	D_3	D_4	D_5	C	C+< 5%	5% – 10%	> 10%	queries to C
2 GB	short	30	2.6	2.0	2.0	2.8	5.6	23.2	72.4%	7.4%	16.2%	27.9%
		100	2.6	3.0	3.2	3.4	8.2	17.4	70.5%	12.2%	17.4%	13.8%
		200	4.0	3.4	2.4	4.8	6.8	16.4	71.5%	11.2%	17.3%	15.3%
		500	4.8	3.0	4.6	6.4	5.8	13.2	66.2%	22.7%	11.1%	15.4%
		Ave.	3.5	2.8	3.1	4.4	6.6	17.5	70.2%	13.4%	15.5%	18.1%
2 GB	long	100	3.6	2.4	1.8	2.8	6.6	25.2	77.5%	13.1%	9.4%	25.9%
		200	4.8	4.0	1.8	3.0	11.2	17.6	70.6%	17.6%	11.8%	13.1%
		500	5.4	4.8	3.2	5.0	10.2	13.8	78.5%	17.4%	4.2%	5.7%
		Ave.	4.6	3.7	2.3	3.6	9.3	18.9	75.5%	16.0%	8.5%	14.9%
20 GB	short	100	0.4	0.4	0.6	0.8	2.2	14.0	84.7%	4.5%	10.8%	37.0%
		200	0.6	0.6	1.2	1.4	1.8	13.4	84.2%	6.3%	9.5%	31.6%
		500	0.4	0.8	1.8	1.2	2.6	12.4	86.4%	6.2%	7.3%	29.0%
		Ave	0.5	0.6	1.2	1.1	2.2	13.3	85.1%	5.7%	9.2%	32.5%
20 GB	301-350 short	100	2	1	0	1	1	31	86.1%	0.0%	13.8%	50.0%
		200	2	1	0	1	1	30	85.7%	0.0%	14.3%	40.0%
		500	1	1	0	1	2	30	85.7%	5.8%	8.5%	40.0%
		Ave	1.7	1.0	0.0	1.0	1.3	30.3	85.8%	1.9%	12.2%	43.3%

(b) Results of replica selection for collection precise queries

Table 4.9. Replica selection for unreplicated queries

collection, long queries on the 2 GB TREC 2+3 collection, and short queries on the 20 GB TREC VLC collection.

In Table 4.9(a), columns 1 and 2 indicate the size of collection and the type of the query sets. Column 3 indicates the number of documents stored for each query. Column 4 lists the number of collection precise queries. Columns 5 through 10 list the average number of replica precise queries over the five trials in each of replicas (D_i) as well as the original collection (C), where the replica is the smallest replica which causes a precision loss less than 5%. In Table 4.9(b), columns 4 through 9 contain the average number of collection precise queries over the five trials the replica selector sends to each of the replicas as well as the original collection. Column 10 contains the average percentage of collection precise queries that the selector directs to the original collection and replicas that causes a precision loss less than 5%. Columns 11 through 12 contain the average percentages of collection precise queries that the selector directs to replicas that causes precision loss from 5% to 10%, and more than 10%. Column 13 contains the average percentage of replica precise queries over the five trials that the selector directs to the original collection. We expect the replica selector to direct the replica precise queries to a replica, because they have replicas that cause precision losses less than 5%.

For short queries on the 2 GB collection, on the average, our replica selector directs 83.6% (70.2%+13.4%) of collection precise queries to the replicas that cause a precision loss less than 10% (our acceptable level) as well as the original collection, and only directs 18.1% of queries which are replica precise to the original collection.

For long queries on the 2 GB collection, on the average, our replica selector directs 91.5% (75.5%+16.0%) of collection precise queries to the replicas that cause a precision loss less than 10% as well as the original collection, and only directs 14.9% of replica precise queries to the original collection.

For short queries on the 20 GB collection, when we experiment with the same setting as the 2 GB collection, on the average, our replica selector directs 90.8% (85.1%+5.7%) of collection precise queries to the replicas that cause a precision loss less than 10% as well as the original collection. When we experiment with queries 301-350 as our unreplicated queries, our replica selector directs 87.7% (85.8%+1.9%) of collection precise queries to the replicas that cause a precision loss less than 10% as well as the original collection.

Precision of Replica Selection versus the Original Collection

We compare the retrieval precision of executing unreplicated queries against replicas or the original collection selected by our replica selector with only searching the original collection.

Table 4.10 lists average precision over 5 test trials for short queries on the 2 GB TREC 2+3 collection, long queries on the 2 GB TREC 2+3 collection, and short queries on the 20 GB TREC VLC collection. In these tables, column 1 lists the number of documents at which we present the precision figures. Column 2 lists the precision figures when all queries go to the original collection. Columns 3 through 6 list the precision figures when building replicas using different numbers of top documents. The numbers in the parentheses show the precision percentage difference.

For the 2 GB collection using short queries, the precision losses range from 6.8% to 17.1%. Increasing the number of replicated documents for each query improves the precision, because the replicas contain more relevent documents for each replicated query, which helps determining the similarity between unreplicated and replicated queries. When the number of top retrieved documents is less than 30 documents, which are the retrieval levels that concern online users most, our replica selector causes an average precision percentage loss within 14.6% and 10% of searching the

at m	Precision				
docs	orig.	Top 30	Top 100	Top 200	Top 500
10	42.8	37.8 (-11.9)	38.9 (-9.2)	39.4 (-8.0)	39.9 (-6.8)
20	39.4	34.0 (-13.8)	35.8 (-9.1)	36.1 (-8.4)	35.6 (-9.7)
30	35.5	30.3 (-14.6)	32.6 (-8.2)	32.7 (-7.8)	32.7 (-7.9)
100	27.2		23.3 (-14.0)	24.0 (-11.6)	24.2 (-10.8)
200	21.8			18.3 (-16.5)	18.8 (-13.9)
500	14.3				11.8 (-17.1)

(a) short queries on the 2 GB collection

at m	Precision			
docs	orig.	Top 100	Top 200	Top 500
10	55.3	52.9 (-4.3)	52.0 (-6.0)	54.8 (-0.9)
20	52.7	49.4 (-6.2)	48.6 (-7.8)	50.9 (-3.4)
30	50.0	46.2 (-7.6)	45.7 (-8.7)	47.9 (-4.3)
100	40.4	35.3 (-12.6)	35.1 (-13.2)	36.8 (-8.8)
200	33.1		27.3 (-17.4)	29.1 (-12.0)
500	21.8			18.2 (-16.6)

(b) long queries on the 2 GB collection

at m	Precision			
docs	orig.	Top 100	Top 200	Top 500
10	12.8	12.4 (-3.1)	12.6 (-1.6)	12.4 (-3.4)
20	12.3	11.6 (-5.5)	11.9 (-3.1)	11.8 (-3.4)
30	11.8	11.4 (-3.1)	11.9 (+0.8)	11.8 (+0.3)
100	10.0	9.1 (-9.6)	9.6 (-4.2)	10.1 (+0.2)
200	8.4		7.7 (-7.9)	8.3 (-1.0)
500	6.4			5.9 (-7.7)

(c) short queries on the 20 GB collection

at m	Precision			
docs	orig.	Top 100	Top 200	Top 500
10	40.4	36.2 (-10.4)	36.2 (-10.4)	37.8 (-6.4)
20	35.4	30.7 (-13.3)	30.7 (-13.3)	31.3 (-11.6)
30	31.3	26.9 (-14.2)	26.9 (-14.2)	27.0 (-14.0)
100	20.2	17.8 (-11.9)	17.8 (-11.9)	17.2 (-15.0)
200	14.4		12.8 (-10.9)	12.2 (-14.0)
500	7.8			6.7 (-14.0)

(d) short queries (topics 301-350) on the 20 GB collection

Table 4.10. Effectiveness of unreplicated queries (each trial has 50 queries)

original collection, when we only replicate the top 30 documents and the top 100 documents for each replicated query, respectively.

For the 2 GB collection using long queries, the precision losses range from 0.9% to 17.4%. For the top 30 retrieved documents, on the average, the precision drops less than 8.7% when we replicate more than 100 documents for each replicated queries, which is slightly better than short queries.

For the 20 GB collection using short queries, when we experiment with the same setting as the 2 GB collection and use the relevance files for the 2 GB collection, the precision ranges from losing 9.6% to improving 0.8%. For the top 30 retrieved documents, the precision loss is less than 5.5%. When we use short queries 301-350 as our unreplicated queries, the precision loss for the top 30 documents is less than 14.2%. Since topics 301-350 were much more thoroughly judged than topics {51-150, 202-250} for the 20 GB VLC collection, although still only the top 30 documents of each query were judged, we think the results using topics 301-350 is more accurate, which means our replica selection performs slightly worse on the 20 GB collection than on the 2 GB collection. However, the precision percentage loss of 14.2% in our context only means we retrieve one less relevant document for the top 30 documents.

4.3.6 Summary

Sofar in this chapter, we investigated how to select a relevant partial replica using the inference network model. Our approach enables a system to efficiently rank partial replicas and select a replica based on relevance for a given query. We developed a replica selection function, and demonstrated its effectiveness and superiority over a collection ranking function using the InQuery retrieval system and TREC collections. Our results show that the inference network model is a very promising tool for selecting a relevant replica. By using our proposed replica selection function, the replica selector can direct at least 82% of replicated queries to a relevant partial replica rather than the original collection, and it achieves a precision percentage loss less than 10% for the 2 GB collection and 14.2% for the 20 GB collection for the top 30 retrieved documents of each query, when we build replicas using more than 100 top documents for each replicated query.

In the rest of chapter, we will investigate load-balancing, space and time overheads for the replica selection database, as well as updating costs and strategies.

4.4 Load Balancing

In the previous section, we have discussed how to select a relevant replica using inference networks. However if we use relevance as the only criterion, the system could overwhelm one or several replicas by sending them too many IR commands when replicas are relevant to significant numbers of requests. In this section, we discuss how to balance the loads among relevant replicas and the original collection in order to produce a response as quickly as possible.

Since we want to select a replica with the quickest response time, we may choose the *response time* as our metric. However simply using the response time will not produce the best result for a new command when several commands have been in the queue of a replica or collection. We may also use the *outstanding number of commands* of each replica/collection as our metric, and send the new command to the replica/collection with the shortest queue. However we need to be aware that the least loaded replica or collection is not the best choice in our situation, because searching the original collection could take longer time than searching a small replica with higher loads.

In our system, we combine the above two metrics and balance loads as follows: We predict the response time of each replica (R_i) and the original collection (C) using the average response time and the number of outstanding commands. When the replica selector chooses a replica based on relevance, we calculate the predicted response time p_resp_j of the replica, the replicas larger than this, and the original collection using

	Original			Largest Replica		Replica Selection Database		
Collection	Size (MB)	Unique Terms	Top n	Size (MB)	Unique Terms	All Terms (MB)	Terms in Replicas (MB)	Percent. of Largest Replica
TREC 2+3	2 GB	838,948	100	104	128,435	44	8	7.7%
			200	199	200,723	45	12	6.0%
			500	476	325,851	47	20	4.2%
TREC VLC	20 GB	13,088,064	100	95	162,279	741	10	10.5%
			200	191	258,107	742	15	7.8%
			500	459	460,044	746	28	6.1%

Table 4.11. Space overhead for the replica selection database.

$ave\_resp_j \cdot (1+num\_wait\_mes_j)$, where ave_resp_j is the average response time for last 200 responses for either the replica or the original collection, and $num\_wait\_mes_j$ is the number of the outstanding commands to which either the collection or the replica has not yet responded. We send the command to the one with the least p_resp_j. Since we have a connection broker to keep track of all servers, we can easily obtain the information on the response time and the number of outstanding commands. We demonstrate the performance of our system with load balancing in Chapter 6.

4.5 Space and Time Overheads for the Replica Selection Database

This section investigates how much space is needed to store a replica selection database, and how much extra time is used in searching the replica selection database.

4.5.1 Space Overhead

Table 4.11 lists the space overhead for the replica selection database for a 6-layer replication hierarchy in test trial 1 that we described in Section 4.3.2. We list the size and the number of unique terms of the original collection, the size and the number of unique terms of the largest replica, and the size of the replica selection database when we replicate the top 100, 200, and 500 documents, respectively.

For the 2 GB TREC 2+3 collection, the size of the replica selection database is around 45 MB, which is approximately 2.2% of the size of the original collection, when we build the index without deleting any terms except stopwords. If we delete the terms that do not occur in any of replicas, the size of the replica selection database is directly proportional to the number of unique terms in the largest replica, ranging from 8, 12 to 20 MB, approximately 6 MB for every 100,000 unique terms, when we replicate the top 100, 200, and 500 documents for each query.

For the 20 GB VLC collection, the size of the replica selection database is around 742 MB, which is approximately 3.6% of the size of the original collection, when we build the index without deleting any terms except stopwords. If we delete the terms that do not occur in any of replicas, the sizes of the replica selection database range from 10, 15 to 28 MB, also approximately 6 MB for every 100,000 unique terms. The size of replicas are larger than those for the 2 GB collection, because the TREC VLC collection consists of data from more diverse sources than the TREC 2+3, and thus contains more unique terms.

Deleting terms that do not occur in any of replicas significantly reduces the size of the replica selection database, and it has no significant impact on retrieval effectiveness. We may modify our ranking algorithm as follows: When a query contains a term that does not occur in any of replicas, the system assigns the smallest tf (term frequency) score for the replicas, and assigns the largest tf score for the original collection.

4.5.2 Time Overhead

Unlike the document database we discuss in Chapter 6.1 where the query evaluation time is strongly related to both the number of terms in the query and the term frequencies of each term, the time overhead for searching the replica selection database is only related to the number of terms in the query. Since we typically replicate a collection several times, the size of an inverted list in the replication selection database

Terms	Cold Start	Warm Start
2	0.095	0.005
4	0.194	0.008
8	0.391	0.014
16	0.736	0.026
32	1.178	0.049
64	2.357	0.098
128	4.492	0.197

Table 4.12. Time overhead for searching the replica selection database for the 20 GB collection(seconds).

is very small. There is no significant time difference between processing an inverted list with one entry and processing one with ten entries.

Since the sizes of the replica selection databases for the 2 GB TREC 2+3 collection and the 20 GB TREC VLC collection are about the same, we only list the time overhead for searching the replica selection database for the 20 GB VLC when we vary the number of terms per query in Table 4.12. We ran our experiments using a single CPU on a lightly-loaded 3-CPU DEC Alpha server 2100 5/250 (clocked at 250 MHZ) with 1024 MB main memory and using InQuery version 3.1. We obtained this data in the following ways: for each data point for the number of terms per query, we randomly chose terms from the set of terms that occur in the 350 TREC short queries, and constructed ten queries with the same number of terms; we ran these ten queries on the replication selection databases in test trial 1 described in Section 4.3.2, and listed their average evaluation time in the Table 4.12. We ran our experiments under two start situations: *cold start* where we chilled the system by reading a large file before the system processes each query such that each term was read from the disk, and *warm start* where each term in the queries has been cached in the memory. For the 20 GB collection, the time overheads for searching a replica selection database range from 0.095 to 4.492 seconds for the cold start, and from 0.005 to 0.197 seconds for the warm start when the number of terms per query varies from 2 terms to 128 terms.

4.6 Updating

This section investigates space and time costs for updating databases related to replica selection, which include costs from updating replicas, and updating the replica selection database. We then suggest updating strategies based on these costs.

4.6.1 Costs for Updating Replicas

We compare two approaches to updating the replicas:

- *delete-and-add* – delete the old documents and their inverted lists from the data and index files of a replica, and add new ones,
- *rebuild* – build the replica from the scratch.

For obtaining the updating measurements, we directly used or modified the programs provided by InQuery version 3.1. For *rebuild*, we used the InQuery indexing program *inbuild* to build the replica from the scratch. For *delete-and-add*, we modified the InQuery program *purgedb-key* to delete documents and modify the corresponding inverted lists, and used *inparse* to parse new documents, and *merge_btl* to merge the inverted lists of new documents into the database. Table 4.13 lists the updating time for replicas with different sizes when we use a single CPU on a 3-CPU DEC Alpha server (clocked at 250 MHZ). We constructed test replicas by picking documents from subcollections of the 2 GB TREC 2+3 collection in a round-robin fashion. The numbers in the table are the average times over three runs. For each data point on the replica size, we collected the times when we update 20%, 40%, 60%, and 80% of a replica. Since *rebuild* builds the replica from the scratch, the time for updating a replica is only related to the size of replica.

The results show that when we update 20% of documents in a replica, *delete-and-add* performs significantly better than *rebuild*. The smaller the replica is, the better *delete-and-add* performs. When we update more than 40% of documents in a replica, *rebuild* is our choice, especially for large replicas. The larger the replica is, the better

Replica Size	Num. of Doc.	Delete-and-Add				Rebuild
		Updating Percentage				
		20%	40%	60%	80%	
100 MB	47,294	0.12	0.17	0.21	0.26	0.18
200 MB	94,588	0.22	0.31	0.43	0.57	0.36
500 MB	141,882	0.50	0.86	1.35	1.97	0.92
1 GB	283,764	1.23	2.35	4.05	6.25	1.89
2 GB	567,529	2.96	6.93	15.46	25.59	3.73

Table 4.13. Updating time for replicas with different sizes (hours).

rebuild performs. For example, for updating 60% of a 1 GB replica, *rebuild* is 2 times faster than *delete-and-add*; for updating 60% of a 2 GB replica, *rebuild* is 4 times faster than *delete-and-add*.

Another advantage of using *rebuild* is the *rebuild* procedure is much more easily parallelized than *delete-and-add*. The *rebuild* procedure consists of two parts: parsing documents into intermediate files, and merging the intermediate files into indices, where the parsing part accounts for around 94% of total building time. We can easily partition a replica into several small collections, and parse each of them in parallel, and then merge the intermediate files. If our machine is n-CPU multiprocessor, we can finish rebuilding in nearly $\frac{1}{n} \cdot T$, where T is the building time using one CPU, as listed in Table 4.13. If we rebuild a 20 GB replica using 4 CPUs, we may finish it in less than 10 hours.

Partitioning an index is not trivial work like partitioning documents. It takes 5-6 minutes to segment the index of a 2 GB collection, and an hour to segment the index of a 20 GB collection. For *delete-and-add*, if we have a n-CPU multiprocessor, we can finish updating in nearly $\frac{1}{n} \cdot T + s$, where T is the updating time using one CPU and s is the segmentation time.

For *rebuild*, the space required to update a replica is around 2 times the size of the replica. For *delete-and-add*, the space required is $(2 \cdot x\% + 1)$ times the size of the

	Replica Size					
	100 MB	200 MB	500 MB	1 GB	2 GB	20 GB
time (min.)	0.5	0.8	1.4	2.9	4.3	66.7

(a) time for generating intermediate file (minutes)

	Collection Size					
	2 GB TREC 2+3			20 GB VLC		
	1 rep.	3 rep.	5 rep.	1 rep.	3 rep.	5 rep.
time (min.)	1.8	1.8	1.9	30.8	31.0	31.3

(b) time for merging intermediate files (minutes)

Table 4.14. Updating time for replica selection database (minutes).

replica, where x% is the updating percentage. For updating percentages less than 50%, *delete-and-add* uses less space.

4.6.2 Costs for Updating the Replica Selection Database

After we update replicas, we must then update the replica selection database. We first generate the intermediate files from the indexes of the original collection and replicas, and then merge the intermediate files to the replica selection database. We wrote a program to generate an intermediate file that only keeps the document frequency and collection term frequency of each term from each collection/replica index, and used the InQuery version 3.1 program *merge_btl* to merge these intermediate files. We obtained our measurements using a CPU on a 3-CPU DEC Alpha server clocked at 250 MHZ.

Table 4.14 lists the updating time for the replica selection databases with different number of replicas and different sizes of the original collection. Table 4.14(a) lists the time for generating the intermediate file from the index of a replica when we vary the size of the replica. Table 4.14(b) lists the time for merging the intermediate files. For both 2 GB and 20 GB collection, we list the merging time for 1 replica with the size of 500 MB, 3 replicas with sizes of 300 MB, 400 MB, and 500 MB, and 5 replicas with sizes of 100 MB, 200 MB, 300 MB, 400 MB, and 500 MB.

When the size of the original collection is 2 GB, it takes around 10 minutes to update a replica selection database with 5 replicas. When the size of the original collection is 20 GB, it takes around one and half hour for updating a replica selection database with 5 replicas.

The space required to update the replica selection database is around 1.5 times the size of the replica selection database. For the 2 GB TREC 2+3 collection, the size of the replica selection database is around 46 MB, and thus updating needs around 70 MB. For the 20 GB VLC collection, the size of the replica selection database is around 750 MB, and thus updating needs 1.1 GB.

4.6.3 Updating Strategies

Although updating a large replica is time-consuming, fortunately, the statistics in Section 4.1.3 show that a replica built using topics that occurred a day ago, even a month ago may satisfy a significant number of queries. We do not need to update replicas hourly, or even daily, which makes the costs for updating replicas affordable. How often to update replicas is domain-dependent. Some domains such as news and finance may need updates more often than others. Sometimes a system needs to be updated immediately due to some bursty events.

One strategy for updating replicas is to update replicas at regular intervals, such as every week. The disadvantages of this strategy are that it could react too slowly to a bursty event such as the Starr report, and it could waste time when the topics that users are interested in change very slowly. In order to overcome these problems, we propose other two more selective strategies:

- Event triggered updating: watch for bursty events, and trigger the updating procedure when some special events happen.

- Performance triggered updating: watch the percentage of workloads that the replica selector sends to the replicas, and trigger the updating procedure when the percentage falls below some threshold.

For the event triggered updating strategy, the easiest way to use it is through human intervention. When the system manager watches a special event, and increasingly many users issue queries to search it, the manager could start the updating procedure before the scheduled time. Automatic event detection is an on-going research topic. When the technology can be very efficiently implemented, we can use it to automatically detect the events and trigger the updating procedure.

The performance triggered updating strategy is very easy to implement in the current system. We can let the replica selector record the percentage of queries that it sends to each replica, when the percentage falls below a threshold, the system informs the system manager. The performance triggered updating strategy also applys to bursty events, if a lot of users search for an event that does not exist in the replicas.

For bursty events, we could just add documents into replicas without deleting others for quick updating, which also means we need to save some extra space for bursty events.

4.7 Summary

In this chapter, we have discussed the issues related to partial collection replication. Based on our analyses of actual system logs, an IR system can take advantage of query locality and replicate a small percentage of collections to improve performance. We presented a method for constructing a hierarchy of partial replicas from a collection where each replica is a subset of all larger replicas. We then extended the inference network model to rank and select partial replicas. We compared our new replica selection function to previous work on collection selection over a range of tuning parameters. For a given query, our replica selection algorithm correctly determines the most relevant of the replicas or original collection, and thus maintains the highest retrieval effectiveness while searching the least data as compared with the other ranking functions. The space overhead of the replica selection database is directly proportional to the number of unique terms in the largest replica. Using a parallel technique to build collections, we can update replicas and replica selection databases in reasonable time. For example, we can update a 20 GB replica within 10 hours when we use four 250 MHZ CPUs. Of course we can do updating faster with a machine with more and/or faster CPUs. Although updating large replicas is very costly, fortunately, the statistics from the real system logs show that updating hourly or daily is unnecessary. We also proposed two simple strategies for updating replicas. In Section 6.2, we will demonstrate the performance gain of partial collection replication under different system configurations and user patterns.

CHAPTER 5
THE SIMULATION MODEL

In order to expedite our investigation of possible system configurations, characteristics of IR collections, and the system performance for distributed information retrieval architectures illustrated in Chapter 3, we implement a simulator with numerous system parameters, and validate it against the prototype implementation. In this chapter, we present the simulation model.

In our simulator, we model collections and queries by obtaining statistics from test collections and real query sets. We model query processing and document retrieval by measuring resource usages for each operation in our multithreaded prototype. The hardware we model include CPUs, disks, I/O bus, memory, and the network.

Our simulator has two types of parameters: system configuration parameters and system measurement parameters. The system configuration parameters include the number of threads, number of disks, number of CPUs, collection size, term frequency distribution, query length, command arrival rate, command mixture ratio, replication percentage, distracting percentage, and selection percentage. The system measurement parameters include query evaluation time, document retrieval time, network time, connection broker time, and time to merge results. The system configuration values deal with the configuration of the system and data layout on the disks, and change with each simulation scenario or group of scenarios. The system measurements deal with a change of the IR system and the underlying hardware platform. The outputs of the simulator include response times for each IR command, utilizations of each hardware resource, such as the CPU utilization and the disk utilization, and utilizations of each IR software system component, such as the InQuery server utilization and the connection broker utilization.

We use YACSIM, a process oriented discrete event simulation language [48] to implement the simulator. YACSIM contains a set of data structures and library routines that manage user created processes and resources. We model a single server (InQuery server, connection broker, collection selector, or replica selector) as a set of processes. Each process simulates the activity of a thread in the real system by requesting services from resources. Our simulation model is simple, yet contains sufficient details to accurately represent the important features of the system, as we show through performance validation.

The remainder of the chapter is organized as follows: we present the system measurements and their validation in Section 5.1, and the configuration parameters in Section 5.2.

5.1 System Measurements and Validation

For the experiments in this dissertation, we model the IR system by analyzing a prototype of a distributed multithreaded information retrieval system based on InQuery and by measuring the resource usage for each operation. We focus on CPU, disk, I/O bus, and the network resources. For memory and cache effects, we simply assume the system is either cold started where no cache effects are considered, or warm started where memory is big enough to hold an index and/or an entire collection. Unless we explicitly state otherwise, the experiments we present are cold started, i.e., we do not include cache benefits.

We model the collection of CPUs as a multi-server processor-sharing infinite length queue. We model a disk, the I/O bus, and the network as a single FCFS infinite length queue. The processing times (service times) at a hardware resource are modeled as a deterministic function of overhead plus a linear function of the data size (i.e.

$\alpha_0 + data\_size * \alpha_1$), where $data\_size$ is replaced with a more meaningful concept in a different operation, such as the term frequency in a query command, the number of documents in a summary command, and the message size in a network transmission. We determine the actual values of α_0 and α_1 of each hardware resource for each operation by obtaining measurements from our multithreaded prototype.

We measure resource usages for query evaluation, document/summary retrieval, result merging, connection brokering, and network transmission. We examine TREC collections (up to 20 GB) to obtain system measurements (see Appendix A). We obtained the measurements using a client-server IR system based on InQuery version 3.1 running on DEC Alpha Server 2100 5/250 with 3 CPUs (clocked at 250 MHz), 1024 MB main memory and 2007 MB of swap space, running Digital UNIX V3.2D-1 (Rev 41).

5.1.1 Query Evaluation Time

In an earlier study, Cahoon and McKinley demonstrated that the query evaluation time is very strongly related to the number of terms per query and the collection frequency of each of the query terms [12, 13]. They model the query evaluation time as a function of the number of terms per query and the frequency of the individual query terms plus additional overheads for the CPU and disk. The overheads account for the time InQuery spends on combining the results of each of the terms. Without these additional overheads, the evaluation times of long queries are underestimated. In this work, we adopt their model, but we recollect the measurements, since we use a faster machine. For each of 1 GB TREC 1, 2 GB TREC 2-3, and 3 GB TREC 1-3 collections, we collected their Pearson's correlation coefficients between term evaluation times and term frequencies. The resulting values were 0.94, 0.95, and

$$
\begin{aligned}
\mathit{query\_eval\_time} &= \sum_{i=1}^{n} \mathit{eval\_term\_time}(\mathit{term}_i) + \mathit{combining\_overhead} \\
\mathit{eval\_term\_time}(\mathit{term}) &= \mathit{cpu\_time}(\mathit{term}) + \mathit{disk\_time}(\mathit{term}) + \mathit{I/O\_bus\_time}(\mathit{term}) \\
\mathit{cpu\_time}(\mathit{term}) &= \begin{cases} 4.643e-03 + (9.334e-06 \times \mathit{term}) & : \ \mathit{term} \leq 1000 \\ 1.462e-02 + (2.209e-06 \times \mathit{term}) & : \ 1000 \leq \mathit{term} \leq 10000 \\ 2.531e-02 + (1.089e-06 \times \mathit{term}) & : \ 10000 \leq \mathit{term} \leq 100000 \\ 1.593e-01 + (1.671e-07 \times \mathit{term}) & : \ \mathit{term} \geq 100000 \end{cases} \\
\mathit{disk\_time}(\mathit{term}) &= \begin{cases} 5.773e-02 + (2.048e-05 \times \mathit{term}) & : \ \mathit{term} \leq 1000 \\ 7.659e-02 + (2.566e-06 \times \mathit{term}) & : \ 1000 \leq \mathit{term} \leq 10000 \\ 9.607e-02 + (5.187e-07 \times \mathit{term}) & : \ 10000 \leq \mathit{term} \leq 100000 \\ 1.463e-02 + (8.208e-07 \times \mathit{term}) & : \ \mathit{term} \geq 100000 \end{cases} \\
\mathit{I/O\_bus\_time}(\mathit{term}) &= \begin{cases} 5.075e-06 + (4.044e-08 \times \mathit{term}) & : \ \mathit{term} \leq 1000 \\ 2.318e-05 + (3.936e-08 \times \mathit{term}) & : \ 1000 \leq \mathit{term} \leq 10000 \\ 1.288e-04 + (3.329e-08 \times \mathit{term}) & : \ 10000 \leq \mathit{term} \leq 100000 \\ 2.869e-03 + (9.514e-09 \times \mathit{term}) & : \ \mathit{term} \geq 100000 \end{cases} \\
\mathit{combining\_overhead} &= 0.075 * \sqrt{(n-1)} * \sum_{i=1}^{n} \mathit{eval\_term\_time}(\mathit{term}_i)
\end{aligned}
$$

Figure 5.1. Query evaluation timing values (seconds).

0.94, respectively, which indicates a very positive relationship[1]. Our results confirm Cahoon and McKinley's findings.

Figure 5.1 lists the formulas for calculating query evaluation time values in our simulation. We created the formulas for *eval_term_time*() by fitting the term evaluation time measurements with four straight lines using a least square method, each of which covers term frequencies below 1,000, 1,000-10,000, 10,000-100,000, and above 100,000, respectively. The model is more accurate using four lines rather than a single straight line for all the data. We divide *eval_term_time()* into CPU, disk, and I/O bus time. Although we created *eval_term_time*() by using measurements obtained from searching the TREC 1 collection, the formulas also match well measurements obtained from TREC 2-3 and TREC 1-3 (see Section 5.1.5).

[1] 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.

Using our measurements, the time to evaluate a single term in the TREC 1 collection ranges from 0.07 seconds for a term appearing once to 1.2 seconds for a term appearing 995,008 times (the maximum term frequency in TREC 1 after removing stopwords). The CPU and disk times account for more than 99% of the total evaluation time. The I/O bus time accounts for 0.007% to 1.0% with an average of 0.16%. The disk access time is typically larger than CPU processing 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 or faster disks, the CPU processing may be larger than disk time.

5.1.2 Document Retrieval Time

We measure InQuery to determine the amount of time it takes to retrieve a document. Since the size of a document in our test collections is not very large, where the average size of a document is 2.3 KB in the 1 GB TREC 1 collection, 2.9 KB in the 3 GB TREC 1-3 collection, and 2.8 KB in the 20 GB VLC collection, the document retrieval time is very short and does not show a strong relationship with the document size. 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 TREC 1 collection. The average size of a document is 2.3 KB. The document retrieval time is also divided into three parts: CPU time, disk time, and I/O bus time. Our measurements show that disk access time accounts for 87% of the total retrieval time on average, while I/O bus time accounts for 0.06%.

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 TREC 1 collection, *i.e.*, 2.3 KB. The simulator also uses the document retrieval time to compute the summary information retrieval time. We implement the summary information

$$
\begin{aligned}
sender\_time(message\_size) &= 2.085e-05 + (3.383e-08 \times message\_size) \\
receiver\_time(message\_size) &= 2.428e-05 + (4.848e-08 \times message\_size) \\
network\_time(message\_size) &= message\_size \times 8/network\_speed
\end{aligned}
$$

Figure 5.2. Network time values (seconds).

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.

5.1.3 Network Time

We model the network time as the sender overhead, the receiver overhead, and the network latency. The sender overhead is the CPU processing time for adding a message to the network. The receiver overhead is the CPU time for removing a message from the network. The network latency is the amount of time that the message spends in transit. These times are a linear function of the message size, when the bandwidth of the network is fixed. We obtained these measurements using TTCP by measuring the time to send messages from 32 bytes to 64 KB between two DEC Alpha Servers connected by a lightly loaded 10 Mbps Ethernet. Figure 5.2 lists the network time equations in our simulator. Using these equations, the sender overhead for sending a 2 KB message is 0.09 milliseconds, the receiver overhead is 0.12 milliseconds, and the network latency is 1.64 milliseconds.

5.1.4 Connection Broker Time

The connection broker time is the CPU time that a connection broker spends to distribute an IR command to the corresponding InQuery servers or send back a result to a client. We divide the time in the connection broker into three categories: the time

Message Encoding and Decoding Time

$$\begin{aligned}
\mathit{query}() &= 7.500e - 05\\
\mathit{summary}(\mathit{nsums}) &= 1.669e - 05 + (2.489e - 05 \times \mathit{nsums})\\
\mathit{document\_list}(\mathit{ndocs}) &= 8.523e - 04 + (4.852e - 06 \times \mathit{ndocs})
\end{aligned}$$

Processing Time

$$\begin{aligned}
\mathit{command}(\mathit{nservers}, \mathit{cmd}) &= \begin{cases} 6.170e - 04 & : \mathit{cmd} = \mathit{document}\\ 1.329e - 03 + (1.694e - 04 \times \mathit{nservers}) & : \mathit{otherwise} \end{cases}\\
\mathit{server\_result}(\mathit{cmd}) &= \begin{cases} 7.111e - 05 & : \mathit{cmd} = \mathit{document}\\ 1.291e - 03 & : \mathit{otherwise} \end{cases}
\end{aligned}$$

Merging Time

$$\begin{aligned}
\mathit{query}(\mathit{nanswers}) &= 2.566e - 05 + (1.814e - 06 \times \mathit{nanswers})\\
\mathit{summary}(\mathit{nsums}) &= -0.751e - 05 + (2.414e - 05 \times \mathit{nsums})
\end{aligned}$$

Figure 5.3. Connection broker time values (seconds).

to encode and decode messages, the processing time for handling messages, and the time to merge results. Figure 5.3 lists the formulas we use to compute the different values.

Each time a message arrives, the connection broker must decode the message to determine an appropriate action. The connection broker also creates new messages and must encode the data into the message format. The time to decode and encode a query command, $\mathit{query}()$, is a constant value. The time to decode and encode a summary information, $\mathit{summary}(\mathit{nsums})$, is a linear function of the number of summaries. The $\mathit{document\_list}(\mathit{ndocs})$ function is the time to encode and decode query results; the value depends upon the number of documents returned from the InQuery server.

The processing time for handling messages depends upon who sends the message and the message type. For commands from clients, the connection broker places them in the queues for the appropriate destination InQuery servers. 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 times using constant values.

The connection broker also spends time merging query and summary results. The time to merge query and summary results depends upon the number of answers or summaries an InQuery server returns. We model the merge times as linear functions of the number of results.

5.1.5 Validation of the Query Evaluation Model

This section presents the validation of the simulator's query evaluation times against the actual implementation. We divide the validation into three steps: (1) we validate the term evaluation time that is the base of the query evaluation model; (2) we validate the query evaluation model; (3) we validate the query evaluation operation in a multithreaded IR system. The actual system was running InQuery 3.1 on a 3-CPU Alpha Server 2100 5/250 running Digital UNIX V3.2D-1 (Rev 41).

Validation of the term evaluation time

We validated the accuracy of *eval_term_time*() in Figure 5.1 by comparing the values calculated using the formulas with measurements we obtained from searching TREC 1, TREC 2-3, and TREC 1-3 collections. For each collection, we randomly chose 500 terms and calculated the percentage difference of the times for each of the terms. The term frequencies for TREC 1, TREC 2-3, and TREC 1-3 we chose range from 1 to 556,548, from 1 to 1,940,988, and from 1 to 2,935,997, respectively. Before processing each term, we chilled the system by reading a large file that fills the memory such that every terms was read from the disk. Table 5.1 shows the validation results. Column 1 lists the names of collections. Column 2 and 3 show the average percentage difference of evaluation time between the simulator and the actual system, and its standard deviation. A positive value means that the simulator overestimates

Query Type	Difference Ave.	 Std.	 ±5%	 ±10%	 ±15%
TREC 1	0.1%	0.9%	100%	100%	100%
TREC 2-3	1.7%	3.0%	88%	97%	100%
TREC 1-3	1.4%	7.8%	85%	96%	100%

Table 5.1. Term evaluation time validation.

the actual system. Columns 4 through 6 show the percentage of terms running on the simulator that fall within ±5%, ±10%, and ±15% of the actual system.

Since the formulas were obtaining by fitting the measurements from the TREC 1 collection, the model matches the measurements from TREC 1 best, the average percentage difference is 0.1% with a standard deviation of 0.9%, and all terms fall within 5% difference. For TREC 2-3, the average percentage difference is 1.7% with a standard deviation of 3.0%. 97% of terms fall within 10% and all terms falls within 15%. For TREC 1-3, the average percentage difference is 1.4% with a standard deviation of 7.8%. 96% of terms fall within 10% and all terms falls within 15%. These numbers indicate the model matches TREC 2-3, and TREC 1-3 very well.

Validation of the query evaluation model

We validated the accuracy of the query evaluation model by creating artificial queries and comparing the performance of query function for searching a 1 GB collection in the actual implementation and the simulator. We randomly generated three sets of queries: 50 short queries with an average of 2 terms per query, 50 medium queries with an average of 12 terms per query, and 50 long queries with an average of 27 terms per query. When obtaining measurements, we chilled the system before the system processes each query.

Table 5.2 shows the validation results. Column 2 and 3 show the average percentage difference of evaluation time between the simulator and the actual system, and its standard deviation. A positive value means that the simulator overestimates the

Query	Difference						Ave. Eval.
Type	Ave.	Std.	±10%	±20%	±30%	±40%	Time (sec)
Short	2.0%	16.2%	42%	72%	96%	100%	0.7
Medium	4.2%	10.1%	68%	90%	100%	100%	4.6
Long	2.3%	8.5%	78%	96%	100%	100%	10.1
Ave.	2.8%	11.6%	63%	86%	99%	100%	5.1

Table 5.2. Query model validation.

	Total		Distribution		
Query			0% to	−10% to	−20% to
Type	Number	Percentage	−10%	−20%	−30%
Short	22	44%	11	7	4
Medium	17	34%	14	3	0
Long	19	38%	18	1	0
Ave.	58	39%	43	11	4

Table 5.3. Distribution of underestimated queries.

actual system. Columns 4 through 7 show the percentage of queries running on the simulator that fall within ±10%, ±20%, ±30% and ±40% of the actual system. The last column lists the average evaluation query time for each set of queries in the actual system. On average the simulator is 2.8% slower than the actual system with the standard deviation of 11.6%. The variation of short queries is twice that for long queries. All queries fall within 40% of the actual system.

Table 5.3 details the underestimated queries. Columns 2 and 3 show the total number of underestimated queries and the corresponding percentage in all query sets. Columns 4 through 6 show the number of the underestimated queries that fall within −10%, −10% to −20% and −20% to −30% of the actual system. Although we underestimate 58 queries out of 150 queries, 43 queries fall within 10% and only four short queries fall outside of 20%. Thus, we usually overestimate queries. Our validation results show that our query evaluation model matches the actual system very closely, although we do not accurately model every query.

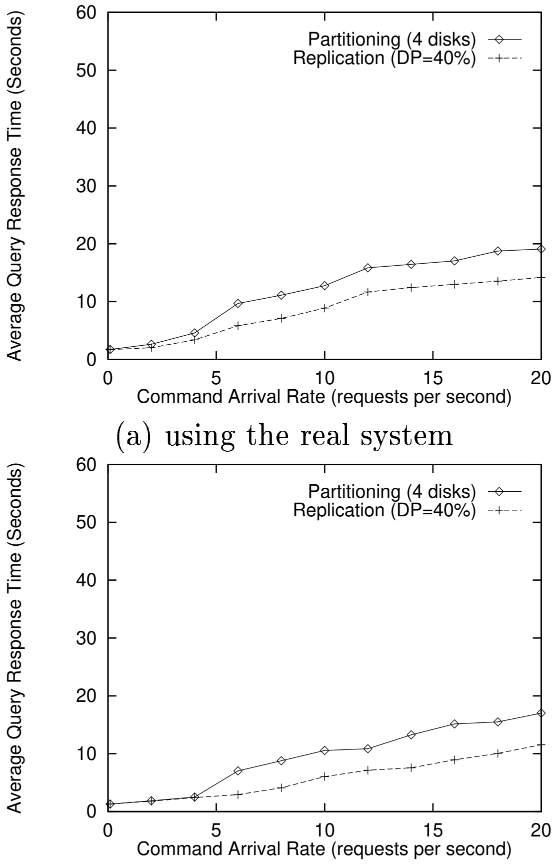
Query Type	Arrival Rate λ per sec.	Number of Threads					
		1	2	3	4	8	16
Short	.5	1.5%	0.5%	-2.5%	2.5%	-0.8%	1.5%
	1	13.7%	3.0%	1.9%	-1.5%	1.0%	-0.3%
	5	**27.2%**	17.6%	3.8%	-0.4%	2.1%	1.0%
Medium	.5	19.8%	9.0%	4.4%	1.9%	1.2%	0.5%
	1	26.0%	12.0%	2.5%	**-3.9%**	1.9%	-0.4%
	5	21.5%	15.7%	7.2%	8.8%	5.0%	4.1%
Long	.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%

Table 5.4. Percentage difference of average response times between the implementation and simulator.

Validation of query operation

This section validates query operations for searching a 1 GB collection on a multi-threaded server using InQuery 3.1 as the query arrival rate and the number of threads increase on a 3-CPU Alpha Server 2100 5/250 running Digital UNIX V3.2D-1 (Rev 41). We assumed queries arrives as a Poisson process. Each thread executes one query. The collection was put on one disk. We used the same query sets as used in validating the query evaluation model.

Table 5.4 lists the percentage difference of the average response time between the actual system and the simulator for each query set as the arrival rate and the number of threads increase. Positive numbers indicate the simulator overestimates the actual system. The simulator reports response times that are 4.5% slower than the actual system on the average, and range from 3.9% faster to 27.2% slower than the actual system. The difference between the actual system and the simulator tends to decrease as the number of threads increases and the query arrival rate decreases. For example, for a system with 16 threads, the simulator is from 0.3% faster to 4.1% slower than the actual system. The difference between the simulator and the actual system increases



(a) using the real system

(b) using the simulator

Figure 5.4. Validation of the performance using partial replication.

as a function of the number of queries waiting in the queue, because our query model usually overestimates the query evaluation time. Overall, the simulator matches the actual system closely.

Validation of partial replication performance

This section compares the simulator and the implementation on partial replication for searching a 16 GB collection on a multi-tasking server using InQuery 3.1 as the query arrival rate increases on a 3-CPU Alpha Server 2100 5/250 running Digital UNIX V3.2D-1 (Rev 41). We used a multi-tasking server instead of a multithreaded server just to save us time from implementing replica selection in a legacy system like InQuery which uses too many global variables. Our earlier work showed that the multitasking server performs similar to the multithreaded server, although the multithreaded server is always slightly faster (90% of measured response times fall within 10% of each other) [55].

In this experiments, we distributed the 16 GB collection over 4 disks and used an extra disk to store a 4 GB replica. We assumed queries arrives as a Poisson process, and used 50 short queries with average 2 terms per query. Figure 5.4 compares the performance of using the real system and the simulator when the replica distracts 40% of commands, and shows that two systems presented the same trend and expected improvements from partial replication. Section 6.1.3 and Section 6.2 further explore the performance of partial replication.

5.2 Configuration Parameters

This section describes the parameters we vary in our experiments. Our configuration parameters include parameters that define command characteristics such as *command arrival rate*, and *command mixture ratio*, those that define query and collection characteristics such as *collection size*, *terms per query*, and *query term frequency distribution*, those that describe data placement features such as *collection size*, *number of collections*, *number of replicas* and *replica size*, and those that describe hardware and software features such as *number of threads*, *number of CPUs*, and *number of disks.* Table 5.5 presents the parameters, their abbreviations, and values we use throughout this dissertation.

Command arrival rate (λ)

In the simulator, we model light or heavy workloads by slowing down or speeding up the command arrival rate. We model the command arrival pattern using a Poisson

Parameters	Abbreviation	Values
Command Arrival Rate Poisson dist. (avg. commands/sec)	λ	.1 2 4 6 8 10 12 14 16 18 20
Command Mixture Ratio query:summary:document	R_{cm}	1:1.5:2
Terms per Query (average) shifted neg. binomial dist.	N_{tpq}	2 8 12 27
Query Term Frequency Distribution	D_{qtf}	Obs. Dist.
Number of Collections	N_{col}	1 8 32
Collection Size (GB)	C_{size}	1 2 4 8 16 32 64 128 256 1024
Server speedup	S_p	4
Number of CPUs	N_{cpu}	1 2 4
Number of Disks	N_{disk}	1 2 4 8
Number of Threads	N_{th}	1 2 4 8 16 32
Number of Replicas	N_{repl}	1 2 4
Replication Percentage	P_{repl}	3% 6.25% 12.5% 25%
Distracting Percentage	P_{dist}	10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
Selection Percentage	P_{sel}	12.5% 25% 50%
Collection Access Skew Zipf-like function	θ	0 (pure Zipf) 0.3 1 (uniform)

Table 5.5. Configuration parameters.

process with rate λ. We vary λ from 0.1 commands per second to 20 commands per second.

Command mixture ratio (R_{cm})

We simulate user activity by issuing query, summary, and document retrieval commands. We vary the mixture of the commands to model different user patterns. For example, 1:2:2 means that for each query command, clients issue 2 summary and 2 document commands. In the THOMAS log that we analyzed in Appendix B, if we assume the response for a summary command contains the summary information for 10 documents, the ratio of query:summary:document is 1:1.5:2.

Number of collections (N_{col}), collection size (C_{size})

These parameters give the number of collections and the total size of all collections in gigabytes in the IR system. We vary these two parameters to examine the scalability of an IR system when it processes an increasing number of collections and an increasing amount of data.

Terms per query (N_{tpq})

Cahoon and McKinley demonstrated that shifted negative binomial distributions closely match the distributions of query length in query sets they investigated [12, 13]. In this dissertation, we use their results, as shown Table 5.6. 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. In Table 5.6, there are three sets of (n, p, s), which produce different lengths of queries. The first row describes short queries with an average of 2 terms per query that mimic those found in the query sets from the 103rd Congressional Record. The third row describes long queries with an average of 27 terms per query that mimic those found in a TREC 1 query set. The second rows describes medium queries with 12 terms per query that is an artificial query set that shows characteristics from short and long queries.

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

Table 5.6. The values used in terms per query.

Query term frequency distribution (D_{qtf})

Since there is no agreement about a commonly accepted distribution for term frequencies in queries [75], we use an observed query term frequency distribution obtained from the TREC 1 query set as described in Cahoon and McKinley's paper [12]. We reproduce the figure that shows the term frequency distributions in the TREC 1 collection and the query set, as shown in Figure 5.5. The figure shows that terms used in

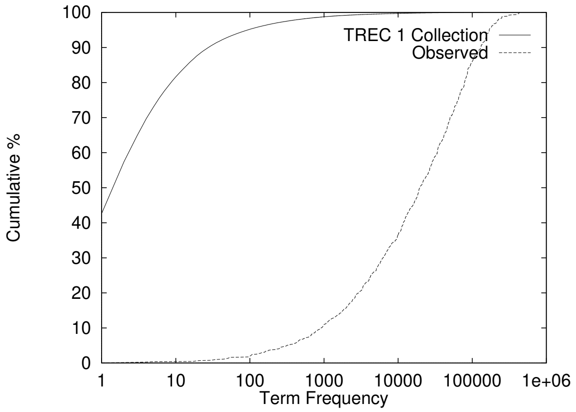
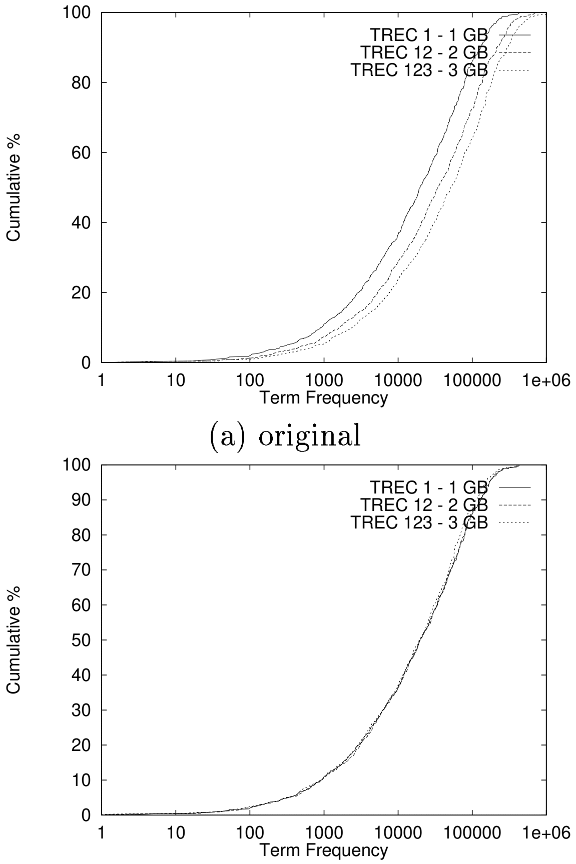


Figure 5.5. Query term frequency distributions.

the TREC query set tend to use frequently used terms in the collection. Frequencies of about 90% of query terms are larger than 1000.

We observe how this distribution changes as the collection size increases, shown in Figure 5.6. We draw Figure 5.6(a) using raw data. We draw Figure 5.6(b) by normalizing the frequencies by a factor of $bsize/size$, where $bsize$ is the size of the base collection (1 GB in this case), and $size$ is the size of the currently observed collection. Figure 5.6(b) show that the three lines for TREC 1, TREC 12, and TREC 123 overlap after normalizing the term frequency. Therefore, in our simulator, we generate a query term with term frequency tf on a particular collection by using the base distribution obtained from the base collection (TREC 1 collection in our case) to generate a base frequency b_tf, and then multiplying b_tf by a factor of $size/bsize$, i.e, $tf = b\_tf * size/bsize$. But we need to note, this relation holds when the collection size increases, due to data coming from the same sources. If the new data comes from different sources, the tf could be less than $b\_tf * size/bsize$, because many new terms could be added.

For replicas, we use $tf = repl\_f * b\_tf * size/bsize$, where $repl\_f \geq 1$, since replicas will contain documents for most frequently used queries, and thus query term frequencies should be larger than the average. By examining query term frequencies of replicated queries in the replicas we built for the experiments in Section 4.3.2, we find that



(a) original

(b) normalized

Figure 5.6. Query term frequency distributions with increasing collection size.

generally $repl\_f$ decreases as b_tf increases. We obtained the following formulas by dividing b_tf into several intervals and averaging the values within each interval:

$$f = \begin{cases} 5.43 & : \quad b\_tf < 1000 \\ 2.92 & : \quad 1000 \leq b\_tf < 5000 \\ 2.60 & : \quad 5000 \leq b\_tf < 10000 \\ 1.50 & : \quad 10000 \leq b\_tf < 50000 \\ 1.31 & : \quad 50000 \leq b\_tf < 100000 \\ 1.21 & : \quad otherwise \end{cases}$$

$$repl\_f = \begin{cases} f & : \quad f * size/ori\_size \leq 1, \text{where } ori\_size \text{ is the original collection size} \\ \frac{ori\_size}{size} & : \quad otherwise \end{cases}$$

Server speedup (S_p)

We adjust this parameter to simulate a server S_p times as fast as our base system.

Number of CPUs/disks/threads per machine ($N_{cpu}/N_{disk}/N_{th}$)

The number of CPUs per machine and the number of disks per machine affect the machine processing capability and data layout of a IR system. The number of threads affects the utilization of hardware components. We vary these numbers to examine how to effectively exploit hardware resources to build a high performance IR server.

Number of replicas (N_{repl}), replication percentage (P_{repl}), distracting percentage (P_{dist})

These three parameters together describe the features of partial replication: how many replicas in the system (N_{repl}), what percentage of data is replicated (P_{repl}), and what percentage of queries is directed to the replicas (P_{dist}). When we say a partial replica satisfies P_{dist} of queries, it also indicates that the replica will satisfy the same percentage of summary and document commands, because we use the same distribution to distribute summary and document commands. Therefore, "a replica distracts P_{dist} of queries" has the same meaning as "a replica distracts P_{dist} of commands" in our context. "A replica distracts or satisfies a query" means that the replica selector sends the query to that replica.

Selection percentage (P_{sel}), collection access skew (θ)

These two parameters describe the features of collection selection: what percentage of collections the system searches at a time (P_{sel}), and the frequency of each collection being accessed after collection selection (θ).

We model the collection access skew using a Zipf-like function as follows: assume that we want to select W collections to search at a time, we use W trials. In each trial, we choose a collection from the collections C that have not been chosen according to the distribution function $Z(i)$:,

$$Z(i) = c/i^{1-\theta}, \text{ where } c = 1/\Sigma_{j=1}^{C}(1/j^{1-\theta}), 1 \leq i \leq C.$$

As θ varies from 1 to 0, the probabilities vary from an uniform distribution to the pure Zipf distribution. We do not have logs for collection access patterns in a real multiple collection searching environment, but since queries have locality, the actual distribution probably closes to a pure Zipf than an uniform function.

CHAPTER 6

PERFORMANCE EVALUATION OF OUR DISTRIBUTED INFORMATION RETRIEVAL SYSTEM

This chapter investigates the performance and scalability of our distributed information retrieval system using the simulator presented in Chapter 5. We evaluate the performance of parallel IR servers using symmetric multiprocessors. We examine how to balance hardware and software resources with respect to numbers of threads, CPUs, disks, and the collection size. We compare the data placement strategies when we have additional disks. We evaluate the performance of partial collection replication and collection selection in a distributed IR system. We compare the performance of partial replication and collection partitioning, and the performance of collection selection and partial collection replication.

Our experiments demonstrate that adding hardware resources improves system performance only when hardware resources are carefully balanced, otherwise it can degrade the performance. Using extra disks or servers to build partial replicas of a collection performs significantly better than using them to partition the collection when the replica(s) distract a certain amount of IR commands. The improvements occur when the replica distracts less than 10% of commands in many cases. Restricting the search to a subset of the collections improves system performance when either the collection access after collection selection is fairly uniform or only a small percentage of collections are searched.

In the experiments in this chapter, we model the command arrival as a Poisson process. The users use short queries with an average of 2 terms per query, and issue query, summary, and document commands, with a ratio of 1:1.5:2, as we found in the Thomas logs (see [22] and Section B). We measure response time, CPU and disk utilization, and determine the largest command arrival rate at which the system supports an average query response time under 10 seconds. We chose 10 seconds arbitrarily as our cutoff point for a reasonable response time.

The remainder of the chapter is organized as follows: we evaluate the performance of parallel servers in Section 6.1, the performance of partial collection replication in Section 6.2, the performance of collection selection in Section 6.3, and summarize the results of this chapter in Section 6.4.

6.1 Parallel Information Retrieval Using Symmetric Multiprocessors

This section explores how to achieve high performance for information retrieval using symmetric multiprocessors. We investigate how to balance software and hardware with respect to multiple threads, CPUs, and disks as the collection size increases. We compare the performance of replication and partitioning over additional disks.

We start with a base system that consists of one thread, CPU, and disk. Our base system is disk bound where the disk is a bottleneck. We improve the performance of our IR server through better software (multithreading), and with additional hardware (CPUs and disks). Our parallel information retrieval server uses InQuery as the retrieval engine [46]. In our parallel IR server, as shown in Figure 6.1, when the server receives an IR command, it assigns the command to one or multiple threads according to the command type and collection partitioning. For example, we distribute a collection over 4 disks; when the server receives a query command, it assigns the query command to four threads, each of which evaluates the command against the data on one disk, and then the server merges the results from the 4 threads. For a summary command, the server assigns it to n threads, where n is the number of disks

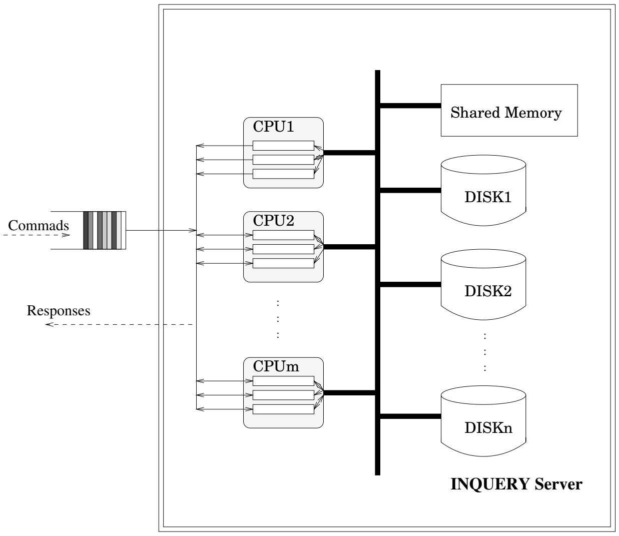


Figure 6.1. The parallel InQuery server

Parameters	Abbreviation	Values
Command Arrival Rate Poisson dist. (avg. commands/sec)	λ	.1 2 4 6 8 10 12 14 16 18 20
Command Mixture Ratio query:summary:document	R_{cm}	1:1.5:2
Terms per Query (average) shifted neg. binomial dist.	N_{tpq}	2
Query Term Frequency dist. from queries	D_{qtf}	Obs. Dist.
Number of CPUs	N_{cpu}	1 2 4
Number of Disks	N_{disk}	1 2 4 8 16
Number of Threads	N_{tn}	1 2 4 8 16 32

Table 6.1. Configuration parameters for parallel experiments.

that contain the documents whose identifiers are described in the command, and then the server merges the summaries. For a document command, the server assigns it to one thread, which retrieves the full text of that document.

Table 6.1 presents parameters, their abbreviations, and values we use in the experiments of this section. The rest of this section is organized as follows: Section 6.1.1 investigates the effects of threading and factors that affect the necessary number of threads. Section 6.1.2 investigates balancing hardware when adding CPUs and disks.

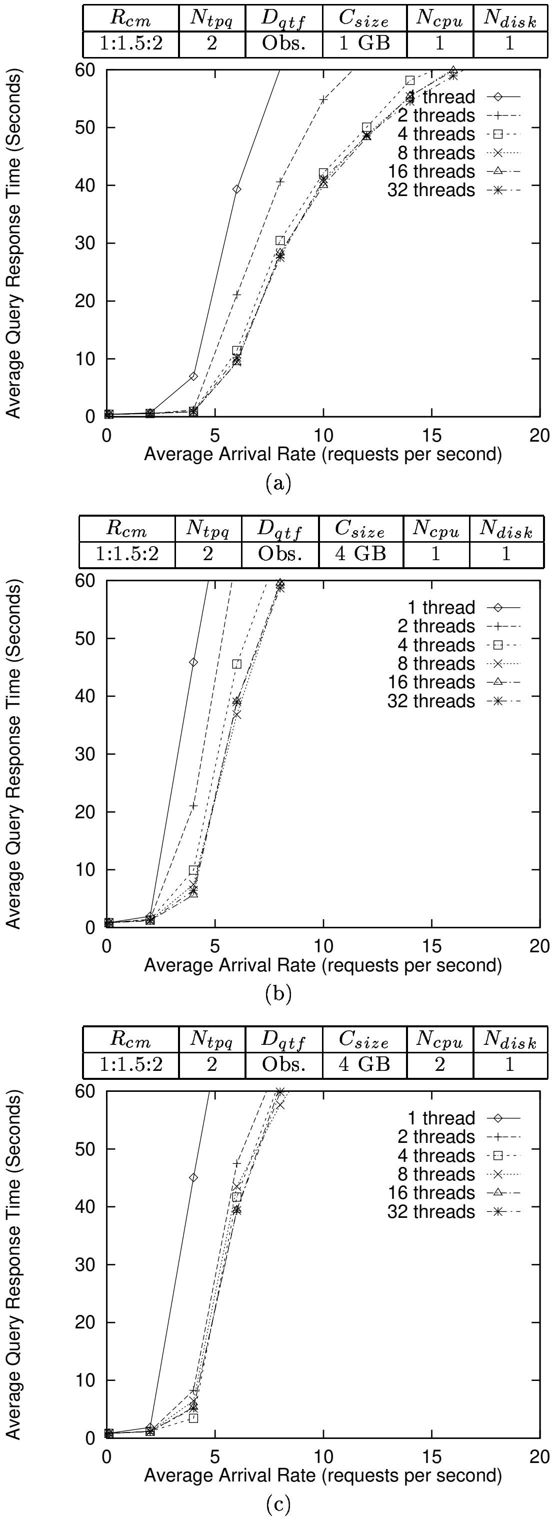
Section 6.1.3 compares the performance of replication with partitioning when we have additional disks. Section 6.1.4 summarizes the results of this section.

6.1.1 Threading

This section examines how the number of threads affects system scalability, and what factors affect the necessary number of threads that leads the system to its peak performance.

Figure 6.2 to Figure 6.4 illustrate how the average query response time and resource utilization change as the number of threads increases with varying numbers of CPUs and disks. When multiple disks exist in the system, we distribute the collection over disks in a round-robin fashion.

Figure 6.2 illustrates configurations when the disk is a bottleneck. We show the relationship between performance and the number of threads when we put more data on the disk, and add CPUs. We start with one disk and one CPU for a 1 GB collection (Figure 6.2(a)), and then increase the collection size from 1 GB to 4 GB (Figure 6.2(b)), and add one CPU (Figure 6.2(c)). Figure 6.2(d) shows the CPU and disk utilizations at some interesting data points in configurations (a) to (c). The box on the top of each figure lists the system parameters for the experiment. In all these configurations, threading significantly improves the query response time as the number of threads increases. However, after the number of threads increases to 4, more threads improve the performance very little. By examining the utilization shown in Figure 6.2(d), 4 threads result in disk utilization of 97.4% for $\lambda = 8$. There is little room for further increasing the disk utilization. When we increase the collection size (see Figure 6.2(b)), the system shows the same trend as configuration (a), the system performance significantly improves until the the number of threads reaches 4, although searching 4 GB takes longer time than searching 1 GB. When we add an additional CPU into the configuration (b) (see Figure 6.2(c)), the system perfor-



R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}
1:1.5:2	2	Obs.	1 GB	1	1

(a)

R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}
1:1.5:2	2	Obs.	4 GB	1	1

(b)

R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}
1:1.5:2	2	Obs.	4 GB	2	1

(c)

	Config. (a)			Config. (b)		Config. (c)	
	$\lambda = 8$			$\lambda = 6$		$\lambda = 6$	
Res.	$N_{th} = 1$	$N_{th} = 2$	$N_{th} = 4$	$N_{th} = 2$	$N_{th} = 4$	$N_{th} = 2$	$N_{th} = 4$
CPU	23.0%	27.5%	30.3%	31.1%	34.7%	17.7%	18.1%
DISK	73.8%	87.9%	97.4%	83.4%	94.3%	93.5%	98.7%

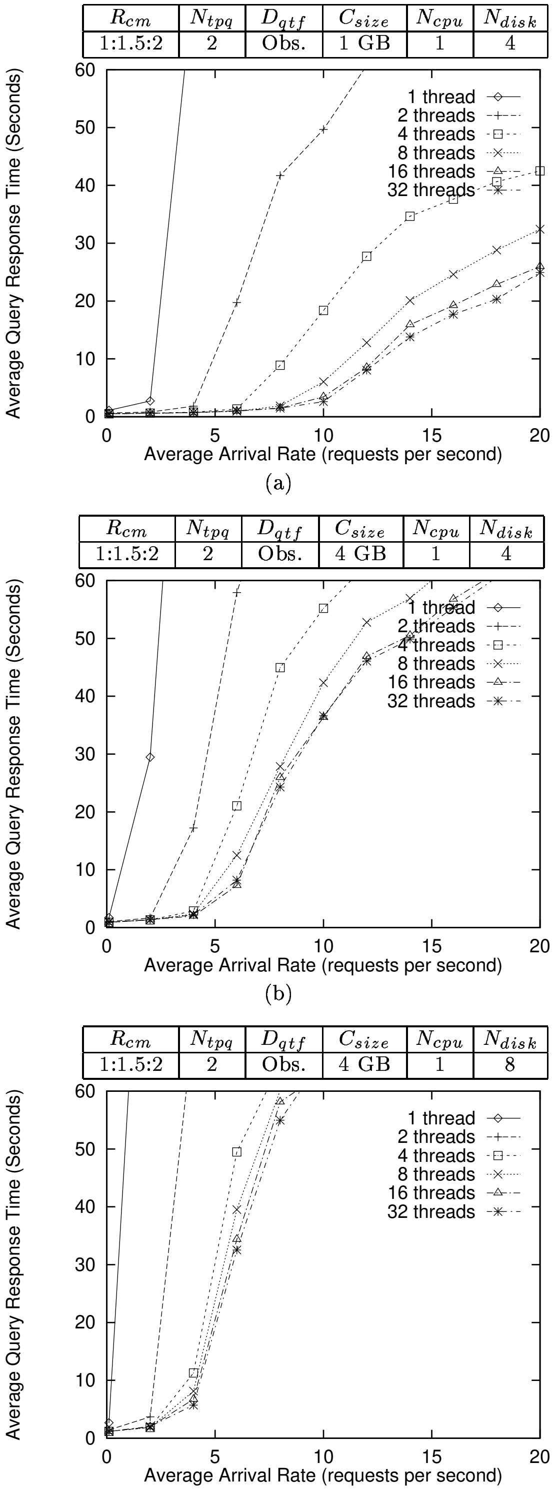
(d) hardware utilization at some interesting data points

Figure 6.2. Performance as the number of threads increases (disk bottleneck).

mance still significantly improves until the number of threads is 4. But 2 threads in configuration (c) is very close to 4 threads, since an additional CPU means more computation power that shortens the waiting time for CPU. Although fewer threads in this configuration could bring more improvement than the other two configurations, the maximum improvement of the system is still limited by the disk utilization.

Figure 6.3 illustrates configurations when the CPU is a bottleneck. We start with a system using 1 CPU for a 1 GB collection distributed over 4 disks (Figure 6.3(a)), and then increase the collection size to 4 GB (Figure 6.3(b)), and add 4 disks which distributes the 4 GB collection over 8 disks (Figure 6.4(c)). Figure 6.3(d) shows the CPU and disk utilization at some interesting data points in configurations (a) to (c). In these configurations, the average query response time significantly improves until the number of threads reaches 8, when the CPU becomes overutilized (see Figure 6.3(d)). As in the case of the disk bottleneck (see Figure 6.2), adding data causes longer searching time, but does not significantly affect the number of threads that leads the system to peak performance. Adding additional 4 disks makes the performance worse than fewer disks due to two reasons. First, although with 8 disks each disk holds half as much data as 4 disks, searching it takes more than half time of searching all the data (it takes around 3/5 of the time according to our measurements). Second, searching more disks requires more coordination time. Therefore the total amount of the CPU time for searching 8 disks is more than searching 4 disks, which overwhelms the CPU sooner.

Figure 6.4 illustrates configurations when CPUs and disks are well balanced as the collection size increases. Figure 6.4(a)-(c) show the configurations using 2 CPUs and 4 disks for 2 GB, 4 GB, and 8 GB of data, respectively. Figure 6.4(d) shows the CPU and disk utilization at some interesting data points in configurations (a) to (c). In these three configurations, the average query response time significantly improves



R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}
1:1.5:2	2	Obs.	1 GB	1	4

(a)

R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}
1:1.5:2	2	Obs.	4 GB	1	4

(b)

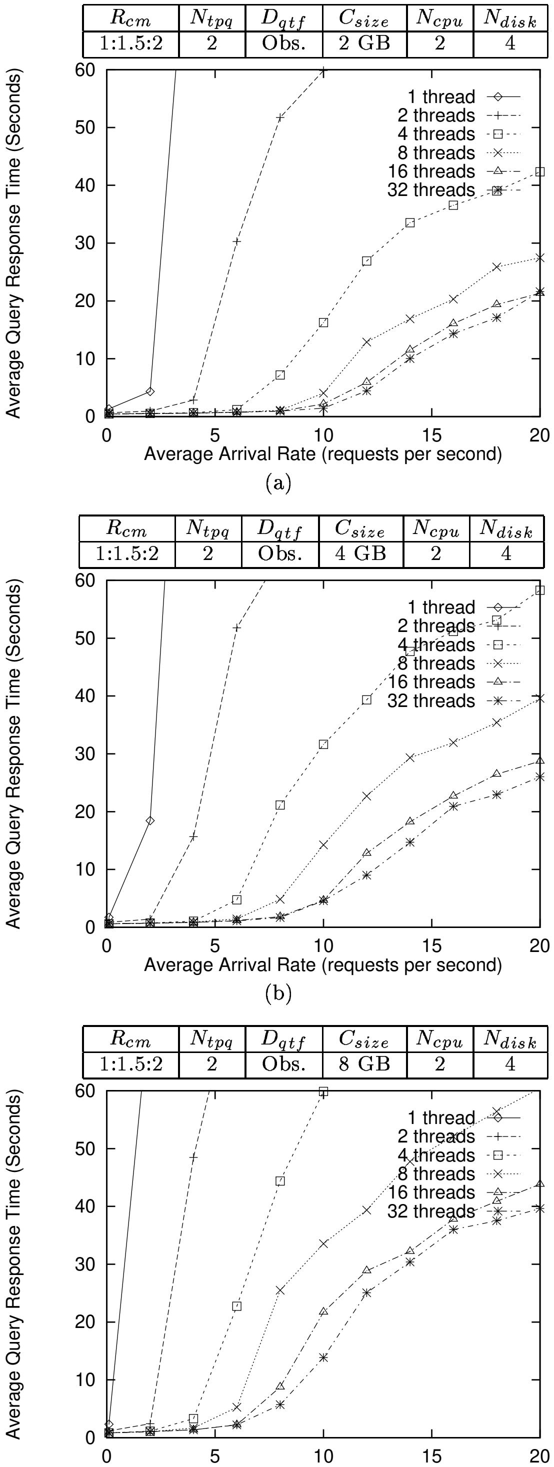
R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}
1:1.5:2	2	Obs.	4 GB	1	8

(c)

	Config. (a)		Config. (b)		Config. (c)	
	$\lambda = 10$		$\lambda = 8$		$\lambda = 8$	
Res.	$N_{th} = 4$	$N_{th} = 8$	$N_{th} = 4$	$N_{th} = 8$	$N_{th} = 4$	$N_{th} = 8$
CPU	68.5%	83.7%	80.4%	92.7%	86.8%	98.2%
DISK	54.1%	65.7%	40.4%	48.1%	23.7%	26.8%

(d) hardware utilization at some interesting data points

Figure 6.3. Performance as the number of threads increases (CPU bottleneck).



R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}
1:1.5:2	2	Obs.	2 GB	2	4

(a)

R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}
1:1.5:2	2	Obs.	4 GB	2	4

(b)

R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}
1:1.5:2	2	Obs.	8 GB	2	4

(c)

	Config. (a)		Config. (b)		Config. (c)	
	$\lambda = 10$		$\lambda = 10$		$\lambda = 6$	
Res.	$N_{th} = 8$	$N_{th} = 16$	$N_{th} = 8$	$N_{th} = 16$	$N_{th} = 8$	$N_{th} = 16$
CPU	56.9%	59.7%	65.1%	76.6%	63.9%	64.1%
DISK	64.7%	65.7%	67.5%	78.7%	58.6%	59.3%

(d) hardware utilization at some interesting data points

Figure 6.4. Performance as the number of threads increases (CPUs and disks are well balanced).

until the number of threads reaches 16. As the collection size increases from 2 GB to 8 GB, more threads bring very small further improvement for a larger collection.

In all the configurations shown in Figure 6.2-6.4, the average query response time improves significantly as the number of threads increases until either the disk or the CPU is overutilized. Too few threads limit the system's ability to achieve its peak performance. For example in Figure 6.4(b), using 4 threads only supports 7 requests per second for an average query response time under 10 seconds, while using 16 threads supports 11 requests per second under the same hardware configuration. Both CPUs and disks affect the necessary number of threads.

In our configurations, when the CPU is not a bottleneck, a disk needs 4 threads to fully utilize its capacity. When the CPU is a bottleneck, the system needs fewer threads. Increasing the collection size causes longer searching time, but does not significantly affect the number of threads that leads the system to its peak performance.

In general, the necessary number of threads is system dependent, which is affected by any change of the command mixture ratio, the IR system itself, and the underlying hardware platform. However we may estimate the necessary number of threads in a server that uses N_{cpu} CPUs and N_{disk} disks for processing a $(N_{disk} \cdot M)$ GB collection as follows:

1. Run a batch of sample queries against the M data on a single disk using a single CPU and a single thread, and collect the total CPU time and the total disk time for all query commands (T_{Qcpu} and T_{Qdisk}), and the total CPU time and the total disk time for all summary and document commands (T_{Dcpu} and T_{Ddisk}).

2. Calculate the base necessary number of threads (BN_{th}) using:

$$BN_{th} = \frac{T_{Qcpu} + T_{Qdisk} + (T_{Dcpu} + T_{Ddisk})/N_{disk}}{min(T_{Qcpu} + T_{Dcpu}/N_{disk}, T_{Qdisk} + T_{Ddisk}/N_{disk})}.$$

Note the workloads involved in the summary and document commands are distributed over disks.

3. Estimate the necessary number of threads (NN_{th}) using

$$NN_{th} = N_{disk} \cdot BN_{th}.$$

Here we give an example how to use the above formula to estimate the necessary number threads for a server using 1 CPU and 4 disks for 4 GB data. From our simulation results for using 1 CPU and 1 thread for 1 GB data on 1 disk, we collected:

$$T_{Qcpu} = 19.9 \text{ seconds}$$
$$T_{Qdisk} = 64.2 \text{ seconds}$$
$$T_{Dcpu} = 35.8 \text{ seconds}$$
$$T_{Ddisk} = 115.8 \text{ seconds}$$

We calculated $BN_{th} = (19.9 + 64.2 + (35.8 + 115.8)/4)/(19.9 + 35.8/4) = 4.2$, which means we need 4 threads for a disk. For a server using 1 CPU and 4 disks for processing 4 GB of data, we need approximately 16 threads in total, which matches our finding in Figure 6.3(b).

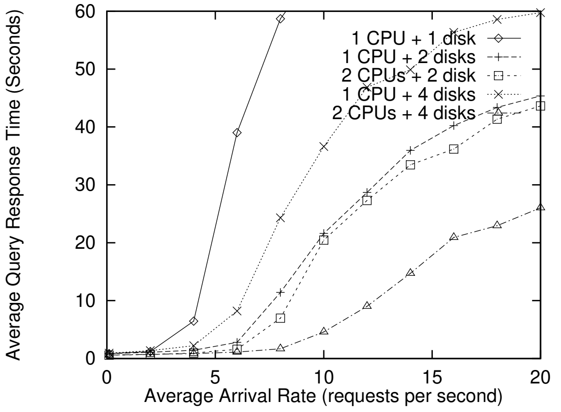
6.1.2 The Hardware Balancing Act

This section investigates how to balance hardware components with respect to varying the numbers of CPUs and disks.

Migration of balancing points with adding hardware components

In general, adding hardware components improves system performance by providing more computation capacity. Figure 6.5 gives an example of how the hardware balancing points migrate as we handle a 4 GB collection by adding CPUs and disks.

R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}	N_{th}
1:1.5:2	2	Obs.	4 GB	varied	varied	32



(a) average response time

Resource	1 CPU+1 disk	1 CPU+2 disks	2 CPUs+2disks	1 CPU+4 disks	2 CPUs+4disks
CPU	30.9%	72.9%	36.3%	92.3%	49.4%
DISK	98.4%	80.7%	80.4%	47.7%	50.2%

(b) hardware utilization when $\lambda = 6$

Figure 6.5. Performance as the hardware configuration changes for a 4 GB collection.

In the case of 1 CPU and 1 disk, the system is disk bound (see Figure 6.5(b)). When we add one disk, the disk bottleneck is relieved, and the system is well balanced. The largest command arrival rate at which the system supports an average query response time under 10 seconds increases by 100%. When we add one CPU into this well-balanced system, the performance further improves, but the improvement is relatively small. The largest arrival rate at which the system supports a query response time under 10 seconds increases by 2%. If we add two more disks instead of adding one CPU, forming a system with 1 CPU and 4 disks, the system becomes CPU bound due to additional overheads to access each disk. The performance using 4 disks is worse than 2 disks, which indicates adding disks can degrade performance when the hardware components are not well balanced. When we add one more CPU, the system is well balanced again, the performance further improves. By comparing two well balanced cases, more hardware components greatly improves the system. For example, for a 4 GB collection, the system with 1 CPU and 2 disks supports 8 commands per second with an average query response time under 10 seconds, while the system with 2 CPUs and 4 disks supports 12 commands per second.

The above observations suggest that adding hardware components improves system performance only for a well-balanced system. If we do not pay attention to the hardware balancing act, we can degrade system performance although we invest more hardware.

As in the case of the number of threads, the right ratio of the number of CPUs and the number of disks is system dependent, which is affected by any change of the command mixture ratio, the IR system itself, and the underlying hardware platform. Assume we want put a M GB collection on a disk, we may estimate the ratio N_{cpu}/N_{disk} as follows:

1. Run a batch of sample IR commands against a M GB collection on a single disk using a single CPU and a single thread, and collect the total CPU time and the total disk time for all query commands (T_{Qcpu} and T_{Qdisk}), and the total CPU time and the total disk time for all summary and document commands (T_{Dcpu} and T_{Ddisk});

2. Collect the time overhead (T_{oh}) for dispatching and merging results for query and summary commands when using two disks, each of which stores a M GB collection.

3. Estimate N_{cpu}/N_{disk} using

$$\frac{N_{cpu}}{N_{disk}} = \frac{N_{disk} \cdot T_{Qcpu} + T_{Dcpu} + (N_{disk} - 1) \cdot T_{oh}}{N_{disk} \cdot T_{Qdisk} + T_{Ddisk}}.$$

Here we give an example how to use the above formula to estimate the number of CPUs when we plan to distribute a 4 GB collection over 4 disks. From our simulation results for using 1 CPU and 1 thread for 1 GB data on 1 disk, we collected:

$$T_{Qcpu} = 19.9 \text{ seconds}$$

$$T_{Qdisk} = 64.2 \text{ seconds}$$

$$T_{Dcpu} = 35.8 \text{ seconds}$$

$$T_{Ddisk} = 115.8 \text{ seconds}$$

From our simulation results for using 1 CPU and 1 thread for 2 GB data over 2 disks, we collected T_{oh} to be 16.2 seconds.

We substituted these value in the formula, we obtained $N_{cpu} = 4 * \frac{4*19.9+35.8+(4-1)*16.2}{4*64.2+115.8} \doteq 2$. Therefore, we need 2 CPUs for handling the 4 GB collection over 4 disks, which matches our finding in Figure 6.5.

Increasing the collection size

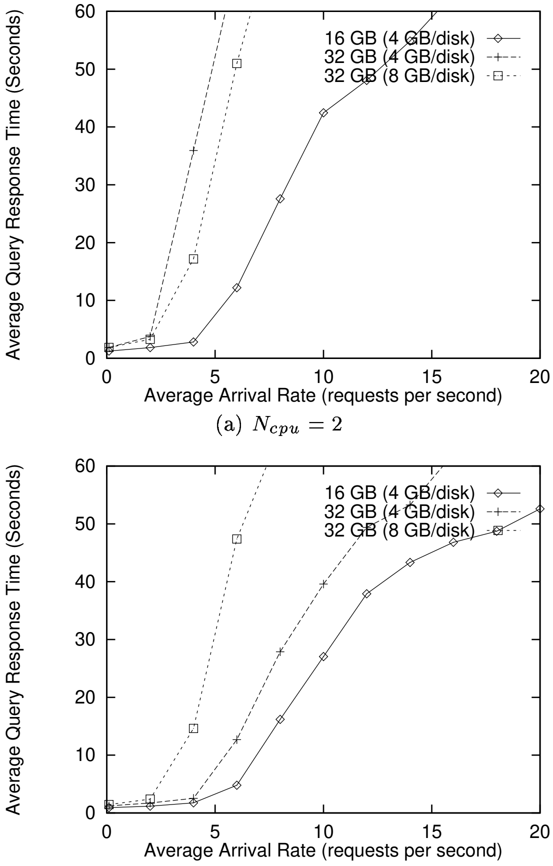
This section examines how to balance hardware components as the collection size increases. When the collection size increases, we may cope with this problem by either buying more disks where each disk holds the same amount of data as before, or replacing the current disks with larger ones where each disk holds more data.

Figure 6.6 illustrates the average query response time and the resource utilization for adding 4 disks and keeping the same amount of data on each disk, and using the same number of disks but putting twice as much data on each disk, when the collection size increases from 16 GB to 32 GB.

From measuring our system, we know it takes each disk less time and each CPU more time to search the same amount of data distributed over twice as many disks, because each disk handles less data, but the total time increases as the number of disks increases because each CPU has more work.

In a 2 CPU server illustrated in Figure 6.6(a), where 2 CPUs and 4 disks constitute a well-balanced server, adding 4 more disks and partitioning 32 GB over 8 disks makes the server become CPU bound, and thus performs worse than a server using the same number of larger disks, because the CPUs are overwhelmed. By examining CPU and

R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}	N_{th}
1:1.5:2	2	Obs.	varied	varied	varied	32



(a) $N_{cpu} = 2$

(b) $N_{cpu} = 4$

		16 GB	32 GB	
Num.		4 disks	8 disks	4 disks
CPUs	Resource	(4 GB/disk)	(4 GB/disk)	(8 GB/disk)
2	CPU	59.0%	96.9%	73.6%
	DISK	57.6%	43.8%	84.4%
4	CPU	29.3%	54.7%	40.5%
	DISK	57.3%	49.5%	86.9%

(c) hardware utilization when $\lambda = 4$

Figure 6.6. Increasing the number of disks versus increasing the data size per disk as the collection size increases.

disk utilization in Figure 6.6(c), putting more data on a disk moves the balance point a little bit (the ratio of CPU and disk utilization moves from 1:02:1 to 0.9:1 for 2 CPUs), but not significantly. The balance point moves because the term frequency of a given term increases as the collection size increases, and the term evaluation time is a linear function of the term frequency (refer to Section 5.1.1).

In a 4 CPU server illustrated in Figure 6.6(b), where 4 CPUs and 4 disks constitute a disk-bound system, adding 4 more disks and partitioning 32 GB over 8 disks performs better than a server using the same number of larger disks, because it relieves the disk bottleneck and makes the server well-balanced.

The above observations suggest that how to handle additional data is determined by the balance between hardware resources. Putting more data on each disk is superior to adding disks when the server is already well-balanced.

6.1.3 Partitioning Versus Replication

In this section, we examine how to place the collection on the available disks. We may either partition the collection over all the disks or partition it over some disks and replicate it on some disks. We investigate both full and partial replication.

Partitioning Versus Full Replication

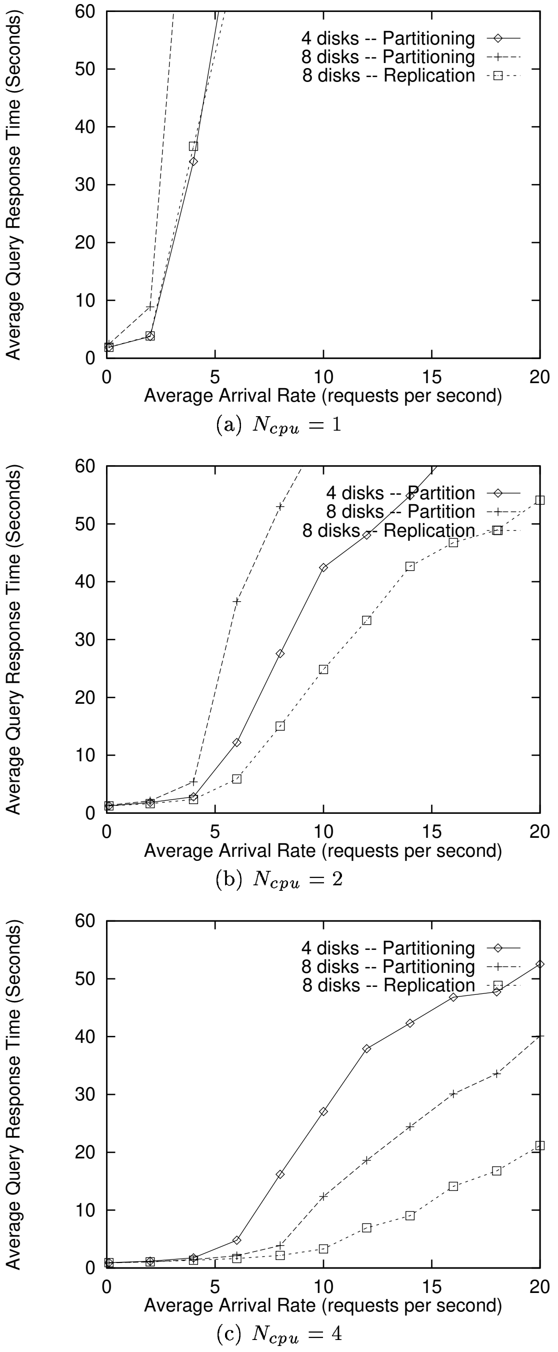
For a unloaded system, partitioning over additional disks reduces the command response time, because a command is processed against the disks with less data in parallel, while full replication does not, because each disk still handles the same amount of data. For a loaded system, by using full replication, each set of disks handles half the workload, and thus reduces the command response time due to reducing the waiting time for the disk service. By partitioning a collection over twice as many disks, each disk handles half of the data, but it handles more than half the load, because it needs to access each disk to process a query command. In addition, searching half of the data takes more than half the time and searching more disks needs more coordination time. Therefore, full replication supports a larger command arrival rate under a given query response time than partitioning for a loaded system. As an example, we demonstrate the performance of searching a 16 GB collection using partitioning and full replication. We assume we have 8 disks, we compare the performance of partitioning the collection over 8 disks and building two copies of the collection, each of which uses 4 disks, illustrated in Figure 6.7.

In Figure 6.7(a) and (b), partitioning over 8 disks performs worse than 4 disks, because the additional overheads overwhelm the CPU(s) sooner. In Figure 6.7(a) where the CPU is a bottleneck for 4 disks, full replication does not degrade performance, because the system still searches 4 disks at a time, and thus does not pose additional overheads on the CPU. However, it does not improve performance either, because the CPU is already overutilized and there is no room for further improvement. In Figure 6.7(b), where 2 CPUs and 4 disks constitute a well balanced system, full replication improves performance, by shortening the waiting time for the disk service and further increasing both CPU and disk utilizations (see Figure 6.7(d). However the high CPU utilization limits its further improvement.

In Figure 6.7(c) where 4 CPUs and 4 disks constitute a system with a disk bottleneck, and 4 CPUs and 8 disks constitute a well balanced system, both partitioning the collection over 8 disks and full replication significantly improves system performance, because the disk bottleneck is relieved. However full replication performs better, because each disk and CPU handles less workloads than partitioning. In this case, because the system is well-balanced, full replication achieves it largest improvement, and supports twice as many commands with an average query response time below 10 seconds compared with partitioning over 4 disks (from 7.3 commands per second to 14.5 commands per second).

The above results confirm that full replication supports a larger command arrival rate under a given cutoff for response time than partitioning the collection over additional

R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}	N_{th}
1:1.5:2	2	Obs.	16 GB	varied	varied	32



(a) $N_{cpu} = 1$

(b) $N_{cpu} = 2$

(c) $N_{cpu} = 4$

Num. CPUs	Resource	part.+ 4 disks	part.+ 8 disks	full repl.
1	CPU	99.3%	99.8%	99.4%
	DISK	48.1%	20.4%	24.7%
2	CPU	85.3%	98.5%	97.9%
	DISK	84.1%	40.3%	48.4%
4	CPU	46.9%	84.0%	70.2%
	DISK	90.6%	71.3%	68.8%

(d) hardware utilization when $\lambda = 10$

Figure 6.7. Partitioning versus full replication for a 16 GB collection.

disks for a loaded system. However, we do use partitioning instead of replication to keep the response time below a given cutoff, when users typically issue very long queries, or when each disk holds a large collection. For example, our measurements show that it takes 2.5, 1.4, and 0.8 seconds to process a query with 2 terms on a 16, 8, and 4 GB collection on a single disk. If we expect our server produces a response time below 1 second for a 16 GB collection, we need to partition it over 4 disks, each of which holds 4 GB of data.

Partitioning Versus Partial Replication

In the case of full replication, we may evenly distribute the workloads over full replicas and thus improve system performance. However, in the case of partial replication, how much partial replication improves performance depends on the distracting percentage, i.e., how many commands partial replicas satisfy. Partial replication performs better than partitioning only when the replicas distract (satisfy) a certain number of commands. Assume partitioning the collection over N_{disk_orig} disks supports λ_{orig} commands per second with an average query response time under some cutoff and partitioning the collection over additional N_{disk_new} disks ($N_{disk\_orig} + N_{disk\_new}$ in total) supports λ_{new} commands per second with the same average response time, then building partial replicas using additional N_{disk_new} disks achieves the same average response time as partitioning when the replicas distract less than $\frac{\lambda_{new}-\lambda_{orig}}{\lambda_{new}} \cdot 100\%$ of commands, if $\frac{\lambda_{new}-\lambda_{orig}}{\lambda_{new}} \cdot 100\%$ of commands do not overload the replicas.

As an example, we present a set of experiments to compare the performance of partitioning and partial replication for a 16 GB collection. We assume each disk can store at most 4 GB of data, and thus we need 4 disks to store the 16 GB collection, which is our baseline. When we have an additional disk, we may store the collection by either partitioning the collection over 5 disks, or building a partial replica on the 5th disk which contains 25% of the original collection. If query locality is high, the replica selector may send more queries to the replica than the original collection, which may result in load imbalance. We balance the loads using the method we described in Section 4.4.

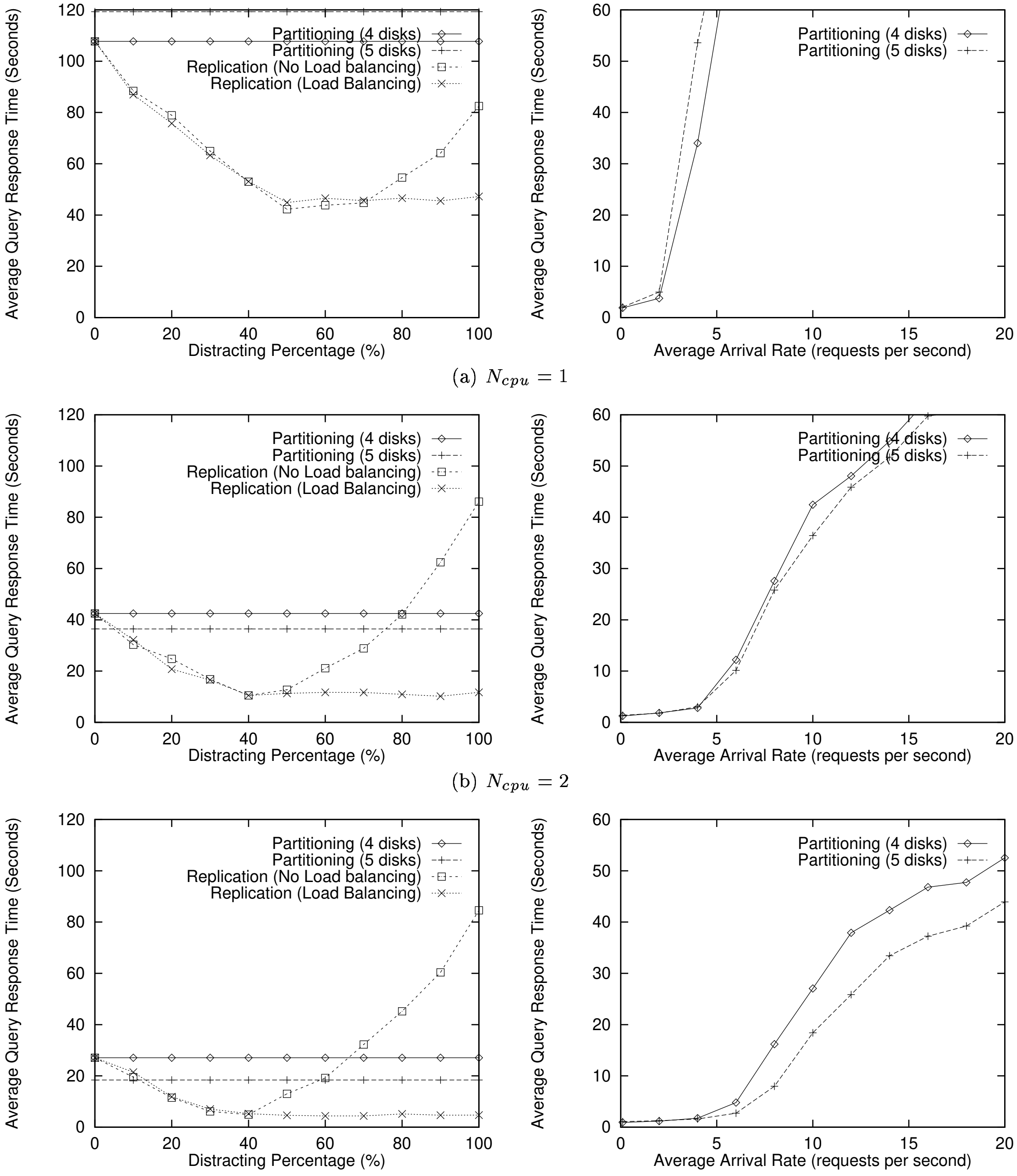
Figure 6.8 presents the average query response time versus the distracting percentage when commands arrive at 10 per second as well as the average query response time versus the command arrival rate for partitioning over 4 or 5 disks. In Figure 6.8(d), the rows of "DISK" list the average disk utilization over all disks for partitioning, and average utilization over the disks that store the original collection; the rows of "DISK (repl)" list the utilization of the disk that stores the partial replica.

In Figure 6.8(a), where one CPU is a bottleneck for 4 disks, partitioning the collection over 5 disks is actually worse than partitioning it over 4 disks, because additional CPU overheads due to searching more disks exacerbate the CPU bottleneck. It is not surprising that partial replication performs much better than partitioning at all data points of the distracting percentage, because searching the replica on single disk takes less CPU time than searching 4 or 5 disks, and thus relieves the CPU bottleneck.

In Figure 6.8(b), where 2 CPUs and 4 disks constitute a well balanced system, partitioning the collection over 5 disks performs slightly better than partitioning over 4 disks. By examining the right figure, partitioning over 4 disks and 5 disks support 5.7 and 6 commands per second with an average query response time under 10 seconds, respectively. Partial replication performs better than partitioning over 5 disks when the replica distracts more than 5% of commands from the left figure, which is consistent with our formula $(6 - 5.7)/6 \doteq 5\%$.

In Figure 6.8(c) where 4 CPUs and 4 disks constitute a disk bound system, both partitioning the collection over 5 disks and partial replication significantly improve system performance, because each disk handles less loads, which relieves the disk bottleneck. Partial replication performs better than partitioning over 5 disks when the replica distracts more than 13% of commands. By examining the right figure,

R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{cpu}	N_{disk}	N_{th}
1:1.5:2	2	Obs.	16 GB	varied	varied	32



(a) $N_{cpu} = 1$

(b) $N_{cpu} = 2$

(c) $N_{cpu} = 4$

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

(d) hardware utilization when $P_{dist} = 50\%$

Figure 6.8. Partitioning versus partial replication for a 16 GB collection.

partitioning over 4 disks supports 7.0 commands per seconds with an average query response time under 10 seconds, and partitioning 5 disks supports 8.4 commands per second. Partial replication should achieve the same average response time as partitioning over 5 disks when the replica distracts less than $(8.4 - 7.0)/8.4 \doteq 17\%$, which is consistent with our finding in the left figure.

By examining Figure 6.8, we also find that the improvement due to partial replication increases as the replica distracts increasingly many commands until the replicated disk load gets too high. At that point, load balancing is necessary. It also suggests that more than one replica will be helpful when the system exhibits high query locality ($> 40\%$).

6.1.4 Summary

Since the IR workload is heterogeneous, i.e., it consists of significant amounts of both CPU and I/O processing, multithreading improves performance of the base system by introducing well-known multiprogramming benefits - increasing the hardware resource utilization because I/O and computation overlap. The necessary number of threads that leads the system to its peak performance is directly related to the number of CPUs and disks. The size of collection has very little effect on the necessary number of threads.

Adding disks improves performance because partitioning the collection across multiple disks introduces a finer-grain execution of IR commands in parallel and using additional disks for building replicas improves the system throughput. Adding CPUs also improves performance because more CPUs provide more computation capability. However adding hardware components significantly improves performance only when they are well-balanced. Otherwise, adding disks can degrade the performance when the CPU is a bottleneck, and adding CPUs does not improve performance when the disk is a bottleneck, as we show through the experiments in this section. Increasing the collection size does not significantly change hardware balance point; the server performance is more related to the balance of hardware components than to the collection size. We present several formulas that use simple measurements to estimate the necessary number of threads, the number of CPUs and disks that constitute a well-balanced system.

Our results also show that when the response time for searching a collection distributed over a set of disks is below a cutoff for the response time, full replication supports a larger command arrival rate under this response cutoff than partitioning the collection over twice as many disks, and partial replication supports a larger command arrival rate under this cutoff than partitioning a collection over additional disks when the replica distracts a modest number of commands. When the response time is above the cutoff, we need to use partitioning instead of replication to keep the response time down.

6.2 Partial Collection Replication in a Distributed Information Retrieval System

This section investigates the performance of a distributed information retrieval system with partial collection replication. During the experiments, we vary the command arrival rate, the replication percentage that indicates what percentage of the original collection is replicated, and the distracting percentage that indicates what percentage of commands the replica selector directs to the replicas instead of the original collection. In the experiments of this section, we configure servers with 4 CPUs and 8 disks, and each server handles 32 GB of text and its index. According to our measurements from InQuery, this server supports at most 7 commands per second with an average query response time under 10 seconds for short queries with an average 2 terms per query. We configure the server in this way just to give an example to show the performance trends of a distributed IR system. Of course we can build a

Parameters	Abbre.	Values
Command Arrival Rate Poisson dist. (avg. commands/sec)	λ	0.1 2 4 6 8 10 12 14 16 18 20
Command Mixture Ratio query:summary:document	R_{cm}	1:1.5:2
Terms per Query (average) shifted neg. binomial dist.	N_{tpq}	2
Query Term Frequency dist. from queries	D_{qtf}	Obs. Dist.
Number of CPUs	N_{cpu}	4
Number of Disks	N_{disk}	8
Number of Threads	N_{th}	32
Collection Size	C_{size}	64 GB 128 GB 256 GB
Replication Percentage	P_{repl}	6.25% 12.5% 25% 50%
Distracting Percentage	P_{dist}	10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

Table 6.2. Configuration parameters for replication experiments.

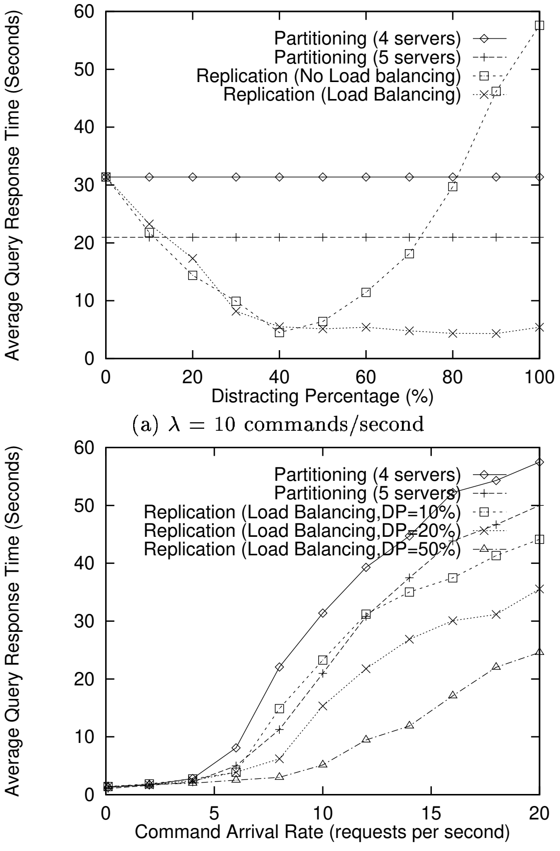
server with the same performance by using fewer and faster CPUs and disks. We include a replica selector in the connection broker. Table 6.2 presents parameters, their abbreviations, and values we use in the experiments of this section.

6.2.1 Varying the Distracting Percentage

Figure 6.9 illustrates performance when we use 4 servers to store the 128 GB original collection and build a 32 GB replica (25% of the original collection) on the 5th server. Figure 6.9(a) illustrates the average query response time versus the distracting percentage when the commands arrive at 10 commands per second. Figure 6.9(b) illustrates the average query response time versus the command arrival rate when the replica distracts 10%, 20%, and 50% of IR commands.

Similar to the behavior on a single server, Figure 6.9(a) shows that the performance of partial replication improves as the distracting percentage increases until too many commands are directed to the replica and load balancing is needed to direct a subset of commands back to the original collection. The redirecting point is when the response time for searching a replica equals to the response time for searching the original collection (40% in this experiments). For using a single replica, the redirecting point is less than 50%, because for the same command arrival rate, the server used to store the replica handles more load than an individual server used to store the original

λ	R_{cmd}	N_{tpq}	D_{qtf}	C_{size}
varied	1:1.5:2	2	Obs.	128 GB
N_{cpu}	N_{disk}	N_{th}	P_{dist}	P_{repl}
4	8	32	varied	25%



(a) λ = 10 commands/second

(b) varying the command arrival rate.

Figure 6.9. Varying the distracting percentage.

collection, since the replica uses fewer servers, and workloads involved in summary and document commands are distributed. More than one replica is useful when the distracting percentage is high.

When the distracting percentage is $p\%$, we may roughly estimate the improvement on the command arrival rate under a certain average query response time due to partial replication as a factor of $\frac{1}{1-p\%}$, if $p\%$ of commands do not overload the replicas. This is a lower bound of the improvement when replicas are not overloaded, since it is derived by assuming both the replica and the original collection produce the same average query response time (the redirecting point). Actually, an under-loaded replica produces much quicker response than the original collection, which makes the original collection tolerate more commands for the same average query response time. When load-balancing redirects some commands back to the original collection for a high distracting percentage, we can not attain this improvement.

As an example, examine Figure 6.9(b). Distracting 10%. 20%, and 50% of commands increases the command arrival rate under 10 seconds by a factor of 1.2, 1.4, and 1.9, while our prediction is 1.1, 1.3, and 2.0. Distracting 50% of commands performs a little worse than we predict, because load balancing redirects some commands back to the original collection (see Figure 6.9(a)).

6.2.2 Partial Replication Versus Collection Partitioning

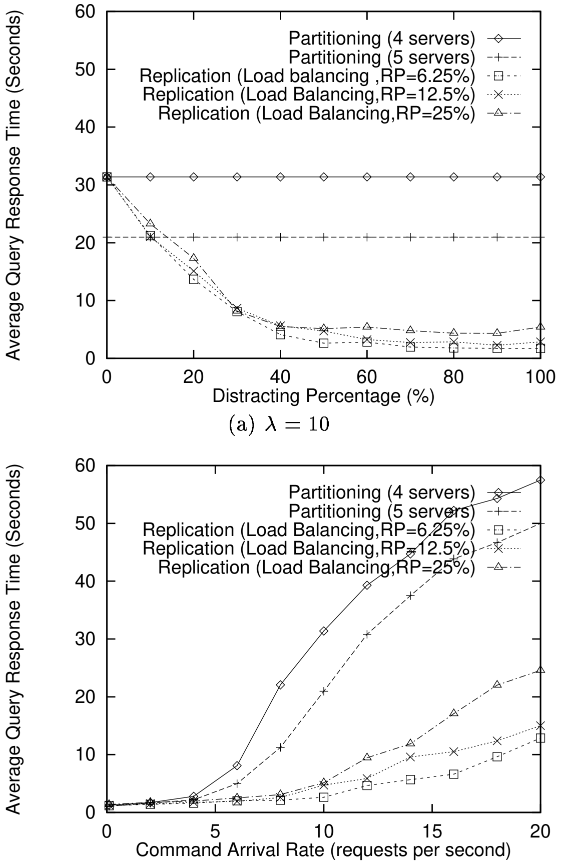
Partial replication in a distributed system performs better than collection partitioning when it distracts a certain number of commands which is less than the command difference that partitioning over additional servers can handle.

As an example, we compare the performance of partial replication with collection partitioning for a 128 GB collection, as shown in Figure 6.9. In Figure 6.9, we either use 4 servers to store the original collection and build a 32 GB replica (25% of the original collection) on the 5th server or partition the collection over 5 servers. As illustrated in Figure 6.9(b), partitioning the collection over 4 and 5 servers support 6.2 and 7.5 commands per second with an average query response time under 10 seconds, i.e., partitioning the collection over 4 servers handles 17% less commands than partitioning it over 5 servers. In Figure 6.9(a), partial replication with load balancing performs better than partitioning over 5 servers when the replica distracts 14% of commands, which is slightly less than the command percentage difference of partitioning (17%).

6.2.3 Varying the Replicating Percentage

This section examines the effect of varying the replica size on performance. Figure 6.10 illustrates the average query response time for a 128 GB collection when we use 4

λ	R_{cm}	N_{tpq}	D_{qtf}	C_{size}
varied	1:1.5:2	2	Obs.	128 GB
N_{cpu}	N_{disk}	N_{th}	P_{dist}	P_{repl}
4	8	32	varied	varied



(a) $\lambda = 10$

(b) $P_{dist} = 50\%$

Figure 6.10. Varying the replication percentage.

servers to store the original collection, each of which stores 32 GB, and an additional server to store a replica. We vary the replica size from 8 GB, 16 GB, and 32 GB, which is 6.25%, 12.5% and 25% of the original collection. We vary the command arrival rate and the distracting percentage.

Figure 6.10 shows that reducing the replication percentage of course reduces the average query response time, and it significantly affects system performance only for high commands rate ($\geq$ 10 commands per second in Figure 6.10(b)). When the replica distracts 50% of commands, reducing the replication percentage from 25% to 6.25% for a 128 GB collection increases the command arrival rate with an average query response time below 10 seconds by a factor of 1.5.

6.2.4 Varying the Collection Size

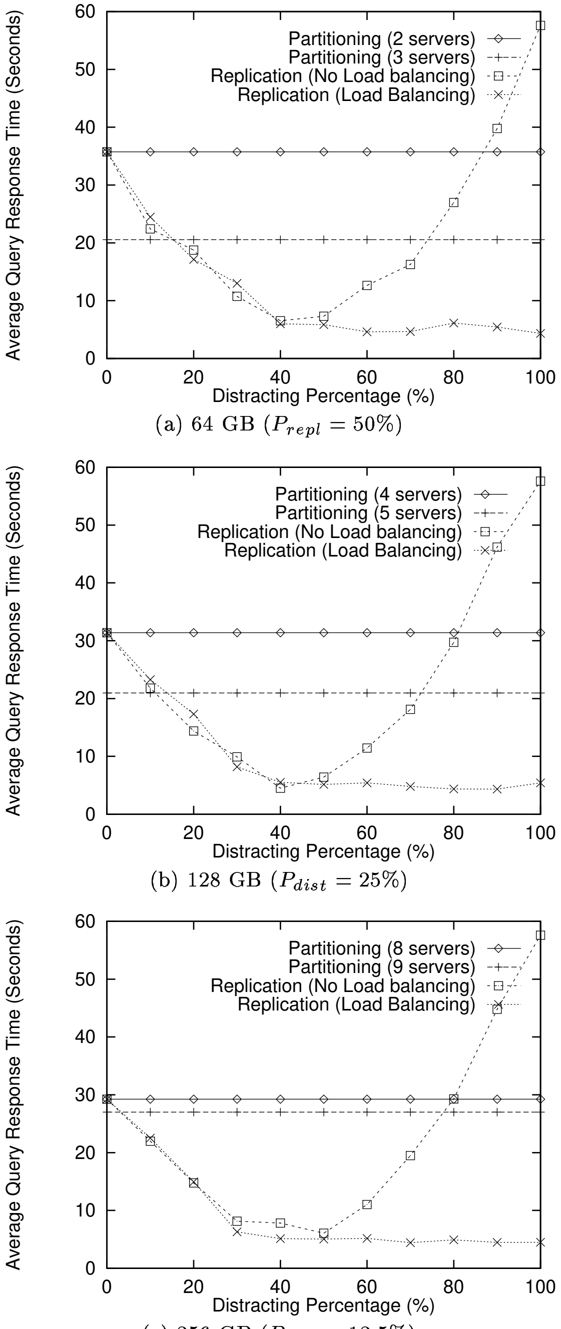
In this set of experiments, we vary the collection size from 64 GB to 256 GB. For a N GB collection, we use $N/32$ servers to store the original collection, and an additional server to store a replica with the size of 32 GB. Figure 6.11 lists the average query response time when the commands arrive at 10 per second for a collection of 64 GB, 128 GB, and 256 GB, i.e., the partial replica contains 50%, 25% and 12.5% of the original collection, respectively.

Figure 6.11(a) to (c) show the query response time when commands arrive at 10 commands per seconds for 64 GB, 128 GB, and 256 GB, respectively. The results show that performance improvement due to partitioning over an additional server decreases as the collection size increases, because an additional server distributes a smaller portion of data for a larger collection, while partial replication on the additional server consistently decreases the query response time as the distracting percentage increases, no matter what the collection size is, because the query locality indicated by the distracting percentage determines the performance improvement. Partial replication brings down the query response times from 35.8 to 3.8 seconds, from 31.4 to 3.5 seconds, and from 29.2 to 3.1 seconds when commands arrive at 10 commands per second for 64 GB, 128 GB, and 256 GB, respectively. Here we can find another interesting phenomenon: when each server holds the same amount of data, searching a 256 GB collection (over 8 servers) is faster than searching a 128 GB (over 4 servers) and 64 GB (over 2 servers), because the workload involved in the summary and document commands is distributed over more servers, and each server handles less work.

6.2.5 Replication Hierarchy

In this set of experiments, we assume we have several additional servers, each of which stores a partial replica, and we organize the servers as a hierarchy of replicas. We

λ	R_{cm}	N_{tpq}	D_{qtf}	C_{size}
10	1:1.5:2	2	Obs.	Varied
N_{cpu}	N_{disk}	N_{th}	P_{dist}	P_{repl}
4	8	32	varied	varied



(a) 64 GB ($P_{repl} = 50\%$)

(b) 128 GB ($P_{dist} = 25\%$)

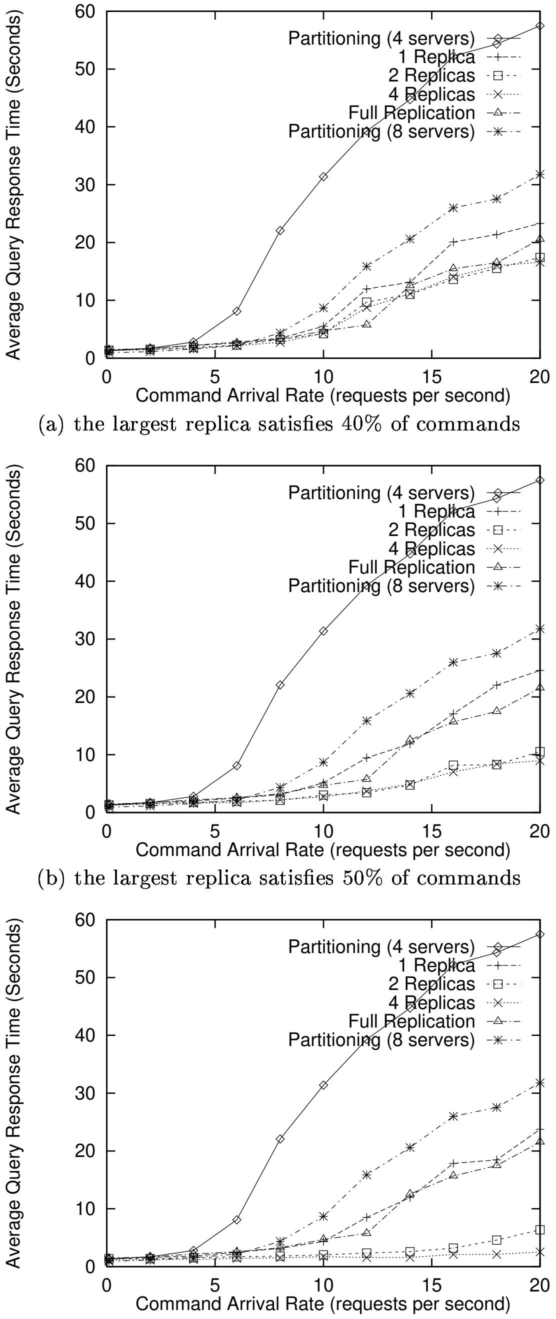
(c) 256 GB ($P_{dist} = 12.5\%$)

Figure 6.11. Varying the collection size.

assume each server handles at most 32 GB of data. For a N GB collection, we use $N/32$ servers to store the original collection, the first additional sever stores 32 GB of data and satisfies p% of commands, the second additional server stores 16 GB of data and satisfies (p% - 10%) less commands, which is a subset of the commands that the first additional server satisfies, and the i-th extra server stores $32/2^{i-1}$ GB and satisfies (p% -(i-1)*10%) of commands, which is a subset of commands that the larger replicas satisfy.

Figure 6.12 illustrates the query response time when we have 1, 2, and 4 additional servers for a 128 GB collection. We compare the performance of partial replication with different hierarchies, and full replication using 4 additional servers. Figure 6.12(a) to (c) illustrate the scenarios where the largest replica satisfies 40%, 50% and 80% of commands, respectively. Our results show that when we have twice as many servers, full replication performs better than partitioning over these servers, and in our example, using additional servers to build a full replica improves the command arrival rate with an average query response time below 10 seconds by a factor of 1.3, as compared with partitioning over 8 servers. Partial replication using fewer servers could achieve similar or better performance than full replication. Partial replication with one replica is able to achieve similar performance to full replication when the partial replica distracts 40% of commands. Partial replication with a hierarchy of two replicas improves the command arrival rate with an average query response time below 10 seconds by a factor of 1.7 as compared with full replication when the replicas distract 50% of commands (see Figure 6.12(b)), because searching a smaller replica takes less time, and it also eliminates the result merging time and reduces the necessary message exchange between the connection broker and servers. When the largest replica satisfies 80% of commands, the query response time for partial replication with a hierarchy of 4 additional servers is insensitive to the command arrival rate that we consider in this experiment. This result indicates that partial replication

λ	R_{cm}	N_{tpq}	D_{qtf}	C_{size}
varied	1:1.5:2	2	Obs.	128GB
N_{cpu}	N_{disk}	N_{th}	P_{dist}	P_{repl}
4	8	32	varied	25%



(a) the largest replica satisfies 40% of commands

(b) the largest replica satisfies 50% of commands

(c) the largest replica satisfies 80% of commands

Figure 6.12. Performance with a hierarchy of replicas.

with a hierarchy of replicas has the potential to improve performance and scalability significantly over full replication.

6.2.6 Summary

In this section, we demonstrate the performance of partial collection replication in a distributed information retrieval system. The performance improvement due to partial collection partition is determined by the distracting percentage that indicates access locality. We compare the performance of partial replication with collection partitioning. We show that using one additional server to build one replica performs better than further partitioning the collection over this additional server, even when the replica distracts less than 10% of commands in many cases. It also achieves comparable performance with partitioning over twice as many servers, and full replication when the replica distracts 40% of commands. When replicas can distract more than 40% of commands, a hierarchy of replicas further improves system performance.

6.3 Collection Selection in a Distributed Information Retrieval System

This section investigates the performance of a distributed information retrieval system using collection selection. As an example to show the performance improvement due to collection selection, we perform a set of experiments that handle 256 GB of data using 8 servers, each of which has 4 CPUs and 8 disks, and handles 32 GB of data. We include a collection selector in the connection broker. During the experiments, we vary the command arrival rate, the collection selection skew, the selection percentage, and the number of collections. We also compare collection selection with partial collection replication. Table 6.3 presents parameters, their abbreviations, and values we use in the experiments of this section.

Parameters	Abbre.	Values
Command Arrival Rate Poisson dist. (avg. commands/sec)	λ	0.1 2 4 6 8 10 12 14 16 18 20
Command Mixture Ratio query:summary:document	R_{cm}	1:1.5:2
Terms per Query (average) shifted neg. binomial dist.	N_{tpq}	2
Query Term Frequency dist. from queries	D_{qtf}	Obs. Dist.
Number of CPUs	N_{cpu}	4
Number of Disks	N_{disk}	8
Number of Threads	N_{th}	32
Collection Size	C_{size}	256 GB
Collection Access Skew Zipf-like function	θ	0.0 (Zipf) 0.3 1.0(uniform)
Selection Percentage	P_{sel}	12.5% 25% 50%
Number of Collections	N_{col}	8 32

Table 6.3. Configuration parameters for collection selection experiments.

6.3.1 Varying the Collection Access Skew and the Selection Percentage

The performance improvements due to collection selection are mainly determined by the popularity of the most frequently used collection after collection selection. In addition, searching a subset of collection saves the time for merging results and exchanging messages, which also contributes to the performance improvements. If p% of commands go to the most frequently used collection, collection selection will improve the largest command arrival rate under a certain average query response time by a factor of $\frac{1}{p\%}$, which means the system can handle $\frac{100\%-p\%}{p\%}$ more commands than a system without collection selection. If nearly 100% of commands go to the most frequently used collection, performance improvements are small and only due to less result merging time and network competition.

There are two factors that determine the popularity of the most frequently used collection: the collection access skew and the percentage of collections being chosen. The system with collection selection achieves the largest improvement when the collections are uniformly accessed and the system selects as few collections as possible while maintaining acceptable precision.

We model the collection access skew using the Zipf-like distribution [50]:

$$Z(i) = c/i^{1-\theta}, \text{ where } c = 1/\Sigma_{j=1}^{C}(1/j^{1-\theta}), 1 \leq i \leq C.$$

As θ varies from 1 to 0, the probabilities vary from an uniform distribution to a pure Zipf distribution. We do not have statistics about the collection access skew resulting from collection selection in a real multiple collection searching environment, but locality suggests that uniform distributions are unlikely.

We use the collection inference retrieval network to select the most relevant collections [15]. The experiments presented in [15, 54, 77] demonstrated that when we built collections such that each collection contains documents from the same source, using this method to select the top 50% of collections achieves comparable accuracy with searching all collections, and selecting the top 10% and 20% of collections causes a loss of precision around 30% and 25% for the top 30 retrieved documents. Xu *et al.* [77] further demonstrated that query expansion can reduce the precision loss for searching the top 10% of collection to 10%, but it is paid by the time for query expansion (executing an original query and analyzing the top documents) and the time to process at least 20 additional 20 terms or phrases added by query expansion for each query.

We demonstrate the performance improvements due to collection selection for 256 GB of data distributed over 8 servers. We view the data on each server as a single collection, i.e., 8 collections in total. Table 6.4 lists the percentage of commands that go to the most frequently used collection based on the Zipf-like distribution when

	Selection Percentage		
θ	12.5%	25%	50%
1 (uniform)	12.5%	25.0%	50.0%
0.3	24.0%	49.0%	76.0%
0.0(pure Zipf)	35.0%	60.0%	87.0%

Table 6.4. The percentage of commands that goes to the most frequently used collection

we vary the collection access skew from an uniform to a pure Zipf function, and the selection percentage from 12.5% to 50%. Figure 6.13 illustrates the corresponding average query response time versus the command arrival rate resulting from our simulator.

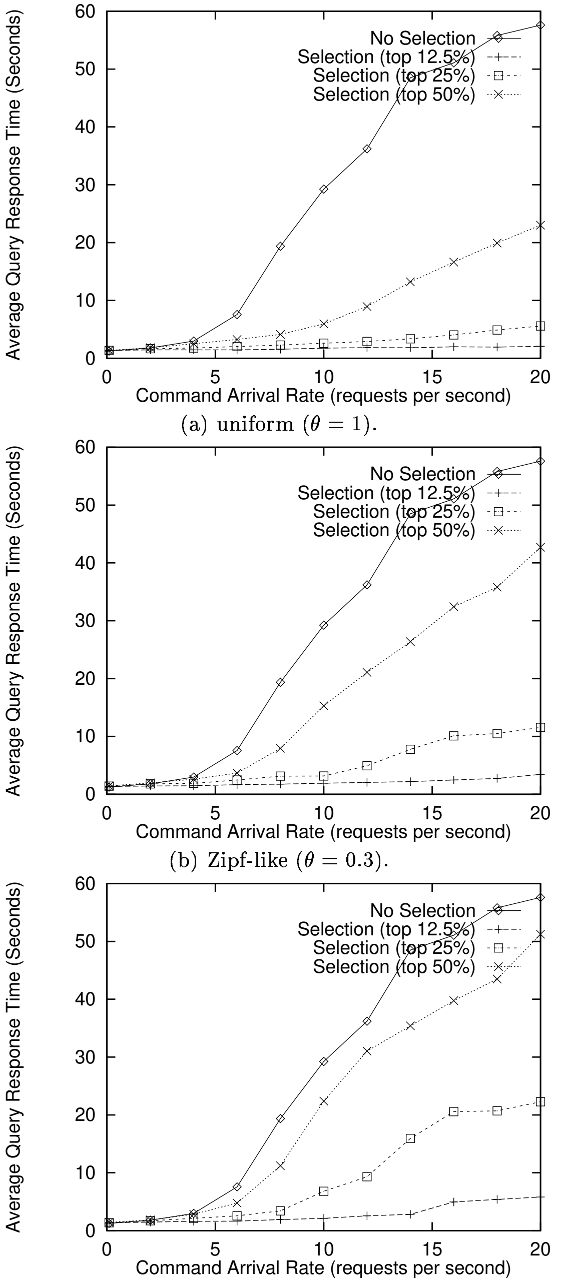
Figure 6.13(a) illustrates the scenarios where the collections are uniformly accessed. Selecting 12.5% of the collections is insensitive to the commands arrival rate that we consider in this experiment; selecting 25% of the collections increases the command arrival rate with a response time below 5 seconds by a factor of 4.0; selecting 50% of the collections increases the command arrival rate with a response time below 10 seconds by a factor of 2.1. These numbers are consistent with our prediction based on the first row in Table 6.4: 8 (1/12.5%), 4 (1/25.0%), 2 (1/50.0%).

Figure 6.13(b) illustrates the scenarios where the collection access skew follows a Zipf-like distribution with $\theta = 0.3$. Selecting 12.5% of the collections is insensitive to the commands arrival rate that we consider in this experiment; selecting 25% and 50% of the collections improve the command arrival rate with a response time below 10 seconds by a factor of 2.1 and 1.3, respectively. These numbers are consistent with our prediction based on the second row in Table 6.4: 4.0 (1/24.0%), 2.0 (1/49.0%), and 1.3 (1/76.0%).

Figure 6.13(c) illustrates the scenarios where the collection access skew follows the pure Zipf distribution. Selecting 12.5% of the collections increases the command arrival rate with a response under 5 seconds by a factor of 3.1, and selecting 25% and 50% of the collections improve the command arrival rate with a response time below 10 seconds by a factor of 1.8 and 1.2, respectively. These numbers are consistent with our prediction based on the third row in Table 6.4: 2.9 (1/35.0%), 1.7 (1/60.0%), 1.1 (1/87.0%).

These results demonstrate that the performance improvements due to collection selection are a function of the collection access skew and the percentage of collections

λ	R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{col}	P_{sel}	θ
varied	1:1.5:2	2	Obs.	256 GB	8	varied	varied



(a) uniform ($\theta = 1$).

(b) Zipf-like ($\theta = 0.3$).

(c) pure Zipf ($\theta = 0$).

Figure 6.13. Performance with collection selection for 256 GB of data on 8 servers.

λ	R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{col}	P_{sel}	θ
varied	1:1.5:2	2	Obs.	256 GB	varied	25%	varied

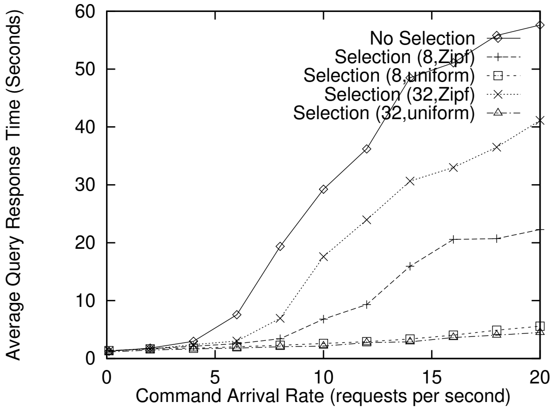


Figure 6.14. Varying the number of collections for 256 GB of data on 8 servers.

being chosen: higher collection access skew and/or more collections being chosen produce smaller performance improvements

6.3.2 Varying the Number of Collections

In this section, we still use 8 servers, but we divide the data into 32 collections, and each server holds 4 collections. We compare the performance of this configuration with the one used in Section 6.3.1 (8 collections).

Figure 6.14 illustrates the average query response time versus the command arrival rate when we select the top 25% of collections. When the collections are uniformly accessed, 8 collections and 32 collections present similar performance. Reducing the selection percentage by a half doubles the largest command arrival rate with an average query response time under a certain cutoff. When the collection access skew follows the pure Zipf distribution, fewer collections perform better, because the most frequently used collection is chosen with a lower probability when selecting a smaller number of collections. For selecting the 25% of 8 collections according to the Zipf distribution, 60% of commands go to the most frequently used collections. For selecting the 25% of 32 collections according to the Zipf distribution, 90% of commands go to the most frequently used collections.

Query	Collection Access Skew + Number of Selected Collections			
Locality	Low + Small	Low + Large	High + Small	High + Large
Low	Col.Sel			Part.Repl
High	Col.Sel	Part. Repl.		

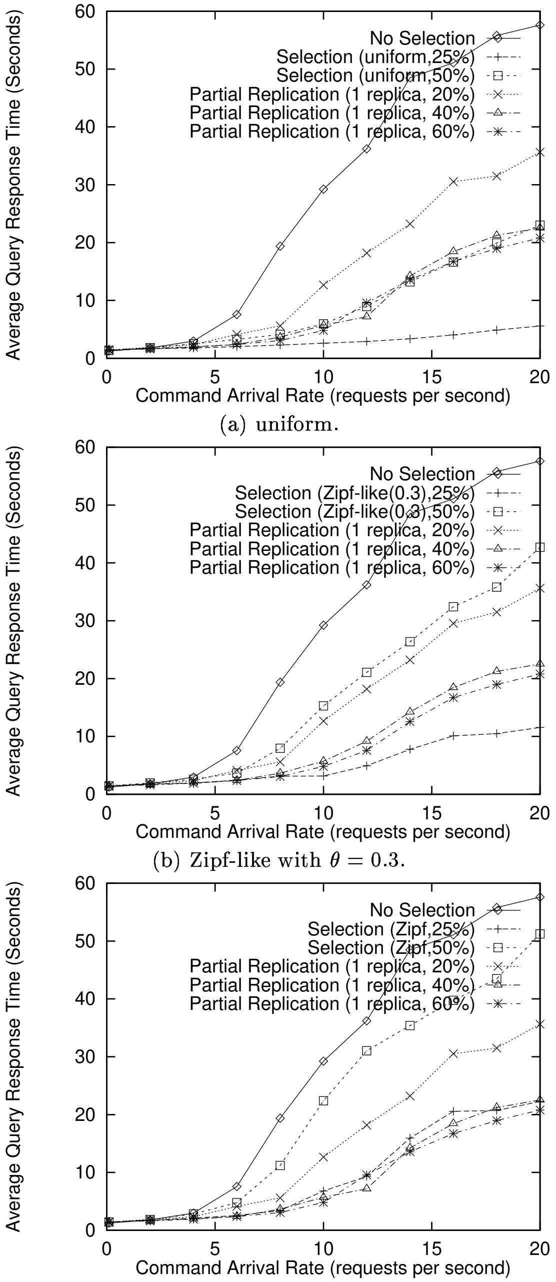
Figure 6.15. Collection selection versus partial collection replication.

6.3.3 Collection Selection Versus Partial Collection Replication

Both collection selection and partial collection replication let us search a subset of data. For collection selection, the collection access skew after collection selection and the number of selected collections affect the system performance. The number of selected collections also affects the retrieval accuracy of the system. Our current techniques only guarantee that selecting half the collections achieves accuracy comparable to searching all collections when we do not apply query expansion. In this case, when the collection access is highly skewed (following the pure Zipf distribution), collection selection contributes little to improving system performance. For partial collection replication, the performance improvement is limited by query locality. If there is not enough query locality in a system, partial collection replication is useless. Therefore, each of these two techniques performs well in different circumstances, as illustrated in Figure 6.15.

As an example, Figure 6.16 compares collection selection with partial replication for 256 GB of data using 8 servers. For collection selection, we distribute the 256 GB data over 8 collections, each of which resides on one server. For partial replication, we use 7 servers to store the 256 GB data, and build a 32 GB replica on the 8th server. Figure 6.16(a) illustrates the scenarios where collections are uniformly accessed after collection selection, which is the best case. Selecting the top 50% of collections presents similar performance to using one replica when the replica distracts 40% of commands. Selecting the top 25% of performs significantly better than all the cases of partial replication we consider in this experiment.

λ	R_{cm}	N_{tpq}	D_{qtf}	C_{size}	N_{col}	P_{sel}	θ
varied	1:1.5:2	2	Obs.	256 GB	varied	varied	varied



(a) uniform.

(b) Zipf-like with $\theta = 0.3$.

(c) pure Zipf.

Figure 6.16. Collection selection versus partial collection replication for 256 GB on 8 servers.

Figure 6.16(b) illustrates the scenarios where the collection access follows a Zipf-like distribution with $\theta = 0.3$ after collection selection. Selecting the top 50% performs worse than all the cases of partial collection replication we consider in this experiments. Selecting the top 25% performs better than all these cases.

Figure 6.16(c) illustrates the scenarios where the collection access follows the pure Zipf distribution after collection selection. Selecting the top 25% of collections presents similar performance to partial replication using one replica when the replica distracts 40% of commands. Selecting the top 50% of performance performs worse than partial replication.

The above results suggest that collection selection performs better than partial collection replication when either the collection access is fairly uniform or a small number of collections are sufficient to maintain retrieval accuracy. In practice, because of locality, uniform access is unlikely to be realized; the current techniques for collection selection require to select 50% of collections for search at a time in order to maintain the retrieval accuracy. In contrast, we find high query locality in real IR system logs (typically 30%-40% in THOMAS and Excite logs). All these indicate that partial collection replication performs better than collection selection using the current techniques.

6.3.4 Summary

In this section, we demonstrated the performance of collection selection in a distributed information retrieval system, and compared the performance of collection selection with partial collection replication. The performance improvements due to collection selection are determined by the collection access skew and the number of selected collections. Each of collection selection and partial collection replication can perform better than the other under different situations. Collection selection performs significantly better when either collection access is fairly uniform or a small number of collections are sufficient to maintain the retrieval accuracy. However, in practice, neither of these two conditions is likely to satisfy by using the current collection selection techniques.

6.4 Summary

In this chapter, we explored how best to build a parallel IR server with respect to the number of threads, CPUs, and disks, evaluated the performance of a distributed IR system, and demonstrated the performance improvement due to partial collection replication and collection selection.

Our results show that on a parallel server, the necessary number of threads that leads the system to its peak performance is directly related to the number of CPUs and disks. The size of collection has very little effect on the necessary number of threads. On a parallel server, adding hardware components significantly improves performance only when they are well-balanced. Otherwise, adding disks can degrade the performance when the CPU is a bottleneck, and adding CPUs does not improve the performance when the disk is a bottleneck. We presented several formulas that use simple measurements to estimate the necessary number of threads, the number of CPUs and disks that constitute a well-balanced system.

In a parallel IR server or a distributed IR system, when we have additional hardware resources, such as an additional disk or an additional server, using additional hardware components to build replicas produces significantly better performance than partitioning when there is modest query locality. The improvements occur even when the replica satisfies as few as 10% of queries.

Collection selection also improves performance, but its improvement is limited by the popularity of the most frequently used collection after collection selection. The collection access skew and the number of selected collections are two factors that affect the popularity of the most frequently used collection. Collection selection performs significantly better than partial collection replication when the collection access is fairly uniform or a small number of collections are sufficient to maintain the retrieval accuracy. Partial collection replication performs better than collection selection when the query locality is high. In practice, partial collection replication performs better than collection selection using the current techniques.

CHAPTER 7

TOWARD SEARCHING A TERABYTE OF TEXT

In this chapter, we present experiments for searching a terabyte of text on a cluster of symmetric multiprocessors using the architectures illustrated in Chapter 3. We first estimate the sizes of replicas, the replica selection database, and the collection selection database in Section 7.1. We then demonstrate the performance for short queries with an average of 2 terms per query by using 32 servers, 8 servers but with larger disks, and 8 faster servers in Section 7.2, Section 7.3, and Section 7.4. At last, we demonstrate the performance for longer queries with an average 8 terms per query. Table 7.1 presents parameters, and their abbreviations and values we use in the experiments of this chapter.

Our results show that a replica with 32 GB is sufficient to store the top 200 documents of the top topics that satisfy around 40% of queries based on the Excite log. Our results show that using one replica to distract 20% and using 2 replicas to distract 40% and 60% of commands improves the largest command arrival rate under a cutoff for the average query response time by a factor of 1.3, 2.2, and 3.5. Selecting the top 25% of 8 collections improves the largest command arrival rate under a cutoff for the average query response time by a factor of 1.8 when the collection skew follows the Zipf distribution. Our results also show that although using fewer large servers, speeding up servers, and using longer queries affect the response time, none of them change the relative improvements due to partial replication and collection selection with uniform access. When the collection access follows the Zipf-like distribution

Parameters	Abbre.	Values
Command Arrival Rate Poisson dist. (avg. commands/sec)	λ	0.1 2 4 6 8 10 12 14 16 18 20
Command Mixture Ratio query:summary:document	R_{cm}	1:1.5:2
Terms per Query (average) shifted neg. binomial dist.	N_{tpq}	2 8
Query Term Frequency dist. from queries	D_{qtf}	Obs. Dist.
Number of CPUs	N_{cpu}	4
Number of Disks	N_{disk}	8
Number of Threads	N_{th}	32
Size of Collection	C_{size}	1 TB
Number of Collections	N_{col}	8 32
Server Speedup	S_p	1 4
Replication Percentage	P_{repl}	3%
Distracting Percentage	P_{dist}	20% 40% 60%
Collection Access Skew Zipf-like function	θ	0.0 (Zipf) 1.0(uniform)
Selection Percentage	P_{sel}	25% 50%

Table 7.1. Configuration parameters for terabyte experiments

with the exponent $\theta < 1$, the number of collections affects the popularity of the most frequently used collection, and thus affects the performance.

7.1 The Sizes of Replicas, Replica Selection Database, and Collection Selection Databases

Before we present the performance of our distributed IR system for searching a terabyte of text, we first estimate how big a replica needs to be in order to cover the documents for top queries, and the sizes of a replica selection database and a collection selection database.

Table 7.2 estimates the replica size for a terabyte of text. The replica size is determined by two factors: the average document size and query locality. The average document size varies from source to source. For examples, the average document size of the USENET News is 2 KB; the average document size of the Associate Press Newswire and Wall Street Journal is 3 KB; the average document size of the websites operated by 10 Australia Universities is 9 KB; the average document size of the 20 GB TREC VLC collection is 2.8 KB. Our estimation uses three different numbers:

Top	% of	Replica Size (top 200 documents per query)		
Topics	Queries	(2 KB per doc)	(3 KB per doc)	(9 KB per doc)
1000	16.0%	400 MB	600 MB	1.8 GB
5000	27.9%	2 GB	3 GB	9 GB
10000	34.4%	4 GB	6 GB	18 GB
20000	42.0%	8 GB	12 GB	36 GB

Table 7.2. The replica size based on the Excite log

2 KB, 3 KB, and 9 KB. For query locality, we use the statistics obtained from the Excite log, since its workloads are at the level of the system we investigate. We collect the top 200 documents for each query. We also assume there are no overlaps among the documents, although there exists overlap in reality. In Table 7.2, columns 1 and 2 show the query locality in the Excite log (refer to table 4.2); columns 3 through 5 show the estimated size of the replica when we vary the average document size. The results show that, for example, the replica of 4 GB, 6 GB and 18 GB satisfies at least 34% of queries for the average document size of 2 KB, 3 KB, and 9 KB, respectively. As discussed in Section 4.5, the size of the replica selection database is directly proportional to the number of unique terms in the largest replica. Based on the statistics we obtained in Section 4.5 (6 MB every 100,000 unique terms) and the number of unique terms in the 20 GB TREC VLC collection (13,088,064 unique terms), we estimate that the size of our replica selection database is from 1GB to 2GB, which is 0.1% to 0.2% of the total collection size.

The size of the collection selection database is around 2% of the total size of collections based on the observation of the collection selection database for the 20 GB TREC VLC collection. For 1 terabyte of text, its size is around 20 GB.

7.2 Performance Using Queries with an Average of Two Terms

In this section, we present a set of experiments for searching a terabyte of text using short queries with an average 2 terms per query. As our baseline, we use 32 servers to store 1 terabyte of text, each of which stores a collection of 32 GB plus its indices. We view all the data on each server as a collection, and we have 32 collections in total. We then compare the performance of the following configurations with the baseline:

- Partitioning over twice as many servers: partitioning 1 TB of text over 64 servers, each of which stores 16 GB of data and its indices.

- Partial collection replication: using one additional server to store a 32 GB replica and using two additional servers to store a hierarchy of replicas with a 16 GB replica and a 32 GB replica, when the largest replica distracts 20%, 40%, and 60% of commands.

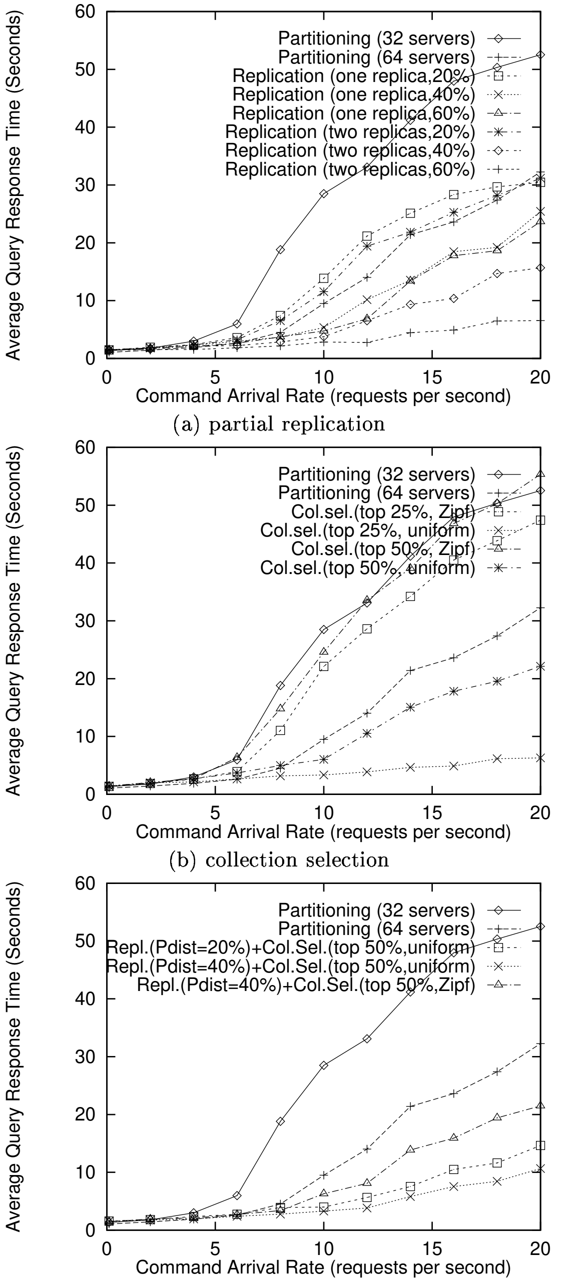
- Collection selection: using collection selection to select the top 25% or 50% of collections.

- Partial collection replication plus collection selection: conducting partial replica selection first, and when the replica selector directs a query to the original collection, the collection selector selects the top 50% of collections for searching.

We put the replica selector and collection selector in the connection broker as shown in Figure 3.1(c).

Figure 7.1 illustrates the average query response time versus the command arrival rate in the above configurations. In Figure 7.1(a), we demonstrate the average response time when we replicate 32 GB (3% of the total size) on 1 or 2 servers. Using one replica to distract 20%, 40%, and 60% of commands improves the largest command arrival rate with an average response time below 10 seconds by a factor of 1.3, 1.8, and 2.0, and using two replicas to distract 20%, 40%, and 60% of commands improves the largest command arrival rate with an average response time below 10 seconds by a factor of 1.3, 2.2, and 3.4, while partitioning over 64 servers improves it by a factor of 1.5. Using one or two additional server to replicate 3% of the data achieves similar performance to partitioning over 64 servers (32 additional servers as compared to

λ	R_{cm}	N_{tpq}	N_{qtf}	C_{size}	N_{cpu}	N_{disk}
varied	1:1.5:2	2	Obs.	1 TB	4	8
N_{th}	P_{dist}	P_{repl}	θ	P_{sel}	N_{col}	S_p
32	varied	3%	varied	varied	32	1



(a) partial replication

(b) collection selection

(c) partial replication (one replica) plus collection selection

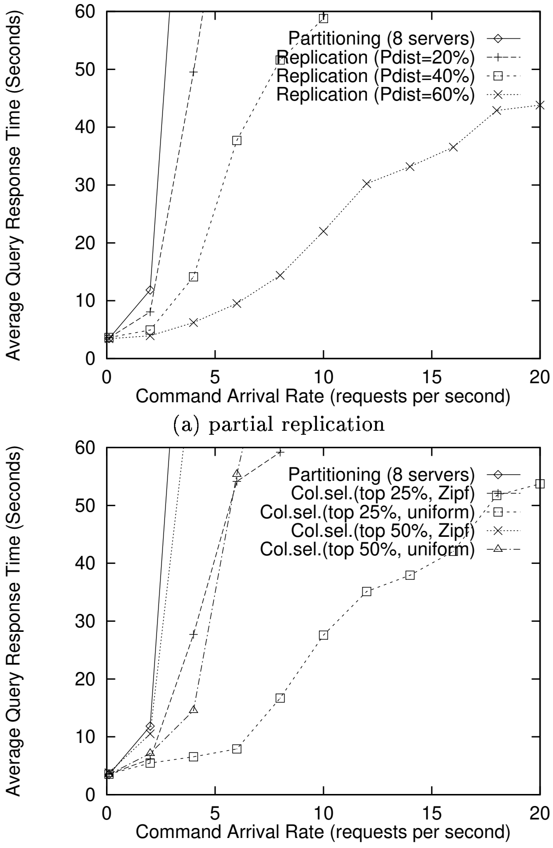
Figure 7.1. Performance when searching a terabyte of text using queries with an average of 2 terms.

the baseline) when the replica distracts 20% of commands, and performs significantly better when the replica(s) distract more than 40%.

In Figure 7.1(b), we demonstrate the average response time when we select the top 25% and 50% of collections using the pure Zipf and uniform distributions. Using the Zipf distribution illustrates a case where collection selection has no significant impact on performance, since almost all commands (100% and 90% when selecting the top 50% and 25%) go to the top 1 collection. Selecting the top 25% and 50% of collections using the pure Zipf distribution improves the largest command arrival rate with an average response time below 10 seconds by a factor of 1.2 and 1.04. Using the uniform distribution illustrates the best case for collection selection. Selecting the top 25% of collections using uniform distribution improves the largest command arrival rate with an average response time below 5 seconds by a factor of 3.9, and selecting the top 50% of collections improves the largest command arrival rate with an average response time below 10 seconds by a factor of 1.9.

In Figure 7.1(c), we illustrate scenarios where partial collection replication and collection selection work together to improve performance. In this example, we use one additional server to store a 32 GB replica and do replica selection first. When the replica selector directs a query to the original collection(s), the collection selector selects the top 50% of collections for processing the query. From Figure 7.1(a), we know using a single replica to distract 20% and 40% of commands improves the command arrival rate with a average response time below 10 seconds by a factor of 1.3 and 1.8. Plus using collection selection with the uniform collection access, the performance further improves to a factor 2.3 and 2.9. When the collection access skew follows the Zipf distribution, using a replica to distract 40% of queries plus selecting the top 50% of collections improves performance to a factor of 1.9.

λ	R_{cm}	N_{tpq}	N_{qtf}	C_{size}	N_{cpu}	N_{disk}
varied	1:1.5:2	2	Obs.	1 TB	4	8
N_{th}	P_{dist}	P_{repl}	θ	P_{sel}	N_{col}	S_p
32	varied	3%	varied	varied	8	1



(a) partial replication

(b) collection selection

Figure 7.2. Performance when searching a terabyte of text using larger disks

7.3 Performance with Larger Disks

In this section, we present another set of experiments for searching a terabyte of text using short queries with an average 2 terms per query. Each server still has 4 CPUs and 8 disks, but each disk stores 16 GB of data and its indices. Therefore each server handles 128 GB, and a terabyte of text needs 8 servers. We view all the data on each server as a collection, and we have 8 collections in total.

Figure 7.2 illustrates the performance gain by using partial replication and collection selection in this case. In Figure 7.2(a), we demonstrate the average query response time when we replicate 32 GB (3% of the total size) on one additional server. Since a server can hold 128 GB in total, we put four copies of the replica on this server.

Comparing the baseline in Figure 7.1(a) where 1 terabyte of text is distributed over 32 servers and it supports an average response time under 10 seconds at 6.7 commands per second, using 8 servers supports an average response time under 10 seconds at 1.7 commands per second, which is 3 times less than the system with 4 times the servers (32 servers). Distracting 20%, 40%, and 60% of commands increases the largest command arrival rate with an average response time below 10 seconds by a factor of 1.3, 2.1, and 3.5, which is similar to the improvement of partial replication for distributing a terabyte over 32 servers (1.3, 2.2, 3.4).

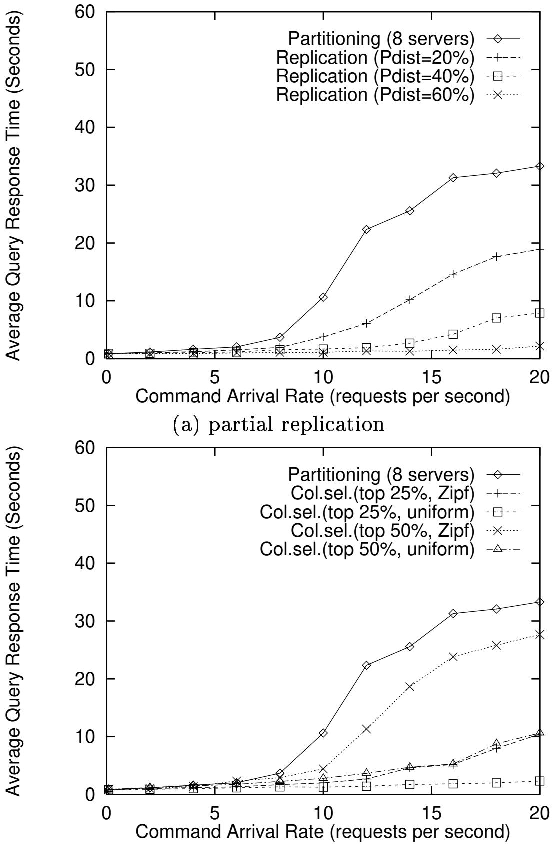
In Figure 7.2(b), we demonstrate the average response time when we select the top 25% and 50% of collections using the pure Zipf and uniform distributions. Selecting the top 25% and 50% of collections using the Zipf distribution improves the largest command arrival rate with an average response time below 10 seconds by a factor of 1.8 and 1.2, which is better than using 32 servers (Figure 7.1(b)), because the number of collections is smaller in this case. Selecting the top 25% and 50% of collections using uniform distribution improves the largest command arrival rate with an average response time below 10 seconds by a factor of 3.9 and 2.0, which are similar to those using 32 servers.

7.4 Performance with Faster Servers and Network

In this section, we present a set of experiments using faster servers and a faster network. We assume we have 8 servers and each server handles 128 GB of data as in Section 7.3. We assume servers and the network are 4 times as fast as those used in Section 7.3. Figure 7.3 illustrates the performance gain by using partial replication and collection selection in this case.

In Figure 7.3(a), we demonstrate the average response time when we replicate 32 GB (3% of the total size) on one additional server. As in Section 7.3, we put four copies of the replica on this server. Comparing with Figure 7.2(a) where the system with 8

λ	R_{cm}	N_{tpq}	N_{qtf}	C_{size}	N_{cpu}	N_{disk}
varied	1:1.5:2	2	Obs.	1 TB	4	8
N_{th}	P_{dist}	P_{repl}	D_{cas}	P_{sel}	N_{col}	S_p
32	varied	3%	varied	varied	8	4



(a) partial replication

(b) collection selection

Figure 7.3. Performance when searching a terabyte of text using faster servers and network

servers supports an average query response time under 10 seconds at 1.7 commands per second, using a system with 4 time as fast supports an average query response time under 10 seconds at 9.9 commands per second, which is 5.8 times as many as the old system. When using partial replication, distracting 20% of commands increases the largest command arrival rate with an average response time below 10 seconds by a factor of 1.4 and distracting 40% of commands increases the largest command arrival rate with an average response time below 5 seconds by a factor of 2.1, which is similar to the improvement of partial replication for the system in Section 7.3 (1.3

and 2.1). Distracting 60% of commands is insensitive to the command arrival rate that we consider in this experiments.

In Figure 7.3(b), we demonstrate the average response time when we select the top 25% and 50% of collections using a pure Zipf and an uniform distribution. Selecting the top 25% and 50% of collections using the Zipf distribution improves the largest command arrival rate with an average response time below 10 seconds by a factor of 1.9 and 1.2. Selecting the top 50% of collections using uniform distribution improves the largest command arrival rate with an average query response time below 10 seconds by a factor of 2.0 and selecting the top 25% of collections using uniform distribution is insensitive to the command arrival rate that we consider in this experiments.

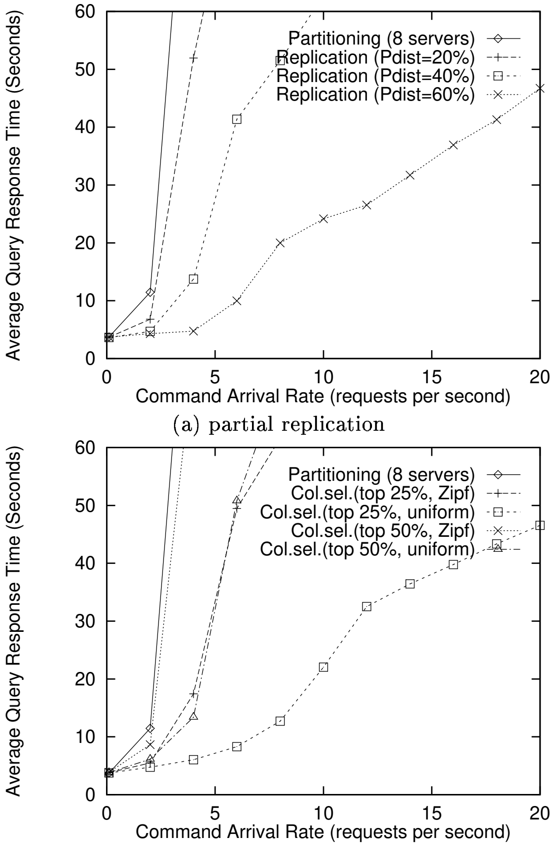
The results of this section show that although using faster machines reduces the response time, it does not significantly change the relative improvement due to collection selection and partial replication.

7.5 Performance with Longer Queries

In this section, we present a set of experiments for searching a terabyte of text using longer queries with an average 8 terms per query. The system configuration of this section is the same as in Section 7.4. Figure 7.4 illustrates the performance gain by using partial replication and collection selection in this case.

In Figure 7.4(a), we demonstrate the average response time when we replicate 32 GB (3% of the total size) on one additional server. As in Section 7.3 and 7.4 we put four copies of the replica on this additional server. Comparing the baseline in Figure 7.3(a) where the system using queries with an average of 2 terms per query supports an average query response time under 10 seconds at 9.9 commands per second, the system using queries with an average of 8 terms per query supports a response time under 10 seconds at 1.7 commands per second, which is 5 times less

λ	R_{cm}	N_{tpq}	N_{qtf}	C_{size}	N_{cpu}	N_{disk}
varied	1:1.5:2	8	Obs.	1 TB	4	8
N_{th}	P_{dist}	P_{repl}	D_{cas}	P_{sel}	N_{col}	S_p
32	varied	3%	varied	varied	8	4



(a) partial replication

(b) collection selection

Figure 7.4. Performance when searching a terabyte of text using an average of 8 terms

than the system using 2 terms. Distracting 20%, 40%, and 60% of commands increases the largest command arrival rate with an average response time below 10 seconds by a factor of 1.3, 2.1, and 3.5.

In Figure 7.4(b), we demonstrate the average response time when we select the top 25% and 50% of collections using Zipf and uniform distributions. Selecting the top 25% and 50% of collections using the Zipf distribution improves the largest command arrival rate with an average response time below 10 seconds by a factor of 1.8 and 1.2. Selecting the top 25% and 50% of collections using uniform distribution improves the largest command arrival rate with an average response time below 10 seconds by a factor of 4.0 and 2.0.

The results of this section show that although the system using longer queries increases the response time, it does not significantly change the relative improvement due to collection selection and partial replication.

7.6 Summary

This section presented the performance of our distributed IR system for searching a terabyte of text. We first justified that a replica with 32 GB is sufficient to store the documents of the top topics that satisfy 40% of queries according to the Excite log. We then presented and compared the performance using partial collection replication and collection selection for different system configurations: 32 GB per server, 128 GB per server with larger disk, using faster servers and networks, and using longer queries. Our results show that using one replica to distract 20% and using 2 replicas to distract 40% and 60% of commands improves the largest command arrival rate under a cutoff for the average query response time by a factor of 1.3, 2.2, and 3.5. Selecting the top 25% of 8 collections improves the largest command arrival rate under a cutoff for the average query response time by a factor of 1.8 when the collection skew follows the Zipf distribution. These results suggest that partial collection replication and collection selection can significantly improve the system performance for searching a terabyte of text. Our results also show that although they affect the response time, none of them change relative improvements due to partial replication and collection selection with uniform access. For collection access with the Zipf-like distribution ($\theta < 1$), the number of collections affects the popularity of the most frequently used collection and thus affects performance.

CHAPTER 8

CONCLUSIONS

The goal of this dissertation was to develop and evaluate distributed information retrieval architectures that attain quick response for rapidly increasing data and workloads while maintaining retrieval accuracy. Our architectures include components that do not appear in previous IR architectures: partial collection replication and selection, and collection selection. We considered both performance and accuracy (efficiency and effectiveness) of IR systems and how to make best use of available resources. In this final chapter, we summarize the results of each chapter, review the research contributions of this dissertation, and discuss future research directions.

8.1 Summaries

In this dissertation, we first investigated how to use partial collection replication for information retrieval, and then presented and evaluated distributed architectures that incorporate parallel information retrieval, partial collection replication and selection, and collection selection. We evaluated the performance improvements due to each of these three components one by one. We also compared the performance of partial collection replication with collection partitioning, as well as compared partial collection replication with collection selection. We used inference networks to select relevant replicas and collections in order to maintain retrieval effectiveness.

In Chapter 4, we investigated using partial collection replication for information retrieval. Our analyses on actual system logs demonstrated that there is sufficient query locality to justify partial collection replication. Our evidence also indicated that a searchable replica achieves better performance than caching queries, because the replica selection algorithm finds similarity between non-identical queries, and thus increases the observed locality. We presented a method for constructing a hierarchy of partial replicas from a collection where each replica is a subset of all larger replicas. We extended the inference network model to rank and select partial replicas. We proposed a new replica selection function and showed that it works better for replica selection than other ranking functions. We examined time and space overheads to build and update replicas and the replica selection database. Although updating large replicas is costly, fortunately, the statistics from the real system logs show that updating hourly or daily is unnecessary that makes the update affordable. We also proposed two simple strategies for updating replicas.

To expedite our investigation of possible system configurations, characteristics of IR collections, and the basic IR system performance, we implemented a simulator with numerous system parameters. We presented and validated the simulator against our prototype implementation in Chapter 5.

In Chapter 6, we first explored how to build high performance IR servers using symmetric multiprocessors. We investigated how to balance the hardware and software resources with respect to number of threads, CPUs, and disks as the collection size increases. We investigated the factors that affect the necessary number of threads. We showed that adding disks can degrade the performance when the CPU is a bottleneck, and adding CPUs does not improve performance when the disk is a bottleneck. Increasing the collection size does not significantly change the hardware balance point, and the server performance is more related to the balance of hardware components than the collection size. We presented several formulas that use simple measurements to estimate the necessary number of threads, the number of CPUs and disks that constitute a well-balanced system. We also compared the performance of partitioning and replicating data over additional disks.

We then investigated the performance of partial replication in a distributed IR system. We showed the query locality determines the improvement due to partial collection replication, load balancing is necessary when the replicas distract too many commands. We also showed that partial collection replication with load balancing consistently performs better than collection partitioning, even with fewer hardware resources, and requires modest query locality. A hierarchy of replicas further increases performance for large command rates and high distracting percentages.

We also investigated the performance of collection selection in a distributed IR system. We showed the performance of collection selection is determined by the popularity of the most frequently used collection and the number of selected collections. We also compared the performance of collection selection with partial collection replication. Our results showed that either of these two techniques can performs better than the other under some circumstances. Partial collection replication is better than collection selection when a system has high query locality. Collection selection performs better than partial collection replication when the collection access is fairly uniform after collection selection, but because of locality, this result is unlikely to be realized in practice.

In Chapter 7, We presented experiments for searching a terabyte of text using our technologies. We presented and compared the performance using partial replication and collection selection for different system configurations: 32 GB per server, 128 GB per server with larger disk, using faster servers and networks, and using longer queries. Our results show that using one replica to distract 20% and using 2 replicas to distract 40% and 60% of commands improve the largest command arrival rate under a cutoff for the average response time by a factor of 1.3, 2.2, and 3.5, respectively. Selecting the top 25% of 8 collections improves the largest command arrival rate under a cutoff for the average response time by a factor of 1.8 when the collection skew follows the Zipf distribution. These results suggest that partial collection replication and collection selection can significantly improve the system performance for searching a terabyte of text. Our results also showed that although using fewer and faster servers and issuing longer queries affect the absolute response time, none of them change relative improvements due to partial replication and collection selection with uniform access. When the collection access follows the Zipf-like distribution ($\theta < 1$) after collection selection, reducing the number of collection achieves better performance.

8.2 Contributions

This dissertation presents a significant step towards providing fast and effective distributed information retrieval in large-scale information retrieval systems. We presented and evaluated distributed architectures that attain quick response for rapidly increasing data and workloads while maintaining retrieval accuracy. Our architectures include components that do not appear in previous IR architectures: partial collection replication and selection, and collection selection. We considered both performance and accuracy (efficiency and effectiveness) of IR systems and how to make best use of available resources. The specific contributions of this dissertation are:

- First work on partial collection replication and selection in information retrieval:
  - justifying the usefulness of partial replication for IR based on traces;
  - developing a replication architecture;
  - developing an effective replica selection function and demonstrating its performance;
  - estimating updating costs for replicas and replica selection database;
  - proposing updating strategies.
- Scalable distributed architectures and an evaluation of their performance:
  - implementing and validating a simulator for distributed IR systems;

  - performance evaluation of parallel servers using symmetric multiprocessors;
  - performance evaluation of partial replication;
  - comparison of partial collection replication and collection partitioning;
  - performance evaluation of collection selection;
  - comparison of collection selection and partial collection replication;
  - mechanisms for searching a terabyte of text.

8.3 Future Work

Although we have largely achieved the research goal set for this dissertation, there are a number of directions in which the work in this dissertation can pursue further:

- Replica Selection.

  Although our current replica selection function works fairly well (approximately 10% precision percentage loss for the top 30 documents), a better selection function may exist that reduces the precision loss. In addition, when we tested our selection function, we built replicas using the top documents of each query resulting from searching all collections. How well our function works when we only use the results from searching a subset of collections (due to collection selection) needs to be further tested. We also need to test our current selection function using larger collections and more queries.

- Collection Selection.

  We need to develop a better collection ranking function that can produce comparable retrieval accuracy when searching a small percentage of collections, since the current ranking function only works well for searching the top 50% of collections. Although query expansion can reduce the precision loss to around

10% when searching the top 10% of collection [77], we prefer to develop a better function instead of query expansion, because query expansion needs additional computation that easily cancels out the performance gain due to searching less data, as compared with using original queries.

- Unifying Replica Selection and Collection Selection.

  In our work, we assume replica selection and collection selection are two distinct processes. It will be interesting to see whether our replica selection works well in collection selection. If this function works well, we may combine these two processes into one.

- Caching.

  During our performance evaluation, we do not consider caching. It will be interesting to investigate how caching further improves the system performance, and to compare caching to building partial replicas.

- Trace Analyses.

  In our work, the traces we obtained only reflect two specific IR applications. We need to obtain more traces to analyze the user access patterns in different applications, and obtain the collection access patterns in a multiple collection searching environment.

- Heterogeneous Environments.

  The work demonstrated in this dissertation used a homogeneous environment. We assumed we used the same machines with the same number of CPUs and disks, and used a local area network. In an environment with heterogeneous computers and/or using a wide area network, many new issues will occur, for example, how to offset the speed difference of each server in order to produce high performance since the best performance of multiple collection searching is achieved when all servers can produce response at the same time. In a wide area network such as the Internet, how to route messages to avoid link congestions is more critical than in a local area network. Therefore how to adapt our technologies in that environment needs to be studied.

- Uncontrollable Environments.

  All our work in this dissertation assumed we work in a controllable environment, i.e., we are able to access all indices to build a replica selection database, and collection selection database. There are environments where documents within a system is protected, and we we can only guess their contents by sending probing queries. How to achieve high retrieval accuracy using partial replication and collection selection in this kind of environment needs to be studied.

- Multimedia Environments.

  In our work, we only investigated and evaluated the architectures for searching text. How to provide fast and effective retrieval in a multimedia environment needs to be studied.

The most important problem for IR systems is to attain quick response for rapidly increasing data and workloads while maintaining retrieval accuracy, regardless of computing environments that will change with time. This dissertation attacked on this key problem and made significant contributions.

APPENDIX A

TREC COLLECTIONS

The TREC collections are large test collections distributed by National Institute of Standards and Technology (NIST) for testing and comparing the current text retrieval techniques [36, 37, 38, 39, 40]. In this thesis, we use TREC collections as our testbed.

A.1 Data Sources

TREC collections come from 18 different data sources, such as news, patents, and the Web. We list all the data sources in this section.

AAG

Legal information made available by the Australian Attorney-General's Department. It includes legislation and judgments.

ADIR

The FATEXT database of industrial relations documents supplied by the Australian Department of Industrial Relations.

AP

Associate Press Newswire, 1988, 1989, 1990.

APLT

A virtually complete collection of proceedings within the Australian Federal Parliament (1970 - 1995) including Hansard reports of both chambers plus notice papers, committee reports etc.

AUNI

Downloads of the websites operated by ten Australian Universities: Australian National University, Victorian University of Technology, Latrobe University, Murdoch University, Ballarat University, Edith Cowan University, University of Newcastle, Charles Sturt University, University of Tasmania, Adelaide University, and the Uniserve academic clearing house.

CR

Congressional Record, the proceedings of the legislative branch of the U.S Goverment (1993).

DOE

Short Abstracts from DOE publications.

FBIS

Selected non-U.S. broadcast and print publication from Foreign Broadcast Information Service.

FR

Federal Register, the official record of the executive branch of the U.S Government (1989, 1988, 1994).

FT

Financial Times (1988 - 1990, 1991-1994).

GH

Glasgow Herald, Glasgow, Scotland. (1995 - 1997).

LATIMES

Articles that appeared in the newspaper *LA Times* in 1989 and 1990.

NEWSxy

The NEWS collections consist of USENET NEWS postings collected by the University of Waterloo. The items have been filtered to remove some or all encoded binaries and image files.

PATENTS

U.S patents (1993).

PGUT

Project Gutenberg collection of electronic transcriptions of out-of-copyright books. Obtained from a Project Gutenberg FTP site.

SJM

San Jose Mercury News (1991).

WEB01

Downloads of websites operated by: Australian Broadcasting Commission, Australian National Library, Australian Department of Defence, Australian Computer Society, Federal Parliament of Canada, Commonwealth Scientific and Industrial Research Organisation.

WSJ

Wall Street Journal (1987 - 1989, 1990 - 1992).

ZIFF

Articles from *Computer Selects* (Ziff-Davis Publishing).

A.2 Statistics of TREC Collections

	Name	Size (Megabytes)	Num. Documents
TREC 1	AP '89	254	84,678
	DOE	184	226,087
	FR '89	260	25,960
	WSJ '87-89	267	98,732
	Ziff 1	242	75,180
TREC 2	AP '88	237	79,919
	FR '88	209	19,860
	WSJ '90-92	242	74,520
	Ziff 2	175	56,920
TREC 3	AP '90	237	78,321
	PATENTS	243	6,711
	SJM '91	287	90,257
	Ziff 3	345	161,021
TREC 4	CR	235	27,922
	FR94	395	55,630
	FT	564	210,158
TREC 5	FBIS	470	130,471
	LATIMES	475	131,896
TREC 6 VLC	AAG	1875	561,566
	ADIR	775	42,841
	APLT	1540	421,681
	AUNI	725	81,334
	GH	394	135,477
	FT	527	202,433
	NEWS01	955	446,106
	NEWS02	943	450,027
	NEWS03	934	482.395
	NEWS04	966	483,145
	NEWS05	1170	590,202
	NEWS06	1121	571,891
	NEWS07	1080	520,282
	NEWS08	1728	856,609
	PGUT	431	3,303
	WEB01	142	8,513
	plus all TREC 1-5 collections		

Table A.1. The TREC Collections

APPENDIX B
ACCESS LOG ANALYSIS

This appendix presents analyses of two server logs we obtained from THOMAS, a legislative information service of the U.S. Congress through its Library [52], and Excite, a Web search engine [27].

B.1 The THOMAS Log

The THOMAS system is a legislative information service of the U.S. Congress through the Library of Congress, which uses InQuery as the retrieval engine. It contains the full text of bills introduced in the Congresses from 101st to 105th as well as the text of the Congressional Records for those Congresses.

B.1.1 Query Locality

We examine the query locality by analyzing the log of THOMAS between July 14 and September 13, 1998. We collect the total number of queries, number of unique queries, number of topics, and overlap of topics between days, and weeks. Since the log does not contain document identifiers returned from each query evaluation, we built a test database using the Congress Record for 103rd Congress (235 MB, 27992 documents), which is a part of the TREC 4 collection [39], and also a part of collection THOMAS uses. We reran all queries against this test database, and view distinct queries be as the same if their top 20 documents completely overlap, i.e., we group these queries and call them a topic. Table B.1 to B.5 summarize the statistics.

Table B.1 lists the number of queries, number of unique queries, number of topics, number of topics occurring once, number of topics occurring more than once, and number of topics that contain more than one query on a day by day base. The number of queries or unique queries with results represents the queries that actually match documents from our test database. We mark the days that we only got a part of the log with a $*$. The statistics show that around 29% of topics occur more than once, and account for 63% of total queries. Among the topics that occur more than once, around 35% contains more than one unique query, which indicates caching queries which requires an exact match would not detect this locality.

Table B.2 presents how many queries account for the daily top 100, 200, 500, 1000, and 2000 topics. On a typical day, the top 100 topics (top 2%) account for around 17% of total queries; the top 200 topics (top 4%) account for around 25%; the top 500 topics (top 10%) account for around 40%; the top 1000 topics (top 20%) account for 54%, and the top 2000 topics (top 25%) account for 70%. We notice that the data from September 10 to September 13 presents significantly different characteristics from the other days. These were days when the Starr report was published on the Internet. The top queries on these days are some variations of starr, starr report, clinton, Monica lewisky, and impeachment. The 100 topics account as high as 73.4% of queries on the days.

Table B.3 presents the percentages of queries whose topics match a topic or one of the top 100, 200, 500, or 1000 topics on the previous day or on July 14, 1998 respectively. The left part of the table lists the statistics on overlap between days. The right part of the table lists the statistics on overlap with July 14. Typically, when we replicate all topics of the previous day, around 43% of queries match a topic in the replica; when we replicate the top 1000 topics of the previous day, around 28% of queries match a topic in the replica. However, on the days the Starr report came out, as high as 85% of queries match a topic on the previous day. If we only replicate the topics on July 14, we find fewer queries match a topic in the replica as time elapses. For the top 1000 topics, the percentage of queries that matches a topic decreases from 30%

date	Number of							
	queries		unique queries		topics			
		with		with		occurring	more than	more than one
	total	results	total	results	total	once	once	unique query
7/14	9969	9619	6226	5973	5182	3620	1562	564
7/15	9656	9281	6067	5805	5036	3544	1492	545
7/16	9730	9338	6027	5776	5007	3491	1516	541
7/17	8429	8071	5399	5158	4469	3172	1297	510
*7/18	2104	1985	1479	1396	1285	969	316	93
*7/19	2128	2036	1555	1486	1385	1076	309	84
7/20	10113	9715	6324	6055	5195	3615	1580	609
7/21	9905	9573	6205	5985	5176	3667	1509	556
7/22	10650	10271	6497	6236	5409	3756	1653	585
7/23	10527	10168	6388	6149	5333	3695	1638	596
7/24	8461	8131	5177	4944	4360	3055	1305	416
*7/25	2095	1986	1479	1405	1288	973	315	94
*7/26	2218	2095	1554	1458	1330	969	361	103
*7/27	1705	1664	1308	1274	1156	878	278	98
*7/28	1542	1498	1140	1109	1001	754	347	87
*7/29	1529	1488	1163	1133	1019	774	245	91
*7/30	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7/31	7850	7527	5053	4830	4183	2919	1264	473
*8/01	1996	1865	1460	1369	1250	914	336	95
*8/02	211	194	160	148	141	106	35	7
*8/03	1288	1247	1026	991	902	704	198	71
8/04	8826	8514	5549	5332	4636	3245	1391	502
8/05	9376	9041	5955	5707	4902	3387	1515	566
8/06	9445	9110	5965	5713	4981	3498	1483	523
8/07	7049	6801	4642	4440	3916	2815	1101	375
*8/08	1687	1610	1213	1154	1059	799	260	75
*8/09	1740	1641	1235	1162	1072	785	287	68
8/10	7577	7334	4951	4766	4193	2983	1210	432
8/11	7811	7542	5110	4920	4328	3031	1297	440
8/12	7428	7117	4877	4674	4155	3027	1128	394
8/13	7084	6796	4722	4529	3989	2812	1177	415
8/14	5160	4934	3542	3389	3024	2189	835	269
*8/15	1450	1329	1052	997	926	725	201	60
*8/16	1458	1361	1101	1029	967	741	226	53
8/17	6129	5881	4110	3938	3510	2513	997	326
8/18	6673	6365	4480	4262	3813	2740	1073	350
8/19	6201	5974	4161	3993	3592	2563	1029	310
8/20	5915	5656	3953	3772	3409	2455	954	301
8/21	4844	4632	3417	3265	2983	2206	777	238
*8/22	1311	1229	980	916	859	652	207	52
*8/23	1445	1357	1093	1033	952	716	236	66
8/24	5823	5569	4012	3823	3394	2414	980	339
8/25	6168	5913	4148	3974	3568	2576	992	323
8/26	6164	5921	4213	4032	3611	2585	1026	324
8/27	6201	5945	4182	4000	3568	2557	1011	346
8/28	5126	4913	3569	3416	3120	2303	817	250
*8/29	1661	1585	1188	1128	1044	777	267	73
*8/30	2028	1921	1459	1389	1281	949	332	93
8/31	6309	6040	4254	4059	3635	2612	1023	339
9/01	6927	6639	4538	4327	3863	2781	1082	347
9/02	7422	7128	4759	4568	4047	2912	1135	388
9/03	7230	6919	4731	4524	4029	2896	1133	393
9/04	6838	5584	3947	3761	3320	2377	943	330
*9/05	1831	1765	1340	1262	1174	856	318	74
*9/06	1711	1614	1251	1179	1096	836	260	73
*9/07	2768	2621	1886	1783	1650	1197	453	119
9/08	8508	8103	5398	5126	4496	3164	1332	455
9/09	8800	8430	5583	5368	4717	3321	1396	485
9/10	14540	13695	6623	6181	5251	3694	1557	580
9/11	15657	13487	4546	4111	3300	2396	904	338
9/12	12114	10220	3017	2696	2020	1466	554	217
9/13	7095	6236	2719	2473	2049	1486	563	197

Table B.1. The statistics from the THOMAS log (I)

date	percentages of queries that top topics account for					date	percentages of queries that top topics account for				
	100	200	500	1000	2000		100	200	500	1000	2000
7/14	16.4%	23.9%	37.5%	50.7%	66.9%	8/14	18.6%	26.8%	42.1%	59.0%	79.2%
7/15	17.5%	25.1%	38.1%	51.2%	67.3%	*8/15	30.2%	45.3%	67.9%	100.0%	100.0%
7/16	17.2%	24.8%	38.5%	51.5%	67.8%	*8/16	27.0%	41.7%	65.7%	100.0%	100.0%
7/17	18.2%	25.8%	39.5%	53.3%	69.4%	8/17	17.2%	25.1%	40.4%	57.3%	74.3%
*7/18	27.4%	39.5%	60.5%	85.6%	100.0%	8/18	15.8%	23.8%	38.9%	54.7%	71.5%
*7/19	25.3%	36.4%	56.5%	81.1%	100.0%	8/19	17.0%	24.5%	39.4%	56.1%	73.4%
7/20	16.4%	24.2%	37.8%	50.8%	67.1%	8/20	17.5%	25.6%	40.5%	57.4%	75.1%
7/21	16.5%	24.3%	37.8%	51.1%	66.8%	8/21	16.4%	24.6%	40.4%	57.2%	78.8%
7/22	17.8%	25.5%	38.2%	50.8%	66.8%	*8/22	29.5%	45.8%	70.8%	100.0%	100.0%
7/23	18.2%	25.4%	38.3%	51.1%	67.2%	*8/23	27.2%	41.9%	66.7%	100.0%	100.0%
7/24	16.8%	23.5%	35.8%	48.6%	63.8%	8/24	16.2%	24.2%	39.4%	57.0%	75.0%
*7/25	27.4%	39.4%	60.3%	85.5%	100.0%	8/25	16.4%	24.2%	39.8%	56.6%	73.5%
*7/26	26.4%	38.4%	60.4%	84.2%	100.0%	8/26	15.1%	23.2%	38.6%	55.5%	72.8%
*7/27	25.7%	37.9%	60.6%	90.6%	100.0%	8/27	16.9%	24.6%	39.8%	56.6%	73.6%
*7/28	30.0%	43.4%	66.6%	999.9%	100.0%	8/28	15.9%	24.4%	40.2%	56.8%	77.2%
*7/29	28.5%	41.9%	65.1%	98.7%	100.0%	* 8/29	29.7%	42.5%	65.7%	97.2%	100.0%
*7/30	n/a	n/a	n/a	n/a	n/a	*8/30	25.0%	36.9%	59.3%	85.4%	100.0%
7/31	17.3%	25.3%	39.7%	54.2%	71.0%	8/31	15.8%	23.9%	39.4%	56.0%	72.9%
*8/01	24.0%	36.4%	59.8%	86.6%	100.0%	9/01	16.3%	24.6%	40.1%	55.6%	71.9%
*8/02	78.9%	100.0%	100.0%	100.0%	100.0%	9/02	18.1%	26.3%	40.9%	55.4%	71.3%
*8/03	27.8%	43.7%	67.8%	100.0%	100.0%	9/03	16.6%	24.6%	39.7%	54.3%	70.7%
8/04	18.3%	25.8%	39.1%	52.7%	69.0%	9/04	17.5%	25.8%	41.6%	58.5%	76.4%
8/05	17.2%	24.6%	37.7%	51.1%	67.9%	*9/05	25.2%	38.1%	61.8%	90.1%	100.0%
8/06	17.0%	24.2%	37.7%	51.0%	67.3%	*9/06	27.9%	40.8%	63.1%	94.1%	100.0%
8/07	18.2%	26.1%	40.7%	55.6%	71.8%	*9/07	23.8%	35.0%	56.1%	75.2%	100.0%
*8/08	29.3%	42.9%	65.3%	96.3%	100.0%	9/08	17.0%	24.9%	38.8%	52.8%	69.2%
*8/09	28.0%	41.6%	65.1%	95.6%	100.0%	9/09	16.3%	23.6%	37.2%	51.2%	67.8%
8/10	16.2%	24.2%	38.6%	53.6%	70.1%	9/10	40.4%	46.0%	55.7%	64.9%	76.3%
8/11	16.2%	23.7%	37.8%	51.9%	69.1%	9/11	65.5%	69.5%	76.2%	82.9%	90.4%
8/12	17.4%	25.1%	39.3%	53.8%	69.7%	9/12	73.4%	78.1%	84.6%	90.0%	99.8%
8/13	15.8%	23.3%	38.1%	53.4%	70.3%	9/13	57.5%	63.4%	74.2%	83.2%	99.2%

Table B.2. The statistics from the THOMAS log (II)

to 22.2% within 2 months. Comparing with the left part of the table, the percentage drop is very gradual, which suggests that we do not need to update replicas everyday. However, on the special days when the Starr report came out, if we replicate the top 1000 topics, as high as 82% of queries match a topic on the previous day, which suggests that we need a specific mechanism to handle this kind of bursty event.

Table B.4 presents the percentage of queries whose topics match a topic or one of the top 100, 200, 500, or 1000 topics on the previous seven days or on the week of July 14 to July 20, 1998, respectively. The left part of the table lists the statistics on overlap between the current day with the previous seven days. The right part of the table lists the statistics on overlap with the week of July 14 to July 20. Typically, when we replicate all topics of the previous seven days, around 58% of queries match a topic in the replica; when we replicate the top 1000 topics of the previous day, around 35% of queries match a topic in the replica. However, on the days the Starr report came out, as high as 85% of queries match a topic on the previous day. If we only replicate the topics on the week of July 14 to July 20, fewer queries match a topic in the replica as time elapses.

Table B.5 presents the percentage of queries whose topics match a topic or one of the top 100, 200, 500, or 1000 topics that we accumulate from the very beginning, i.e. we count the frequency of each topic without considering the time. The statistics are very similar to those in Table B.4 except when we replicate all topics, which is not feasible in real situations. The statistics suggest we may only need to count frequencies of topics within a period of time.

B.1.2 Document Access Patterns

In this section, we examine the ratio of query commands and document retrieval commands, and how many documents being viewed fall into the top n documents. Unlike a typical IR system where users can issue more than one summary commands

date	percentages of queries that match a top topic on the previous day					percentages of queries that match a top topic on 7/14				
	all	100	200	500	1000	all	100	200	500	1000
7/14	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7/15	43.3%	13.4%	18.0%	24.8%	30.1%	43.3%	13.4%	18.0%	24.8%	30.1%
7/16	44.4%	12.9%	18.0%	24.4%	30.4%	42.6%	12.7%	17.9%	24.0%	29.3%
7/17	44.0%	14.1%	18.9%	25.5%	30.4%	42.0%	12.9%	17.4%	23.5%	28.7%
*7/18	40.2%	13.7%	17.7%	23.3%	28.2%	41.0%	12.8%	17.9%	23.5%	28.8%
*7/19	25.7%	11.3%	15.3%	19.5%	24.0%	39.4%	13.8%	18.2%	23.6%	28.0%
7/20	22.3%	8.2%	11.1%	15.3%	19.2%	41.3%	12.1%	16.5%	23.1%	28.4%
7/21	43.0%	12.5%	17.9%	24.7%	29.8%	41.6%	12.3%	16.7%	23.6%	28.5%
7/22	44.4%	14.0%	19.9%	27.4%	31.9%	42.5%	12.4%	17.2%	24.0%	29.6%
7/23	45.9%	14.3%	19.7%	27.3%	31.5%	41.4%	12.6%	17.2%	23.4%	28.7%
7/24	45.1%	15.7%	20.3%	27.3%	31.8%	42.5%	12.6%	17.4%	23.4%	28.6%
*7/25	42.5%	15.0%	18.4%	24.5%	30.1%	42.1%	13.0%	17.8%	23.9%	28.1%
*7/26	27.1%	11.7%	14.2%	18.9%	23.5%	43.4%	11.3%	17.3%	25.4%	30.2%
*7/27	22.2%	8.1%	11.2%	16.7%	20.7%	43.9%	10.0%	14.1%	23.0%	28.9%
*7/28	30.6%	10.3%	15.0%	20.0%	26.7%	44.3%	9.5%	13.3%	23.2%	29.2%
*7/29	26.7%	10.4%	12.9%	17.7%	26.7%	40.8%	10.0%	14.4%	21.9%	27.1%
*7/30	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7/31	n/a	n/a	n/a	n/a	n/a	38.5%	11.2%	15.1%	21.9%	26.4%
*8/01	38.0%	11.6%	17.0%	22.6%	26.5%	36.5%	11.0%	15.3%	21.5%	26.1%
*8/02	28.9%	8.76%	14.4%	23.7%	26.8%	43.3%	8.8%	14.4%	20.6%	35.1%
*8/03	2.9%	2.3%	2.9%	2.9%	2.9%	40.3%	8.7%	13.3%	22.1%	27.0%
8/04	19.1%	6.4%	9.4%	13.7%	19.1%	39.8%	11.0%	15.1%	22.7%	27.7%
8/05	41.5%	13.4%	17.3%	23.9%	28.6%	38.4%	10.6%	15.4%	21.6%	26.6%
8/06	41.5%	12.5%	17.8%	24.3%	28.7%	37.8%	10.4%	14.3%	21.3%	26.0%
8/07	43.1%	13.9%	17.9%	24.9%	29.4%	38.6%	10.4%	15.3%	21.9%	26.0%
*8/08	39.4%	14.2%	19.0%	24.8%	29.3%	35.6%	11.6%	16.8%	22.1%	25.9%
*8/09	24.7%	9.9%	13.0%	18.3%	23.5%	41.6%	11.6%	16.3%	23.9%	28.2%
8/10	20.4%	7.2%	9.8%	14.8%	19.6%	37.2%	9.8%	13.9%	20.1%	24.7%
8/11	38.6%	11.3%	15.5%	20.9%	26.0%	37.3%	10.3%	14.6%	20.1%	24.5%
8/12	40.7%	12.1%	16.6%	23.2%	28.3%	38.5%	10.0%	14.4%	21.3%	26.2%
8/13	37.1%	10.9%	15.2%	22.0%	26.8%	36.1%	9.8%	13.8%	20.0%	24.5%
8/14	36.6%	11.7%	15.4%	21.9%	26.1%	38.1%	11.6%	16.1%	21.3%	26.0%
*8/15	35.1%	12.3%	16.0%	21.7%	26.2%	37.9%	12.5%	17.3%	23.0%	26.3%
*8/16	20.0%	6.8%	9.8%	13.8%	20.0%	35.8%	10.0%	14.0%	19.0%	23.4%
8/17	18.3%	7.2%	8.8%	12.3%	18.3%	34.6%	10.0%	14.0%	19.3%	22.9%
8/18	33.4%	9.4%	13.9%	19.0%	24.4%	36.2%	9.7%	13.4%	19.2%	23.7%
8/19	34.6%	10.9%	15.0%	20.0%	24.6%	35.3%	9.8%	13.3%	19.3%	23.3%
8/20	34.8%	10.8%	13.3%	19.0%	24.4%	34.5%	9.6%	12.9%	18.6%	22.7%
8/21	31.2%	8.7%	12.3%	16.7%	21.3%	35.2%	9.4%	13.2%	18.4%	23.3%
*8/22	32.9%	10.1%	14.2%	20.1%	24.8%	35.5%	10.3%	15.3%	20.0%	25.1%
*8/23	20.0%	7.7%	11.0%	15.0%	20.0%	37.1%	10.4%	15.0%	19.1%	24.2%
8/24	16.6%	5.3%	8.3%	11.6%	16.6%	37.2%	10.1%	13.8%	19.2%	24.1%
8/25	33.7%	11.0%	14.6%	19.4%	25.0%	36.7%	10.4%	14.3%	19.8%	24.3%
8/26	35.6%	10.3%	13.9%	19.4%	24.5%	36.9%	9.2%	13.0%	19.0%	24.1%
8/27	36.1%	9.7%	13.5%	20.2%	25.9%	37.4%	9.9%	13.6%	19.7%	25.2%
8/28	32.9%	9.3%	13.5%	19.1%	23.0%	34.3%	9.5%	12.8%	18.3%	23.4%
*8/29	35.2%	10.2%	15.8%	20.9%	26.3%	40.6%	11.9%	16.7%	22.8%	28.4%
*8/30	24.3%	11.4%	14.7%	19.5%	24.3%	37.0%	10.0%	14.6%	20.6%	26.3%
8/31	20.4%	6.9%	9.4%	13.6%	17.5%	37.3%	9.9%	13.9%	19.6%	24.8%
9/01	35.4%	10.8%	14.1%	20.2%	25.7%	37.6%	9.5%	13.6%	19.6%	24.5%
9/02	40.0%	11.6%	15.7%	24.3%	29.4%	39.1%	10.5%	14.6%	21.0%	26.2%
9/03	39.3%	11.4%	16/4%	22.6%	27.4%	38.0%	9.8%	13.6%	20.1%	25.7%
9/04	38.3%	11.4%	15.6%	22.5%	26.4%	36.5%	9.5%	12.9%	18.9%	23.9%
*9/05	36.3%	12.5%	15.4%	21.7%	26.1%	36.2%	10.6%	14.8%	20.5%	24.9%
*9/06	24.0%	11.6%	14.7%	18.6%	22.8%	38.3%	8.8%	14.9%	21.7%	26.9%
*9/07	22.8%	9.6%	12.1%	16.3%	21.3%	36.1%	11.1%	15.1%	21.0%	25.7%
9/08	24.6%	8.2%	11.8%	17.4%	20.2%	36.1%	9.9%	13.9%	19.7%	24.3%
9/09	38.8%	11.3%	15.9%	21.9%	27.0%	36.5%	9.8%	13.4%	18.9%	23.4%
9/10	57.1%	30.8%	43.1%	40.9%	47.0%	38.2%	7.5%	10.4%	14.3%	20.5%
9/11	78.1%	63.1%	65.5%	69.2%	71.7%	44.0%	4.1%	6.2%	8.7%	22.2%
9/12	81.1%	71.5%	73.3%	76.2%	78.1%	47.6%	3.0%	5.2%	6.8%	24.6%
9/13	65.4%	53.0%	56.1%	59.7%	62.0%	44.8%	5.5%	8.6%	11.2%	25.6%

Table B.3. The statistics from the THOMAS log (III)

date	percentages of queries that matches a top topics on the previous seven days					percentages of queries that matches a top topic on the week of 7/14-7/20				
	all	100	200	500	1000	all	100	200	500	1000
7/14	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7/15	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7/16	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7/17	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
*7/18	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
*7/19	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7/20	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7/21	60.2%	13.2%	18.9%	28.2%	33.2%	60.2%	13.2%	18.9%	28.2%	33.2%
7/22	61.6%	13.5%	19.8%	28.2%	34.9%	60.7%	14.6%	21.2%	29.8%	36.8%
7/23	62.6%	14.0%	19.6%	28.8%	34.6%	61.0%	15.2%	21.1%	29.2%	36.8%
7/24	62.0%	15.8%	21.6%	28.8%	34.2%	60.4%	16.4%	22.2%	29.9%	36.7%
*7/25	60.2%	16.6%	21.0%	27.4%	33.3%	58.3%	16.4%	21.8%	28.8%	35.5%
*7/26	63.1%	14.1%	19.9%	29.2%	35.8%	61.0%	13.7%	20.4%	30.5%	38.6%
*7/27	63.0%	12.6%	18.7%	27.4%	34.4%	61.1%	12.6%	19.6%	27.7%	35.9%
*7/28	64.7%	13.4%	20.1%	27.8%	35.4%	61.5%	13.5%	19.4%	26.6%	36.4%
*7/29	55.2%	12.9%	18.5%	25.8%	32.5%	58.0%	12.7%	18.3%	26.9%	35.1%
*7/30	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7/31	49.8%	12.4%	17.6%	24.5%	30.0%	59.1%	13.6%	19.0%	27.3%	34.2%
*8/01	48.5%	11.3%	16.9%	23.4%	28.5%	56.4%	12.0%	17.6%	26.6%	31.7%
*8/02	47.4%	8.2%	12.4%	26.8%	31.4%	62.9%	12.4%	14.9%	27.3%	33.5%
*8/03	48.4%	11.2%	17.3%	26.3%	32.4%	59.1%	11.5%	17.3%	26.1%	33.9%
8/04	47.0%	13.9%	18.8%	26.1%	31.5%	61.3%	14.3%	20.4%	29.1%	36.3%
8/05	50.4%	13.9%	18.7%	26.1%	31.1%	59.4%	13.5%	19.2%	27.6%	35.0%
8/06	52.7%	13.6%	18.9%	26.2%	32.5%	58.7%	13.5%	19.0%	26.7%	34.2%
8/07	55.9%	13.7%	19.2%	27.1%	33.5%	57.9%	13.5%	18.5%	26.6%	34.6%
*8/08	56.3%	15.4%	20.8%	28.2%	31.9%	57.5%	13.3%	20.0%	28.8%	34.3%
*8/09	56.7%	13.5%	19.8%	28.2%	33.8%	57.0%	13.3%	19.7%	31.3%	37.6%
8/10	56.0%	12.7%	17.8%	25.8%	31.9%	58.8%	12.8%	17.8%	26.8%	33.8%
8/11	56.3%	12.8%	17.3%	25.2%	31.5%	59.0%	12.1%	17.6%	25.7%	32.2%
8/12	58.9%	13.7%	19.1%	27.0%	33.3%	59.6%	13.4%	18.3%	27.3%	34.2%
8/13	55.3%	11.7%	16.2%	24.1%	29.6%	57.0%	12.3%	17.0%	24.9%	31.5%
8/14	54.1%	12.7%	18.4%	24.9%	29.8%	54.6%	13.8%	19.3%	26.8%	32.3%
*8/15	56.5%	13.5%	19.9%	26.4%	32.2%	58.4%	14.1%	18.4%	26.6%	34.5%
*8/16	53.1%	11.9%	16.2%	24.8%	30.9%	56.7%	13.4%	17.9%	25.8%	31.4%
8/17	52.9%	12.1%	16.2%	23.3%	29.5%	56.5%	12.0%	17.2%	24.5%	31.5%
8/18	51.7%	11.2%	16.4%	23.4%	28.8%	59.4%	11.9%	16.4%	25.1%	32.0%
8/19	52.1%	11.4%	15.8%	23.1%	28.1%	58.2%	12.6%	16.9%	24.5%	31.2%
8/20	50.3%	11.3%	15.1%	21.7%	27.5%	56.9%	11.3%	15.9%	23.9%	30.2%
8/21	49.1%	10.5%	14.5%	20.5%	26.2%	53.4%	11.0%	15.7%	23.4%	29.1%
*8/22	53.7%	12.4%	17.5%	24.6%	30.7%	56.8%	14.0%	18.8%	27.6%	35.3%
*8/23	53.6%	11.7%	15.0%	21.7%	28.4%	56.7%	11.5%	16.7%	23.2%	30.7%
8/24	52.4%	11.3%	16.2%	23.6%	29.5%	58.1%	11.9%	16.4%	25.5%	32.5%
8/25	51.5%	11.2%	16.3%	23.7%	29.0%	57.9%	12.7%	17.0%	25.2%	32.0%
8/26	51.4%	11.3%	15.5%	22.9%	28.2%	57.6%	11.0%	16.0%	24.2%	31.6%
8/27	51.2%	11.3%	16.1%	22.3%	28.2%	58.0%	11.8%	16.7%	25.0%	32.9%
8/28	49.5%	11.2%	15.6%	22.2%	26.7%	54.0%	11.5%	16.0%	23.5%	30.7%
*8/29	53.9%	14.6%	18.9%	27.7%	32.5%	58.4%	13.9%	18.5%	28.0%	34.7%
*8/30	49.8%	12.8%	17.0%	25.5%	30.8%	57.6%	11.6%	16.6%	26.1%	33.3%
8/31	51.5%	10.9%	15.7%	23.5%	28.5%	57.5%	11.4%	16.8%	25.0%	32.3%
9/01	51.8%	11.3%	16.3%	24.3%	30.0%	60.1%	11.3%	16.5%	26.5%	33.4%
9/02	55.5%	12.2%	17.7%	27.0%	31.1%	60.3%	12.5%	17.3%	27.0%	34.9%
9/03	53.3%	11.5%	16.9%	25.3%	30.4%	58.8%	11.6%	16.7%	25.9%	32.9%
9/04	53.8%	11.0%	16.2%	25.1%	30.3%	55.9%	11.0%	16.3%	25.1%	31.8%
*9/05	53.2%	13.5%	17.5%	25.6%	32.1%	55.8%	12.0%	17.1%	24.5%	31.9%
*9/06	54.2%	13.1%	19.0%	26.3%	32.2%	56.3%	10.7%	15.9%	26.4%	33.1%
*9/07	52.6%	13.4%	18.9%	24.6%	29.6%	71.0%	12.7%	16.5%	25.6%	33.1%
9/08	52.1%	11.8%	17.3%	25.0%	30.5%	72.9%	12.1%	17.0%	26.0%	32.6%
9/09	53.6%	11.9%	16.4%	23.9%	28.9%	75.6%	12.0%	17.0%	24.8%	32.0%
9/10	66.9%	23.3%	34.4%	41.3%	45.8%	81.1%	28.3%	35.1%	41.7%	50.0%
9/11	82.6%	60.3%	63.9%	68.6%	71.3%	82.8%	33.5%	47.8%	54.5%	64.5%
9/12	86.6%	68.8%	71.6%	75.8%	77.8%	84.3%	28.0%	47.1%	57.0%	67.9%
9/13	77.5%	53.4%	57.1%	61.6%	64.9%	91.1%	23.2%	38.2%	48.7%	58.9%

Table B.4. The statistics from the THOMAS log (VI)

date	percentages of queries that matches a top topic on all previous days				
	all	100	200	500	1000
7/14	n/a	n/a	n/a	n/a	n/a
7/15	43.3%	13.4%	18.0%	24.8%	30.1%
7/16	52.5%	12.8%	18.9%	26.2%	31.6%
7/17	55.1%	14.1%	19.7%	27.5%	33.6%
*7/18	56.9%	13.9%	19.3%	27.2%	34.4%
*7/19	54.5%	14.1%	19.3%	26.3%	31.2%
7/20	57.8%	13.1%	18.4%	26.0%	32.3%
7/21	60.2%	13.2%	18.9%	28.0%	33.2%
7/22	62.8%	13.5%	19.4%	28.7%	35.0%
7/23	64.7%	14.2%	19.3%	28.7%	34.8%
7/24	65.2%	15.4%	20.9%	28.7%	34.4%
*7/25	63.9%	14.6%	20.5%	28.2%	33.6%
*7/26	67.1%	12.4%	19.8%	29.9%	35.5%
*7/27	67.8%	12.1%	17.7%	27.8%	34.9%
*7/28	70.6%	11.1%	17.5%	28.2%	35.8%
*7/29	68.5%	11.3%	17.5%	27.2%	33.5%
*7/30	n/a	n/a	n/a	n/a	n/a
7/31	65.1%	12.5%	18.3%	25.5%	31.6%
*8/01	61.2%	12.0%	16.6%	25.4%	31.7%
*8/02	69.1%	13.4%	18.6%	27.8%	40.7%
*8/03	69.3%	10.0%	15.9%	27.1%	33.2%
8/04	68.1%	12.5%	18.9%	27.7%	34.9%
8/05	67.5%	13.2%	19.1%	27.2%	32.8%
8/06	67.6%	12.4%	18.7%	26.5%	32.5%
8/07	68.8%	12.2%	18.4%	26.7%	33.2%
*8/08	69.1%	13.7%	19.1%	27.5%	33.2%
*8/09	67.2%	13.0%	20.0%	29.4%	35.4%
8/10	68.9%	11.4%	17.3%	25.3%	31.6%
8/11	69.1%	12.1%	17.6%	25.6%	31.9%
8/12	70.5%	12.6%	18.1%	27.0%	33.8%
8/13	68.7%	12.1%	17.0%	25.2%	30.9%
8/14	67.2%	14.0%	19.2%	25.9%	31.4%
*8/15	69.7%	13.8%	18.7%	26.1%	32.6%
*8/16	68.7%	12.7%	17.0%	22.9%	30.1%
8/17	68.8%	11.4%	16.2%	23.7%	29.8%
8/18	70.6%	11.0%	16.3%	24.2%	30.7%
8/19	70.6%	11.4%	16.7%	23.9%	29.8%
8/20	70.9%	10.5%	15.5%	23.3%	28.9%
8/21	67.9%	10.7%	15.6%	23.0%	28.6%
*8/22	68.9%	11.7%	18.9%	25.4%	32.1%
*8/23	71.1%	11.1%	16.7%	23.0%	29.6%
8/24	71.1%	11.4%	16.5%	24.6%	30.6%
8/25	69.7%	11.6%	16.8%	24.8%	30.7%
8/26	70.6%	10.4%	15.8%	23.4%	29.7%
8/27	70.5%	11.1%	16.4%	24.3%	31.1%
8/28	69.2%	10.7%	16.0%	23.4%	29.2%
*8/29	70.1%	12.4%	18.8%	26.4%	33.6%
*8/30	71.0%	11.0%	16.7%	25.5%	31.4%
8/31	72.1%	10.8%	17.0%	24.5%	31.0%
9/01	73.3%	10.6%	16.3%	25.6%	32.1%
9/02	73.6%	11.9%	17.4%	26.6%	34.1%
9/03	72.8%	10.6%	16.6%	25.7%	31.5%
9/04	71.0%	10.7%	15.8%	24.4%	30.6%
*9/05	69.6%	11.7%	18.0%	24.7%	29.8%
*9/06	73.6%	11.0%	17.5%	25.4%	31.5%
*9/07	79.1%	11.7%	17.3%	25.6%	31.4%
9/08	82.0%	11.4%	17.0%	24.5%	30.8%
9/09	83.8%	10.7%	15.9%	24.0%	30.1%
9/10	87.3%	8.2%	15.9%	24.9%	44.6%
9/11	89.2%	46.4%	57.7%	63.3%	69.5%
9/12	91.0%	62.2%	65.7%	71.3%	75.0%
9/13	95.4%	48.9%	54.0%	59.3%	63.9%

Table B.5. The statistics from the THOMAS log (V)

for a query, i.e., the system returns the summary information of the top n documents based on the user's requests, the THOMAS system returns all summary information of the top n documents at a time, i.e., it does not provide explicit summary commands for users to use. Table B.6 summarizes the statistics. Column 1 lists the date. Column 2 lists the number of queries. Column 3 lists the number of document being accessed. Column 4 lists the ratio of document to query command. Columns 5 through 14 list the number of document commands that access a document that is ranked as 1-10, 11-20, 21-30, 31-40, 41-50, 51-60, 61-80, 80-100, 101-200, and larger than 200.

On the average for each query command, the user views 1.9 documents. 97.7% of the documents the users viewed are ranked as the top 30. Only 0.6% of documents the users viewed are ranked higher than 200. On a typical day, for each query, the user views around 2 documents. But on the days when the Starr report came out, for each query, the user viewed as few as 0.4 documents, i.e., the users did not find any interesting document for many queries. The reason is that many users entered into the system just to read the Starr report, however, at that time, the Starr report had not input to the system. Actually, several days later, the THOMAS system became aware of this problem, and set up another link to lead the users who just want to read the Starr report to another page instead of searching the whole database.

B.2 The Excite Log

The Excite log we obtained contains one day log information on September 16, 1997. We used the same methodology as we used in analyzing the THOMAS log. We built a test database to automatically disseminate queries by using downloads of the websites operated by ten Australian Universities (725 MB, 81334 documents), which is a part of the TREC 5 collection [40].

Table B.7 lists the statistics for the Excite log: the numbers of queries, unique queries, topics, topics occurring once, topics occurring more than once, and topics that contain

date	Num. of qry	Num. of doc.	doc : qry	percentage of documents falling within ranks									
				1-10	11-20	21-30	31-40	41-50	51-60	61-80	81-100	101-200	> 200
7/14	9969	18046	1.8	78.1%	8.9%	4.6%	3.3%	3.4%	0.3%	0.6%	0.6%	0.2%	0.1%
7/15	9656	17867	1.9	77.5%	9.7%	5.3%	3.2%	3.1%	0.2%	0.2%	0.2%	0.4%	0.1%
7/16	9730	17661	1.8	78.2%	9.2%	4.8%	3.4%	3.2%	0.3%	0.2%	0.2%	0.3%	0.2%
7/17	8429	14830	1.8	79.1%	9.4%	4.6%	3.0%	3.0%	0.2%	0.3%	0.3%	0.1%	0.1%
*7/18	2104	3938	1.9	72.7%	11.4%	5.7%	4.3%	4.8%	0.0%	0.3%	0.3%	0.4%	0.3%
*7/19	2128	4410	2.1	68.7%	13.9%	8.2%	4.6%	4.1%	0.0%	0.1%	0.1%	0.3%	0.1%
7/20	10113	19162	1.9	78.3%	9.6%	4.8%	3.4%	3.1%	0.2%	0.2%	0.1%	0.2%	0.1%
7/21	9905	18380	1.9	77.7%	9.3%	4.9%	3.0%	2.9%	0.2%	0.2%	0.2%	0.2%	1.5%
7/22	10650	19600	1.8	77.2%	9.6%	5.5%	3.3%	3.3%	0.2%	0.4%	0.3%	0.2%	0.1%
7/23	10527	20805	2.0	77.3%	9.1%	4.9%	3.3%	3.3%	0.3%	0.3%	0.3%	0.2%	1.1%
7/24	8461	15372	1.8	77.8%	9.5%	4.7%	3.2%	3.0%	0.3%	0.4%	0.4%	0.6%	0.2%
*7/25	2095	4081	2.0	75.1%	9.6%	5.4%	4.6%	4.1%	0.1%	0.2%	0.3%	0.3%	0.3%
*7/26	2218	4319	2.0	75.7%	9.7%	5.3%	4.3%	4.2%	0.2%	0.2%	0.2%	0.3%	0.1%
*7/27	1705	3108	1.8	77.9%	9.1%	3.9%	2.0%	1.7%	1.3%	1.9%	1.0%	0.3%	0.9%
*7/28	1542	2252	1.5	80.0%	9.0%	3.8%	1.6%	1.2%	0.8%	1.0%	1.5%	1.0%	0.1%
*7/29	1529	2876	1.9	79.3%	7.7%	4.9%	2.1%	1.4%	0.6%	0.9%	0.9%	0.2%	2.0%
*7/30	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
7/31	7850	16601	2.1	81.6%	7.8%	3.8%	2.6%	3.2%	0.2%	0.3%	0.2%	0.1%	0.1%
*8/01	1996	4302	2.2	75.6%	9.1%	5.1%	4.0%	4.2%	0.2%	0.4%	0.2%	0.5%	0.7%
*8/02	211	485	2.3	66.8%	12.0%	8.3%	6.0%	5.8%	0.0%	0.0%	0.4%	0.0%	0.0%
*8/03	1288	2250	1.8	84.5%	6.0%	4.0%	1.6%	1.5%	1.0%	1.0%	0.4%	0.0%	0.0%
8/04	8826	18634	2.1	80.8%	7.8%	4.1%	3.2%	3.3%	0.1%	0.3%	0.2%	0.1%	0.1%
8/05	9376	19987	2.1	80.2%	8.4%	4.4%	3.2%	2.9%	0.2%	0.2%	0.3%	0.2%	0.2%
8/06	9445	19321	2.1	79.2%	9.3%	4.7%	3.1%	3.0%	0.2%	0.2%	0.2%	0.1%	0.1%
8/07	7049	14605	2.1	79.1%	8.5%	4.8%	3.2%	3.1%	0.3%	0.5%	0.3%	0.1%	0.2%
*8/08	1687	3275	1.9	73.2%	11.0%	5.6%	3.9%	4.5%	0.4%	0.4%	0.1%	0.4%	0.6%
*8/09	1740	4135	2.4	63.9%	9.4%	6.0%	4.2%	5.1%	1.0%	2.1%	1.8%	4.1%	2.4%
8/10	7577	15558	2.1	78.3%	8.9%	4.5%	3.2%	3.1%	0.2%	0.4%	0.4%	0.7%	0.3%
8/11	7811	16317	2.1	77.4%	9.2%	5.0%	3.1%	3.2%	0.2%	0.5%	0.4%	0.7%	0.2%
8/12	7428	15195	2.1	77.4%	9.1%	5.0%	3.6%	3.1%	0.3%	0.5%	0.3%	0.7%	0.1%
8/13	7084	14259	2.0	74.4%	8.9%	5.2%	3.6%	3.5%	0.7%	1.0%	0.8%	0.8%	1.2%
8/14	5160	10569	2.1	75.6%	8.6%	4.8%	3.8%	3.5%	0.5%	1.0%	0.6%	0.6%	0.9%
*8/15	1450	2879	2.0	73.2%	10.3%	5.7%	4.9%	4.6%	0.2%	0.5%	0.1%	0.4%	0.1%
*8/16	1458	2728	2.0	73.5%	11.0%	6.2%	4.4%	3.9%	0.3%	0.2%	0.1%	0.3%	0.0%
8/17	6129	12113	2.0	77.7%	8.7%	5.3%	3.7%	3.1%	0.2%	0.2%	0.2%	0.9%	0.1%
8/18	6673	12561	1.9	79.1%	8.4%	4.4%	3.1%	2.9%	0.3%	0.5%	0.3%	0.5%	0.6%
8/19	6201	12576	2.0	76.4%	7.9%	4.6%	3.5%	3.4%	0.2%	0.5%	0.6%	1.6%	1.5%
8/20	5915	10980	1.9	77.0%	9.5%	4.7%	3.5%	3.3%	0.3%	0.4%	0.4%	0.7%	0.3%
8/21	4844	9775	2.0	76.7%	9.7%	5.1%	3.8%	3.6%	0.2%	0.2%	0.1%	0.1%	0.7%
*8/22	1311	2480	1.9	75.0%	10.4%	5.6%	3.7%	3.9%	0.1%	0.0%	0.3%	0.0%	1.1%
*8/23	1445	2610	1.8	70.5%	12.1%	6.1%	4.8%	5.4%	0.2%	0.1%	0.2%	0.5%	0.3%
8/24	5823	11146	1.9	75.2%	9.2%	5.2%	3.8%	3.5%	0.4%	0.7%	0.5%	0.6%	1.1%
8/25	6168	11915	1.9	75.3%	9.2%	5.7%	3.6%	3.7%	0.5%	0.6%	0.2%	0.3%	0.9%
8/26	6164	12269	2.0	76.9%	8.7%	4.8%	3.2%	3.5%	0.4%	0.5%	0.5%	1.0%	0.6%
8/27	6201	12312	2.0	73.5%	9.5%	5.7%	4.0%	3.4%	0.2%	0.3%	0.3%	0.8%	2.2%
8/28	5126	9501	1.9	76.6%	8.7%	5.1%	3.6%	3.4%	0.4%	0.6%	0.5%	0.8%	0.4%
*8/29	1661	3321	2.0	64.3%	10.5%	5.8%	4.2%	5.1%	0.2%	0.8%	0.9%	3.2%	5.0%
*8/30	2028	3233	1.6	72.1%	11.0%	7.1%	4.7%	4.3%	0.0%	0.1%	0.0%	0.4%	0.2%
8/31	6309	11591	1.8	77.3%	9.4%	4.8%	3.4%	3.4%	0.3%	0.5%	0.4%	0.3%	0.2%
9/01	6927	12527	1.8	78.1%	9.1%	5.1%	3.2%	3.4%	0.1%	0.4%	0.2%	0.3%	0.1%
9/02	7422	14608	2.0	75.0%	8.5%	4.9%	3.6%	3.3%	0.4%	0.6%	0.6%	1.4%	1.6%
9/03	7230	13238	1.8	78.1%	9.2%	4.8%	3.2%	3.4%	0.2%	0.2%	0.2%	0.6%	0.2%
9/04	6838	10839	1.6	75.5%	9.2%	5.3%	3.7%	3.1%	0.2%	0.5%	0.5%	1.3%	0.7%
*9/05	1831	3227	1.8	73.3%	10.1%	5.6%	4.6%	4.7%	0.1%	0.1%	0.1%	0.5%	0.9%
*9/06	1711	3242	1.9	69.5%	10.7%	6.5%	5.4%	5.2%	0.5%	1.0%	0.9%	0.2%	0.1%
*9/07	2768	5434	2.0	66.1%	10.2%	6.8%	5.6%	6.3%	1.2%	1.7%	1.7%	0.4%	0.0%
9/08	8508	15591	1.8	78.3%	9.2%	4.7%	3.5%	3.4%	0.2%	0.3%	0.2%	0.2%	0.1%
9/09	8800	16127	1.8	77.3%	9.6%	5.1%	3.4%	3.5%	0.1%	0.4%	0.4%	0.2%	0.1%
9/10	14540	16784	1.2	78.2%	9.3%	4.8%	3.4%	3.6%	0.2%	0.2%	0.2%	0.2%	0.1%
9/11	15657	8144	0.5	77.1%	8.7%	4.9%	3.7%	5.0%	0.1%	0.1%	0.0%	0.2%	0.2%
9/12	12114	5005	0.4	73.6%	10.5%	6.0%	4.2%	4.9%	0.2%	0.2%	0.1%	0.1%	0.2%
9/13	7095	5488	0.8	72.2%	10.8%	6.4%	5.0%	5.1%	0.2%	0.1%	0.1%	0.1%	0.1%
ave	5928	10433	1.9	75.8%	9.5%	5.2%	3.7%	3.6%	0.3%	0.5%	0.4%	0.5%	0.6%

Table B.6. Document access statistics from the THOMAS log

	Number of							
	queries		unique queries		topics			
date	total	w/results	total	w/results	total	occur. once	more than once	more than one unique query
09/16/97	499836	444899	365276	320987	249405	196672	52733	32750

	percentages of queries that the top topics account for					
	500	1000	5000	10000	15000	20000
09/16/97	12.3%	16.0%	27.9%	34.4%	38.6%	42.0%

Table B.7. The statistics from the Excite log.

more than once unique query. It also list how many of queries the top 500, 1000, 5000, 10000, and 20000 topics cover respectively. The statistics show that the Excite queries also have high query locality; the top 500 topics (around 2% of the total topics) cover 11.5% of queries, and the top 20000 topics (around 10% of the total topics) cover 39.6%. Among the topics that occurred more than once, 62% (32750) contain more than one unique query. Since we only have this one day log, we can not collect the information about overlap of topics between days.

BIBLIOGRAPHY

[1] Acharya, S., and Zdonik, S.B. An efficient scheme for dynamic data replication. Tech. Rep. CS-93-43, Department of Computer Science, Brown University, Sept. 1993.

[2] Agarwal, D., and Abbadi, A.E. The tree quorum protocol: An efficient approach for managing replicated data. In *Proceedings of the 16th VLDB Conference* (1990).

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

[4] AltaVista. *http://www.altavista.com/*.

[5] Baentsch, M., Molter, G., and Sturm, P. Introducing application-level replication and naming into today's Web. In *Proceedings of Fifth International World Wide Web Conference* (Paris, France, May 1996).

[6] Bailey, P., and Hawking, D. A parallel architecture for query processing over a terabyte of text. Tech. Rep. TR-CS-96-04, The Australian National University, June 1996.

[7] Bestavros, A. Demand-based document dissemination to reduce traffic and balance load in distributed information systems. In *Proceedings of SPDP'95: The 7th IEEE Symposium on Parallel and Distributed Processing* (San Anotonio, Texas, Oct. 1995).

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

[9] Bowman, C.M., Danzig, P.B., Hardy, D.R., Manber, U., and Schwartz, F. Harvest, a scalable, customizable discovery and access system. Tech. Rep. CU-CS-732-94, Department of Computer Science, University of Colorado at Boulder, July 1994.

[10] Buckley, C. Implementation of the smart information retrieval system. Tech. Rep. 85-686, Computer Science Department, Cornell University, Ithica, NY, May 1985.

[11] Burkowski, F. J. Retrieval performance of a distributed text database utilizing a parallel process document server. In *1990 International Symposium On Databases in Parallel and Distributed Systems* (Trinity College, Dublin, Ireland, July 1990), pp. 71–79.

[12] Cahoon, B., and McKinley, K. S. Performance evaluation of a distributed architecture for information retrieval. In *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Zurich, Switzerland, Aug. 1996), pp. 110–118.

[13] Cahoon, B., McKinley, K. S., and Lu, Z. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Transaction on Information Syetems (submitted)* (1999).

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

[15] Callan, J. P., Lu, Z., and Croft, W. B. Searching distributed collections with inference networks. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Seattle, WA, July 1995).

[16] Carey, M.J., and Livny, M. Distributed concurrency control performance: A study of algorithms, distribution and replication. In *Proceedings of the 14th VLDB Conference* (Los Angeles,CA, 1988).

[17] Carter, R.L., and Crovella, M.E. Dynamic server selection using bandwidth probing in wide-area networks. Tech. Rep. BU-CS-96-007, Boston University, Mar. 1996.

[18] Chakravarthy, A.S., and Haase, K.B. Netserf: Using semantic knowledge to find internet information archives. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Seattle, WA, July 1995), pp. 4–11.

[19] Cheung, S.Y., Ammar, M.H., and Ahamad, M. The grid protocol: a high performance scheme for maintaining replication data. In *Proceedings of the sixth International Conference on Data Engineering* (Jan. 1990), pp. 438–445.

[20] Couvreur, T. R., Benzel, R. N., Miller, S. F., Zeitler, D. N., Lee, D. L., Singhai, M., Shivaratri, N., and Wong, W. Y. P. An analysis of performance and cost factors in searching large text databases using parallel search systems. *Journal of the American Society for Information Science 7*, 45 (1994), 443–464.

[21] Cringean, J. K., England, R., Mason, G. A., and Willett, P. Parallel text searching in serial files using a processor farm. In *Proceedings of the Thirteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Brussels, Belgium, Sept. 1990).

[22] Croft, W. B., Cook, R., and Wilder, D. Providing government information on the Internet: Experiences with THOMAS. In *The Second International Conference on the Theory and Practice of Digital Libraries* (Austin, TX, June 1995).

[23] Danzig, P. B., Ahn, J., Noll, J., and Obraczka, K. Distributed indexing: A scalable mechanism for distributed information retrieval. In *Proceedings of the Fourteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Chicago, IL, 1991), pp. 221–229.

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

[25] Eager, D.L., and Sevick, K.C. Achieving robustness in distributed database systems. *ACM Transactions on Database Systems 8*, 3 (Sept. 1983), 354–381.

[26] Efraimidis, P., Glymidakis, C., Mamalis, B., Spirakis, P., and Tampakas, B. Parallel text retrieval on a high performance supercomputer using the vector space model. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Seattle, WA, 1995), pp. 58–66.

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

[28] Faloutsos, C., and Christodoulakis, S. Signiture files: An access methods for documents and its analytical performance evaluation. *ACM Transaction on Office Information Systems 2*, 4 (Oct. 1984), 267–288.

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

[30] Gifford, D. Weighted voting for replicated data. In *Proceedings of 7th ACM Symposium on Operating System Principles* (1979), pp. 150–162.

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

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

[33] Gray, J., Helland, P., O'Neil, P., and Shasha, D. The dangers of replication and a solution. In *Proceedings of ACM SIGMOD'96* (Montreal, Canada, June 1996), pp. 173–182.

[34] Guyton, J., and Schwarz, M. Locating nearby copies of replicated internet servers. Tech. Rep. CU-CS-762-95, University of Colorado at Boulder, Feb. 1995.

[35] Hardy, D.R., and Schwartz, M.F. Customized information extraction as a basis for resource discovery. *ACM Transactions on Computer Systems 14*, 2 (May 1996), 171–199.

[36] Harman, D., Ed. *The First Text REtrieval Conference (TREC-1).* National Institute of Standards and Technology Special Publication 200-217, Gaithersburg, MD, 1992.

[37] Harman, D., Ed. *The Second Text REtrieval Conference (TREC-2).* National Institute of Standards and Technology Special Publication 300-217, Gaithersburg, MD, 1993.

[38] Harman, D., Ed. *The Third Text REtrieval Conference (TREC-3).* National Institute of Standards and Technology Special Publication 500-225, Gaithersburg, MD, 1994.

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

[40] Harman, D., Ed. *The Fifth Text REtrieval Conference (TREC-5).* National Institute of Standards and Technology Special Publication, Gaithersburg, MD, 1996.

[41] Harman, D., McCoy, W., Toense, R., and Candela, G. Prototyping a distributed information retrieval system that uses statistical ranking. *Information Processing & Management 27*, 5 (1991), 449–460.

[42] Hawking, David, Craswell, Nick, and Thistlewaite, Paul. Overview of trec-7 very large collection track. In *Proceedings of the 7th Text Retrieval Conference* (1998).

[43] Howard, J. H., Kazar, M. L., Menees, S. G., A.Nichols, D., Satyanarayanan, M., Sidebotham, R. N., and West, M. J. Scale and performance in a distributed file system. *ACM transactions on Computer Systems 6*, 1 (Feb. 1988).

[44] Huang, Y., and Wolfson, O. A competitive dynamic data replication algorithm. In *IEEE Proceedings of 9th International Conference on Data Engineering* (Vienna, Austria, 1993), pp. 310–337.

[45] Infoseek. *http://guide.infoseek.com.*

[46] InQuery. *http://ciir.cs.umass.edu/info/highlights.html.*

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

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

[49] Katz, E.D, Butler, M., and McGrath, R. A scalable HTTP server: the NCSA prototype. In *Proceedings of First International World Wide Web Conference* (Geneva, Switzerland, May 1994).

[50] Knuth, D.E. *The Art of Computer Programming Vol 3: Sorting and Searching.* Addison-Wesley.

[51] Kumar, A., Rabinovich, M., and Sinha, R.K. A performance study of general grid structures for replicated data. In *Proceedings of 13th International Conference on Distributed Computing Systems* (1993).

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

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

[54] Lu, Z., Callan, J. P., and Croft, W.B. Applying inference networks to multiple collection searching. Tech. Rep. TR96-42, Department of Computer Science, University of Massachusetts at Amherst, 1996.

[55] Lu, Z., McKinley, K. S., and Cahoon, B. The hardware/software balancing act for information retrieval on symmetric multiprocessors. In *Proceedings of Europar98* (Southhampton, UK, 1998).

[56] Macleod, I. A., Martin, T. P., Nordin, B., and Phillips, J. R. Strategies for building distributed information retrieval systems. *Information Processing & Management 23*, 6 (1987), 511–528.

[57] Mamalis, B., Spirakis, O., and Tampakas. Parallel techniques for efficient searching over very large text collections. In *Proceedings of The Fifth Text REtrieval Conference (TREC-5)* (Gaithersburg, MD, 1996), National Institute of Standards and Technology Special Publication.

[58] Martin, T. P., Macleod, I. A., Russell, J. I., Lesse, K., and Foster, B. A case study of caching strategies for a distributed full text retrieval system. *Information Processing & Management 26*, 2 (1990), 227–247.

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

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

[61] Ribeiro-Neto, B. A., and Barbosa, R. A. Query performance for tightly coupled distributed digital libraries. In *Proceedings of ACM Digital Libraries Conference* (Pittsburgh, PA, 1998).

[62] Robertson, S.E. The probability ranking principle in ir. *Journal of Documentation 33*, 4 (1977), 294–304.

[63] Salton, G. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, 1989.

[64] Salton, G., and McGill, M. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[65] Shah, A., and Marzull, K. Trade-offs between replication and availability in distributed databases. Tech. Rep. TR89-1065, Cornell University, Dec. 1989.

[66] Stanfill, C. Partitioned posting files: A parallel inverted file structure for information retrieval. In *Proceedings of the Thirteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Brussels, BELGIUM, 1990), pp. 413–428.

[67] Stanfill, C., and Kahle, B. Parallel free-text search on the connection machine system. *Communications of the ACM 29*, 12 (Dec. 1986), 1229–1239.

[68] Stanfill, C., Thau, R., and Waltz, D. A parallel indexed algorithm for information retrieval. In *Proceedings of the Twelfth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Cambridge, MA, June 1989), pp. 88–97.

[69] Thomas, R.H. A majority consensus approach to concurrency control for multiple copy database. *ACM Transactions on Database Systems 4*, 2 (June 1979), 180–209.

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

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

[72] Turtle, H. R. *Inference Networks for Document Retrieval.* PhD thesis, University of Massachusetts, Feb. 1991.

[73] Van Rijsbergen, C.J. *Information Retrieval.* Butterworths, 1979.

[74] Voorhees, E. M., Gupta, N. K., and Johnson-Laird, B. Learning collection fusion strategies. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Seattle, WA, 1995).

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

[76] Wolfson, O., and Jajodia, S. An algorithm for dynamic replication of data. In *Proceedings of 11th ACM Symposium on the Principles of Database Systems* (San Diego, California, June 1992).

[77] Xu, J., and Callan, J.P. Effective retrieval with distributed collections. In *Proceedings of SIGIR98* (Melbourne, Australia, Aug. 1998).

[78] Yu, P. S., and MacNair, E. A. Performance study of a collabortive method for hiearchical caching in proxy servers. In *Proceedings of 7th International World Wide Web Conference* (Brisbane, Australia, Apr. 1998).

[79] Zobel, J., Moffat, A., and Ramamohanarao, K. Inverted files vesus signiture files for text indexing. Tech. Rep. CITRI/TR-95-5, Collaborative Information Technology Research Institute, Deparment of Computer Science, Royal Melbourne Institute of Technology, Australia, July 1995.