The Hardware/Software Balancing Act for Information Retrieval on Symmetric Multiprocessors

Zhihong Lu Kathryn S. McKinley Brendon Cahoon

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

Abstract. Web search engines, such as AltaVista and Infoseek, handle tremendous loads by exploiting the parallelism implicit in their tasks and using symmetric multiprocessors to support their services. The web searching problem that they solve is a special case of the more general *information retrieval* (IR) problem of locating documents relevant to the information need of users. In this paper, we investigate how to exploit a symmetric multiprocessor to build high performance IR servers. Although the problem can be solved by throwing lots of CPU and disk resources at it, the important questions are *how much* of *which* hardware and *what* software structure is needed to effectively exploit hardware resources. We have found, to our surprise, that in some cases adding hardware *degrades* performance rather than improves it. We show that multiple threads are needed to fully utilize hardware resources. Our investigation is based on InQuery, a state-of-the-art full-text information retrieval engine.

1 Introduction

As information explodes across the Web and elsewhere, people increasingly depend on search engines to help them to find information. Web searching is a special case of the more general *information retrieval* (IR) problem of locating documents relevant to the information need of users. In this paper, we investigate how to balance hardware and software resources to exploit a symmetric multiprocessor (SMP) architecture to build high performance IR servers. Our IR server is based on InQuery [2, 3], a state-of-the-art full-text information retrieval engine that is widely used in Web search engines, large libraries, companies, and governments such as Infoseek, Library of Congress, White House, West Publishing, and Lotus [5]. Our work is novel because it investigates a real, proven effective system under a variety of realistic workloads and hardware configurations on an SMP architecture. The previous research investigates either the IR system on massively parallel processing (MPP) architecture or it investigates only a subset of the system on SMP architecture such as the disk system or it compares the cost factors of SMP architecture with other architectures. (See [6] for a more thorough comparison with the related work). Our results provide insights for building high performance IR servers for searching the Web and other environments using a symmetric multiprocessor.

The remainder of this paper is organized as follows. The next section describes the implementation of our parallel IR server and simulator. Section 3 presents results that demonstrate the system scalability and hardware/software balancing with respect to multiple threads, CPUs, and disks. Section 4 summarizes our results and concludes.

2 A Parallel Information Retrieval Server

This section describes the implementation of our parallel IR server and simulator. We begin with a brief description of the InQuery retrieval engine [2,3,5]. We next present the features we model and summarize our validation of the simulator against the multithreaded implementation.

InQuery Retrieval Engine

InQuery is one of the most powerful and advanced full-text information retrieval engines in commercial or government use today [5]. It uses an inference network model, which applies Bayesian inference networks to represent documents and queries, and views information retrieval as an inference or evidential reasoning process [2,3]. The inference networks are implemented as inverted files. In this paper, we use "collection" to refer to a set of documents, and "database" to refer to an indexed collection.

The InQuery server supports a range of IR commands such as query, document, and relevance feedback. The three basic IR commands we model are: *query*, *summary*, and *document*. A *query* command requests documents that match a set of terms. A query response consists of a list of top ranked document identifiers. A *summary* command consists of a set of document identifiers. A summary response includes the document titles and the first few sentences of the documents. A *document* command requests a document using its document identifier. The response includes the complete text of the document.

The Parallel IR Server

To investigate the balance between hardware and software in a IR system on a symmetric multiprocessor, we implemented a parallel IR server using InQuery as the retrieval engine. The parallel IR server exploits parallelism as follows: (1) It executes multiple IR commands in parallel by multithreading; and (2) It executes one command against multiple partitions of a collection in parallel. To expedite our investigation of possible system configurations, characteristics of IR collections, and the basic IR system performance, we implement a simulator with numerous system parameters, such as the number of CPUs, threads, disks, collection size, and query characteristics. Table 1 presents all the parameters and the values we use in our experiments.

The simulation model is driven by empirical timing measurements from the actual system. For queries, summaries, and documents, we measure CPU and

Table 1. Experimental Parameters

Parameters	Abbreviation	Values
Query Number	QN	1000
Terms per Query (average) shifted neg. binomial dist. (n,p,s)	TPQ	2 (4,0.8,1)
Query Term Frequency dist. from queries	QTF	Observed Distribution
Client Arrival Pattern Poisson process (requests per minute)	λ	6 30 60 90 120 150 180 240 300
Collection Size (GB)	CSIZE	1 2 4 8 16
Disk Number	DISK	1 2 4 8 16
CPU Number	CPU	1 2 3 4
Thread Number	TH	1 2 4 8 16 32

disk usage for each operation, but do not measure the memory and cache effects. We model the collection and queries by obtaining document and term statistics from test collections and real query sets (See [1,6] for more details.)

We validate our simulator against the multithreaded implementation. The simulator reports response times that are 4.5% slower than the actual system on the average (See [6] for more details).

3 Experiments and Results

This section explores how software and hardware configurations affect system scalability with respect to multiple threads, CPUs, and disks.

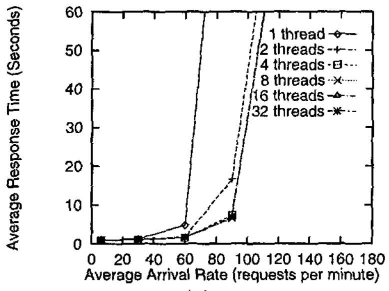
We start with a base system that consists of one thread, CPU, and disk. This system is disk bound. We improve the performance of our IR server through better software: multithreading; and with additional hardware: CPUs and disks. We demonstrate the system scalability using two sets of experiments. In the first set of experiments, we explore the effects of threading on system scalability. In the second set of experiments, we explore system scalability by increasing the collection size from 1 GB to 16 GB. When multiple disks exist, we use a round-robin strategy to distribute the collection and its index over disks.

We assume the client arrival rate is a Poisson process. Each client issues a query and waits for response. For each query, the server performs two operations: query evaluation and retrieving the corresponding summaries. Since users typically enter short queries, we experiment with a query set that consists of 1000 short queries, with an average of 2 terms per query that mimic those found in the query set down loaded from the Web server for searching the 103rd Congressional Record [4]. All experiments measure response time, CPU and disk utilization, and determine the largest arrival rate at which the system supports a response time under 10 seconds. We chose 10 seconds arbitrarily as our cutoff point for a reasonable response time.

3.1 Threading

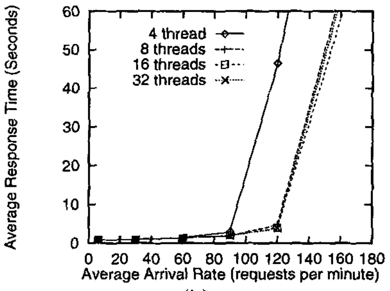
This section examines how the software structure, i.e., number of threads, affects system scalability. Figure 1 illustrates how the average response time and

QN	TPQ	QTF	CSIZE	CPU	DISK	TH
1000	2	Obs.	1 GB	1	1	Varied



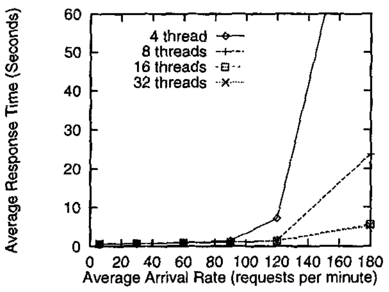
(a)

QN	TPQ	QTF	CSIZE	CPU	DISK	TH
1000	2	Obs.	1 GB	1	4	Varied



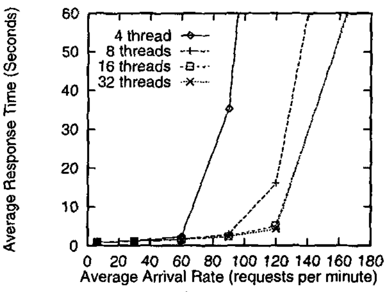
(b)

QN	TPQ	QTF	CSIZE	CPU	DISK	TH
1000	2	Obs.	1 GB	2	4	Varied



(c)

QN	TPQ	QTF	CSIZE	CPU	DISK	TH
1000	2	Obs.	4 GB	2	4	Varied



(d)

	Configuration (a)		Configuration (b)		Configuration (c)		Configuration (d)	
	$\lambda = 120$		$\lambda = 150$		$\lambda = 180$		$\lambda = 150$	
Resource	TH=2	TH=4	TH=4	TH=8	TH=8	TH=16	TH=8	TH=16
CPU	34.4%	36.2%	72.6%	89.2%	55.1%	56.7%	73.3%	85.2%
DISK	92.7%	97.30%	51.1%	62.8%	77.4%	79.4%	68.0%	79.3%

(e) hardware utilization at some interested data points

Fig. 1. Performance as the number of threads increases

resource utilization changes as the number of threads increases with varying number of CPUs and disks.

In all the configurations, the average response time improves significantly as the number of threads increases until either the disk or the CPU is overutilized (see Figure 1 (a) and (b)). Too few threads limits the system's ability to achieve its peak performance. For example in configuration (c), using 4 threads only supports 120 requests per minute for a response time under 10 seconds, while using 16 threads supports more than 180 requests per minute under the same hardware configuration. When either the CPU or the disk is a bottleneck, the system needs fewer threads to reach its peak performance. When CPUs and disks are well balanced (configuration (c) and (d)), the necessary number of threads is influenized more by the number of disks than the collection size. In both configuration (c) and configuration (d), the system achieves its peak performance using 16 threads. Additional threads do not bring further improvement.

3.2 Increasing the collection size

This section examines system scalability and hardware balancing as the collection size increases from 1 GB to 16 GB. In order to examine different hardware configurations, we consider two disk configurations as the collection size increases: fixing the number of disks, and adding disks. We vary the number of CPUs in each disk configuration.

Figure 2 illustrates the average response time and resource utilization when the collection is distributed over 16 disks which means each disk stores a database for 1/16 of the collection. Partitioning the collection over 16 disks illustrates when the system is CPU bound (see Figure 2(c)). Although performance degrades as the collection size increases, the degradation is closely related to the CPU utilization. With 1 CPU where the CPU is overutilized for 1 GB and 60 requests per minute, increasing the collection size from 1 GB to 16 GB decreases the largest arrival rate at which the system supports a response time under 10 seconds by a factor of 10 (see Figure 2(a)). With 4 CPUs where CPUs are overutilized for 1 GB and 180 requests per minute, the performance degrades much more gracefully (see Figure 2(b)). Increasing the collection size from 1 GB to 16 GB only decreases the largest arrival rate for a response time under 10 seconds by a factor of 3 for 4 CPUs.

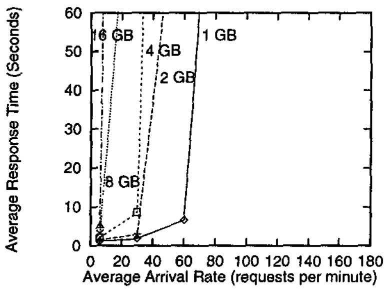
Figure 3 illustrates the average response time and the resource utilization when the number of disks varies with the collection size and each disk stores a database for 1 GB of data. The system produces response times better than 1 GB for 2 GB using 1 CPU and 8 GB using 4 CPUs. A single CPU system thus handles a 2 GB collection faster than a 1 GB collection and a 4 CPU system handles a 2, 4, or 8 GB collection faster than a 1 GB collection in our configuration. The performance improves because work related to retrieving summaries is distributed over the disks such that each disk handles less work, relieving the disk bottleneck. By examining the utilization of CPU and disks in Figure 3(c), we see that the performance improves until the CPUs are overutilized. In the example of the single CPU system, the CPU is overutilized for a 4 GB collection. For a 2 GB collection distributed over 2 disks, the system handles 27.8% more requests than for a 1 GB collection on 1 disk.

By comparing Figure 2(c) and Figure 3(c), we find that the CPU utilization is more closely related to the number of disks rather than the collection size. We also find that adding disks degrades system performance when CPUs and disks are not well balanced. For example, for a 8 GB collection, partitioning over 8 disks using 4 CPUs results in 67.5% CPU utilization (see Figure 3(c)), while partitioning over 16 disks results in 88.0% CPU utilization (see Figure 2(c)) due to the additional overhead to access each disk. In this configuration, a system with 16 disks performs worse than 8 disks because the CPUs are a bottleneck.

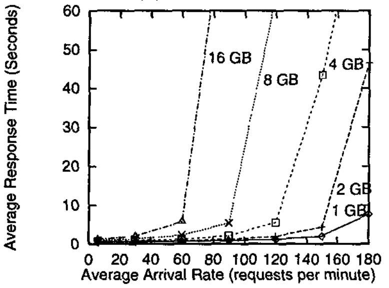
4 Conclusion

In this paper, we investigate building a parallel information retrieval server using a symmetric multiprocessor to improve the system performance. Our results show that we need more than one thread to fully utilize hardware resources (4 to

QN	TPQ	QTF	CSIZE	CPU	DISK	TH
1000	2	Obs.	Varied	Varied	16	16



(a) CPU = 1



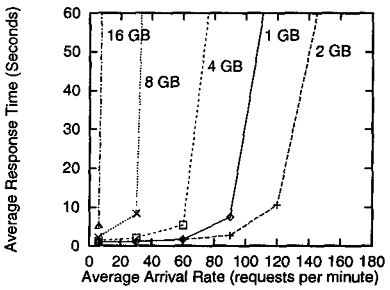
(b) CPU = 4

Num. CPUs	Resource	Size of Collection (size/16 GB per disk)				
		1 GB	2 GB	4 GB	8 GB	16 GB
1	CPU	96.6%	97.4%	98.1%	98.6%	99.0%
	DISK	21.6%	18.4%	14.7%	11.9%	9.8%
4	CPU	38.9%	49.7%	68.0%	88.0%	93.5%
	DISK	34.9%	37.8%	41.1%	42.6%	37.4%

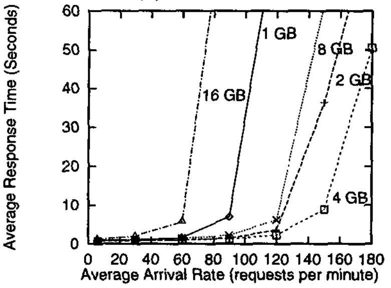
(c) Resource utilization when $\lambda = 120$

Fig. 2. Average response time and resource utilization for a collection distributed over 16 disks

QN	TPQ	QTF	CSIZE	CPU	DISK	TH
1000	2	Obs.	Varied	Varied	Varied	16



(a) CPU = 1



(b) CPU = 4

Num. CPUs	Resource	Size of Collection (1 GB per disk)				
		1 GB	2 GB	4 GB	8 GB	16 GB
1	CPU	36.2%	75.3%	94.3%	98.1%	99.0%
	DISK	97.4%	82.2%	43.5%	20.6%	9.8%
4	CPU	9.0%	19.0%	35.7%	67.5%	93.5%
	DISK	97.4%	82.9%	66.7%	57.1%	37.4%

(c) Resource utilization when $\lambda = 120$

Fig. 3. Average response time and resource utilization when the number of disks varies with the size of the collection

16 threads for the configurations we explored). We also show that adding hardware components can improve the performance, but these components must be well balanced since the IR workload performs significant amounts of both I/O and CPU processing. Our results show that we can search more data with no loss in performance in many instances. Although performance eventually degrades as the collection size increases, the performance degrades very gracefully if we keep the hardware utilization balanced. Our results also show that system performance for our system is more strongly related to the number of disks rather than the collection size.

Acknowledgment

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. Kathryn S. McKinley is supported by an NSF CAREER award CCR-9624209. Any opinions, findings and conclusions or recommendations expressed in this material are the authors and do not necessarily reflect those of the sponsor.

We thank Bruce Croft, Jamie Callan, James Allan, and Chris Small for their support of this work. We also thank all the developers of InQuery without whose efforts this work would not have been possible.

References

1. B. Cahoon and K. S. McKinley. 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*, pages 110–118, Zurich, Switzerland, August 1996.
2. J. P. Callan, W. B. Croft, and J. Broglio. TREC and TIPSTER experiments with INQUERY. *Information Processing & Management*, 31(3):327–343, 1995.
3. J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings of the 3rd International Conference on Database and Expert System Applications*, Valencia, Spain, September 1992.
4. W. B. Croft, R. Cook, and D. Wilder. Providing government information on the internet: Experiences with THOMAS. In *The Second International Conference on the Theory and Practice of Digital Libraries*, Austin, TX, June 1995.
5. InQuery. `http://ciir.cs.umass.edu/info/highlights.html`.
6. Zhihong Lu, Kathryn S. McKinley, and Brendon Cahoon. The hardware/software balancing act for information retrieval on symmetric multiprocessors. Technical Report TR98-25, University of Massachusetts, Amherst, 1998.