PERFORMANCE ANALYSIS AND MODELING OF DISK I/O SUBSYSTEM IN HIGH THROUGHPUT SERVERS

BY

AMISU OLUWAKEMI SALAM-ALADA

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

JANUARY 2004

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **AMISU OLUWAKEMI SALAM-ALADA** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING**.

Thesis Committee

A.W. Taheed 1/14/04
Dr. AbdulWaheed M. A. Sattar (Chairman)

24/01/04
Prof. Mostafa Abd-El-Barr (Member)

U. Baroudi 1/19/04
Dr. Uthman Baroudi (Member)

Mar. 23/04
Department Chairman
Prof. Sadiq M. Sait

Dean of Graduate Studies
Prof. Osama A. Jannadi

4-4-2004
Date

DEDICATION

This thesis is dedicated to my parents and my daughter

AmturRahman Oluwatomi Oluwafeyikemi

ACKNOWLEDGMENTS

In name of Allah, the Most Gracious, the Most Merciful.

"Read! In the Name of your Lord Who has created all that exists. He has created man from a clot (a peace of thick coagulated blood). Read! And your Lord is the Most Generous. Who has taught (the writing) by the pen. He has taught man that which he knew not. (Qur'an 96: 1-5).

All praises belong to Almighty Allah, the most Beneficent and the most Merciful, who enabled me to complete my thesis work and made it possible for me to come this far. I sincerely thank Allah for His endless blessings on me, as His infinite blessings can never be thanked for. I pray Allah to bestow peace and blessings on the Holy Prophet Muhammad (Sal-allah-'Alaihe-Wa-Sallam) and on all his righteous followers till the day of judgement.

Acknowledgement is due to King Fahd University of Petroleum & Minerals for supporting this research. In addition, I thank Information Technology Center (ITC) of King Fahd University of Petroleum & Minerals for providing IBM Netfinity server machine and other accessories used for the measurement-based experiment.

My sincere appreciation goes to Dr. AbdulWaheed M. A. S, for his efforts, advice, encouragement and invaluable support given as my advisor. I also wish to thank my thesis committee members Prof Mostafa. Abd-El-Barr and Dr. Uthman Baroudi for their help, support, and contributions. In addition, I appreciate the support rendered to me by the department Chairman, Dr. Sadiq Sait.

I would like to thank Lindy Shriver, Justin Burke and Alex Ruskov for various discussions and comments during the early stage of my work. In addition, my fellow Performance Engineering Laboratory (PEL) associates, Yau, Sanaullah and Faheem for the great time we shared in the lab.

I acknowledge my colleagues and dear friends and the loving support and encouragement from my dear wife and daughter. Finally, I wish to express my gratitude to my family members especially my parents for their patience, love and sincere prayers from the moment I started my education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THESIS ABSTRACT (ENGLISH)

NAME: Amisu Oluwakemi Salam-Alada

TITLE: Performance Analysis and Modeling of Disk I/O Subsystem in High Throughput Servers.

MAJOR FIELD: Computer Engineering.

DATE OF DEGREE: January 2004.

While processor speeds are rapidly increasing, the Input/Output (I/O) especially disk I/O speeds are lagging far behind. The implication of this trend is that the performance of several high throughput networking application servers is limited by I/O subsystem overheads. This thesis considers three I/O subsystem configurations for high throughput servers: single disk, disk array, and Redundant Array of Inexpensive Disks (RAID). In addition, we consider two disk I/O scheduling policies: First-Come First-Served (FCFS) and Shortest Position Time First (SPTF) with and without prefetching. We characterize the high throughput server workloads in terms of three categories: Sequentially Distributed Strides Accesses (SDSA), Widely Distributed Strides Accesses (WDSA) and Narrowly Distributed Strides Accesses (NDSA). Using this characterization, we employ trace-driven simulation to compare the alternative configurations and scheduling policies. Our simulation-based analysis shows that SPTF scheduling can reduce the average disk seek time by 50% compared to FCFS scheduling with possibility of further 10% improvement when prefetching is used under WDSA workloads. Additionally, using disk array or RAID configurations can reduce effective seek time by a factor ranging from 2 to 8 under WDSA workload. These improvements in disk seek time enhance the overall server throughput. We use measurements on a real server to validate some of the above findings that are related to three types of I/O subsystem configurations.

THESIS ABSTRACT (ARABIC)

الإســـــم : أميسو الوواكيمي سلام-ألادا
عنوان الرسالة : تحليل ونمذجة كفاءة نظام إدخال/إخراج القرص في أجهزة الخادمات ذات الطاقة الإنتاجية العالية.
التخصص : هندسة الحاسب الآلي
تاريخ التخرج : يناير 2004

بينما تزداد سرعات المعالج بصورة مفاجئة ، فإن سرعة الإدخال والإخراج (د/خ) وخاصة إدخال/إخراج القرص قد تخلف كثيراً. نتيجة لذلك فإن كفاءة أجهزة الخادمات ذات الطاقة العالية قد تقيدت بتكاليف نظام د/خ. يعرض هذا البحث ثلاث مواصفات لنظام د/خ : قرص أحادي ، جمع من الأقراص و جمع فائض من الأقراص غير المكلفة (RAID). بالإضافة إلى ذلك، نأخذ بعين الإعتبار طريقتين لجدولة نظام د/خ القرص وهما: أول داخل-أول خارج (FCFS) وأول أقصر وقت للموضع (SPTF) باستخدام و بدون استخدام تمهيد الاستجلاب. صنّفنا قدرة أجهزة الخادمات ذات الطاقة العالية بثلاثة طبقات: المداخل الموسعة والمنتشرة على نحو تسلسلي (SDSA) والمداخل الموسعة والمنتشرة على نطاق واسع (WDSA) و المداخل الموسعة والمنتشرة على نطاق ضيق (NDSA). باستخدام هذا التصنيف، استخدمنا المحاكاة المنقادة نحو الأثر لمقارنة المواصفات الموجوده ومقارنة أساليب الجدولة. فالنتائج المبنية على المحاكاة أظهرت أن استخدام أسلوب الجدولة SPTF يمكن أن يقلل من معدل زمن بحث القرص إلى 50% مقارنة بأسلوب الجدولة FCFS مع احتمالية زيادة في تحسن الأداء بمقدار 10% عند استخدام طريقة تمهيد الاستجلاب تحت قدرات WDSA. إضافة لذلك ، فإن استخدام جمع الأقراص أو RAID يمكن أن يقلل بصورة ملحوظة زمن البحث بمعامل يتراوح بين 2 إلى 8 تحت قدرة WSDA. هذه التحسينات في زمن البحث يزيد من القدرة الإجمالية لجهاز الخادم. أجرينا قياسات حقيقة على جهاز خادم لإثبات النتائج التي حصلنا عليها آنفاً والمتعلقة بأنواع المواصفات الثلاث لنظام د/خ القرص.

درجة الماجستير في العلوم

جامعــة الملك فهــد للبترول والمعــادن ، الظهران – السعودية

يناير 2004

CHAPTER 1

INTRODUCTION

In the majority of institutions with high volume web sites, Input/Output (I/O) operations have been recognized as critical system performance inhibitors. Performance of I/O is particularly important for those who want to serve large volumes of user requests simultaneously. This is due to the fact that whatever the users want to obtain are residing on disk storage. As a web site grows in popularity and the traffic increases, the performance requirements for such disk I/O on high throughput can be a challenge to meet. Hence the I/O handling program in such a high throughput sever should be able to accommodate thousands of requests per minute in order to give better throughput.

In the last decades, innovations in technology have resulted in extraordinary advances in computer processing speed. These advances have led many computer system performance analysts to focus their attention on measuring processor performance to the near exclusion of all other metrics; some have even equated a computer system's performance with how well its CPU performs. This viewpoint, which makes system-wide performance synonymous with CPU speed, is becoming increasingly impractical due to growing disparity with I/O speeds.

If CPU performance continues to improve at its current fast pace and disk I/O at a moderate pace, eventually the performance of all applications such as the web server that

perform excessive I/O operations will be limited by these I/O operations. Thus further CPU performance improvements will have no effect on performance [30]. An ever-widening mismatch between disk I/O and processor performance is rendering disk I/O performance evaluation increasingly important.

Several studies have analyzed disk I/O subsystem using simulation modeling and I/O workload characterization. Some have modeled each of the disk system components. However, to the best of our knowledge, none has looked into the performance requirements of disk I/O behavior when it is used in a high throughput network infrastructure device such as web server or web proxy server. Such servers serve thousands of users simultaneously and the users expect a timely response. Therefore, in order for such high throughput servers to perform, each component of the server, i.e., the hardware components like CPU, memory, disk I/O and network I/O and software components like the HTTP demon, the TCP/IP implementation and the file system must also perform well. We have noticed that one area where bottlenecks occur frequently is the disk I/O component.

1.1 High Throughput Servers

The web is an evolving system that incorporates new devices and services at a very fast rate and servers are key components of the web. They deliver information upon request in the form of text, images, sound, video and multimedia combinations of these. Applications such as video on-demand, e-commerce, and digital libraries continue to increase Internet web traffic at an alarming rate. These applications require high

throughput servers, such as web servers, proxy server, cache servers, and video on-demand servers that can respond to large volumes of requests simultaneously with minimal waiting time at the user end. Performance problems on the Internet are characterized by the unpredictable nature of requests for information and services over the web and servers are expected to meet up with these great demands.

Due to the complexity and importance of disk I/O performance for high throughput servers, we focus on this problem.

1.2 Disk I/O Performance Evaluation

In order to conduct performance evaluation of a system, there are three well-known techniques. These techniques include analytical modeling, simulation and measurement-based evaluation. Disk I/O subsystem evaluation has leveraged from each one of these techniques. Combination of any two or all of these techniques is possible to ensure a thorough investigation and proper validation of results.

Analytical modeling uses equation to express disk I/O characteristics under simplifying assumptions to predict the performance of I/O subsystem. However, the results of such analysis are often not accurate, but it can be used together with other techniques such as measurement-based or simulation modeling. Such schemes are convenient to analyze an I/O subsystem at early design stages but they do not provide accurate insight into the system under consideration.

Simulation modeling is an effective technique for evaluating computer system performance to investigate several what-if questions that may not be possible otherwise. Many Computer system analysts have used this method especially when the system to be characterized is either not available or cannot be conveniently evaluated using measurement. It thus provides an easy way to predict the performance of such systems or to compare different alternatives. In addition, it allows comparison of these alternatives under wider variety of workloads and implements schemes that are impossible under real-world scenarios.

Measurement-based evaluation techniques provide most realistic evaluation of the performance of a system and its behavior. However, such a technique is time consuming and expensive to conduct. In many cases, benchmarking tools are helpful in conducting a thorough evaluation of the system under test.

The goal of any performance study is either to compare different alternatives or to find the optimal parameter values that the system under test can utilize at any time. The outcome often helps in improving the design of such systems or if need be, redesign it. In order to conduct performance evaluation of any system, intimate knowledge of the system is required and proper methodology and appropriate workload and tools are necessary. In addition, understanding the right technique to use for evaluating the system is important.

In this thesis research, we use trace-driven simulation-based analysis technique to compare multiple disk I/O configurations under various I/O access patterns. We use measurements to validate simulation results.

1.3 Problem Statement

A high throughput server performance is often limited by the slowest component on the data path that spans processor, caches, main memory, and disks. This component may be the main memory, the CPU-memory bus, bus adapter, the I/O bus, the I/O controller or the disks themselves. In addition, the complex policies of the operating system can also affect the I/O performance. Compared to all other components, a disk is the only device that uses mechanical components. Consequently, the performance of a disk (in terms of access latency) lags behind memory performance by several orders of magnitude. Therefore, our work focuses the performance impact of the disk I/O subsystem on the high throughput servers in a network infrastructure.

Figure 1.1 presents an illustration of disk I/O bottleneck problem in a typical high throughput server. Considering a typical general-purpose processor with clock speed of more than 1 GHz, we can assume that it can accomplish N transactions/sec for a typical networking application assuming that all data is available in on-chip caches. A processor can typically access an on-chip cache memory location with one cycle latency. Now consider what happens if this data continues to move farther away from the processor in the storage hierarchy due to requirement of larger storage capacity. When data relocates to an off-chip secondary cache, the access latency typically increases to tens of clock cycles resulting in reduction of throughput by same factor (N/10). When transactions run on data from main memory, the access latency can increase to hundreds of clock cycles. Consequently, the peak transaction throughput that the server can deliver comes down in the range of N/100. However, when the storage capacity requirements grow even further

and the data has to be located on a disk, the access time latency (i.e., average seek time) soars to millions of cycles (typically, in the range of 5—10 msec). With such latency for processing every transaction, throughput can potentially reduce to the range of one millionth of N. Unless the disk I/O subsystem accesses and configurations are properly analyzed, it may not be possible to effectively hide such large latencies to reap the benefits of ever increasing processor speeds for a high throughput server.

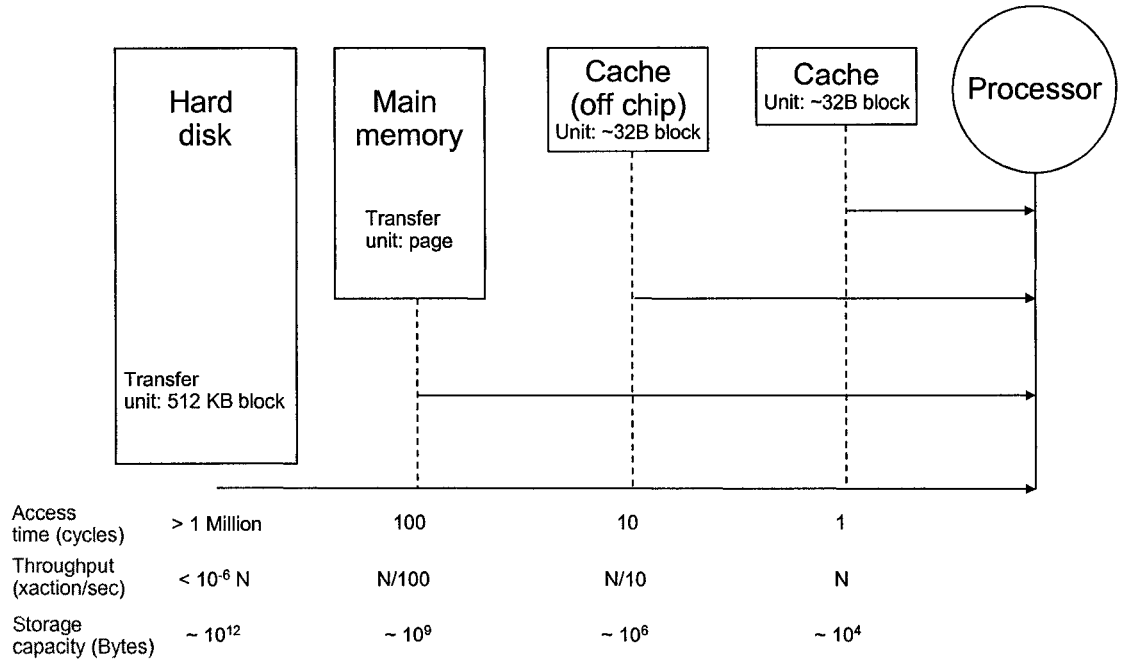


Figure 1.1: An illustration of I/O bottleneck in the context of performance disparity with processors for high throughput server applications.

Performance of the disk I/O subsystems plays a critical role in determining the overall performance of any high throughput systems. The objective of our effort is to model and analyze the disk I/O subsystem behavior for a high throughput server. Various configuration alternatives result in different severities of I/O bottlenecks for different I/O access patterns due to network infrastructure applications executed on high throughput

servers. These alternatives include: stand-alone disks, disk arrays and Redundant Array of Inexpensive Disks (RAIDs). However, implementing each configuration in a high throughput server is impractical due to cost and time. Thus, these systems need to be modeled so that performance impact of these alternative configurations could be evaluated to select a suitable configuration for a particular application. Identifying appropriate scheduling and latency hiding mechanisms is the key to alleviate I/O subsystem performance bottlenecks.

We will first identify three disk I/O subsystem access classes for high throughput server as well as three alternate configurations (Chapter 3). We also define two disk I/O scheduling policies, with and without prefetching. We will use trace-driven simulation to compare alternative configurations under three types of access patterns to identify scheduling and latency hiding mechanisms that are appropriate for alleviating I/O bottlenecks for high throughput servers. Finally, we shall use measurement-based evaluation to validate some of the simulation results.

1.4 Main Contributions

The main contributions of this thesis are summarized as follows:

- Characterization of I/O workload due to high throughput networking applications in terms of three access patterns: Sequentially Distributed Strides Accesses (SDSA), Widely Distributed Strides Accesses (WDSA), and Narrowly Distributed Strides Accesses (NDSA);

- Performance comparisons of disk I/O impact on high throughput servers through trace-driven simulations under three configurations: single disk, disk arrays, and RAIDs;
- Comparison of disk I/O subsystem performance using trace-driven simulation under two disk I/O scheduling algorithms to reduce latency and to improve throughput for our target networking servers, namely: First-Come First-Served (FCFS) and Shortest Position Time First (SPTF);
- Evaluation of a customized disk prefetching algorithm in terms of its impact on access latency and throughput using trace-driven simulation; and
- A measurement-based evaluation of three real disk I/O subsystem configurations using realistic workloads belonging to SDSA, WDSA, and NDSA classes.

1.5 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 provides the background of high throughput servers and related work on disk I/O performance evaluation and tuning. In Chapter 3, we present our approach of performance modeling and analysis of disk I/O subsystems for high throughput servers. Chapter 4 presents workload characterization, which is used in Chapter 5 for trace-driven simulation based analysis of two scheduling algorithms, with and without prefetching. In chapter 6, we present measurement–based performance evaluation of two high throughput servers of interest to our work: web and proxy servers. Finally, in Chapter 7, we draw conclusion about our research and outline the future direction of this work.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Introduction

This chapter discusses the background of our work and some of the related research efforts in this area. The performance analysis and modeling will mainly focus on high throughput servers such as web and proxy servers. We present brief overview of web and proxy servers as well as available I/O bus architectures. In addition, we discuss different efforts aimed at improving I/O performance.

2.2 High Throughput Server Architecture

Considering the high utilization of the web these days, server performance requirements in terms of high throughput continue to increase. Some of the high throughput servers that are often used on the Internet include: web servers, proxy servers, streaming media servers, ftp servers, etc. In this section, we shall discuss relevant server architecture and performance issues that have greatly helped in improving their throughputs.

The architecture of a general-purpose processor based computing platform is shown in Figure 0.1. The figure shows a system with its I/O devices connected to a high bandwidth internal system bus, through a lower bandwidth I/O bus and I/O bus controller. All of the

components indicated from individual I/O devices to the processor including the system software will affect the performance of any I/O tasks.

The rest of this section presents three types of high throughput network application servers: proxy servers, web servers, and multimedia servers.

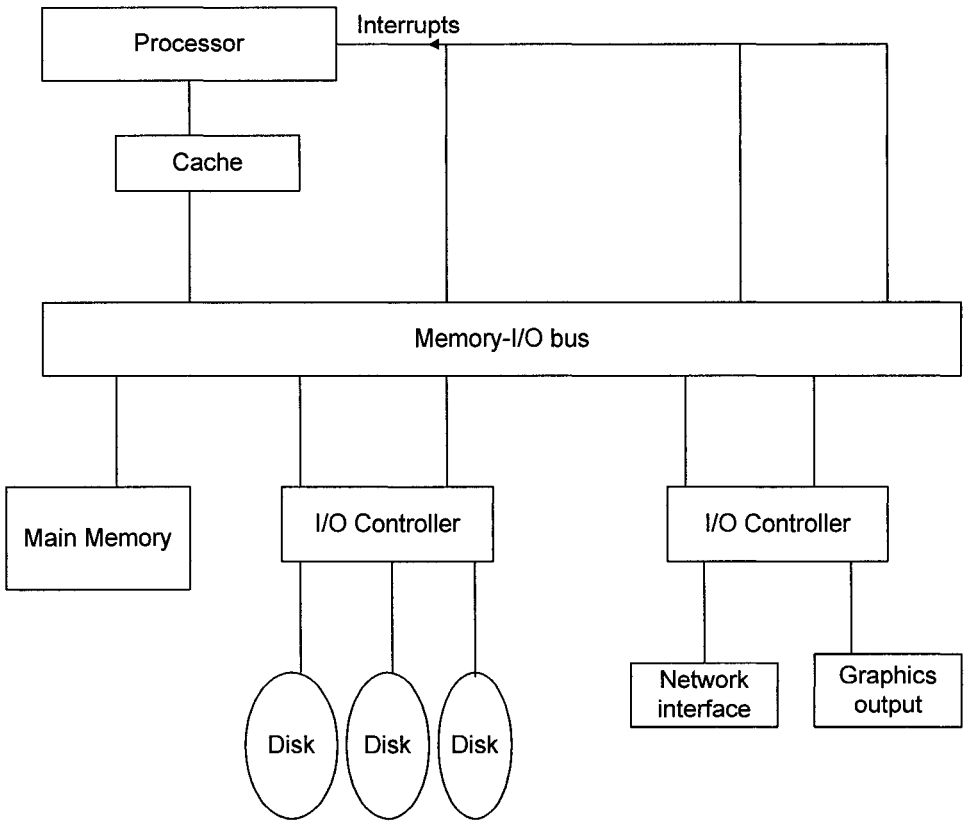


Figure 2.1: Architecture of a typical high throughput server with multiple I/O devices.

2.2.1 Web Proxy Server

A web proxy server is a software system that runs on a dedicated server platform and typically acts as a proxy for a client and forwards web traffic from servers to client and vice versa. The primary purpose is to save network bandwidth and to reduce user perceived network latency by filtering and caching web traffic [9]. Using the caching proxy may result in reduction of external network traffic and reducing the response time for request [61]. The explosive growth of web traffic in recent years, the increasing user demand for low latency service and high cost of bandwidth make the use of proxy servers a very attractive option for saving resources.

The proxy behaves similar to a web server; it listens for and responds to clients' requests. The methods by which a proxy server meets the client request depends on the type of proxy server that is being use. If it does not support caching the user request will be forwarded to the remote server that is specified by the URL in the client request. Once it receives the request it passes to the client. But if it uses cache, the cache will be searched for the requested document before passing request to the remote server.

High throughput servers often make use of disks for caching. A few research efforts report work on reducing the overhead of disk I/O in this area. Carlos Maltzahn and K. J. Richardson [13] noted that the most common bottleneck for large caching proxy server is disk I/O. In their work, they evaluated several ways to reduce the amount of required disk I/O for a proxy server. This was done by comparing the file system interaction of two existing web proxy servers and showing how design adjustment to the cache architecture

could reduce disk I/O and improve performance. They discovered that preserving locality of HTTP reference stream while translating them into cache references and use of virtual memory instead of file system for objects smaller than the system page size are effective strategies that also reduce disk I/O overhead on the server.

With continued growth in demand for large caches and higher performance for web proxy servers, the common bottleneck facing proxy servers is disk I/O. Since not all information being requested can stay in the memory, therefore a web proxy server will have to make use of disk I/O, thereby causing I/O overheads and reducing its performance.

2.2.2 Web Server

A Web server is a combination of a hardware platform, operating system, server software and contents. All of these components influence the performance of a web server. Its purpose is to provide document to the clients on-demand. The server listens on a designated port for a request from the client to establish TCP connection. Once a TCP connection is opened and the client makes a request, the server will then respond to the request. The response always includes a status code to inform the client that the request succeeds or fails. If the request is successful, a document is returned to the client otherwise a reason for failure is returned. Thousands of clients can make this type of request at the same time; hence, if the web server cannot withstand it, the clients will always be getting request failure.

Iyengar et al. [1] focus on a method of improving the web server performance in the situation that the CPU processing power is a limiting resource. Hu, J. C. et al [35] build a new web server mechanism and report several performance results showing that their proposed server is efficient and scalable. On the other hand, many researchers have examined performance of web servers mainly by benchmarking [59] to compare the web server hardware and to analyze the influence on response times by different document sizes. In addition, Slothouber, L. P [43] presents a study using web server modeling. Web server and the Internet is modeled as an open queuing network but the validation of the model is not provided in this study.

With widespread use of Internet, the performance of a web server to respond to large volume of requests is of critical importance. Disk I/O is the single most important aspect that limits the high throughput expected of such a web server.

2.2.3 Multimedia Server

For a multimedia server, disk requests require constant data rates and guaranteed services. Narasimha A. L. and Wyllie J. C. [52] discuss the impact of I/O requests on various I/O system components as well as the impact of disk scheduling algorithms on the performance of a multimedia server. They propose a hybrid disk-scheduling algorithm SCAN Earliest Deadline First (SCAN-EDF) for I/O of a multimedia system. This is a variation of the Scan disk access algorithm for use in a real-time environment where, in general, requests are served according to Earliest Deadline First. If two requests share the same deadline, they may be reorganized according to Scan. The SCAN-EDF combines the

advantages of seek-time optimizing schedule SCAN with optimal real-time schedule Earliest Deadline First (EDF). They show that deferring deadlines yields a better tradeoff than transferring large fragments of data. SCAN-EDF scheduling also works well with multiple data rates.

2.3 Server Disk I/O Controller and Mechanism

Disk drives consist of a disk controller and disk mechanism. Figure 2.2 indicates the disk drive mechanism in a server system. The controller assists in managing the data transfer between the mechanism and the host and maps any incoming logical addresses to the physical disk sectors that store the information. The description of how a small computer system interface (SCSI) disk drive works is described in [18]. The controller consists of a processor, a speed-matching buffer, a SCSI bus interface and sometimes a queue of incoming requests to allow request reordering. Whenever it receives a request, it decodes the request, maps the logical address to the physical one and then sends the request to the mechanism. If the request is a read, the mechanism accesses the data and places them in the speed-matching buffer. If the request is a write, the mechanism accesses the data from the buffer and writes them to disk. The speed-matching buffer can be used as a data cache.

The mechanism consists of a number of platters that rotate on a common axis known as a spindle, with both surface platters coated with magnetic media. Each platter surface has a head that reads and writes data by recording and sensing the magnetic flux variations on the surface of the platter. A platter surface is divided into tracks that look like concentric circles having different diameter. All the tracks on the spindle, that have the same distance from the center, make up a cylinder. The arm assembly of the disk drive moving all of the

heads until the heads is over the desired cylinder. On the other hand, the tracks are divided into sectors where each sector has the same number of bytes. The set of cylinders are partitioned into zones. All cylinders in the same zone have the same number of sectors.

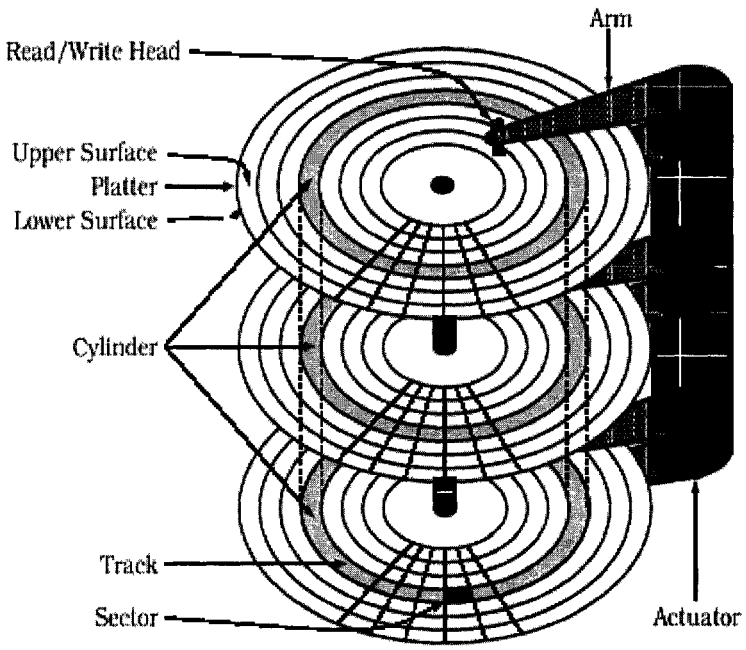


Figure 2.2: An illustration of the disk drive mechanism [18].

Whenever a disk services a request, it experiences mechanical delays of seek and rotational latency. Seek is the time for the actuator to move the disk arm to the desired cylinder while rotational latency is the time for the desired sector to rotate into position after the disk arm is on the correct cylinder. Seek operation in most cases goes through the following operations:

- Speedup where the arm is accelerated until it reaches half of the seek distance or a fixed maximum velocity
- Coast for long seeks, where the arm is moving at maximum velocity

- Slowdown, where the arm is brought to rest close to the desired track and,
- Settle where the disk controller adjusts the head to access the desired location.

Therefore a seek time can be approximated by a function such as

$$\text{Seek time} = \begin{cases} 0 & dis = 0 \\ a + b\sqrt{dis} & 0 < dis \le e \\ c + ddis & dis > e \end{cases} \qquad (2.1)$$

Where a, b, c, d, and e are device specific parameters and dis is the number of cylinder to be traveled. Equation 2.1 is used in Pantheon simulator to approximate seek time.

If a server system has multiple disks, the seek time seen by the server becomes average of seek time contributed by each of the disk as shown in equation 2.2.

Seek time for multiple disks = (1/n) * seek time for stand alone disk

$$= (1/n) * \begin{cases} 0 & dis = 0 \\ a + b\sqrt{dis} & 0 < dis \le e \\ c + ddis & dis > e \end{cases} \qquad (2.2)$$

Where n is the number of disks in the array and a, b, c and d are device specific parameters and dis is the number of cylinder.

2.4 I/O Bus Architecture

The increasing demands for highly available, high throughput servers pose a great challenge to computer architects to design new I/O bus architectures as solutions to these needs. A bus can therefore best be defined as an avenue for transferring data between the CPU and peripherals. The increasing demands for Internet server applications indicate that high throughput servers' I/O bandwidth requirements can no longer be satisfied by conventional methods. Hence, new technologies are introduced to help reduce or remove bottlenecks often created by conventional I/O architectures.

2.4.1 Peripheral Component Interconnect (PCI)

The PCI local bus was jointly developed by Intel and others to bring current and next-generation PCs to higher levels of system performance. PCI uses a 32-bit data path, 33MHz clock speed and a maximum data transfer rate of 132MB/sec. A 64-bit specification exists for future PCI designs. This doubles data transfer performance to 264MB/sec. TABLE 2.1 compares PCI with other types of I/O interfaces.

PCI is not a true local bus; instead, it occupies an intermediate level between the CPU local bus (processor/memory/cache subsystem) and a standard expansion bus (ISA, EISA, and Micro-Channel). The PCI bus is isolated from the CPU local bus by a PCI bridge/controller. The CPU can write data to PCI peripherals such as a hard drive and the PCI bridge/controller can immediately store the data in its buffer. This lets the CPU go on to its next operation rather than waiting for the transfer to complete. The buffer then feeds the data to the peripheral at the most efficient rate possible.

PCI also supports busmasters, intelligent devices that, when attached to a system bus, can gain control of the bus and perform tasks independent of the CPU. The CPU can also run concurrently with busmaster peripherals. In addition, it can operate independently when a PCI peripheral is active because of its buffered design.

TABLE 2.1: Comparing PCI with other I/O interfaces

Attributes	I/O Interfaces					
	ISA	**EISA**	**Micro Channel**	**VL-Bus**	**PCI**	**PCI-X**
Data Path Width	8/16	32	16/32/64	32/64	32/64	64
Data Bus Speed (MHz)	5.33/8.33	8.33	10.00	33/50	33.00	100/133
Data Transfer Rates (MB/sec)	5.33/8.33	33.00	20/40/80/160	132/264	132/264	800/1066
Data Rates Implemented (MB/sec)	5.33/8.33	33.00	40 (PS/2)/80 (RS/6000)	132.00	132.00	800
Number of Slots	0-8	EISA	0-8	0-2	0-4	0-4

2.4.2 Fibre Channel

Fibre channel was the first wiring technology adopted for connecting and managing large storage networks, designed to run on fiber optic and copper cables. It blends Gigabit-networking technology with I/O channel technology in a single integrated stack. Fibre channel combines many SCSI bus in parallel for better performance and can support Gigabit transmission speed in the range of 200MB/s, 400MB/s or 1GB/s especially when implemented on fiber optic cables, however, distance is the limitation here. The maximum distance is 10Km. With longer distance, the transmission rate of the fibre channel decreases. [27]

2.4.3 Internet Small Computer System Interface (iSCSI)

iSCSI is an internet protocol that allows SCSI data to be passed across an Ethernet network. SCSI data is contained within an Ethernet packet, limited in size by 1500Bytes. Compared with Fibre Channel, which is capable of sending data in sequences of up to 128MBytes, iSCSI brings functional advantages as opposed to performance.

2.4.4 InfiniBand

Server I/O technologies, particularly the PCI bus, have not kept up with the tremendous cost and performance improvements of processor, memory, storage and other related computer technologies. InfiniBand bus architecture is a new architecture designed to ease data traffic congestion among hardware devices, and is seen as a successor to PCI bus architecture.

InfiniBand is a high performance I/O technology used in networked computing platforms and defines the requirements for creating an InfiniBand network. The benefits of InfiniBand over existing technologies includes: more scale for growth, higher speed data transfer and easy integration with legacy systems.

InfiniBand can support bandwidths greater than prevailing I/O media such as SCSI, Fibre Channel and Ethernet. It uses IPv6 headers that support efficient interconnection between InfiniBand architecture fabrics and traditional Internet and intranet infrastructures [27].
One of the most significant changes that InfiniBand will bring is the removal of the I/O complex from the server. By replacing the bus architecture with InfiniBand technology, a

server has the ability to remove I/O from the server chassis, creating greater server density, and allowing for a more flexible and scalable data center, as independent fabric nodes may be added as needed.

InfiniBand-enabled servers connect seamlessly into existing Ethernet LANs and Fibre Channel-based storage networks. Unlike a typical ordinary server that connects directly to a LAN, a storage-area network (SAN), and sometimes to an Interprocess Communication network, through an I/O subsystem dedicated to each server/appliance. In TABLE 2.2, different I/O bus architectures are compared in term of data transfer rate.

The InfiniBand I/O architecture [33] is developed to address the weak link of the server's I/O subsystem, resulting in a more balanced server design. InfiniBand I/O achieves this goal by providing an industry-standard means for building the high-end features of direct channel architecture into the high-volume server. Prior to InfiniBand I/O, the benefits of channel I/O were realizable on high volume servers through proprietary solutions. InfiniBand I/O therefore blends the scalability, performance, and reliability that was previously reserved for high-end, higher-cost servers with the volume economies of being industry standard.

TABLE 2.2: Comparison of different I/O bus architectures.

	Clock rate (MHz)	**Nominal transfer rate (MB/s)**	**Maximum transfer rate (MB/s)**	**Bus type**
PCI	33/32 33/64 64/66 64/133	133 266 533 1066	1000	Shared
SCSI	33/32 33/64 64/66	80 160 320		
Fibre Channel		25 100 1000	100 (is mostly used, limited by distance)	Switch fabric
Infiniband		2500 – 30,000 per link	30,000 (depending on number of links)	Switch fabric

2.5 I/O Performance Evaluation

2.5.1 Benchmarks

There have been great developments in disk I/O subsystems that serve as the storage system for the high throughput servers. These developments often create new challenges for storage system evaluation. Benchmarks used in the evaluation process must evolve to comprehensively stress these I/O systems. For instance, disk arrays that serve as a means to improve disk I/O performance are able to service many I/O operations at the same time. Benchmarks, therefore, need to issue many simultaneous I/Os if they are to stress a disk array.

Because of the way files are accessed on high throughput servers by the clients, the file system often reduces the number of access to data on the disk I/O subsystem. Its behavioral and functional characteristics do affect the I/O performance from the server point of view. Due to the complexity in measuring this behavior, standard benchmarks are needed that can stress the server and measure performance.

2.5.2 Metrics

More than other areas of computer performance evaluation, disk I/O evaluation involves many varied types of metrics. In this section, we present an overview of some of the metrics commonly used in choosing and evaluating disk I/O systems. The value of most metrics depends strongly on the workload used. Hence, the dual emphasis on metrics and benchmarks.

The most basic metric for I/O performance is *throughput*. Throughput is a measure of speed—the rate at which the storage system delivers data. Throughput is measured in two ways: I/O rate measured in accesses/second and data rate measured in bytes/second or megabytes/second (MB/s). I/O rate is generally used for applications where the size of each request is small, such as transaction processing [8]; data rate is generally used for applications where the size of each request is large, such as scientific applications [23] .

Response time is the second basic performance metric for storage systems. Response time measures how long a storage system takes to access data. This time can be measured in several perspectives. For example, one could measure time from the user's perspective,

the operating system's perspective, or the disk controller's perspective, depending on what is viewed as the storage system.

To have better I/O performance, higher throughput and low response time from the disk is expected. Sometime, these two can be combined together. Such a combination will eventually show that higher utilization leads to slower response time. Many computer system analysts combine throughput and response time by reporting throughput at a given response time [8][56]. For example, the transaction performance processing council B (TPC-B) benchmark reports maximum throughput with 90% of all requests completed within 2 seconds [65].

Therefore, for a web server or web proxy server to perform very well, and have minimal overhead of I/O, the disk I/O subsystem should be capable of delivering high throughput at low response time.

2.6 Related Work

2.6.1 Trends in Storage Systems

The trend towards smaller, less expensive disks creates an opportunity to combine many of these disks into a parallel storage system known as a disk array. The concept of constructing an array of multiple disks has been used for many years for special purposes. It is becoming popular for general use only now.

The basic concept behind disk array is straightforward, it combines many small disks and distribute data among them. This increases the aggregate throughput available to an application. The array of disks can either service many small accesses in parallel or cooperate to deliver a higher data rate to a single large access.[20][28][55]

Disk array compensates for the lower reliability inherent in using more disks by storing redundant, error-correcting information. Current disk array research is focusing on how to distribute (stripe) data across disks to get optimal performance [26][57], how to spread redundant information across disks to increase reliability and minimize the effect of disk failures [28][29][60], and how to reduce the penalties associated with small writes in certain types of disk arrays [22].

Disk array improves throughput by using more disks to service requests. Requests that are serviced by a single disk, however, see the same response time. File caches, disk caches, and solid-state disks use dynamic random access memory (DRAM) to decrease response time. Caches can be placed in a variety of places in the system memory hierarchy [2].

In [37][47], response times for writes is decreased by writing the data to RAM, acknowledging the request, and then transferring the data to disk asynchronously. This technique, called write-behind, leaves the data in RAM more vulnerable to system failures until written to disk. As with any cache, read response time is decreased if the requested data is found in cache RAM.

Two storage metrics have been addressed, these are: throughput and response time. Some studies have focused on tape devices, which addresses how to migrate data from tape to

faster storage [2][25], how to increase tape throughput using striping [57], and how to decrease response time by prefetching and caching [29][34]. Our work will not consider tape devices.

Reported disk reliability has improved dramatically over the past ten years. The most common metric for reliability, mean-time-to-failure, has increased from 30,000 hours to 150,000-200,000 hours. This jump in apparent reliability comes mostly from changing the method of computing mean-time-to-failure and is not expected to continue improving as quickly [28].

Innovation is also taking place in the file system design. A good example of how file systems have improved I/O system performance is the Log-Structured File System (LFS) [48][36] LFS writes data on the disk in the same order that it is written. This leads to highly sequentialized disk writes and thus improves the sustainable disk write throughput. Although the raw performance in storage technology has improved much slower than processor technology, innovation such as file caches, disk array, robot-driven tape systems, and new file systems have helped close the gap to some extent.

2.6.2 Approaches for Improving the Disk I/O Performance

Several research efforts have addressed the issue of enhancing the performance of disk I/O subsystem and to fill the performance gap between the CPU and I/O subsystem. Most of these approaches are considered effective in improving disk performance and are

widely used. Others may be limited in effectiveness or excessive cost for general-purpose usage.

It has been observed that if corresponding improvement in I/O subsystem is not achieved as the processor is improving, the system performance may not achieve better overall performance improvement. N. Reddy [53] considers several options in designing an I/O system and analyzes their impact on the performance. He used trace-driven simulation for his study on disk I/O subsystem with a nonvolatile cache. He considered major parameters such as cache size, block size and fetch size. He noticed that decoupling the cache block size and the fetch size has significant effect on improving the performance.

2.6.3 Redundant Array of Inexpensive Disks (RAID)

Use of RAID arrays is a well-known way of increasing I/O throughput. They have built – in redundancy and support accesses to multiple disks by striping. Paterson et. al. [20] enumerates several ways of storing redundant data on disk arrays. They define a five level RAID taxonomy. RAIDs extend the mean time to failure of disk arrays by storing information redundantly. If a single disk in the array fails, the contents of that disk are reconstructed from the redundancy information. The RAID levels differ in the amount of overhead to store redundant data and the effort to recover data when a single disk fails.

Since Patterson et al defined the RAID taxonomy; many groups have analyzed the performance of RAID systems. The original RAID document defined minimal requirement for each RAID level. Within the taxonomy, there is a wide range of possible

implementations. Lee [44] studied eight different RAID parity placement strategies and compared their performance to simple disk striping. Chervenak and Katz [16] studied the performance of a RAID prototype developed at U.C Berkeley. Unlike previous studies that relied on simulation, their study was based on a real RAID prototype. Their study concentrated on the effects of cache, memory and interface contention on the performance of the RAID array. They reported that I/O performance optimization consists of both device level improvements and an improved caching interface.

Cabrera and Long [11][12] analyze a distributed disk array consisting of disks that were distributed across a local network. In addition to building a prototype, they perform a simulation study, which examines the relative performance of several different types of disks in their system. They showed that seek delay can affect performance as much as the number of disks in the array. For small requests, the disks seek time dominate service time and limits performance. For larger requests, however, seek delays are amortized and have a smaller effect.

Chen and Towsley [15] describe analytical models for a level five RAID and parity stripped disk array architecture. For a typical disk, they compare the performance of these two disk array architectures for several request sizes and request arrival rates. They finally conclude that RAID level five can offer better performance than parity stripped architectures when request size is smaller than the striping unit.

2.6.4 Disk Array

Kim proposes a byte interleaving disk array system and uses queuing models to analyze their advantages and drawbacks [42]. She concludes that disk arrays offer good performance only for light traffic. Livny [46] proposes a so-called declustering or striped configuration of several disk devices. Reddy [3] studies some performance tradeoffs for disk array systems by means of simulation. He also proposes hybrid combinations of synchronized and striped disks. Chen, et al, [55] study the effect of unit of interleaving for a RAID configuration. They deal only with highly stripped configurations [49].

For the purpose of this thesis, we use the term Disk Array to identify multiple similar disks. The goal is not to provide redundancy but to ensure high throughput by distributing the data across multiple disks.

2.6.5 Analytical Modeling of Disk Subsystem

It is well known that as the processor speed increases, I/O becomes a performance bottleneck. Therefore, several researchers have been looking for ways to improve disk I/O performance. Chris Ruemmler and John Wilkes [17] provide description of the current state of the art in disk drive technology and the underlying technology trends with particular emphasis on simulation features. They demonstrate how to construct accurate disk simulation models and provide quantitative information on how the different portions of the model contribute to its accuracy.

Using disk arrays have been described as one of the method that helps in improving the disk I/O performance. But having an analytic performance model of a disk array is very difficult to achieve. Lee E. K and Katz R. H. [26] develop and validate an analytic performance model for disk arrays. In their work, an equation for approximating disk utilization, response time and throughput is presented. The validation was done using simulation, which is used to investigate the error introduced by approximations used in deriving the analytic model. Their model is based on a closed queuing system with a fixed number of processes. Their model is limited to a disk component of disk system whereas a disk array has many components

There is a large body of work on analytical models of disk arrays. Bitton and Gray [21] present a model of seek time in shadowed disks, an early version of RAID1. Kim and Tantawi [42] analyzed the positioning time delays in asynchronous disk interleaving, a variety of disk striping where sub-blocks of a data block are placed independently of one another. Chen and Towsley, in their work [40][61], present an analytical queuing model of the response time in a disk array. Lee and Katz [26] present a model of disk striping under a simple synchronous workload. This model treats reads and writes similarly and assumes that the mean service time of a block request at a disk is known. Merchant and Yu [4][5][6] present several queuing models of response times for disk arrays using RAID1 and RAID5 layouts, both for normal and degraded modes; Thomasian and Menon [7] analyze the performance of RAID5 with distributed sparing in normal mode, degraded mode, and rebuild mode in an online transaction processing environment. Menon and Mattson [4][38] present response-time models of disk arrays using RAID 5 layout. Unlike most other models, these include some cache effects, albeit very simple ones. Disk drives

are not directly modeled as a part of these models. All of the above studies were conducted using analytical techniques.

While several of the above studies compare their results against simulation results, none of the simulations used are validated against real arrays, nor are any of the analytical results compared directly against real hardware. Many make simplifying assumptions, such as exponentially distributed disk service times, and most do not model the effects of either the disk or the array cache. None of the models above take into account the effects of the queue sizes on the disk service times. As a consequence, these pre-existing models typically mis-predict by a considerable margin, even when validated against a simulator [45]. Errors are likely to be even more significant when validated against a commercial array that implements many optimizations not contemplated by the simulator.

An analytical modeling effort is presented in [24], which solves the problem of having an analytical model that cannot predict performance. A new analytical model is developed that performs readhead and request re-ordering. This was done by developing performance models for components of the disk drive and composing them together afterward. The result is a model that was able to predict the behavior of a variety of real disk drives.

Mustafa et al. [51] report their work on reducing the likelihood of having to wait for an access that goes all the way down to the physical media. They propose an analytical method to predict the throughput of a disk array. An analytical model is built using a decomposition scheme of the internal device. One good thing about this analytical

modeling is that the models are less costly to develop and allow the users to concentrate on the relevant parts of the system. In addition, it is faster to generate predictions than simulations. The analytical models are validated using real life systems using measurements. The result of their work show reasonably accurate prediction. They use analytical modeling to predict the throughput in terms of the number of I/O requests serviced per unit time that a disk array could attain under a specific workload.

John O. and Fred Douglis [41] discuss several techniques for improving the disk I/O throughput performance. These techniques include: file caching, battery-backed-up caches and cache logging. They found that large memories and disk array are one way to eliminate the bottleneck problem that is often encountered in I/O subsystems.

2.6.6 Benchmarking

Chen and Patterson [14] propose an approach to I/O performance analysis known as self-scaling benchmark. This automatically and dynamically adjusts several aspects of its workload according to the performance characteristics of the system being measured. Some of the experimental factors of performance analysis are:

- Number of unique bytes: this is the number of unique data bytes read or written in a workload, which is essentially the total size of the data set.
- Percentage of reads: this is the number of read access relative to other type of accesses in percentage.
- Average I/O request size: this is done by choosing from a distribution with a coefficient of variance of one.

- Percentage of sequential requests: this is the percentage of requests that sequentially follow the prior request
- Number of processes: represent the concurrency in the workload, i.e., the number of processes simultaneously issuing I/O request.

2.6.7 Simulation of Disk I/O Models

Several simulation models of disks and disk arrays exist. DiskSim [31] and Pantheon [39] are two simulation environments for a disk subsystem, includes disks, buses, adapters, controllers and drivers. Both include disk modules that have been carefully validated against real disks, while the rest of the components are varied. The Pantheon disk simulator incorporates bus contention and other bus effects but no results have been published that describe the idle periods and head-limited bus transfer. RaidFrame [68] is a software RAID controller, which can also be used as a stand-alone discrete-event simulator for disk arrays. Hennessy et al [32] present a method for approximating the throughput of multiple disks on a SCSI bus by summing the seek time, rotational latency, and transfer time, and derating the performance by a bus contention factor derived from a general queuing model. The Parallel Disk Model (PDM) [67] is an abstract model for the design and analysis of algorithms on parallel disk systems. Cormen et al. [19] present an application level study of accuracy of the PDM. Worthington et al. [71] propose many experiments to measure the performance parameters of an individual disk drive.

2.6.8 Prefetching Techniques

Prefetching is considered an effective option to alleviating the I/O bottleneck and improving throughput performance. In order to achieve this goal, file system predicts the

disk blocks that will be needed in the nearest future and fetches them into the memory ahead computation. Shriver and Small [63] report that prefetching could improve the throughput as much as 50%. They proposed an analytic performance model for file system reads based on 4.4 BSD file system. This was parameterized by the access patterns of the files, file layout on the disk and underlying characteristic of the disk system. They finally propose the best way to tune file system and disk parameters in order to improve system throughput. Their work has some limitations in the sense that it only addresses file read traffic without considering file writes. Also the workload they used was limited to files that read from begin to end without think time between the read requests.

2.7 Conclusion

In this chapter, we presented background to our work on high throughput servers and described different studies that have been done in improving performance of disk I/O subsystems using different techniques.

CHAPTER 3

MODELING AND ANALYSIS METHODOLOGY

3.1 Introduction

This chapter presents our modeling and analysis methodology to address the problem of analyzing disk I/O subsystem of the high throughput servers, which was formally stated in Section 1.3. The key aspect of our analysis methodology is based on classifying the disk I/O accesses of high throughput servers into three types. This classification is presented in Section 3.4. This classification will lead to collecting workload traces (Chapter 4) that will be used to conduct trace-driven simulation (Chapter 5) and measurement-based evaluation (Chapter 6).

3.2 Research Methodology

The objective of our study is to model and analyze the disk I/O subsystem behavior for a high throughput server, since performance bottleneck most often lies in the I/O subsystem. Various configuration alternatives result in different severities of such bottlenecks for high throughput servers. Alleviating such bottlenecks is critical for obtaining high performance from these servers.

In order to achieve the objective, we use measurement-based technique to evaluate the disk impact on such high throughput servers. We characterize the workloads obtained

from measurement-based study and use the results in the trace-driven simulation In the trace-driven simulation, we will implement two disk scheduling algorithms, since such implementations are not possible on commercial operating system-based platforms.

We use an existing disk simulation tool, called DiskSim (version 3.0) for the simulation-based analysis to evaluate the impact of different access patterns observed from the measurement study. The essence is to see how they affect the disk subsystem and how we can improve disk performance. The performance of any disk contributes to a great extent to the overall server performance, therefore, a performance tuned disk subsystem, will surely provide higher server throughput.

3.3 Disk Access Patterns

Disk access patterns represent different ways in which clients make requests to a high throughput server. Depending on the application of a high through server, access patterns can vary. In contrast to access frequency that specifies temporal characteristics, here our focus will be on spatial locality of an I/O reference. Additionally, mutual distance among successive reference is of more interest rather than the absolute location of an I/O reference. Thus, we use access strides as a measure of spatial disk access patterns. We classify I/O accesses in terms of three patterns: *Sequentially Distributed Strides Access, Widely Distributed Strides Access and Narrowly Distributed Stride Access patterns.*

3.3.1 Sequentially Distributed Strides Access (SDSA)

Sequential access is simply that each process begins reading at an optional interleaving factor (or at the beginning of file). The interleaving factor can be adjusted such that regions of the file that may reside on different nodes can be accessed independently. Figure 3.1 indicates how sequential stride works with each process reading the file sequentially.

SDSAs are common to ftp servers that provide access to large files or programs for download by remote clients. In such cases, server opens the file from the beginning, sequentially reads block(s) of data, and sends them to the client over the network. In such cases, the disk accesses will have unit strides to read the (logically) contiguous data from the disk.

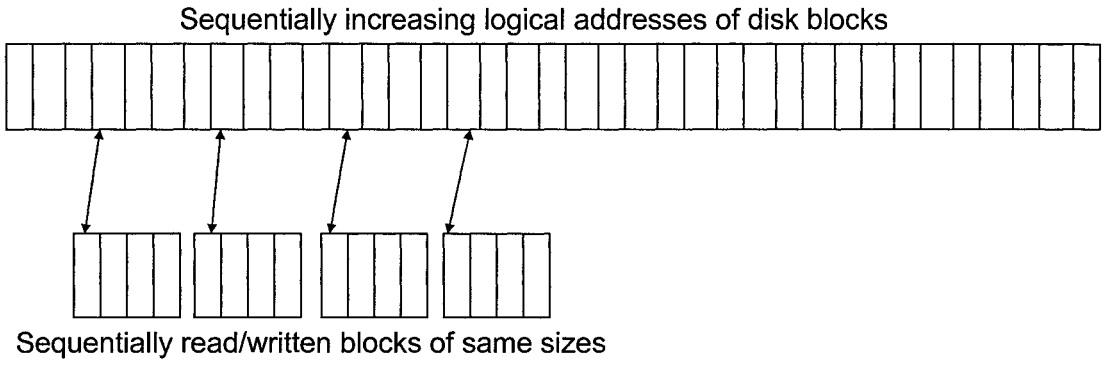


Figure 3.1: An illustration of SDSA disk I/O pattern.

3.3.2 Widely Distributed Strides Accesses (WDSA)

A class of high throughput server applications accesses the I/O in a manner that various spatial locations are equally likely to be the target. An example of such an application is a

web proxy server. Documents are often widely spread at various locations of a disk-based cache. Ignoring a small number of highly popular documents, client request will result in disk accesses that are widely distributed, i.e., various spatial locations are equally likely to be accessed.

In case of proxy server accesses, strides will be widely distributed only during reading of such data from I/O device. While writing data, for instance during a cache-fill phase, strides will be most likely of sequential nature. Therefore, unless otherwise specified, we will use the term of WDSAs in the context of read accesses to the disk by a web proxy application. WDSA patterns are illustrated in Figure 3.2.

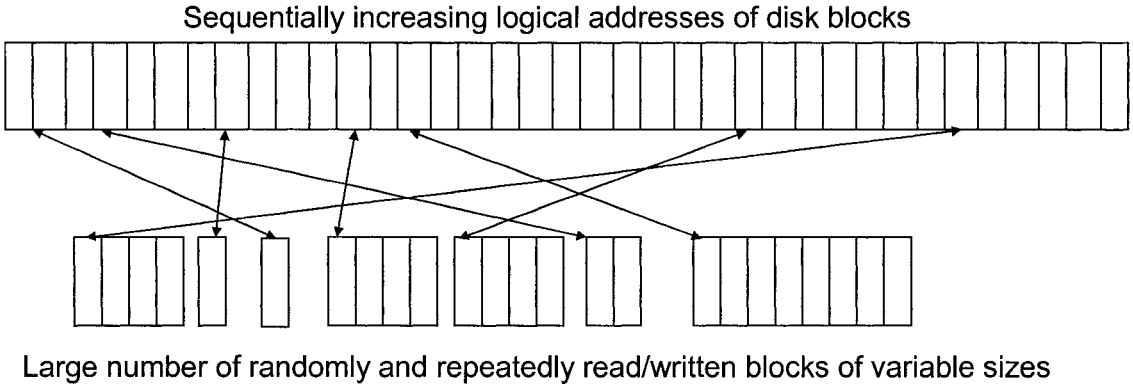


Figure 3.2: An illustration of WDSA disk I/O pattern.

3.3.3 Narrowly Distributed Strides Accesses (NDSA)

Not all applications require sequential and widely distributed accesses. Some high throughput server applications require distribution of spatial references that are in close proximity. One example of such an application is a web server. A web server typically

holds a smaller number of documents compared to a proxy server. Out of these documents, only few are accessed frequently. In such cases, disk locations storing data blocks corresponding to popular files will be accessed much more frequently compared to the others. Figure 3.3 presents an illustration of the NDSA pattern.

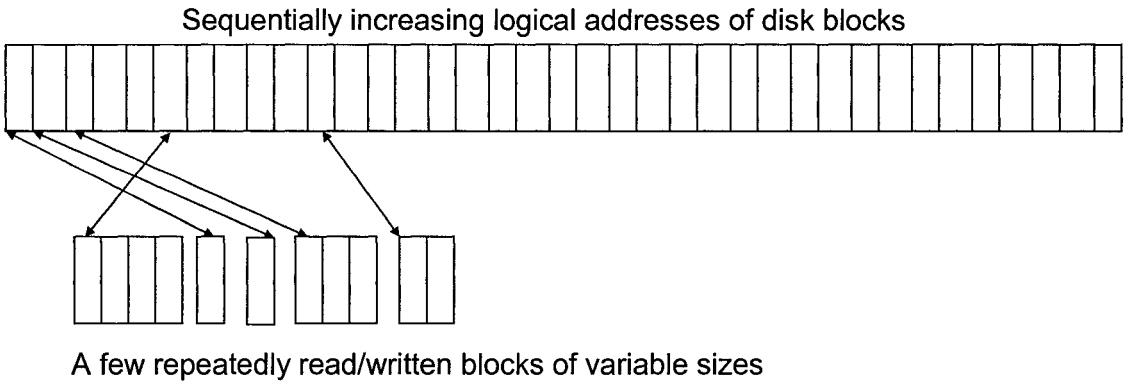


Figure 3.3: An illustration of NDSA disk I/O pattern.

3.4 Modeling of Disk I/O Subsystems

There are many different ways by which disk I/O subsystem can be represented in analytic or simulation models. The fundamental objective of any modeling study, including an I/O subsystem, is to determine which devices should be represented in the model. The major components of disk service time that are necessary in evaluating disk performance are: *seek time* (the time required to position the arm to the correct cylinder), and *data transfer rate*. In general, performance modeling can be categorized into two; these are open and closed system models

Our trace-driven simulation-based study uses a simple I/O subsystem model initially, and was later modified to incorporate other type of configurations such as disk array and RAID 5. We consider a generic networked server for the purpose of system modeling. In order to keep the characterization process simple, we consider that the server has only one hard disk attached to it. Various types of disk I/O transactions (discussed in the following section) arrive at the disk buffer and are scheduled according to First-Come First-Served (FCFS) or Shortest Position Time First (SPFT) disciplines. Read accesses return data to an appropriate location in the main memory. Figure 3.4 illustrates this model.

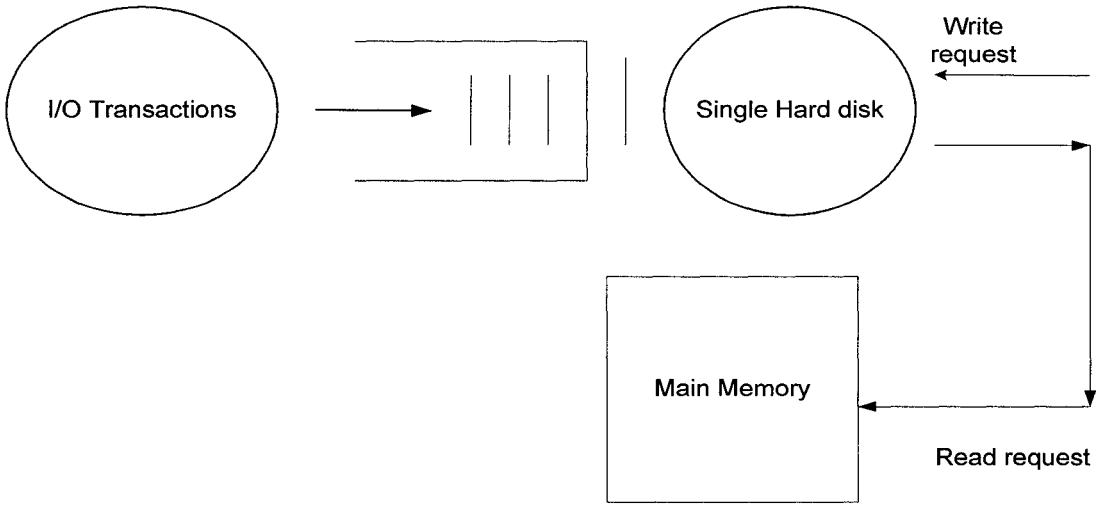


Figure 3.4: Disk I/O subsystem model for simulating various types of access patterns for a high throughput server.

3.5 Performance Analysis Methodology

In this section, we describe different methodologies that we employed in our work to evaluate the performance of disk I/O subsystems. Due to the advancement in computer technology and application areas, disk I/O performance is becoming a dominant factor in

the overall system performance. As a result of this, there is need to understand the performance impact of disk I/O on high throughput servers under different workloads.

3.5.1 Workload Characterization

Having a repeatable workload allows testing different alternatives under identical conditions. However, a real situation is not always repeatable, hence the need to study such an environment by observing the essential characteristics. Then use this to develop a workload model that can be used repeatedly and still give similar behavior as the real situation. It then becomes easier to control the effect of changes in the workload with the availability of the model.

3.5.2 Simulation

We used trace-driven simulation tool called DiskSim to see how the disk I/O subsystem behaves under different conditions. This simulator will be used to model different I/O subsystem configuration, namely single disk, RAID 5, and disk array systems. We make use of the traces that we obtain from a real single disk based server under various load conditions.

DiskSim is a highly efficient and configurable disk system simulator developed University of Michigan and enhanced at CMU to support research into various aspects of storage subsystem architecture. It is written in C and requires no special system software. It has different modules to simulate disks, buses, device drivers and array organizations.

The simulator is driven by I/O request traces obtained from our measurement-based experiment. It contains modules for most secondary storage components such as device drivers, buses, controllers, adapters and disk drives. The storage component can be configured in different ways depending on the objective. Figure 3.5 shows a diagrammatic representation of the simulator we used.

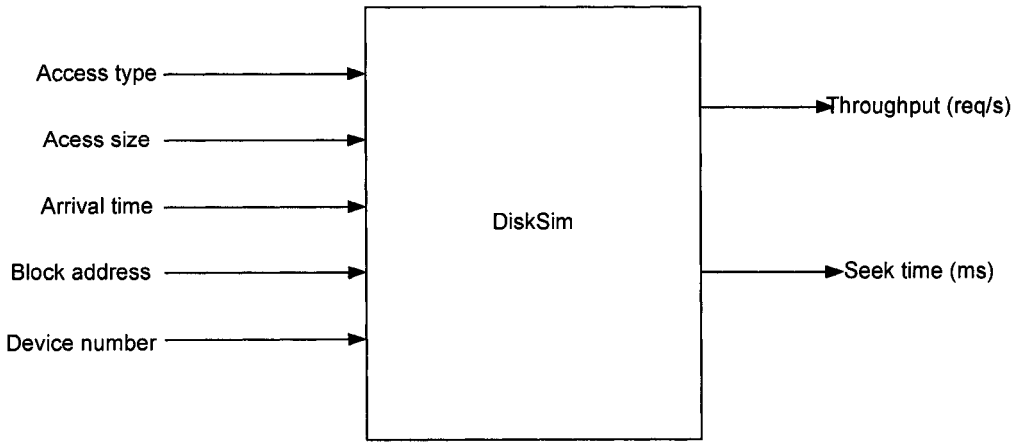


Figure 3.5: Simulation model representation

3.5.2.1 Trace Data Collection

To collect the traces for simulation analysis from the server machine, we enabled a low (kernel) level trace log collector available in the performance monitor of Windows 2000 server. Monitor collects the traces in the kernel memory buffer and transfers to a separate disk provided for logging once the allocated buffer is full. When the trace collection has ended, a post-processing tool called Tracedmp.exe translates the event traces into other traces that the simulator can understand. This is a command-line tool that produces a summary of event trace-log data.

TraceLog does not produce output that is readable without the use of an additional tool. The tool used to make the traces readable is TraceDmp. It takes TraceLog output and parses it into readable form. It can also poll real-time trace-buffer data and convert it to a .csv file. Collection of traces begins when the system is at its optimum level so that we can actually measure the performance of the high throughput server we are considering.

3.5.2.2 Comparison of Scheduling Algorithms

Using trace-driven simulation, we compared the performance of two known disk scheduling algorithms. These are first come first serve (FCFS) and shortest position time first (SPTF). In addition to this, we added a simple prefetching scheme to improve the performance of these algorithms.

The FCFS algorithm services requests in order of their arrival. This type gives low performance compare to any known scheduling algorithm. Since request are only service as they come into the queue, hence, offers no intelligence in servicing such requests.

The other scheduling algorithm takes into consideration the relative seek distance that is the current location of the read/write head. With this information, the scheduler can choose request with minimal positioning delay [10]. The SPTF always select the request that will incur the smallest positioning delay, however, not necessarily the one with the lowest address. After processing the first request, the media platter may rotate past the next sequential request if it does not have prefetching scheme in it.

3.5.2.3 Prefetching Scheme

Prefetching scheme is an effective way of removing bottlenecks in I/O and improving their throughput. In order to achieve this goal, file system predicts the disk blocks that will be needed in the nearest future and fetch them into the memory ahead of computation. In our case, we allow prefetching to the end of the track following the track containing the last sector of the read request. Assuming that the request was preceded by another read request in the previous track. This allows requested *reads* to stay a logical track ahead of the following request. This method is used along with the two basic scheduling algorithms. We found out that it greatly improves performance whenever prefetching is incorporated.

3.5.3 Measurement-Based Evaluation

Measurement-based evaluation is often regarded as an accurate representation of the true behavior of the real system. At the same time, it can serve as a useful technique to validate a simulation-based analysis. In our work, we conducted a measurement-based evaluation of the high throughput servers under consideration. Some of the outputs of these experiments are obtained as disk access traces and used in driving the simulation.

3.6 Conclusion

In this chapter, we have described the methodology used in our work to evaluate the impact of disk I/O on high throughput servers such as web server. We presented the disk I/O subsystem model that we used in characterization of our workload. In addition, we presented different performance evaluation techniques. We also presented different I/O modeling approaches and their applicability.

CHAPTER 4

WORKLOAD CHARACTERIZATION

4.1 Introduction

This chapter presents the workload characterization of disk I/O subsystems for high throughput network servers. Workload characterization is a process whereby a workload model is developed by observing key characteristics of a real environment. While the measurements obtained from real system are not repeatable for other experiments, workload characterization can help repeated experiments using simulation.

We gather performance measurements from disk I/O subsystems of various high throughput servers while stressing them with realistic workloads using industry-standard benchmarks. Using these measurements, we extract key parameters for the simulation model. We employ the parameterized model in Chapter 5 for analyzing various configurations of disk I/O subsystems of high throughput servers under three access patterns that were presented in Chapter 3.

4.2 Workload Parameters

We describe I/O workload to a high throughput server in terms of four parameters: access type, size, frequency, and stride. Following subsections explain the significance of these parameters.

4.2.1 Access Type

Disk I/O subsystem receives two types of requests: read and write. Read accesses provide the location on the disk from where data blocks need to be transferred to the host. A read access results in transferring the required disk block(s) to a specified main memory location. A write request carries data block(s) with it and accomplishes when data are written to the specified location on the disk.

4.2.2 Access Size

Access size represents the number of types transferred to or from a disk. Instead of using disk blocks, we use bytes, which is a generic unit of data transfer.

4.2.3 Access Frequency

Access frequency is related to the temporal characteristics of I/O access requests (read as well write requests). Frequency is expressed in terms of reads/sec or writes/sec transaction rate. Maximum access frequency (i.e., read/write transaction rate) is one of the manufacturer specified parameters of a disk.

4.2.4 Access Stride

We identify three types of strides in the context of three types of high throughput server applications: bulk data transfer from an ftp server, data transfer according to a heavy-tailed document popularity from a proxy server, and transfer of a limited set of documents from a web server. The corresponding access patterns are classified into three major access patterns: *Sequentially Distributed Strides Access (SDSA), Widely Distributed*

Strides Access (WDSA) and Narrowly Distributed Stride Access (NDSA) patterns, respectively.

In the following sections, we use the above stride based access classification to characterize I/O accesses using measurements from a real system. The system is a dual boot IBM Netfinity server based on a Pentium II 667MHz processor, 1GB RAM, one SCSI hard disk, and one 100 Mbits/sec network adapter. We use Microsoft Window 2000 Advance Server and Internet Security & Acceleration (ISA) proxy server [50] performance monitor tool to gather disk related performance statistics.

4.3 Characterization of SDSAs

In order to simulate SDSA pattern, we need to read or write a large file from/to the single disk. We use a CD ROM with about 712 MB long compressed video and transfer it to the disk while collecting measurements for this characterization. We used Windows 2000 copy/paste function to initiate this transfer. We made this choice for two reasons: complexity of using a large file with standard benchmarking tools (that are used for other two patterns) and involvement of network latencies with this relatively simple access pattern.

Figure 4.1(a) presents the read vs. write accesses to the disk under SDSA workload. As expected, the number of write accesses to the disk during our measurement greatly exceeds the read accesses. The small number of read accesses comes from the OS management and monitoring activities. Figure 4.1 (b) presents the strides of the write

accesses. All of the strides are equal to 65,536 bytes, which represent sequential accesses to contiguous disk (logical) addresses as data is read from the CD and transferred to the disk. Data is not transferred one byte at a time; instead, it is transferred as multiple disk blocks to the disk cache. Thus, the system monitors record the logical addresses with respect to writing those blocks of data. Clearly, no logical addresses are re-used during these operations. Disk simply accesses contiguous locations to store the incoming data.

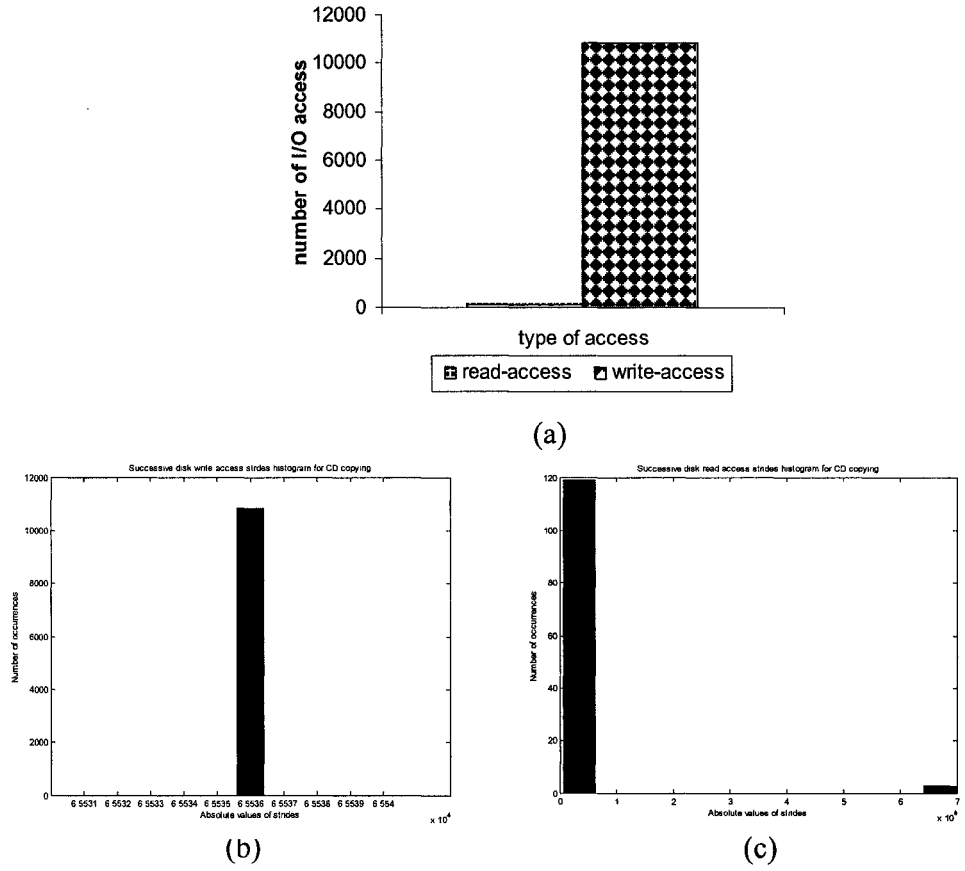
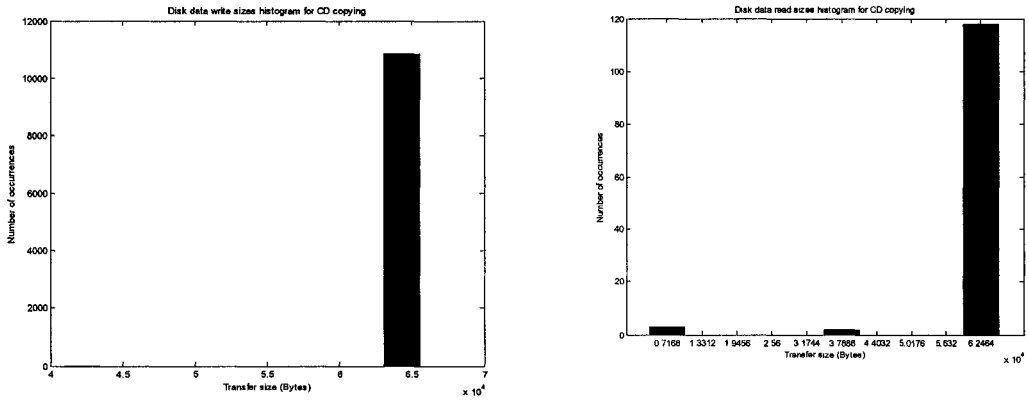


Figure 4.1: Distribution of (a) read and write accesses; (b) write strides; and (c) read strides to disk under SDSA workload.

Figure 4.2 presents the request size distributions for read and write accesses during CD copying. All of the write accesses transfer same amount of data: 64 Kbytes. This size is dependent on operating system assigned disk buffer and the maximum number of bytes that the buffer can hold before transferring to the disk.



Histogram of write accesses

Histogram of read accesses

Figure 4.2: Distributions of read and write request sizes to disk under SDSA workload.

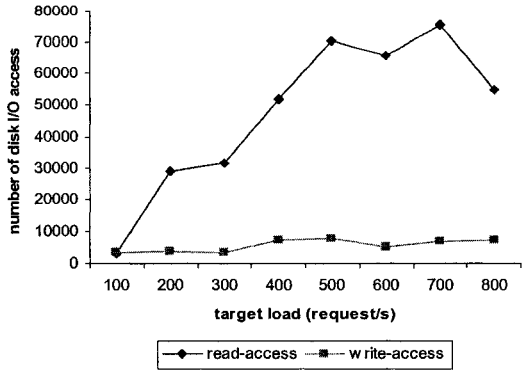
In order to characterize this workload, we ran the experiment many times, we took the average values. Using the information obtained from the experiment, we can summarize following characteristics about SDSA workloads:

- All access strides are identical;
- All accesses are to contiguous disk (logical) addresses;
- All accesses result in same amount of data transfer; and
- No disk address is repeated during such accesses.

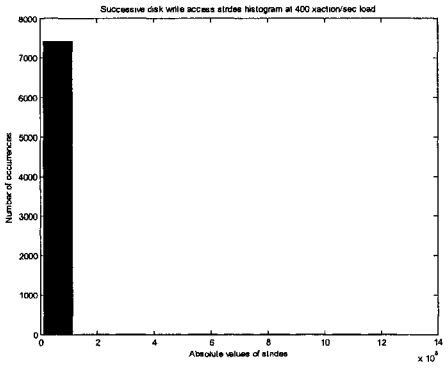
4.4 Characterization of WDSAs

We use our testbed platform to run Microsoft ISA proxy server under Windows2000 Server. In order to generate WDSA workload, we use an industry-standard web proxy benchmark known as web polygraph [69]. This benchmark has two phases: cache fill phase and maximum load phase. We defer further description of this benchmark until Chapter 6. WDSA workload is generated through accesses during the second phase, which accesses documents in a random fashion such that all disk locations are equally likely to be accessed.

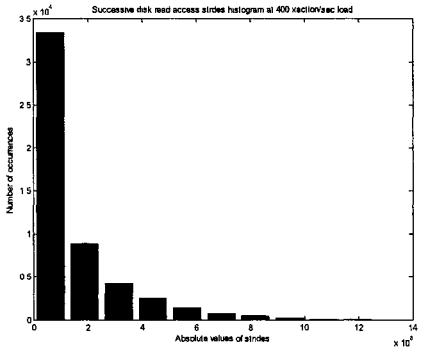
Figure 4.3(a) shows that the number of read accesses dominates the write accesses, especially at higher target loads. The percentage of IO reads is highest when the transaction load is at 400 request/s. In most cases, read accesses make up around 85% of all the accesses coming to the server. Moreover, the percentage was never lower than 50% for read access in all the load transaction cases considered. Figure 4.3 (b) shows that write access strides are small and similar to contiguous accesses. Figure 4.3 (c) indicates that the read accesses are non-contiguous and widely distributed. Hence from the experimental results, we can say that for a high throughput server such as web proxy server, there are more read access than write with non-contiguous WDSA pattern.



a



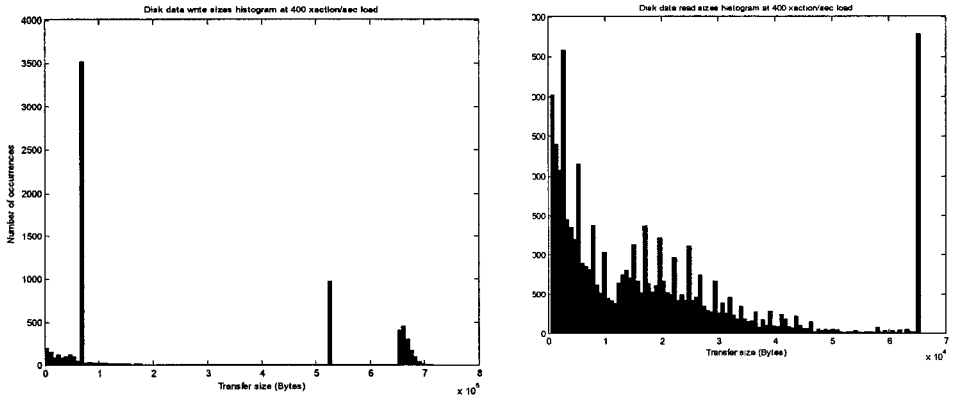
b



c

Figure 4.3: Distributions of (a) read and write accesses; (b) strides for write accesses; and (c) strides for read accesses under WDSA workload.

Figure 4.4 shows the distributions of request sizes for read and write operations under moderate transaction load such as 400 transaction/sec. The write access sizes show two modes and smaller number of accesses with large data transfers. On the other hand, read accesses show a large number of small sized accesses and a long tail. Ignoring the contribution of reading unusually large files from disk cache, this behavior is typical of a zipf distributed web page popularity model.



Histogram of write accesses

Histogram of read accesses

Figure 4.4: Distribution of request sizes for read and write operations under WDSA workload.

During the course of above experiments, we noticed the following characteristics of WDSA workloads:

- Read accesses dominate the write accesses;
- All the read accesses are non-contiguous with a random stride while the write accesses are in a contiguous manner with multiple sequential strides and three distinct clusters;
- Size of data transferred due to each access varies widely; and
- Repetitive accesses to same disk address are possible and frequent.

4.5 Characterization of NDSAs

In order to generate NDSA patterns, we run Microsoft IIS web server on Windows 2000 Server platform. We use Webstone benchmark, which is another industry-standard tool

for performance evaluation of web servers. More details of this benchmark will be presented in Chapter 6.

Figure 4.5(a) shows the frequency of read against write operations under NDSA workload. Numbers of reads dominate the number of writes with the distribution consistent with the Webstone specified frequency of accesses to various files (see Appendix B). Figure 4.5 (b) presents a histogram of successive read strides during this experiment. It indicates that all strides belong to two groups: first one consists of short strides while the second one consists of long strides. This is consistent with IIS webserver loading using standard Webstone load that allows reading certain files more compared to others. Figure 4.5 (c) presents the histogram for write strides during this experiment. These accesses are a result of OS activity and is not of interest to NDSA workload characterization.

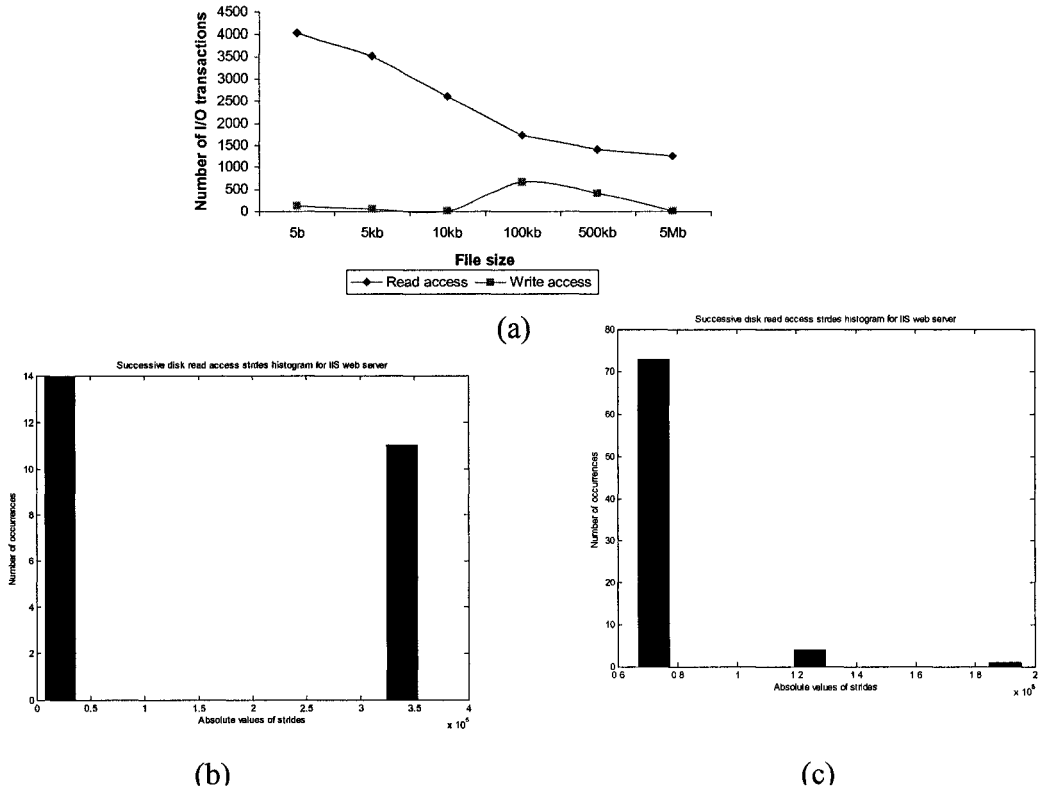


Figure 4.5: Distributions of (a) read and write I/O accesses; (b) strides for write accesses; and (c) strides for read accesses under NDSA workload.

Figure 4.6 presents the request size distributions for read and write accesses. Again, write accesses are not of primary interest for us to characterize NDSA workload. The read accesses mostly result in transfer of 64 KB blocks of disk data. While most of the Webstone clients access relatively small files (about 10 KB) with high frequency, the disk transfers multiple contiguous blocks to disk buffer of the OS to hide latency for subsequent addresses. This is the same behavior that was observed under SDSA workload where instead of individual writes, chunks of 64 KB were transferred to the disk buffer to hide latency.

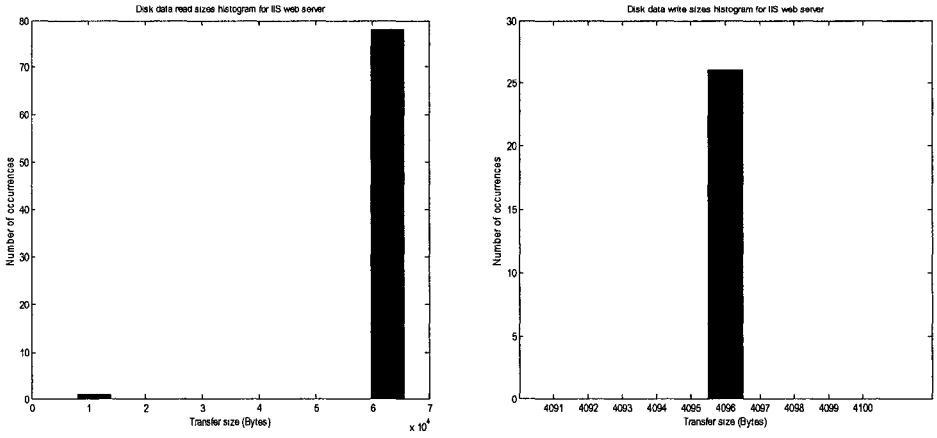


Figure 4.6: Distributions of read and write request sizes to disk under NDSA workload. Most of the read operations result in transfers of 64 KB blocks of data.

We can now summarize the characteristics of NDSA workloads as follows:

- Number of reads dominate the number of writes;
- Strides are confined to a very small group of addresses;
- Repeated accesses to same addresses are highly likely; and
- Amount of data transfer from disk to disk buffer is close to buffer capacity allocated by the operating system (OS) regardless the actual number of bytes requested by the application.

4.6 Model Parameterization

In this section, we summarize the workload characterization in terms of SDSA, WDSA, and NDSA patterns. This parameterization will be used for the simulation-based analysis presented in Chapter 5.

TABLE 4.1, TABLE 4.2 and TABLE 4.3 present the parameters for these three types of workloads. These parameterizations are not extensive. As we shall see in Chapter 5, our simulation analysis uses trace-driven simulation rather than event-driven simulation. Thus, the measurements that we gathered and presented in this Chapter are directly used as input traces to drive the simulation model. These traces will belong to SDSA, WDSA, and NDSA classes with characteristics presented above. Workload characterization effort is still useful as it helps us understand the disk I/O access patterns of high throughput servers.

TABLE 4.1: Disk I/O access parameters for SSA workloads.

Configurations	Type of access	Number of IO	Mean (bytes)	Median (bytes)	Min (bytes)	Max (bytes)
CD-copying	Read	123	63571	65536	4096	65536
	Write	10872	65534	65536	40960	65536

TABLE 4.2: Disk I/O access parameters for WDSA workloads.

Configurations	Type of access	Number of IO	Mean (bytes)	Median (bytes)	Min (bytes)	Max (bytes)
Single-disk Low	Read	3038	46345	65536	512	65536
	Write	3467	97913	73728	512	693248
Single-disk Medium	Read	51859	18102	13312	512	65536
	Write	7434	252350	65536	512	715264
Single-disk Heavy	Read	54884	15559	10752	512	65536
	Write	7412	229640	65536	512	713216

TABLE 4.3: Disk I/O access parameters for NDSA workloads.

Configurations	Type of access	Number of IO	Mean (bytes)	Median (bytes)	Min (bytes)	Max (bytes)
Low	Read	200	25452	4096	8192	104480
	Write	15	4096	4096	4096	4096
Med	Read	79	29156	16384	8192	65536
	Write	26	4096	4096	4096	4096
Heavy	Read	65	5960	8192	8192	65536
	Write	28	4096	4096	4096	4096

4.7 Conclusion

In this chapter, we have presented a workload characterization of disk I/O for high throughput servers. We classified these patterns in terms of three types of accesses: sequential, widely, and narrowly distributed strides accesses. We determined some key parameters that will be used for simulation models in Chapter 5. Measurements obtained as disk activity traces will be directly used for trace-driven simulations in Chapter 5.

CHAPTER 5

TRACE-DRIVEN SIMULATION BASED ANALYSIS

5.1 Introduction

This chapter presents the results of trace-driven simulation based analysis. We use a disk simulation tool known as DiskSim for this analysis. The simulation model uses the parameters determined in Chapter 4 as well as disk accesses traces obtained from a single SCSI disk based server (also from Chapter 4). Using these parameters and traces, we pose several what-if types of questions to the simulation model and analyze the results.

5.2 Scope of Simulation Based Analysis

Trace-driven simulation based analysis investigates two questions related to the disk I/O subsystem performance of a high throughput server:

- How does the disk I/O subsystem performance vary under various alternative scheduling and prefetching policies? and
- What is the performance impact of choosing one of three possible disk I/O subsystem configurations: single disk, disk array, and RAID?

The main goal of using disk scheduling algorithms is to hide mechanical delays associated with disk accessing requests. The first of the above two question considers two alternative scheduling algorithms: First-Come First-Served (FCFS) and Shortest Position Time First

(SPTF). FCFS algorithm services requests in arrival order. In contrast to FCFS, SPTF uses full knowledge of processing overheads, logical-to-physical data block mappings, mechanical positioning delays, and current read/write head location. Schedule for pending requests depends on shortest positioning time constraint. SPTF selects a request that incurs shortest positioning delay and thereby improving disk throughput and latency. The second question uses three access patterns to provide a comparison among three possible configurations: single disk, disk array, and RAID. Simulation model is capable of providing answers to these what-if questions.

We will consistently use two metrics to analyze the simulation results: disk throughput and seek time. Disk throughput determines what the maximum transaction rate that the server can support, especially with disk being a performance bottleneck. The seek time of a hard disk measures the amount of time required for the read/write heads to move between tracks over the surfaces of the platters to read or write to a particular sector. Seek time is one of the most important positioning performance specifications. Seek time is normally expressed in milliseconds; with average seek times for most modern drives today in a rather tight range of 8 to 10 milliseconds. Of course, in the modern PC, a millisecond is an enormous amount of time, because the system memory has a speed measured in nanoseconds, for example (one million times smaller). A 1 GHz processor can (theoretically) execute over one million instructions in a millisecond! Obviously, even small reductions in seek times can result in improvements in overall system performance, because the rest of the system is often waiting for the hard disk during this time. Therefore, seek time is usually considered one of the most important hard disk performance metric.

5.3 Scheduling and Prefetching Policies

In this section, we investigate the impact of two scheduling policies and prefetching on disk performance under WDSA workload. We use disk access traces from the measurement-based testing of Microsoft ISA proxy server under Web Polygraph benchmark [69] load (See Chapter 4) because these traces represent widely distributed strides accesses.

Figure 5.1 presents the throughput comparison of two scheduling algorithms: FCFS and SPTF at three types of server transaction loads: low, medium, and heavy. Throughput is obtained at three levels of read vs. write accesses: 0% read (all accesses are write), 50% read (equal number of read and write accesses), and 100% read (no write accesses). For both scheduling policies, throughput slightly decreases with increasing number of read accesses but remains relatively unaffected due to server transaction load. A more important observation, however, is the improvement in throughput that may result due to using SPTF scheduling instead of FCFS scheduling. In almost all cases, SPTF results in more than 20% increase of disk throughput in terms of I/O transactions rate. This result is intuitive because a scheduling algorithm that considers various physical disk characteristics should result in greater throughput. Throughput decreases with increasing number of read accesses because unlike a write access, a read access results in writing data to main memory. An access to main memory incurs additional latency and reduces throughput.

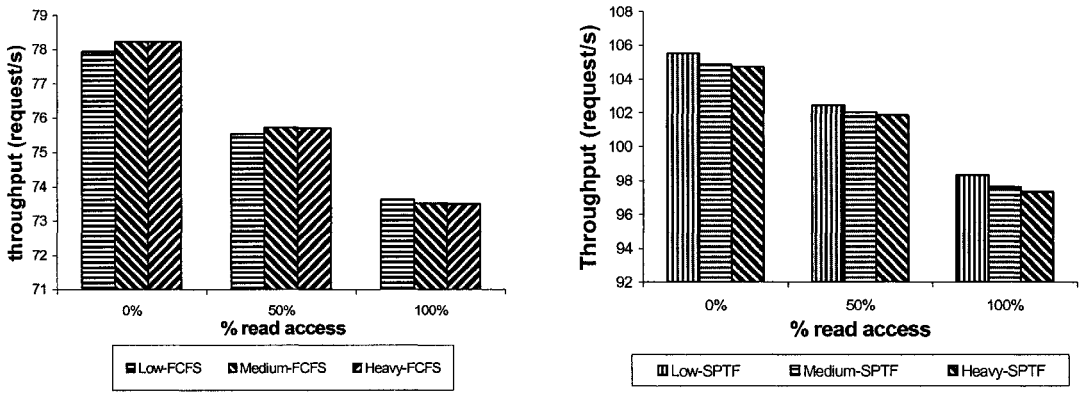


Figure 5.1: Disk throughput comparisons under FCFS and SPTF scheduling policies at various levels of read vs. write accesses and three levels of server loads: low, medium, and heavy.

In addition to choosing from FCFS and SPTF policies, an operating system can decide to employ prefetching using past history of I/O accesses. Figure 5.2 compares the disk throughputs under FCFS and SPTF scheduling with prefetching. Clearly, prefetching has a profound impact in terms of improving disk throughput when larger numbers of accesses are read accesses. Prefetching simply hides the latency by copying a disk block in advance of the access resulting in higher throughput. Prefetching is equally beneficial for FCFS as well as SPTF scheduling. The impact is more obvious when percentage of reads is larger.

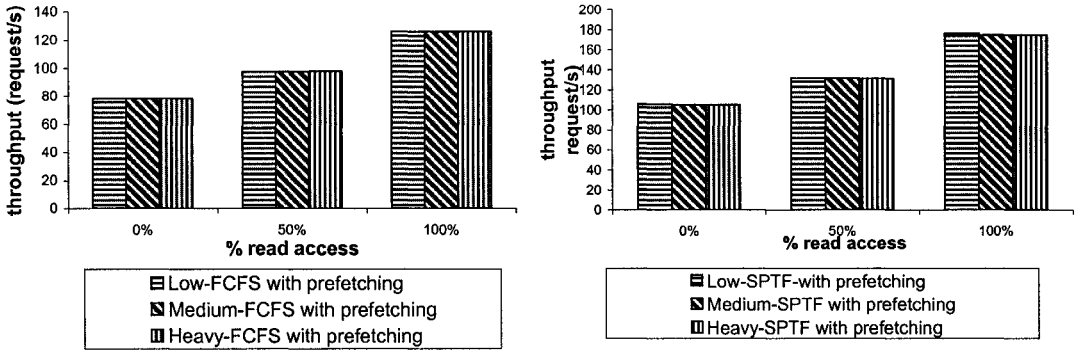


Figure 5.2: Disk throughput comparisons under FCFS and SPTF scheduling policies with prefetching at various levels of read vs. write accesses and three levels of server loads.

Figure 5.3 and Figure 5.4 present the disk seek time and throughput under two scheduling policies, both with and without prefetching. As expected, seek time is lower and throughput is higher under SPFT scheduling. However, the important observation is that prefetching can reduce the seek time by more than 10% under both FCFS and SPTF scheduling and increase throughput by more than 20%. Under SPTF scheduling, the value of seek time reaches in the range of 4 msec, which is only 50% of many commercially available disks.

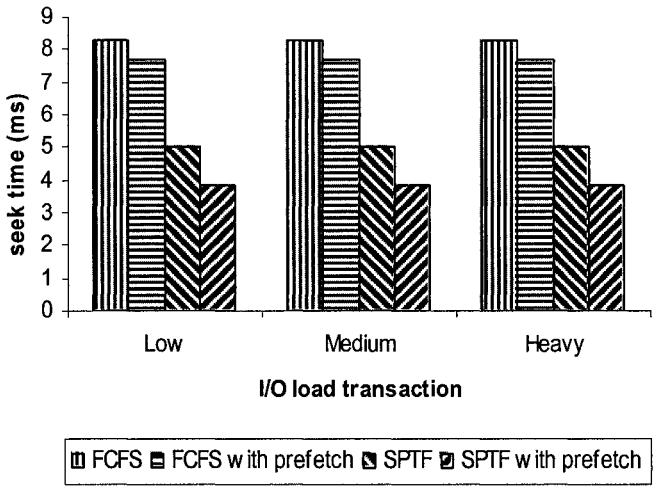


Figure 5.3: Average disk seek time comparisons under FCFS and SPTF scheduling policies with and without prefetching at various levels of read vs. write accesses and three levels of server loads.

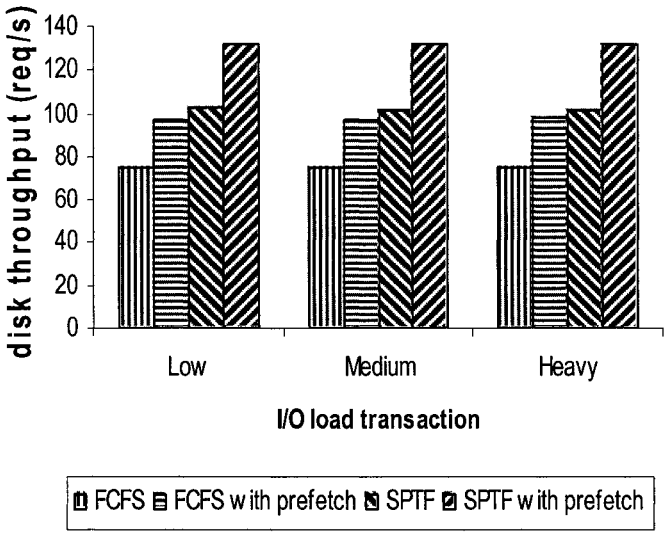


Figure 5.4: Disk throughput under WDSA for single disk comparisons under FCFS and SPTF scheduling policies with and without prefetching at various levels of read vs. write accesses and three levels of server loads.

Trace-driven simulation results indicate that prefetching combined with SPTF scheduling can significantly reduce the average seek time and enhance throughput. It means that

using SPFT with prefetching can potentially enhance throughput of I/O bound server applications by reducing the seek time by over 50%.

5.4 Comparison of Access Pattern Based Workload

In this section, we investigate the impact of disk I/O configurations on performance using three access patterns. We use both regular FCFS as well as SPTF scheduling with and without prefetching for these trace-driven simulations.

5.4.1 SDSA Workload

In this section, we use the traces obtained from CD to disk copying experiment (see Chapter 4) to drive the simulation model. These traces present a sequentially distributed strides access pattern.

Figure 5.5 presents the disk throughput and seek time under two scheduling policies with and without prefetching. Both plots indicate that the impact of prefetching on throughput and seek time is minimal under SDSA workload. The reason for this is that under sequential access, the disk fetches a whole block; once this is done, another is fetched. Therefore, we cannot really see any appreciable difference due to a prefetching scheme. However, it is clear that SPTF scheduling improves throughput as well as reduce seek time compared to FCFS scheduling for SDSA workload.

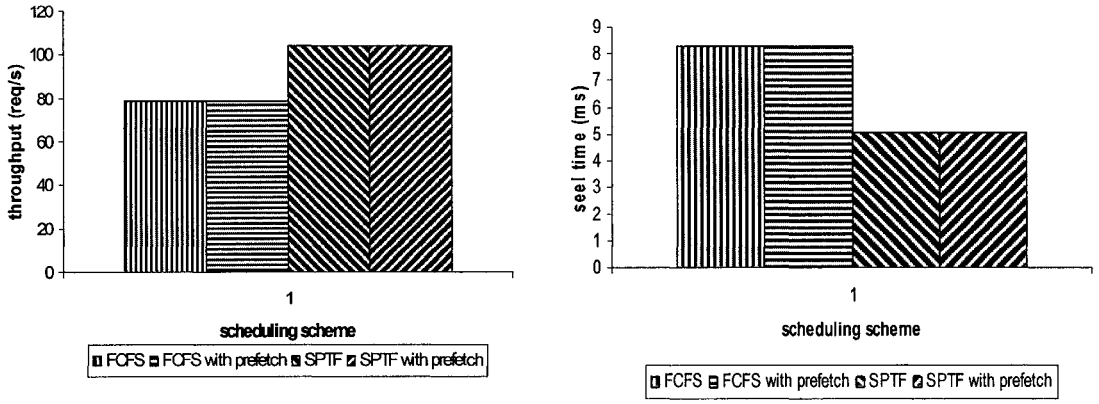


Figure 5.5: Disk throughput and seek time under FCFS and SPTF scheduling with and without prefetching for SDSA workload.

5.4.2 WDSA Workload

We have already presented the throughput and seek time results for trace-driven simulation under WDSA workload in Section 5.3. Figure 5.2 and Figure 5.3 indicate that throughput increases and seek time reduces when SPTF algorithm is combined with prefetching.

5.4.3 NDSA Workload

In this section, we use the traces obtained from loading Microsoft IIS web server through Webstone benchmark (see Chapter 4) to drive simulation model. These traces present a narrowly distributed strides access pattern.

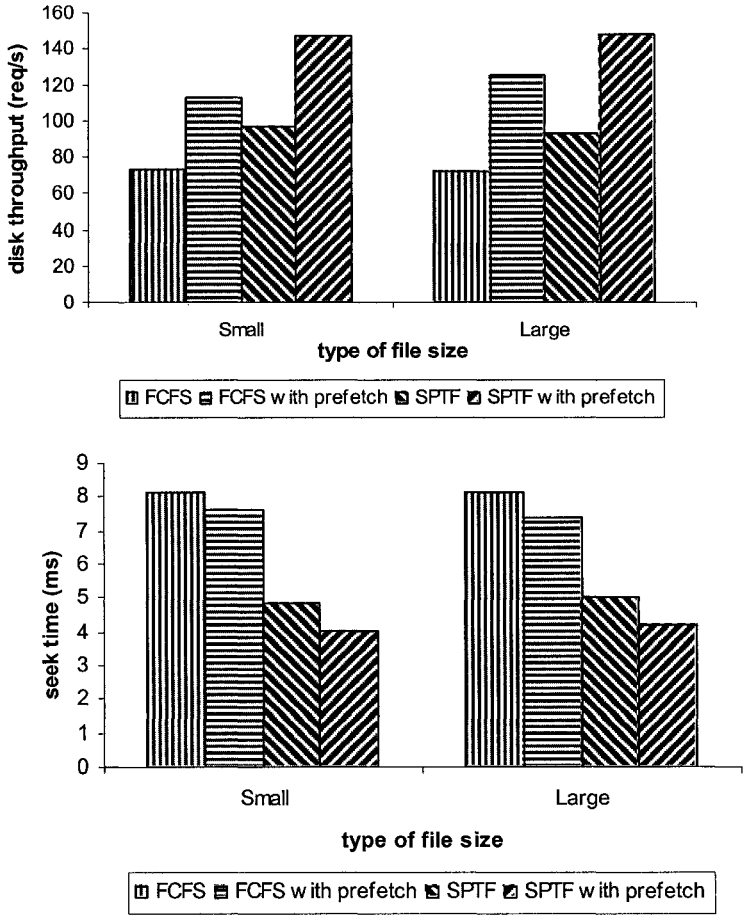


Figure 5.6: Disk throughput and seek time under FCFS and SPTF scheduling with and without prefetching for NDSA workload.

Figure 5.6 presents the disk throughput and seek time under two scheduling policies with and without prefetching using NDSA workload. As in the case of WDSA pattern, we obtain a higher throughput and lower seek time when prefetching is implemented with both scheduling algorithms. The maximum seek time obtained is around 8 msec, which is close to the actual seek time of the disk used. We achieve shorter seek time when we implement prefetching with the SPTF algorithm.

5.5 Comparison of Alternative Configurations

In this section, we compare three disk I/O subsystem configurations: single disk, disk array, and RAID. While a single disk based I/O subsystem with various types of access patterns is analyzed in Section 5.4, we present trace-driven simulation results of disk array and RAID configurations in this section.

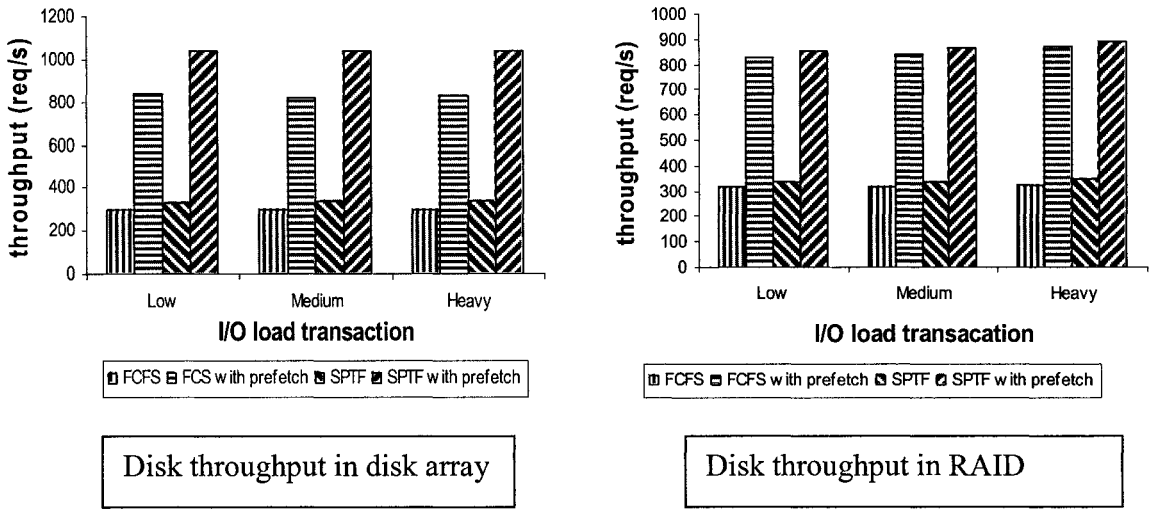


Figure 5.7: Disk throughput comparison in multi-disk and RAID. Throughput is 2 to 8 times higher compared to single disk configuration.

Figure 5.7 shows the disk throughput obtained for multiple disks and RAID configuration. From the figures, we observe that both configurations give very close throughput. However, these throughputs are 2-8 times higher than the case of single disk as we observe in Figure 5.4.

In Figure 5.8, we show the seek time obtained from our simulation when we implement disk array and RAID configurations. Here we considered three different cases of I/O load transactions. The average seek time over multiple disks (2 disks for disk array and 3 disks

for RAID 5 simulations) obtained are significantly lower than the case of single disk as shown in Figure 5.3. In case of disk array, the best seek time (0.5 msec) is 16 times lower compared to worst-case single disk seek time (about 8 msec). However, RAID 5 configuration does not show more than about 2 times reduction in seek time under WDSA workload. This is due to the unbalanced RAID configuration where one of the multiple (three, in this case) disks acts as a coordinator for other disks.

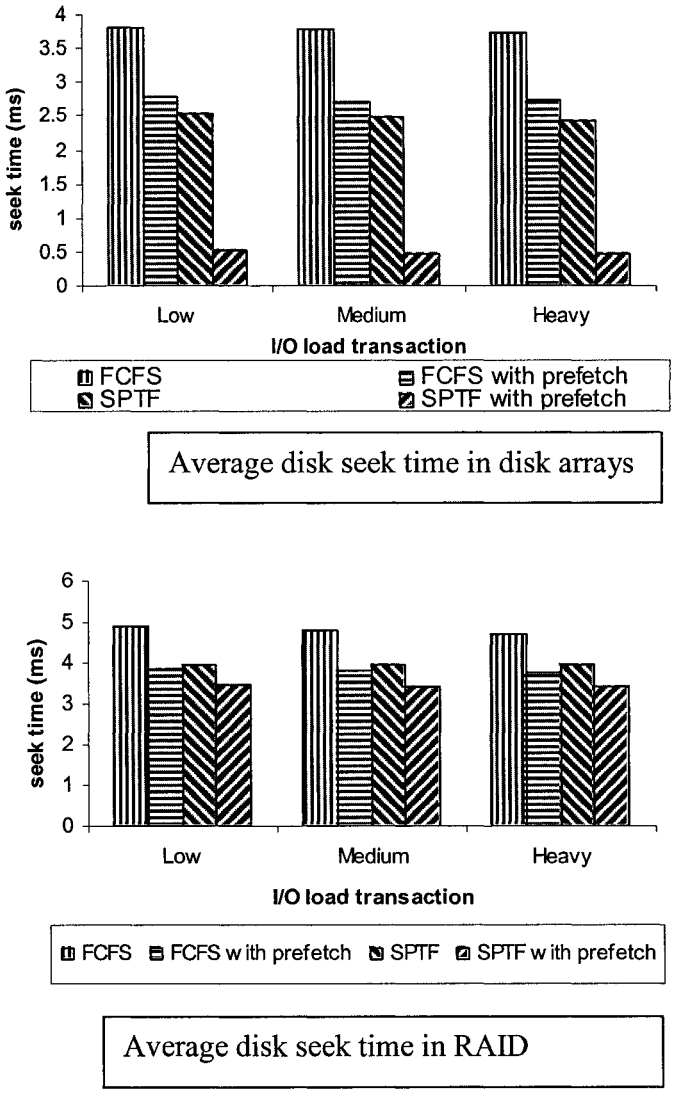


Figure 5.8: Disk seek time in array of disks and RAID 5 configurations.

5.6 Conclusion

In this chapter, we have presented the trace-driven simulation results for various what-if types of questions. We experimented with two different disk-scheduling algorithms. In addition to this, we added prefetching scheme to improve the disk performance. Under all access patterns, throughput improves and seek time reduces under SPTF scheduling with prefetching. When I/O bottleneck due to limited throughput of a single disk is removed in disk array or RAID configurations, trace-driven simulation results show significant throughput enhancement of the overall system.

CHAPTER 6

MEASUREMENT-BASED EVALUATION

6.1 Introduction

We investigated various what-if questions related to I/O subsystem performance of high throughput servers using trace-driven simulation in Chapter 5. In this chapter, we use measurement-based evaluation to verify some of the simulation based results.

In contrast to simulation-based evaluation, our analysis will be limited to FCFS disk scheduling without prefetching as our target server platform (Windows 2000 Server) does not support the other features implemented in the trace-driven simulation. However, we shall compare three I/O subsystem configurations of our interest: single disk, disk array, and RAID. We evaluate three classes of access patterns that have remained in our focus: SDSA, WDSA, and NDSA.

6.2 Experimental Environment

Before presenting any measurement-based evaluation, we describe our experimental setup in this section. Experimental setup consists of hardware platform and software tools. We present testbed and relevant software tools in the following subsection.

6.2.1 Testbed

The experimental testbed, as shown in Figure 6.1, consists of an IBM Netfinity server machine that hosts the high throughput servers under consideration. We conduct measurement-based experiments after booting the server under Windows 2000 Advanced Server operating system. Server machine uses a 1 Gbps 3Com Ethernet adapter.

There are eight client machines, each capable of booting under one of these operating systems: Window 2000 server, RedHat Linux and FreeBSD. The clients run on PCs, comprising of Pentium II 300 MHz, 96 MB RAM minimum and Pentium III 600MHz, 512MB SDRAM and 100Mbps NIC. Figure 6.1 gives the layout of the test bed used for our experiments. These clients generate workload for the server.

This setup employs a closed-LAN with a Cisco 1 Gbps multilayer switch (catalyst 3550). The high throughput servers run on a machine with Pentium II 667MHz, 1GB SDRAM, and customized storage combination depending on the experiment to be performed.

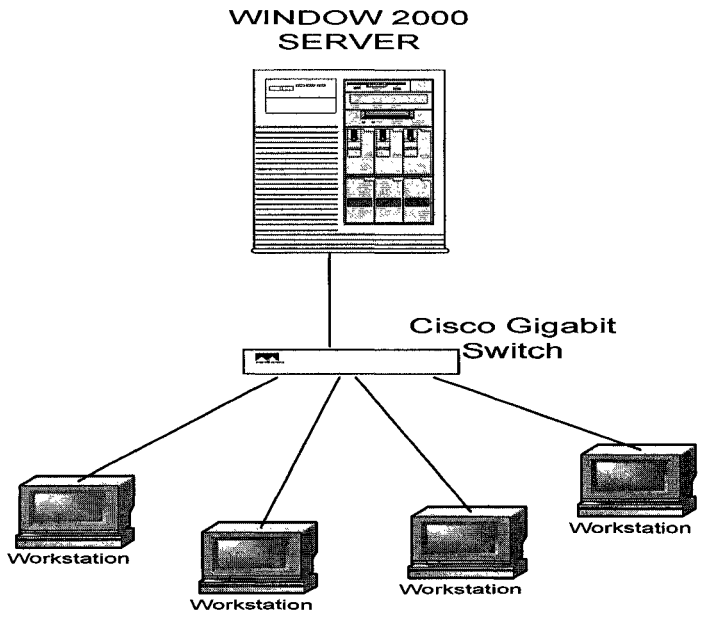


Figure 6.1: Experimental testbed for measurement-based evaluation.

6.2.2 Tools for Server Measurement

We collect performance measurements using some available software tools that can run on the server and client machines. Some of these tools are:

- **Windows 2000 performance monitor:** This is a Windows platform performance tool used to take the necessary measurements on the Microsoft ISA web proxy server and IIS web servers. In addition, we used this tool to collect traces that were used for workload characterization (see Chapter 4) as well as to drive trace-driven simulation (see Chapter 5).
- **Netstat:** It is a tool for measuring protocol-specific throughput and observing a host of network related activities. We monitor network connection status using this tool. This tool is available both under Windows as well as Linux and FreeBSD platforms.

- **Linux tools:** Some tools like vmstat, iostat and sar, are only available on Linux. Hence, they are useful for collecting measurements related to Linux operating system performance.
- **Netperf:** This tool is widely used for measuring network bandwidth between a pair of networked hosts. We often use this tool to ensure the proper functionality of our testbed before conducting any measurement-based experiment.
- **Tracedmp**. This tool is used to process tracelog which are in .etl format obtained under window 2000 server operating system.

  In the following sections, we present measurement-based evaluation of high throughput servers under three types of workloads: SDSA, WDSA, and NDSA.

6.3 Evaluation under SDSA Workload

Disk I/O accesses from sever types of high throughput servers can result in sequential addresses. Examples include ftp and video-on-demand servers. To avoid complexities of working directly with such servers, we copy a 712 MB long compressed video from a CD to disk. In this case, we restrict the measurement-based evaluation to a single disk. Results from this experiment were used in Chapter 4 for workload characterization and capturing access traces that were used in Chapter 5.

6.3.1 Experimental Design

In this section, we described our experimental design as well as the factors and metrics we used for running our experiments.

6.3.1.1 Factors

To carry out our experiment, we considered only one factor and that is the file size. We wanted to know how a high throughput server behave under a very large file size.

6.3.1.2 Metrics

Under this scenario, our choice of performance metrics is meant to allow us to evaluate the disk throughput that the server can deliver under certain workload conditions, especially when a client is downloading large document from servers like an ftp server. Some of the metrics we use in our experiment are as follows:

Disk throughput: this is to determine the response of the server under the workload condition we described earlier. Throughput in this case is measured as transfer/s.

Disk transaction rate: It is a measure of how fast the web server receives client requests and responds to the clients after processing. Transaction rate is measured as transactions/sec or connections/sec.

Server CPU Utilization: This refers to the overall system CPU time that is spent in doing useful work.

6.3.2 System Performance

In the case of sequential access scheme, we copied a very large document size from a CDROM device unto the disk system. We notice that the CPU was consistently 100% utilized throughout this process. This shows that sequential access scheme use more CPU cycles during the disk I/O write access than read under the scenario considered.

Figure 6.2 shows that the CPU was consistently 100% before it eventually dropped after the copying has ended.

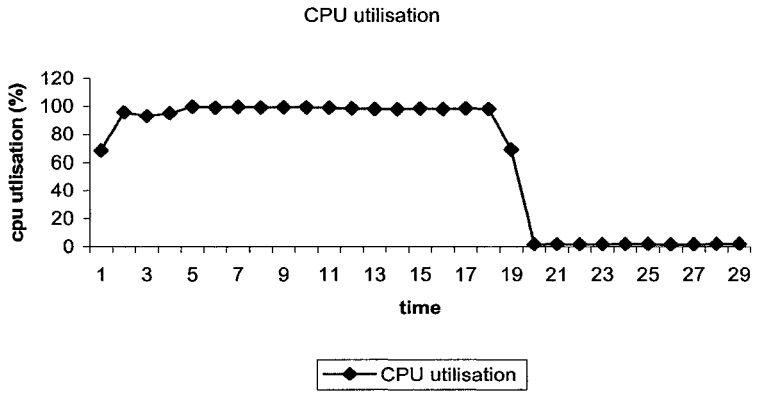


Figure 6.2: CPU utilization for during SDSA workload based I/O operations.

6.3.3 I/O Performance

Figure 6.3 indicates a uniform disk throughput in transfer/second close to the pattern observed in the CPU utilization graph. The throughput suddenly dropped to zero after a while. This is when the system has finished writing the document to the disk system.

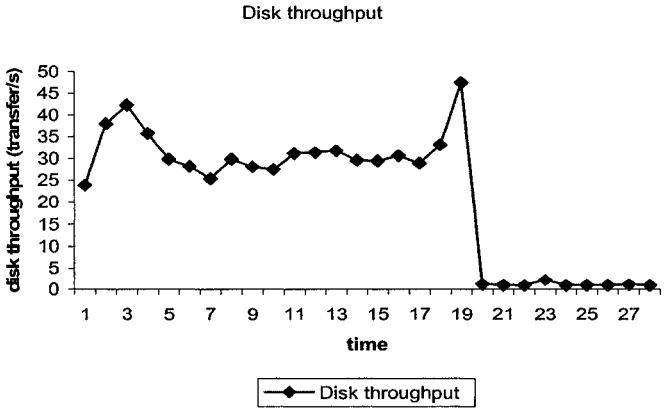


Figure 6.3: Disk throughput for SDSA workload.

Figure 6.4 shows the disk transaction rate for sequential access scheme. It indicates both read and write transaction rates. This shows that the number of writes was greater than the number of reads. Since the document size is large, read transactions are expected to be low.

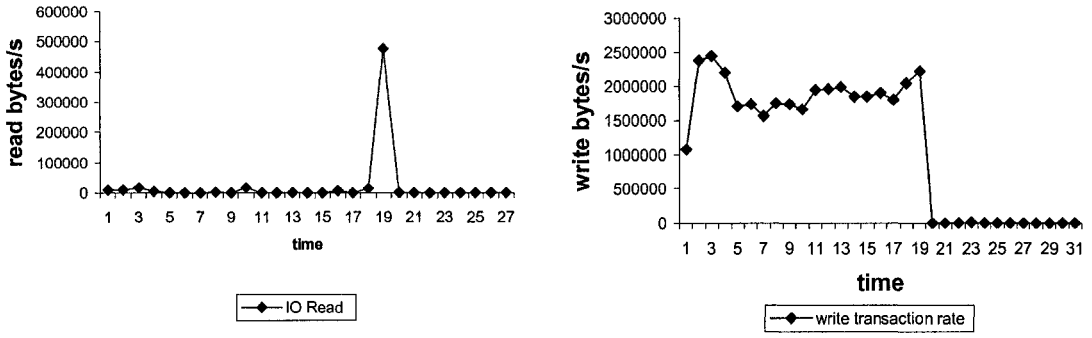


Figure 6.4: Disk I/O transaction rate in byte/s; (a) reads (b) writes under SDSA workload.

Figure 6.5 presents the request frequency in terms of read transactions/sec and write transactions/sec. As expected, the number of write transactions is significantly larger compared to reads. System can sustain up to about 30 write transactions/sec due to large amount of data (64 KB), which is written during every disk transaction.

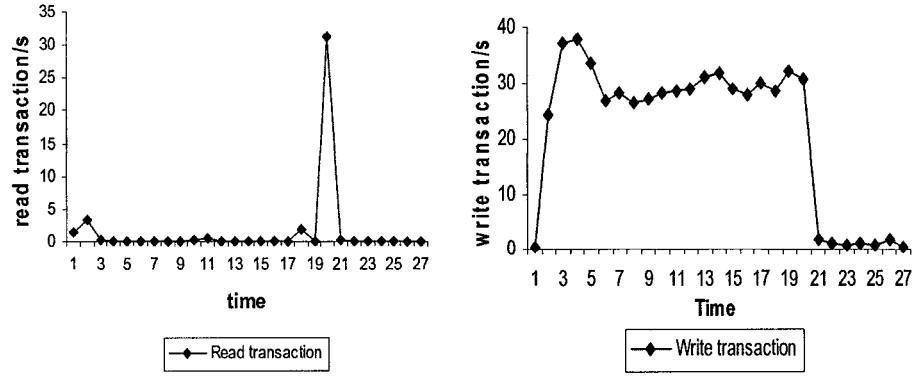


Figure 6.5: Distributions of read vs. write request rates to disk under SDSA workload.

6.4 Evaluation under WDSA Workload

Disk I/O accesses from a typical web proxy server can result in widely distributed strides access pattern. In this section, we use Microsoft ISA Proxy server on Windows 2000 Server platform and load it with Web Polygraph [69] benchmark to generate WDSA workload. Web Polygraph uses a standard workload, called Polymix-3 for proxy server evaluation at a target load rate expressed in terms of HTTP transactions per second.

Our goal is to accomplish three objectives by running measurement-based tests under WDSA workload:

- To generate traces using single server disk for using as input to trace-driven simulation (these were used for analysis presented in Chapter 5);
- To evaluate I/O performance in the high throughput server using three configurations: single disk, multiple disks (disk array), and RAID5; and
- To verify the corresponding simulation-based analysis results.

6.4.1 Experimental Design

In order to conduct these experiments, we vary two main factors: target HTTP transaction rate and disk configuration. Standard workload from Web Polygraph benchmark (Polymix-3) [69] was used to test ISA proxy servers under identical conditions. For this proxy server, we gradually increase the target transaction rate from 120 to 800 transactions/sec with an increment of 400 in each test to determine the peak transaction rate.

The Microsoft ISA proxy was tested in stand-alone mode with single server. Finally, we evaluated the performance impact of downloading large files onto the disk subsystem through ISA proxy server.

6.4.1.1 Factors

To carry out our experiment, we considered two basic factors, in order to evaluate their impact on the high throughput server. These factors are: (1) three different disk configurations, that is, single, multiple and RAID system; and (2) the target load that server can handle for each test. The Hit rate of the proxy server is fixed by Polymix-3 to more than 50% and the proxy servers are expected to meet that hit rate when load is within their normal operating range.

6.4.1.2 Metrics

We use two sets of performance metrics: the first focuses on proxy server performance and the second set of metrics focuses on the system performance (of proxy server). Metrics to evaluate the proxy server performance include:

- HTTP request rate (or connection rate);
- Throughput (in Mbps);
- Hit ratio; and
- Average response time (per transaction).

Metric related to the system performance include:

- CPU utilization of the proxy server host;
- Disk utilization, disk transaction rate, and disk queue length.

6.4.1.3 Benchmarking Tool

In order to benchmark proxy server, the most extensive public-domain benchmark is Web Polygraph [69]. This generates robots PCs that can make connection to the server through the proxy.

Web Polygraph [69] is a public-domain benchmark and is extensively used by Proxy Caching industry to evaluate the performance of various products under realistic workload scenarios. This benchmarking tool simulates web clients, servers, and a large number of client/server IP addresses to stress a proxy server. It provides standard workloads (e.g., Polymix-3) that describe the content type, client and server behaviors, document popularity distribution, target proxy server load, target proxy server hit rate, and a number of other configurable parameters to make the test realistic. It simulates a user as a "robot" that can open multiple connections (client agents) simultaneously that access origin servers through proxy. Polymix-3 workload consists of two phases of heavy load (top1 and top2) and an initial (fill) phase to warm-up the cache.

6.4.2 Server Performance

In order to evaluate the disk I/O performance under USA workload, we present the results of Web Polygraph tests on Microsoft Internet Server Accelerator (ISA) proxy server. The experiment used an advanced Windows2000 Advance Server platform that runs ISA proxy server. Hardware platform is a PIII 677MHZ processor IBM Netfinity server with three SCSI hard disks (one for OS and two for caching), 1 GB RAM and 1000 Mbps NIC

and connected to (simulated) clients and servers through a Gigabit Ethernet switch (Cisco3550).

We modified several parameters in the Windows2000 Advance Server registry to enable it to become a highly available server with possibility of handle thousands of simultaneous connections. Appendix A.1 provides a list of specific registry parameters, their location, and their modified values.

Figure 6.6 presents server performance in terms of transaction rate, data throughput (in Mbps), target document hit ratio, and average transaction latency with respect to offered load (in terms of transactions/sec). In this figure, server performance is presented in terms of actual transaction rate, bit throughput, achieved document hit ratio, and average transaction latency with respect to varying offered HTTP transactions load.

We notice that the throughput increases linearly from target load of 120 transactions/sec to 600 transactions/sec and thereafter starts to decrease. This is because of the server limitation, which can hardly support any transactions beyond 600 transactions/sec. We observe similar trend in all the three configurations; however, the RAID configuration provides higher transaction rate and throughput with heavy offered load (in the range of 800 transactions/sec). This observation is consistent with simulation-based results under heavy load (see Section 5.5). Note that the simulation results were presented in terms of disk throughput while here we present overall server throughput.

We notice a decrease in the measured hit ratio delivered by the server as the transaction rate increases. With higher transaction load, the rate at which the server responds to client requests drops. Therefore, the document-hit ratio is expected to reduce under overload. Three configurations have similar behavior of transaction latency except at large offered load. Latency increases significantly when offered load reaches 600 transactions/sec. A single disk configuration appears to cope well with high load compared to other two configurations. In disk array and RAID configurations, requests are made to different disks at the same time; hence, we expect the latency to respond to increase under heavy load.

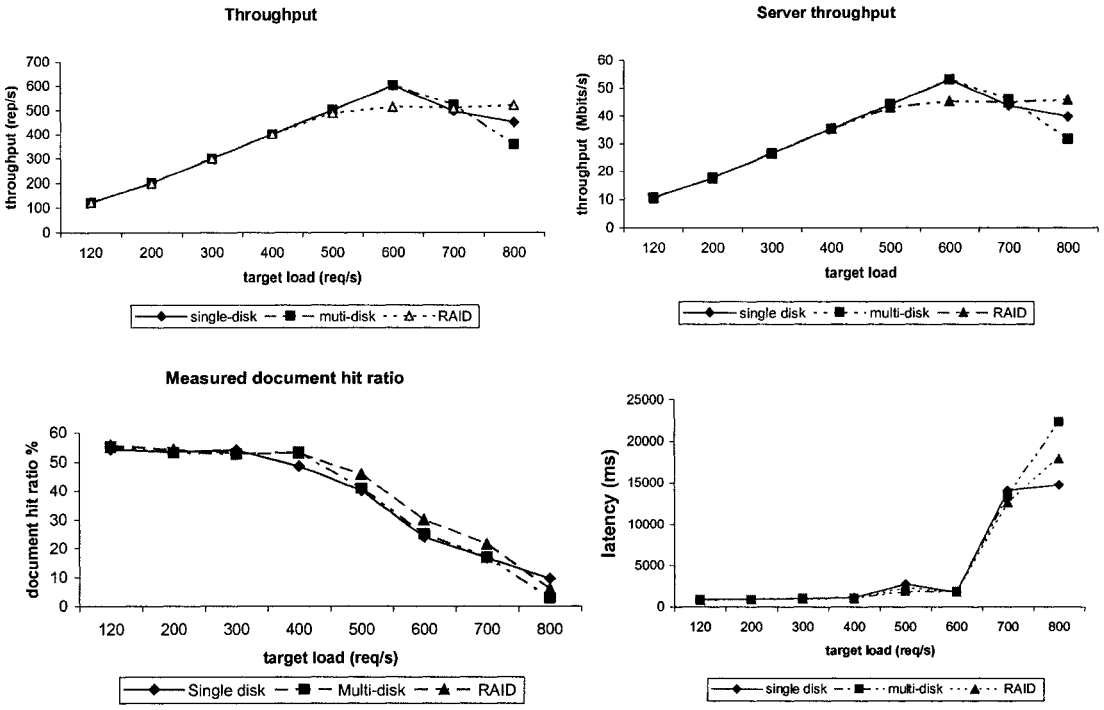


Figure 6.6: Server performance measurement under WDSA workload with varying target load and three configurations: single disk, disk array, and RAID5.

6.4.3 System Performance

Figure 6.7 presents the average CPU utilization during the maximum load phase (top2) of each polygraph test under three configurations: single disk, disk array, and RAID. Although three configurations are similar in terms of CPU utilization, RAID configuration shows higher CPU utilization between 400 to 600 transaction rate. This is largely because the RAID configuration adds additional fault tolerance to the system.

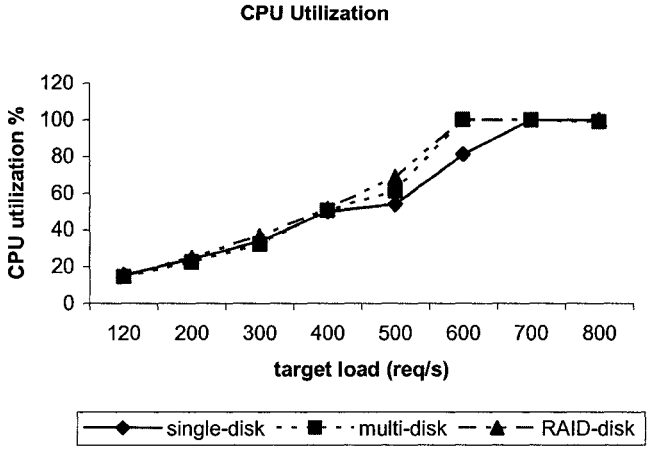


Figure 6.7: System performance in terms of average CPU utilization during the maximum loading phases of Web Polygraph benchmark runs that generates WDSA workload.

In addition to considering average CPU utilization, we also present the instantaneous samples of CPU utilization during top loading phase for all configurations. These measurements are plotted over observed phase in Figure 6.8. CPU utilization is obtained under three levels of loads: 120, 400, and 800 transactions/sec for each of the three configurations: single disk, disk array, and RAID5. Web proxy server CPU saturates whenever the target load increases to 800 transactions/sec. While the three graphs represent the three configuration tested, the graphs have similar nature, indicating that

disk system has very little to do when increasing the target load. Recall that in Chapter 5, simulation-based analysis involving multiple configurations show appreciable performance improvements only under non-standard disk scheduling (SPTF) and prefetching—something not available on a real server running on a commercial platform.

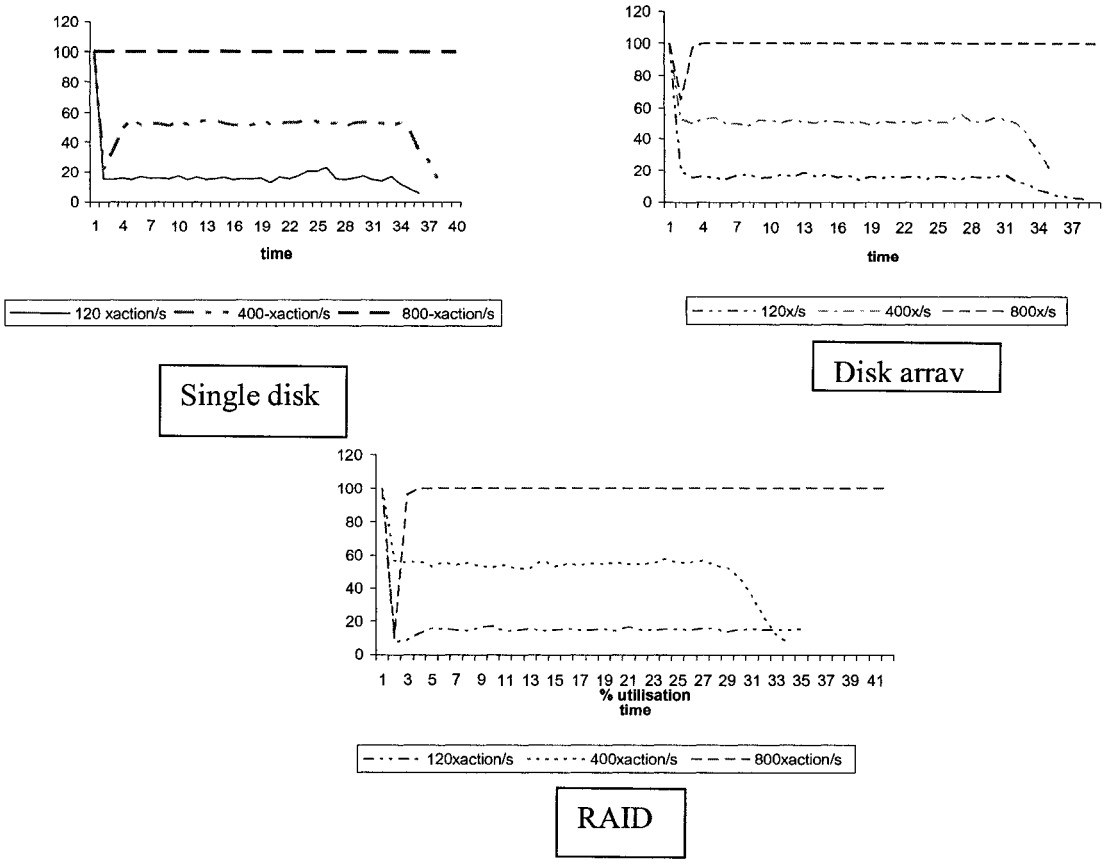


Figure 6.8: System performance in terms of CPU utilization during the maximum loading phases of Web Polygraph benchmark runs that generates WDSA workload.

6.4.4 I/O Performance

Figure 6.9 presents the disk access rate in terms of read and write operations to a single disk under WDSA workload. As expected, frequency of read transactions is significantly greater than the frequency of write transactions during the proxy server peak period referred to as top2 phase of Web Polygraph experiments. This is especially true when the proxy server operates close to its peak throughput level around 400-600 transactions/sec range and reduces when load increases beyond this limit. These are typical symptoms of a performance bottleneck due to disk I/O throughput limitations. A maximum of about 140 disk transactions/sec throughput is achieved, which is a limit for the SCSI disk used for this experiment (see disk specifications in Appendix F).

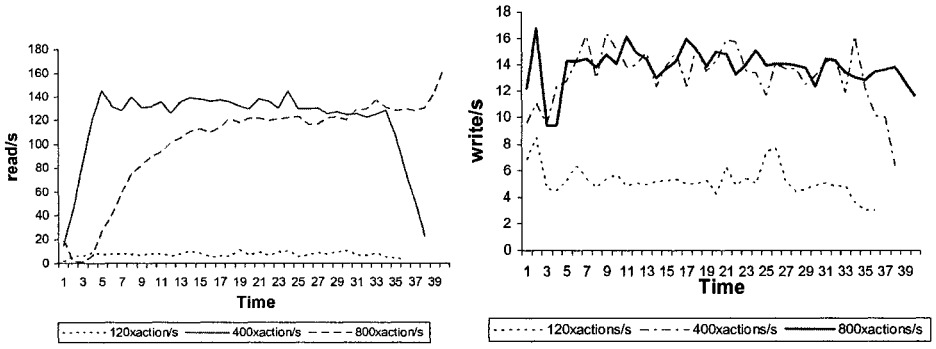


Figure 6.9: Frequency of read and write transactions at three different levels of load to the proxy server: 120, 400, and 800 transactions/sec.

Disk throughput under different disk configurations is shown in Figure 6.10. At high target transactions load, the throughput observed in single and multiple disks system is close to 10 transfer/s, however, the RAID 5 configuration shows a relatively higher (and unbalanced among three disks) throughput in the range of 40 transactions/sec.

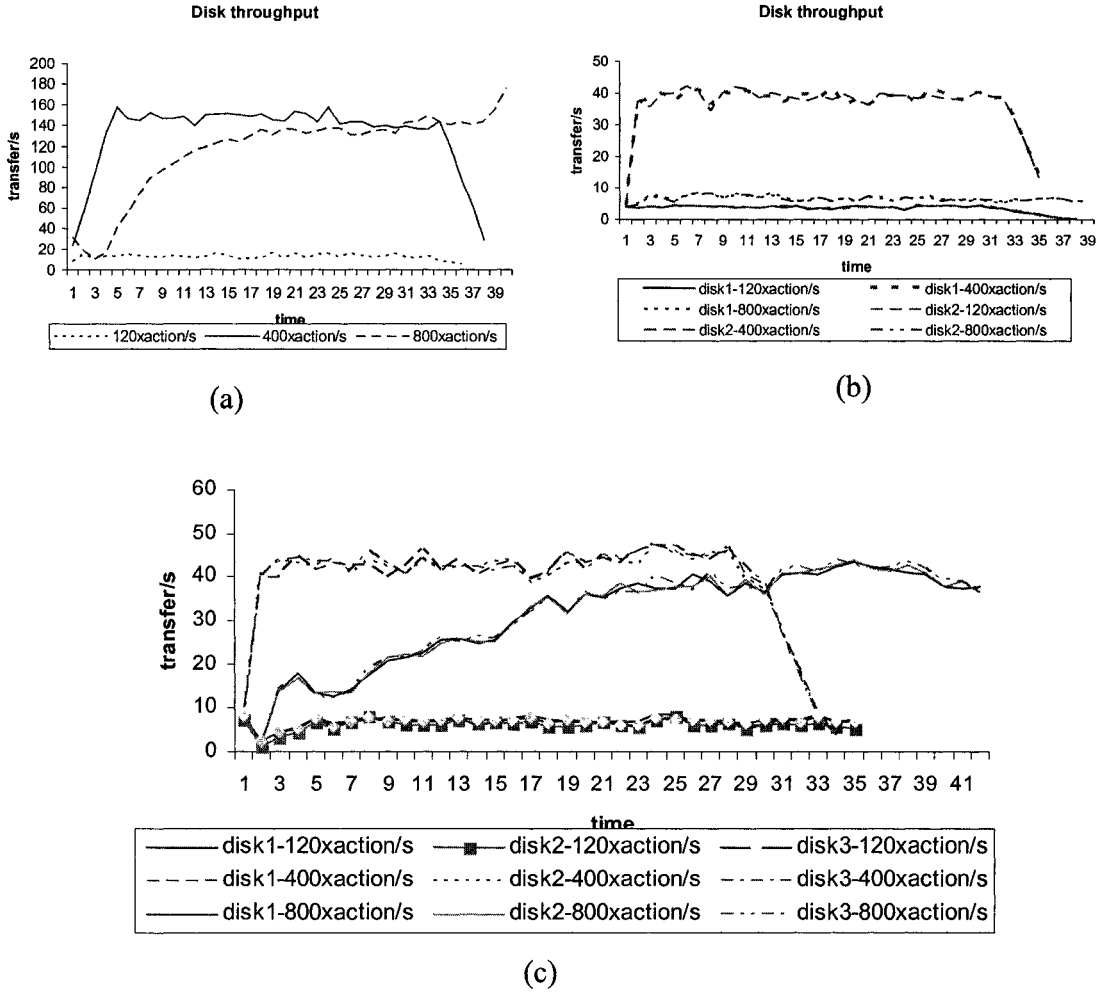


Figure 6.10: I/O performance in terms of disk throughput under WDSA workload for three configurations: single disk (b) disk array and (c) RAID5.

Figure 6.11. breaks down the I/O transactions of disk arrays with respect to reads and writes over top2 phases at three load levels: 120, 400, and 800 transactions/sec for each of the three configurations: single disk, disk array, and RAID5 which is equivalent to low, medium, and heavy.

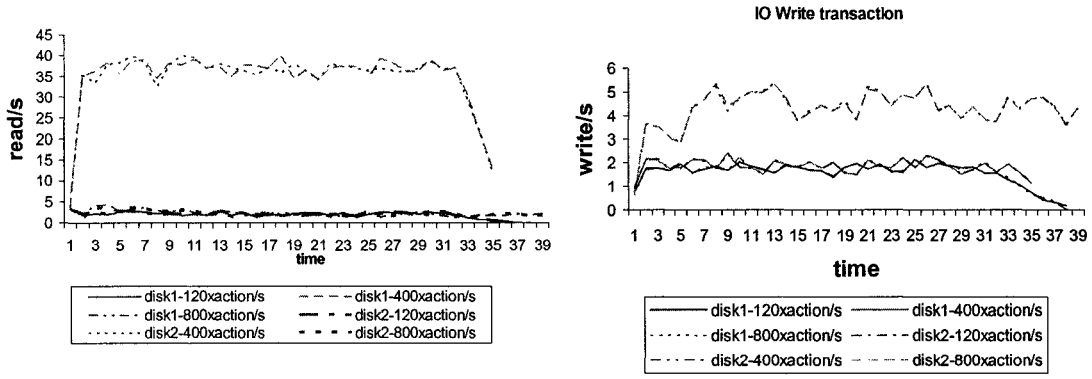


Figure 6.11: Throughput of disk array configuration in terms of reads and writes under three levels of WDSA workload generated by Web Polygraph.

Figure 6.12 indicates the disk queue length. Generally, if disk queue length is greater than 2, it indicates that the disk subsystem is a bottleneck provided enough memory is available. In our experiments, we use 1GB of RAM. On the average, single disk configuration shows that the disk is heavily used and cannot sustain heavy transaction load.

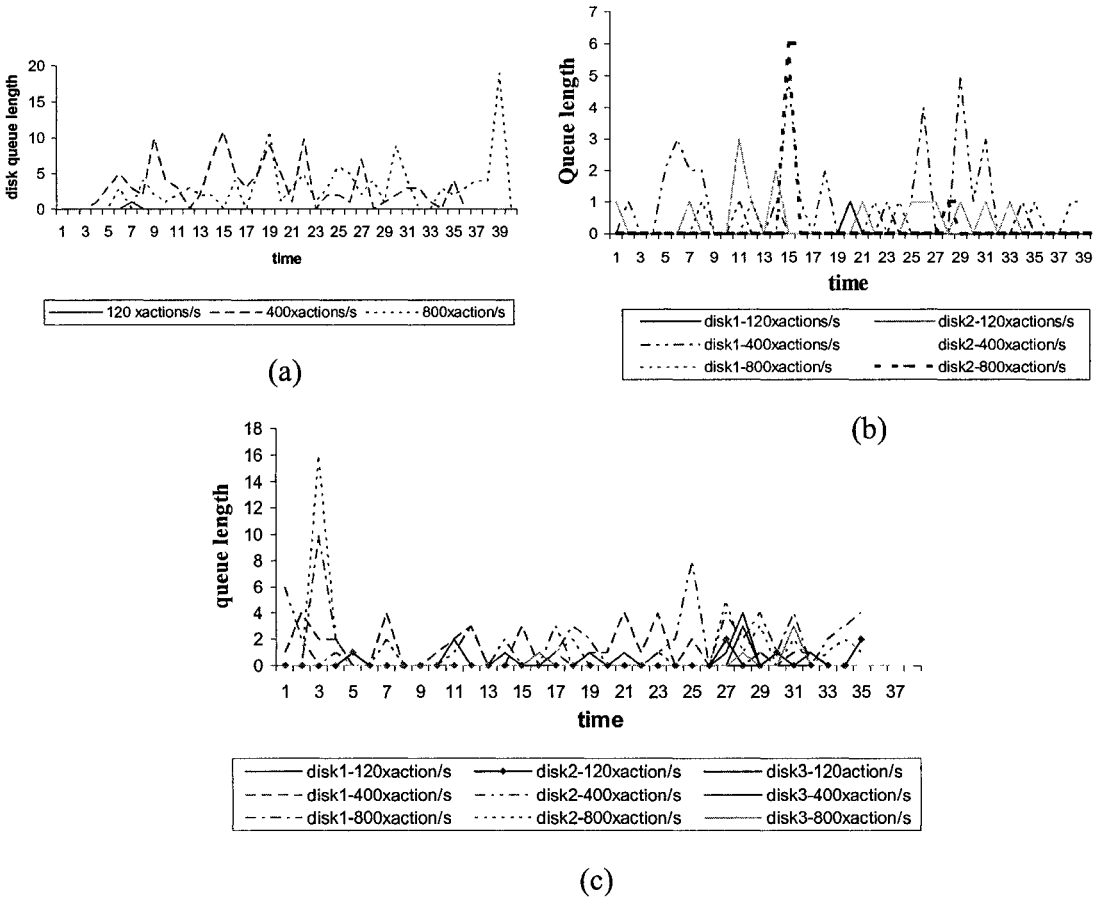


Figure 6.12: Average disk queue length for (a) single disk, (b) disk array, and (c) RAID5 configurations for USA workload and three levels of load.

Using WDSA workload for measurement-based evaluation, we have validated the simulation results related to corresponding case for three configurations. Measurements validate that using either disk array or RAID can improve throughout and reduce latency at high server load.

6.5 Evaluation under NDSA Workload

The narrowly distributed strides access patterns are obtained by loading a web server. For this purpose, we run Microsoft IIS web server on the server platform of our testbed. This web server will be loaded through Webstone benchmark to provide NDSA workload. Set of experiments performed with this setup uses either a single SCSI disk or a RAID5 configuration consisting of three SCSI disks. We did not employ a disk array configuration as it is essentially equivalent to mirroring the single disk—a case that is adequately covered by the RAID configuration.

6.5.1 Experimental Design

In this section, we described our experimental design as well as the factors and metrics we used for running our experiments.

6.5.1.1 Factors

Generally, a careful choice of experimental factors is helpful in emphasizing the important aspects of performance for system under test. Our selection of experimental factors allows us to evaluate the performance of the servers from the disk I/O perspective. In the case of a web server, we used following factors:

I/O system configuration: We used two configurations for these experiments: single disk and RAID5.

Number of clients: This refers to the number of clients requesting for HTTP documents from the web server. Each clients sends HTTP request for a particular web document while the server responds to the request as fast as it can. All HTTP requests are for static documents. We vary the number of clients to represent low, medium and heavy

transaction load to the web server. In this case, we made use of three levels of this factor: 1, 500 and 1000 clients.

Document size: We varied the document size from 500bytes (i.e., a very small size) to 5 MBytes (i.e., a large file size). These document sizes represent two opposite extremes of a wide range of possible document sizes, which are often found on the Internet. Of particular interest is the range between 10KB to 100KB. The reason for choosing document size as one of the factors is to enable us to evaluate web server under realistic conditions.

6.5.1.2 Metrics

Our choice of performance metrics for the web server experiments enable us to evaluate the disk throughput the server can deliver under certain workload condition. In addition, we considered the server CPU utilization as well as latency. Some of the metrics we use in our experiment are as follows:

Throughput: Since web servers process client requests, they are expected to respond at high throughput level, especially under extremely large volumes of client requests. Throughput can be measured in terms of Mbits/sec of data transferred from web server to clients.

Transaction rate: It is a measure of how fast the web server receives client requests and responds to the clients after processing. Transaction rate is measured as transactions/sec or connections/sec.

Average transaction latency: It refers to the latency observed by a clients when it sends a request to the server and until it receives a response from the server. If this latency is high, it implies that the server performance is poor (assuming no network congestion).

Server CPU Utilization: This refers to the overall system CPU time that is spent in doing useful work.

6.5.1.3 Benchmarking Tool

We choose Webstone as our web server benchmarking tool. Webstone [73] is an industry-standard benchmark for comparing/evaluating web server performance. It is a configurable benchmark, which is convenient for performance evaluation of web servers. Webstone simulates only the clients. It does not simulate multiple client IP addresses or multiple connections to the server from each client. WebStone consists of two major components: webmaster and webclients. The webmaster is the program that controls entire measurement-based experiment while webclients simly sends request to the web server under test.

WebStone can distribute the WWW clients easily among the available client computers, while the webmaster controls experimental control activities. It takes a set of files with relative frequencies that it accesses during its tests. For any test in webStone, a real web server has to be used.

6.5.2 Server Performance

In this section, we evaluate Microsoft IIS web server performance in terms of its throughput (transaction rate as well as MBits/sec) and average transaction latency. This evaluation is carried out with varying file sizes, which are requested by Webstone clients from the web server. Appendices B.1 and B.2 give the exact configuration details and filelist for the clients. Note that each experiment uses a single file in the filelist rather than

a large number of files with different probabilities of requesting them, which will be the case for a generic Webstone test. We used a single file for each experiment to ensure non-uniform access pattern to the disk at web server.

Instead of considering the I/O performance directly, we first consider the server and system performance to get a full perspective of the performance. I/O performance is deferred until Section 6.3.4 and will have to be correlated with server and system performance for evaluation.

Figure 6.13 and Figure 6.14 present the connection rate, data throughput, and average transaction processing latency of the server. Server transaction rate represents the rate at which the clients send request to the server and get response back. For a web server transaction to be complete, the clients establish a TCP connection and sends an HTTP request to the server; the server responds to the request if the document is available on the disk; otherwise, it sends an error code; and finally the TCP connection is terminated.

The document size that the clients request from the server has significant effect on the server throughput. We notice that as the document size increases, the number of transactions to the server continues to decrease. It simply takes longer time for the server to send larger number of bytes in response to requests for large-sized documents. With smaller files sizes, both server and clients will be able to establish and terminate their connections frequently, because of this, the transaction throughput of the server can be impaired. Byte throughput increases with file size as well as number of clients accessing the server.

We observe that the server (byte) throughput does not go beyond 90Mbits/s considering TCP overhead when a 10/100Mbit/s network interface card was used. We were able to get better throughput (close to a peak throughput of 300Mbit/s) when the network interface card was replaced with a Gigabit network adapter. In addition, we were able to increase the number of clients accessing the server to 1000, which is close to the maximum the benchmark can accept. When the Gigabit interface was used, it took longer time for the server to become saturate, as seen in Figure 6.13 and Figure 6.14, the server gets to a saturation point when the document size is 500KB, beyond this size, the performance begins to deteriorate. Client experience low latency but latency abruptly increases when the file size increases from 500KB to 5 Mbytes. For this unusually large file size, clients frequently timeout, especially when the number of clients is high in such case. Definitely, having a single disk affects the latency.

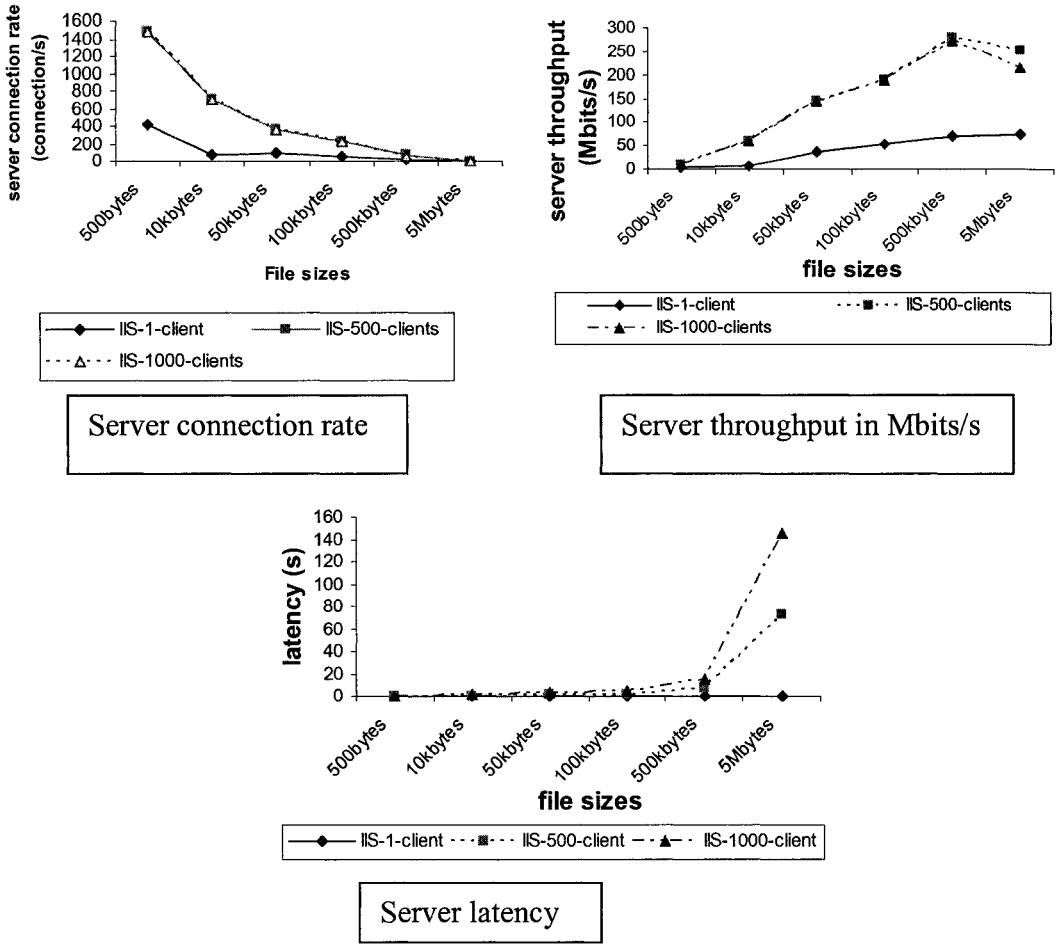


Figure 6.13: Server throughput under single disk (in terms of connection rate and Mbps of data rate) and average latency for HTTP transaction processing. Webstone clients send HTTP transaction requests to the server under NDSA workload pattern.

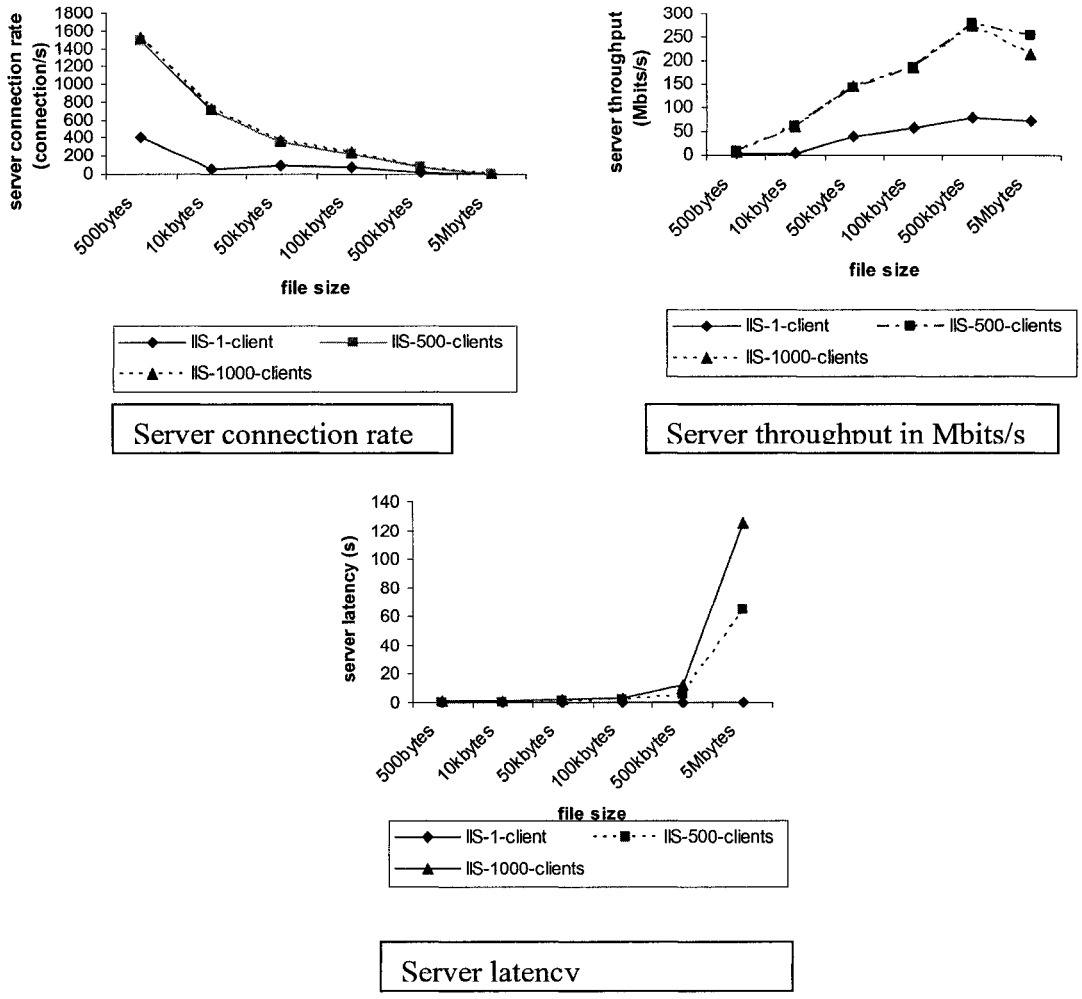
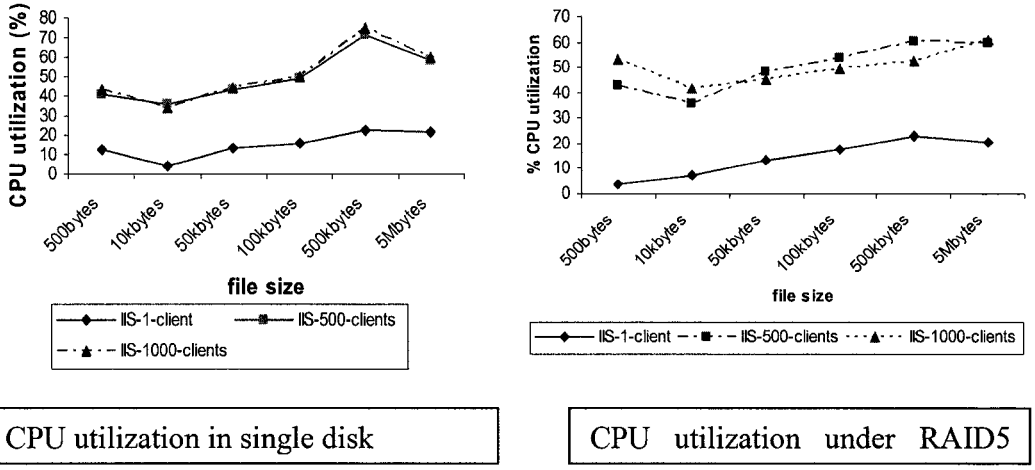


Figure 6.14: Server throughput under RAID configuration (in terms of connection rate and Mbps of data rate) and average latency for HTTP transaction processing. Webstone clients send HTTP transaction requests to the server under NDSA workload pattern.

6.5.3 System Performance

Figure 6.15 shows server CPU utilization for the NDSA workload under single disk and RAID5 configuration. The CPU utilization increases with the increasing file sizes up to 500KB file size, beyond this value, the utilization begin to decrease. With very large file size, the server is no longer receiving many files as it was when the file size is small. With

very large file size, CPU time is tied down, this increases the server latency. However, the number of clients accessing the server contributes to the CPU utilization. Figure 6.15 shows that we have higher CPU utilization when the number of clients accessing the server increases to 1000. The server could not reach 100% utilization because of the large memory size (1GB RAM) present on the server machine.



CPU utilization in single disk

CPU utilization under RAID5

Figure 6.15: Server CPU utilization with varying file sizes and number of simultaneous client accesses under NDSA workload, which is generated by Webstone clients.

6.5.4 I/O Performance

In this section, we present the I/O performance of the web server and correlate it with the server and system performance. In addition, we compare the measurement-based results with the corresponding simulation-based cases of Chapter 5 for verification.

Figure 6.16 shows a low disk throughput in terms of I/O transaction rate regardless of the number of clients or file size. As all clients access the same file during a given experiment, server can read the file from disk once and subsequently serve it from its

main memory. Therefore, multiple levels of storage hierarchy of the server effectively hide the limitation of disk throughput.

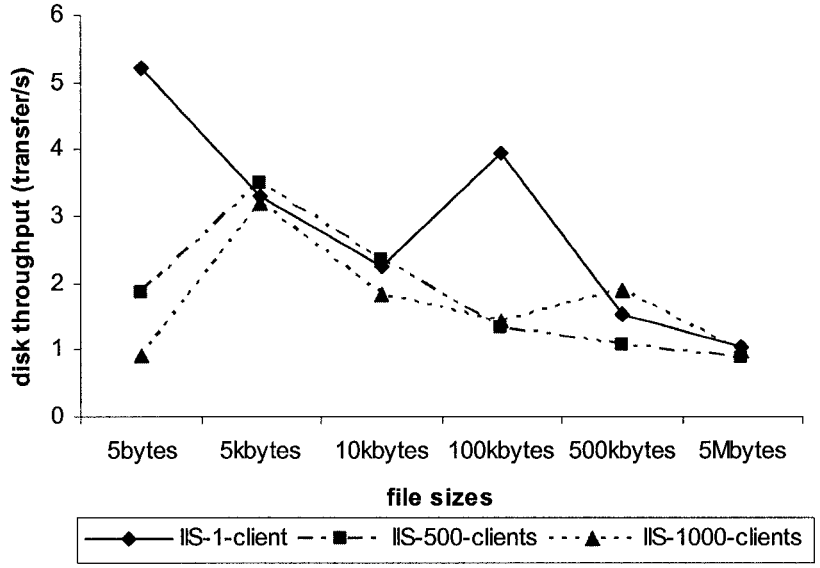


Figure 6.16: Server I/O performance in terms disk throughput due to NDSA workload under single disk.

Figure 6.17 presents the request frequency in terms of read transactions/sec and write transactions/sec under different load conditions. Transaction rate for reads dominates the writes. We observed that there were heavy I/O transactions with fewer clients and the number of read operations reduces with growing number of clients. This is because with the webstone benchmark, few numbers of clients can establish and terminate TCP connections with the server many times. This results in increasing number of I/O transactions that tend to decrease with increasing document sizes for read operations. This behavior is expected of a web server, which performs more reads than writes.

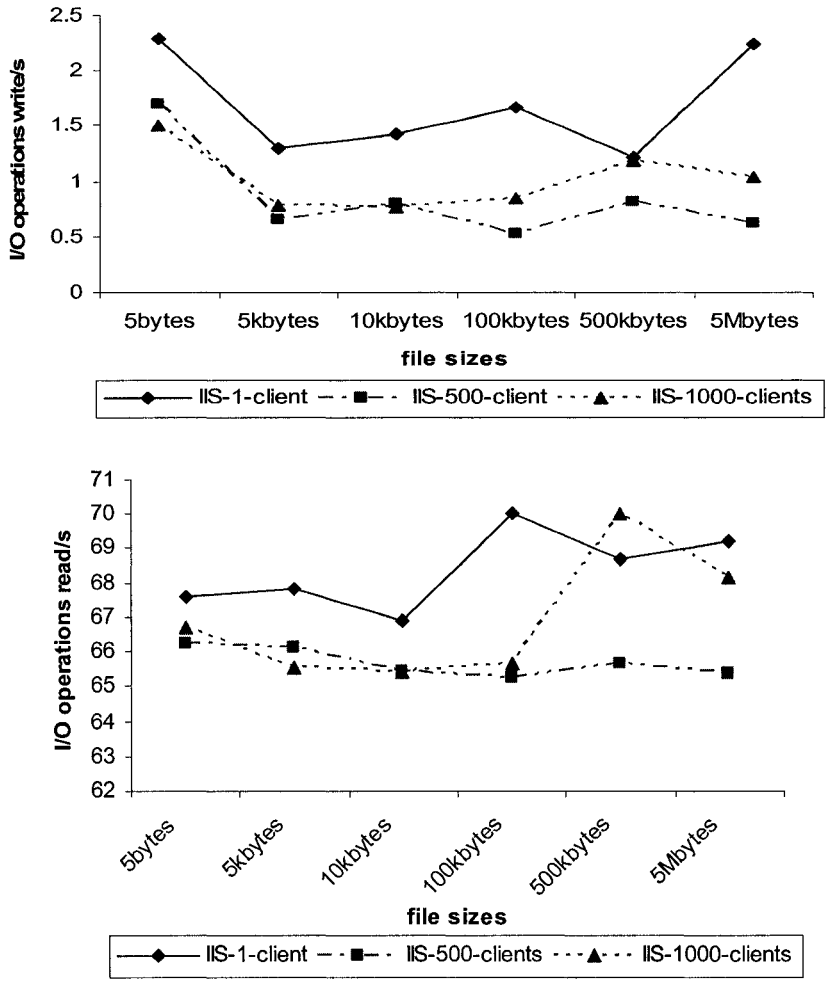


Figure 6.17: Distributions of read vs. write request rates to disk under NDSA workload in term of operation/s.

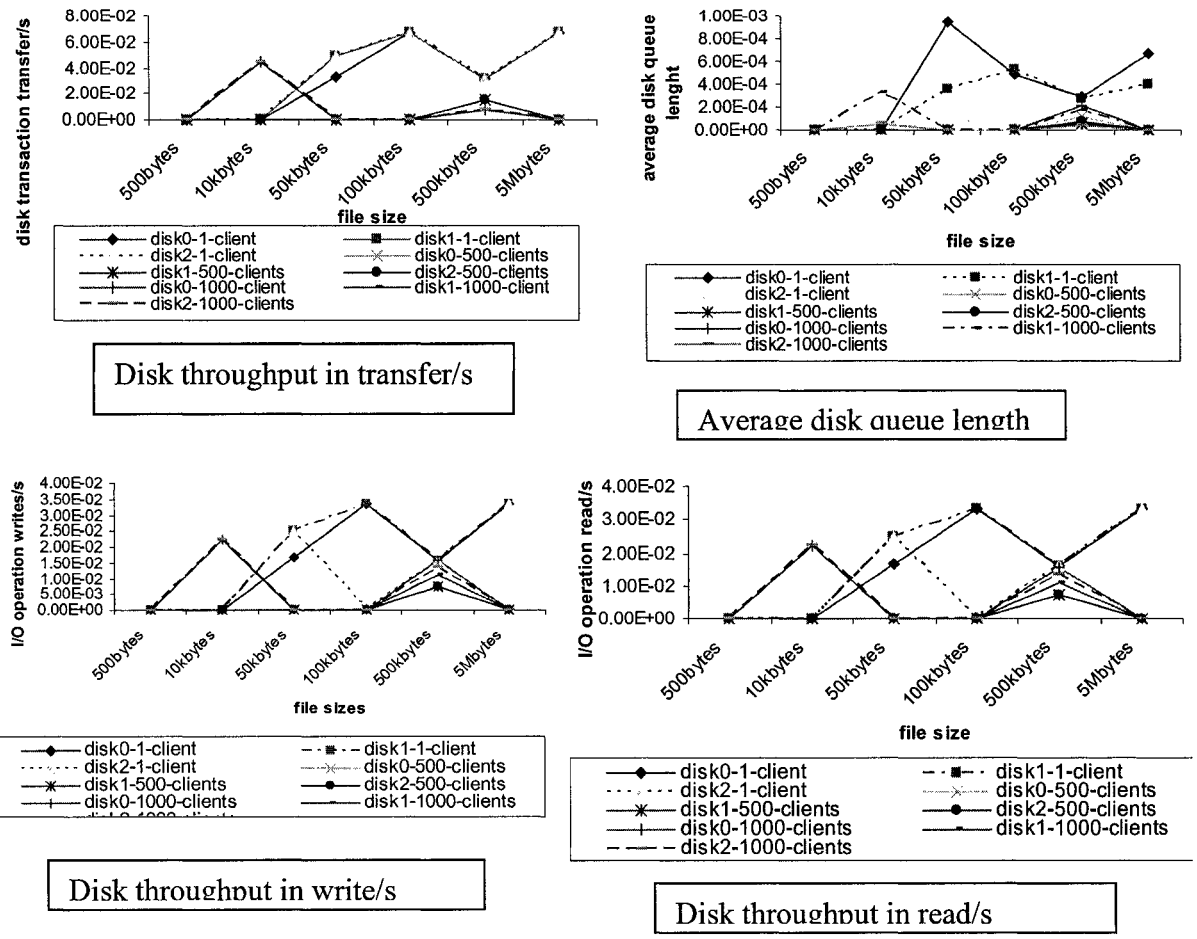


Figure 6.18: Server I/O performance in terms disk throughput, utilization, and queue length due to NDSA workload under RAID configuration.

Measurement-based evaluation of disk performance under NDSA work load shows that the disk system is hardly utilized, especially under the RAID configuration. This can be attributed to the large disk buffer present in the memory of the server system. The documents requested by the clients are kept in the memory; therefore the disk could not cause any bottleneck. A situation that is far different under WDSA workload. We also notice that I/O transaction rate in NDSA is much less as compared to WDSA.

6.6 Conclusion

In this chapter, we have presented the results of our measurement-based evaluation of the impact of disk I/O subsystem on the server performance. Performance of the server system under different disk configurations was evaluated. We report that disk array and RAID 5 configurations exceed the single disk performance under heavy loads resulting in higher server throughput.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this work, we have characterized disk I/O workload for high throughput server using three basic access patterns: SDSA, WDSA, and NDSA. We used trace-driven simulation-based evaluation for comparing three types of disk I/O configuration: single disk, disk arrays, and RAIDs. We have also presented trace-driven simulation based comparisons among two disk I/O scheduling algorithm, FCFS and SPTF, to reduce latency and to improve throughput for our target networking servers. Finally, we evaluated the effect of prefetching on latency and throughput.

The simulation results comparing two scheduling algorithm, show that SPTF reduces the average seek time by about 50% compared to FCFS under WDSA workload. When prefetching in employed in addition to SPTF, further 10% reduction in seek time (compared with regular SPTF) was obtained. Using disk arrays and RAIDs can further reduce the effective seek time and improve the server throughput by properly distributing the data over multiple disks.

Measurement-based results validate the simulation-based results related to comparison of three disk I/O subsystem configurations. Measurements show that in the case of WDSA under disk array or RAID configurations, the server can deliver higher throughput over time, especially when the server is stressed at its maximum peak. However, CPU utilization is higher compare to when single disk is used.

In all the cases considered, the number of disk I/O accesses affects server throughput. Any latency hiding mechanism at the server side will improve performance as measurement-based results show.

7.2 Future Research

Our work reported measurement-based evaluation results as well as simulation analysis comparing the impact of two disk scheduling algorithm on high throughput servers. Since large number of I/O accesses at the server side, degrades performance, therefore implementing prefetching scheme on the real server can go a long way alleviating the I/O bottleneck issues.

In addition, implementation of new bus architecture such as infiniband technology can as well improves server throughput by reducing bottleneck cause by disk I/O.

REFERENCES

[1] Iyengar, E. MacNair, and T. Nguyen. "An analysis of Web Server Performance," In Proceedings of GLOBE COM 1997.

[2] A. J. Smith, ''Optimization of I/O Systems by Cache Disk and File Migration: A Summary'', Performance Evaluation 1, 3 (November 1981), 249-262.

[3] A. L. Reddy and P. Banerjee. "An Evaluation of Multiple –Disk I/O Systems," IEEE Transaction on Computers, vol.38, no 12, pp. 1680-1690, Dec. 1989.

[4] A. Merchant and P. S. Yu. "An analytical model of reconstruction time in mirrored disks." Performance Evaluation, 20(1–3): 115–29, May 1994.

[5] A. Merchant and P.S. Yu. "Analytic modeling and comparisons of striping strategies for replicated disk arrays." IEEE Transactions on Computers, 44(3): 419–33, March 1995.

[6] A. Merchant and P.S. Yu. "Analytic modeling of clustered RAID with mapping based on nearly random permutation." IEEE Transactions on Computers, 45(3): 367–73, March 1996.

[7] A. Thomasian and J. Menon. "Raid5 performance with distributed sparing." IEEE Transactions on Parallel and Distributed Systems, 8(6): 640–57, June 1997.

[8] Anon and et al.. ''A Measure of Transaction Processing Power'', Datamation, 31, 7 (April 1985), 112-118.

[9] Ari Luotonen. "Web Proxy Servers." Prentice Hall, Upper Saddle River, NJ, 1998.

[10] B. Worthington, G. Ganger and Y. Patt. "Scheduling Algorithms for Modern Disk Drives." Proc. ACM Sigmetrics Conf. pp. 241-251, May 1994.

[11] Cabrera, L. F. and Long, D. E. "Exploiting Multiple I/O Streams to provide High Data rates." Technical Report UCSC-CRL-91-08, UC San Diego / CSE, April 1991.

[12] Cabrera, L. F. and Long, D. E. Swift. "Using Distributed Disk Striping to provide High I/O data rates." Technical Report UCSC-CRL-91-46, UC San Diego / CSE, December 1991.

[13] Carlos Maltzahn, Kathy J. Richardson, and D. Grunwald. " Reducing the Disk I/O of Web Proxy Server Caches." In Proceeding of USENIX 1999 Annual Technical Conference California USA June 1999.

[14] Chen, P. M and D. A. Patterson. "A new approach to I/O performance evaluation – self scaling I/O benchmark, predicted I/O performance." ACM Trans. On Computer Systems 12:4 Nov 1994.

[15] Chen, S., and Towsley, D. "The design and evaluation of RAID 5 and Parity Striping Disk Array Architecture." Journal of Parallel and Distributed Computing 17 (1993), 58-74.

[16] Chervenak, A. L. and Katz, R. H. "Performance of a Disk Array Prototype." Proceedings of the 1991 ACM SIGMETRICS Conference 19 May 1991, 188-197.

[17] Chris Ruemmler and John Wilkes. "Modeling Disks," computer System Laboratory HPL-93-68, July 1993.

[18] Chris Ruemmler and John Wilkes. "An Introduction to disk drive modeling." IEEE Computer, 27(3):17-28, March 1994.

[19] Cormen, T. H., and Hirschl, M. "Early experiences in evaluating the parallel disk model with the ViC* implementation." Parallel computing 23, 4 June 1997), 571-600.

[20] D. A. Patterson, G. Gibson and R. H. Katz. ''A Case for Redundant Arrays of Inexpensive Disks (RAID)'', International Conference on Management of Data (SIGMOD), June 1988, 109-116.

[21] D. Bitton and J. Gray. "Disk shadowing." In Proc. of 14th Int'l. Conf. on Very Large Data Bases (VLDB), pages 331–8, August 1988.

[22] D. Stodolsky and G. A. Gibson. ''Parity Logging: Overcoming the Small Write Problem in Redundant Disk Arrays'', Proceedings of the 1993 International Symposium on Computer Architecture, May 1993.

[23] E. L. Miller and R. H. Katz. ''Input/Output Behavior of Supercomputing Applications'', Proceedings of Supercomputing '91, November 1991, 567-576.

[24] E. Shriver, A. Merchant, and J. Wilkes. "An analytical behavior model for disk drives with readahead caches and request reordering." In Proc. of Int'l. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS), pages 182–91, June 1998.

[25] E. Tan and B. Vermeulen. ''Digital audio tape for data storage'', IEEE Spectrum, October 1989, 34-38.

[26] E.K. Lee and R.H. Katz. "An analytic performance model of disk arrays." In Proc. of ACM Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS), pages 98 109, May 1993.

[27] Farley, Marc. "Building Storage Networks." 2nd Ed McGraw Hill 2001.

[28] G. A. Gibson. ''Redundant Disk Arrays: Reliable, Parallel Secondary Storage'', UCB/Computer Science Department. 91/613, University of California at Berkeley, December 1991. Also available from MIT Press, 1992.

[29] G. A. Gibson, R. H. Patterson and M. Satyanarayanan. ''Disk Reads with DRAM Latency'', Third Workshop on Workstation Operating Systems, Key Biscayne, Florida, April 1992.

[30] G. M. Amdahl. ''Validity of the single processor approach to achieving large scale computing capabilities'', Proceedings AFIPS 1967 Spring Joint Computer Conference 30 (April 1967), 483-485.

[31] G. R. Ganger and Y. N. Patt. "Using system-level models to evaluate I/O subsystem designs." IEEE Transactions on Computers, 47(6): 667–78, June 1998.

[32] Hennessy, J. L., and Patterson, D. A. "Computer Architecture: A Quantitative approach." Morgan Kaufmann Publishers Incorporated, San Francisco, CA, 1996. 2nd edition.

[33] Infiniband Architecture. www.infinibandta.org

[34] J. A. Fine, T. E. Anderson, M. D. Dahlin, J. Frew, M. Olson and D. A. Patterson. ''Abstracts: A Latency-Hiding Technique for High-Capacity Mass Storage Systems'', Sequoia Technical Report 92/11, University of California at Berkeley, March 1992.

[35] J. C. Hu, I. Pyarali, and D. C. Schmidt. "Measuring the Impact of Event Dispatching and Concurrency Models on Web Server Performance Over High-Speed Networks," In Proceedings of GLOBECOM 1997.

[36] J. K. Ousterhout and F. Douglis. ''Beating the I/O Bottleneck: A Case for Log-Structured File Systems'', SIGOPS 23, 1 (January 1989), 11-28.

[37] J. K. Ousterhout, A. Cherenson, F. Douglis and M. Nelson. ''The Sprite Network Operating System'', IEEE Computer 21, 2 (February 1988), 23-36.

[38] J. Menon and J. Kasson. "Methods for improved update performance of disk arrays." In Proc. of 25th Int'l. Conf. on System Sciences, volume 1, pages 74–83, January 1992.

[39] J. Wilkes. "The Pantheon storage-system simulator." Technical Report HPL-SSP-95-14, HP Laboratories, December 1995.

[40] J.B. Chen and B.N. Bershad. "The impact of operating system structure on memory system performance." In Proc. of 14th ACM Symposium. on Operating Systems Principles (SOSP), pages 120–33, December 1993.

[41] John Ousterhout and Fred Douglis. "Beating the I/O Bottleneck: A Case for Log-Structured File Systems". ACM Operating System Reviews, Vol. 23, No. 1, pp. 11-28, January 1989.

[42] Kim M. Y. and Tantawi A. N. "Asynchronous Disk Interleaving: Approximating Access Delays." IEEE Transactions on Computers, 40(7), July 1991.

[43] L. P. Slothouber. "A model of Web Server performance." Available at http:// www.starnine.com/webstar/overview.html.

[44] Lee, E. K. "Software and Implementation of a RAID Prototype." Technical Report UCB/CSD 90/573, UC Berkeley/CSD, May 1990.

[45] M. Holland and G. Gibson. ''Parity Declustering for Continuous Operation in Redundant Disk Arrays.'' Proceedings of the 5th International Conference on

Architectural Support for Programming Languages and Operating Systems (ASPLOS-V), October 1992, 23-35.

[46] M. Livny, S. Khoshafian and H. Boral. ''Multi-Disk Management Algorithms'', Proceedings of the 1987 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, May 1987, 69-77.

[47] M. N. Nelson, B. B. Welch and J. K. Ousterhout. ''Caching in the Sprite Network File System'', ACM Transactions on Computer Systems 6, 1 (February 1988), 134-154.

[48] M. Rosenblum and J. K. Ousterhout. ''The Design and Implementation of a Log-Structured File System'', Proceedings of the 13th ACM Symposium on Operating Systems Principles, October 1991.

[49] M. Y. Kim. "Synchronised Disk Interleaving," IEEE Transactions on Computers, Vol. C-35, no. 11 pp. 978-988, Nov 1986.

[50] Microsoft Internet Security & Acceleration Server. Available at http://www.microsoft.com/isaserver/.

[51] Mustafa U., Guillermo A. A., and Arif Merchant. "A Modular, Analytical throughput Model for Modern Disk Arrays." In Proceeding 9th International Symposium on MASCOT 2001, August 2001 USA.

[52] Narasimha Reddy A. L and Wylie Jim. "Disk Scheduling in a Multimedia System'" In Proceedings of ACM Multimedia '93, Anaheim, CA, 225-234, August 1993.

[53] Narasimha Reddy. "A Study of I/O System Organization" Proc. of the 19th Annual International Symposium on Computer Architecture, Vol. 2, pp. 145-185, June 1994.

[54] National Science Foundation Workshop on Next Generation Secondary Storage architecture, National Science Foundation, Bodega Bay, CA, May 1989.

[55] P. M. Chen and D. A. Patterson. ''Maximizing Performance in a Striped Disk Array'', Proceedings of the 1990 International Symposium on Computer Architecture, Seattle WA, May 1990, 322-331.

[56] P. M. Chen, G. Gibson, R. H. Katz and D. A. Patterson. ''An Evaluation of Redundant Arrays of Disks Using an Amdahl 5890'', Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Boulder CO, May 1990.

[57] R. H. Katz, D. W. Gordon and J. A. Tuttle. ''Storage System Metrics for Evaluating Disk Array Organizations.'' UCB/Computer Science Department. 90/611, University of California at Berkeley, December 1990.

[58] R. H. Katz, G. A. Gibson, and D. A. Patterson. "Disk System Architecture for High Performance Computing, "Proc. Of the IEEE, vol.77, no. 12, pp.1842-1858, Dec 1989.

[59] R. McGrath. "Performance of Several Web Server Platforms." Available at http://www.ncsa.uiuc.edu/InformationServers/Performance/Platforms/report.html.

[60] R. R. Muntz and J. C. S. Lui. ''Performance Analysis of Disk Arrays under Failure'', Proceedings of the 16th Conference on Very Large Data Bases, 1990. VLDB XVI.

[61] S. Chen and D. Towsley. "A performance evaluation of RAID architectures." IEEE Transactions on Computers, 45(10): 1116–30, October 1996

[62] S. Glassman. "A Caching Relay for the World Wide Web", First International conference on the World Wide Web, Geneva, Switzerland, 1994.

[63] Shriver E. and Small C. "Why does file system prefetching work?" in Proceedings of the USENIX Annual Technical Conference Monterey, California, USA, June 6-11, 1999.

[64] T. Sterling, P. Messina, and P. H. smith. "Enabling Technologies for Petaflops Computing." MIT Press, 1995.

[65] TPC Benchmark B Standard Specification, Transaction Processing Performance Council, August 1990.

[66] U.S. DOE, "United States Department of energy Accelerated Strategic Computing Initiative (ASCI)." Available at www.llnl.gov/asci. Jan. 1997.

[67] Vitter, J. S, and Shriver, E. A. M. "Algorithms for parallel memory I: two level memories." Algorithmica 12, 2/3 (August/September 1994), 110-47.

[68] W. V. Courtright II. "A transactional approach to redundant disk array implementation." PhD thesis, Department of Electrical Engineering and Computer Science, Carnegie-Mellon University, May 1997.

[69] Web polygraph version 2.7.6 available at http://www.web-polygraph.org/

[70] WebStone2.5 available at http://www.mindcraft.com/webstone/

[71] Worthington, B. L., Ganger, G. R., Patt, Y. N., and Wilkes, J. "On-line extraction of SCSI disk drive parameters." In Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (Ottawa, Canada, May 1995), ACM Press, New York, NY, pp. 146-156.

APPENDIX A

Appendix A.1. Registry parameters to enhance the availability of a Windows 2000 server platform.

Registry Key	Location	Value (dword)
TZPersistIntervalThreshold	W3Cache\Parameters	0x1
RecoveryMruSizeThreshold	W3Cache\Parameters	0x5
MaxClientSession	W3Proxy\Parameters	0x2800
OutstandAccept	W3Proxy\Parameters	0x3e8
MaxUserPort	Tcpip\Parameters	0x0xffff
TcpTimedWaitDelay	Tcpip\Parameters	0x3c
StrictTimeWaitSeqCheck	Tcpip\Parameters	0x1

APPENDIX B

Appendix B.1:
Webstone configuration file

```
ITERATIONS="1"
MINCLIENTS="1"
MAXCLIENTS="1000"
CLIENTINCR="500"
TIMEPERRUN="5"
PROXYSERVER=
SERVER="172.16.70.55"
PORTNO=80
RCP=rcp
RSH=rsh
SERVERINFO=
OSTUNINGFILES=
WEBSERVERTUNINGFILES=
WEBSERVERDIR=
WEBDOCDIR=/wwwroot
CLIENTS=" feel_2 ababeel_2 ababeel_3 ababeel_4"
CLIENTACCOUNT=pel
CLIENTPASSWORD=the,great
FIXED_RANDOM_SEED=true
CLIENTINFO="uname -a"
TMPDIR=/tmp
CLIENTPROGFILE=/home/alada/soft/WebStone2.5/src/web
client
DEBUG=
```

Appendix B.2 : Webstone Filelist

```
# Filelist for WebStone 2.5 Standard Run Rules, same as
filelist.sample
#/file500_1.html 20000
#/file10k_1.html 35000          #1000
#/file50k_1.html 20000
#/file100k_1.html 10000
/file500k_1.html 5000
#/file5m_1.html 2000
```

APPENDIX C

Appendix C.1 Shell script for running web proxy server experiment

```
#!/bin/csh -f
set clients = " feel_3 feel_1 feel_2 "
set servers = "feel_4 ababeel_3"
set echo
echo "Running aka on the clients"
rsh -n feel_3 /usr/local/polygraph/bin/aka xl0 10.16.1-4.1-100
rsh -n feel_1 /usr/local/polygraph/bin/aka xl0 10.16.1-4.101-
200
rsh -n feel_2 /usr/local/polygraph/bin/aka xl0 10.16.1-4.201-
250
echo "Runing aka on the servers"
rsh -n feel_4 /usr/local/polygraph/bin/aka xl0 10.16.129-
131.1-100
rsh -n ababeel_3 /usr/local/polygraph/bin/aka xl0 10.16.129-
131.101-200
foreach i ($clients)
        rsh -n $i /sbin/ipfw -f flush
end
foreach i ($servers)
        rsh -n $i /sbin/ipfw -f flush
        rsh -n $i /sbin/ipfw pipe 1 config delay 40ms plr
0.0005
        rsh -n $i /sbin/ipfw pipe 2 config delay 40ms plr
0.0005
        rsh -n $i /sbin/ipfw add pipe 1 ip from any to
10.16.0.0/16 in
        rsh -n $i /sbin/ipfw add pipe 2 ip from
10.16.0.0/16 to any out
end
foreach i ($clients)
        rcp my_source.pg $i\:\/tmp/my.pg
        rsh -n $i rm -f /tmp/*.con /tmp/*.log /tmp/*.output
end
foreach i ($servers)
        rcp my_source.pg $i\:\/tmp/my.pg
        rsh -n $i rm -f /tmp/*.con /tmp/*.log /tmp/*.output
end
set server_stat_pids =
set server_pids =
foreach i ($servers)
        rsh -n $i /usr/local/polygraph/bin/polysrv --config
/tmp/my.pg --verb_lvl 10 --log /tmp/srv_$i.log --console
/tmp/srv_$i.con                                    --cfg_dirs
/usr/local/polygraph/workloads/include             --notify
10.16.0.20:18256 &
        set server_pids = "$server_pids $!"
        echo started polysrv on $i with server_pids
$server_pids
        rsh -n $i /usr/bin/vmstat 20 >&
/tmp/vmstat_$i.output &
        set server_stat_pids = "$server_stat_pids $!"
end
sleep 40
echo "Starting polyclt on the client machines"
set client_stat_pids =
set client_pids =
foreach i ($clients)
        rsh -n $i /usr/local/polygraph/bin/polyclt --config
/tmp/my.pg --verb_lvl 10 --log /tmp/clt_$i.log --console
/tmp/clt_$i.con --proxy 10.16.0.2:8080 --cfg_dirs
/usr/local/polygraph/workloads/include --dump err --notify
10.16.0.20:18256 &
        set client_pids = "$client_pids $!"
        echo started polyclt on $i with client_pids
$client_pids
        rsh -n $i /usr/bin/vmstat 20 >&
/tmp/vmstat_$i.output &
        set client_stat_pids = "$client_stat_pids $!"
end
echo Waiting for the client processes to finish
foreach i ($client_pids)
        wait($i);
        echo Client processes $i finished; cleaning up
end
```

Appendix C.2: Web polygraph benchmark Polymix-3

```
Bench TheBench = benchPolyMix3; // start with the default
settings
TheBench.peak_req_rate =400/sec;
size ProxyCacheSize = 2.00GB;
rate FillRate = 320/sec;
int clientHostCount = clientHostCount(TheBench);
addr[] rbt_ips = [ '10.16.1-4.1-250' ];
addr[] srv_ips = [ '10.16.129-131.1-200:9090' ];
PopModel popModel = {
        pop_distr = popUnif();
        hot_set_frac = 1%; // fraction of WSS (i.e.,
hot_set_size / WSS)
        hot_set_prob = 10%; // prob. of req. an object
from the hot set
};
Server S = {
        kind = "PolyMix-3-srv";
        contents = [ cntImage: 65%, cntHTML: 15%,
cntDownload: 0.5%, cntOther ];
        direct_access = [ cntHTML, cntDownload,
cntOther ];

        xact_think = norm(1.5sec, 1sec);
        pconn_use_lmt = zipf(16);
        idle_pconn_tout = 10sec;
        addresses = srv_ips;
};
float HitIfRepeat = 80% * (100% - 5%);
// describe PolyMix-3 robot
Robot R = {
        kind = "PolyMix-3-rbt";
        origins = srv_ips;
        recurrence = 55%/HitIfRepeat; // recurrence is
not hit ratio
        embed_recur = 100%;
        public_interest = 50%;
        req_types = [ "Ims200": 5%, "Ims304": 10%,
"Reload" : 5%, "Basic" ];
        pop_model = popModel;
        req_rate = 0.4/sec;
        pconn_use_lmt = zipf(64);
        open_conn_lmt = 4;          // open connections
limit
        addresses = rbt_ips;
};
int RobotCount = count(R.addresses);
rate PeakRate = RobotCount * R.req_rate;
time platDur = 15min;    // plateau phases duration (except
idle phase)
time rampDur = platDur/8; // ramp phases duration   (and
idle phase)
float fillable_ratio = 75%; // ~(cachable, not reloaded, HTTP
200 response)
rate peak_fill_rate = PeakRate * 0.45 * fillable_ratio;
```

```
int wsc = int(peak_fill_rate * platDur); // number of objects in
the WS
working_set_cap(wsc / clientHostCount);
 Phase phWarm = {
            name = "warm";
            goal.duration = 3min;
            recur_factor_beg = 5%/55%;
            special_msg_factor_beg = 0.1;
            load_factor_beg = 0.1;
            log_stats = false;
};
Phase phFRamp = {
            name = "framp";
            goal.duration = rampDur;
            load_factor_end = FillRate/PeakRate;
};
Phase phFill = {
            name = "fill";
            goal.fill_size =  ProxyCacheSize /
clientHostCount;
            wait_wss_freeze = no; // will finish only if WSS is
frozen
};
Phase phFExit = {
            name = "fexit";
            goal.duration = rampDur;
            recur_factor_end = 1;
            special_msg_factor_end = 1;
            load_factor_end = 0.1;
};
Phase phInc1 = {
            name = "inc1";
            goal.duration = rampDur;
            load_factor_end = 1.0;
};
Phase phTop1 = { name = "top1"; goal.duration = platDur; };
Phase phDec1 = { name = "dec1"; goal.duration = rampDur;
load_factor_end = 0.1; };
Phase phIdle = { name = "idle"; goal.duration = rampDur; };
Phase phInc2 = { name = "inc2"; goal.duration = rampDur;
load_factor_end = 1.0; };
Phase phTop2 = { name = "top2"; goal.duration = platDur; };
Phase phDec2 = { name = "dec2"; goal.duration = rampDur;
load_factor_end = 0.1; };
/* collect stats samples in the middle of plateau phases */
StatsSample topSample = { capacity = 10000; };
StatsSample fillSample = { capacity = int(10% *
topSample.capacity); };
phFill.stats_samples = [ fillSample ];
phTop1.stats_samples = [ topSample ];
phTop2.stats_samples = [ topSample ];
schedule(
            phWarm,
            phFRamp, phFill,
            phFExit,
            phInc1, phTop1, phDec1,
            phIdle,
            phInc2, phTop2, phDec2,
            phCool
);
use(S, R);
use(TheBench);
```

APPENDIX D

Appendix D.1: Matlab script to analyze web server traces output

```
% analysis web server reads (no writes) due to Webstone load
% reads
clear;
load webs_read.dat;
disp('histogram of address offsets');
hist(webs_read(:, 4));
xlabel('Address offsets (32-bit)');
ylabel('Number of occurrences');
title('Disk read address offset histogram for IIS web server');
disp('press any key to continue to histogram of strides');
pause;
read_oset = webs_read(:, 4);
l = length(read_oset);
a = read_oset(1:l-1);
b = read_oset(2:l);
read_strides = abs(b - a);
[N, X] = hist(read_strides);
bar(X, N);
xlabel('Absolute values of strides');
ylabel('Number of occurrences');
title('Successive disk read access strdes histogram for IIS web
server');
disp('press any key to view histogram of read data sizes');
pause;
hist(webs_read(:,2));
xlabel('Transfer size (Bytes)');
ylabel('Number of occurrences');
title('Disk data read sizes histogram for IIS web server');
disp('press any key to plot queue sizes')
pause;
plot(webs_read(:,3));
xlabel('Event records');
ylabel('Disk queu size (Bytes)');
title('Disk data read queus sizes for IIS web server');
disp('press any key to view histogram of queue sizes');
pause;
hist(webs_read(:,3));
xlabel('Queue size (Bytes)');
```

Appendix D.2: Matlab script to analyze web proxy server traces output.

```
% analysis of 200 xactions/s load
% reads
clear;
load proxy200_read.dat;
disp('histogram of address offsets');
hist(proxy200_read(:, 4));
xlabel('Address offsets (32-bit)');
ylabel('Number of occurrences');
title('Disk read address offset histogram at 200 xaction/sec
load');
disp('press any key to continue to histogram of strides');
pause;
read_oset = proxy200_read(:, 4);
l = length(read_oset);
a = read_oset(1:l-1);
b = read_oset(2:l);
read_strides = abs(b - a);
```

```
[N, X] = hist(read_strides);
bar(X, N);
xlabel('Absolute values of strides');
ylabel('Number of occurrences');
title('Successive disk read access strdes histogram at 200
xaction/sec load');
disp('press any key to view histogram of read data sizes');
pause;
hist(proxy200_read(:,2));
xlabel('Transfer size (Bytes)');
ylabel('Number of occurrences');
title('Disk data read sizes histogram at 200 xaction/sec load');
disp('press any key to plot queue sizes')
pause;
plot(proxy200_read(:,3));
xlabel('Event records');
ylabel('Disk queu size (Bytes)');
title('Disk data read queus sizes at 200 xaction/sec load');
disp('press any key to view histogram of queue sizes');
pause;
hist(proxy200_read(:,3));
xlabel('Queue size (Bytes)');
ylabel('Number of occurrences');
title('Disk data read queue size histogram at 200 xaction/sec
load');
disp('press any key to analyze writes at 200 xactions/sec
load');
pause;
% writes
clear;
load proxy200_write.dat;
disp('histogram of address offsets');
hist(proxy200_write(:, 4));
xlabel('Address offsets (32-bit)');
ylabel('Number of occurrences');
title('Disk write address offset histogram at 200 xaction/sec
load');
disp('press any key to continue to histogram of strides');
pause;
write_oset = proxy200_write(:, 4);
l = length(write_oset);
a = write_oset(1:l-1);
b = write_oset(2:l);
write_strides = abs(b - a);
[N, X] = hist(write_strides);
bar(X, N);
xlabel('Absolute values of strides');
ylabel('Number of occurrences');
title('Successive disk write access strdes histogram at 200
xaction/sec load');
disp('press any key to view histogram of write data sizes');
pause;
hist(proxy200_write(:,2));
xlabel('Transfer size (Bytes)');
ylabel('Number of occurrences');
title('Disk data write sizes histogram at 200 xaction/sec load');
disp('press any key to plot queue sizes')
pause;
plot(proxy200_write(:,3));
xlabel('Event records');
ylabel('Disk queu size (Bytes)');
title('Disk data write queus sizes at 200 xaction/sec load');
disp('press any key to view histogram of queue sizes');
pause;
hist(proxy200_write(:,3));
xlabel('Queue size (Bytes)');
ylabel('Number of occurrences');
title('Disk data write queue size histogram at 200 xaction/sec
load');
disp('press any key to analyze reads/writes at 400 xactions/sec
load');
pause;
% analysis of 400 xactions/s load
% reads
clear;
load proxy400_read.dat;
disp('histogram of address offsets');
hist(proxy400_read(:, 4));
xlabel('Address offsets (32-bit)');
ylabel('Number of occurrences');
title('Disk read address offset histogram at 400 xaction/sec
load');
disp('press any key to continue to histogram of strides');
pause;
read_oset = proxy400_read(:, 4);
l = length(read_oset);
a = read_oset(1:l-1);
b = read_oset(2:l);
read_strides = abs(b - a);
[N, X] = hist((read_strides), 100);
bar(X, N);
xlabel('Absolute values of strides');
ylabel('Number of occurrences');
title('Successive disk read access strdes histogram at 400
xaction/sec load');
disp('press any key to view histogram of read data sizes');
pause;
hist(proxy400_read(:,2), 100);
xlabel('Transfer size (Bytes)');
ylabel('Number of occurrences');
title('Disk data read sizes histogram at 400 xaction/sec load');
disp('press any key to plot queue sizes')
pause;
plot(proxy400_read(:,3));
xlabel('Event records');
ylabel('Disk queu size (Bytes)');
title('Disk data read queus sizes at 400 xaction/sec load');
disp('press any key to view histogram of queue sizes');
pause;
hist(proxy400_read(:,3));
xlabel('Queue size (Bytes)');
ylabel('Number of occurrences');
title('Disk data read queue size histogram at 400 xaction/sec
load');
disp('press any key to analyze writes at 400 xactions/sec
load');
pause;
% writes
clear;
load proxy400_write.dat;
disp('histogram of address offsets');
hist(proxy400_write(:, 4));
xlabel('Address offsets (32-bit)');
ylabel('Number of occurrences');
title('Disk write address offset histogram at 400 xaction/sec
load');
disp('press any key to continue to histogram of strides');
pause;
write_oset = proxy400_write(:, 4);
l = length(write_oset);
a = write_oset(1:l-1);
b = write_oset(2:l);
write_strides = abs(b - a);
[N, X] = hist(write_strides);
bar(X, N);
xlabel('Absolute values of strides');
ylabel('Number of occurrences');
title('Successive disk write access strdes histogram at 400
xaction/sec load');
```

```
disp('press any key to view histogram of write data sizes');
pause;
hist(proxy400_write(:,2), 100);
xlabel('Transfer size (Bytes)');
ylabel('Number of occurrences');
title('Disk data write sizes histogram at 400 xaction/sec load');
disp('press any key to plot queue sizes')
pause;
plot(proxy400_write(:,3));
xlabel('Event records');
ylabel('Disk queu size (Bytes)');
title('Disk data write queus sizes at 400 xaction/sec load');
disp('press any key to view histogram of queue sizes');
pause;
hist(proxy400_write(:,3));
xlabel('Queue size (Bytes)');
ylabel('Number of occurrences');
title('Disk data write queue size histogram at 400 xaction/sec
load');
disp('press any key to analyze reads/writes at 800 xactions/sec
load');
pause;
% analysis of 800 xactions/s load
% reads
clear;
load proxy800_read.dat;
disp('histogram of address offsets');
hist(proxy800_read(:, 4));
xlabel('Address offsets (32-bit)');
ylabel('Number of occurrences');
title('Disk read address offset histogram at 800 xaction/sec
load');
disp('press any key to continue to histogram of strides');
pause;
read_oset = proxy800_read(:, 4);
l = length(read_oset);
a = read_oset(1:l-1);
b = read_oset(2:l);
read_strides = abs(b - a);
[N, X] = hist(read_strides);
bar(X, N);
xlabel('Absolute values of strides');
ylabel('Number of occurrences');
title('Successive disk read access strdes histogram at 800
xaction/sec load');
disp('press any key to view histogram of read data sizes');
pause;
hist(proxy800_read(:,2));
xlabel('Transfer size (Bytes)');
ylabel('Number of occurrences');
title('Disk data read sizes histogram at 800 xaction/sec load');
disp('press any key to plot queue sizes')
pause;
plot(proxy800_read(:,3));
xlabel('Event records');
ylabel('Disk queu size (Bytes)');
title('Disk data read queus sizes at 800 xaction/sec load');
disp('press any key to view histogram of queue sizes');
pause;
hist(proxy800_read(:,3));
xlabel('Queue size (Bytes)');
ylabel('Number of occurrences');
title('Disk data read queue size histogram at 800 xaction/sec
load');
disp('press any key to analyze writes at 800 xactions/sec
load');
pause;
% writes
clear;
load proxy800_write.dat;
disp('histogram of address offsets');
hist(proxy800_write(:, 4));
xlabel('Address offsets (32-bit)');
ylabel('Number of occurrences');
title('Disk write address offset histogram at 800 xaction/sec
load');

disp('press any key to continue to histogram of strides');
pause;
write_oset = proxy800_write(:, 4);
l = length(write_oset);
a = write_oset(1:l-1);
b = write_oset(2:l);
write_strides = abs(b - a);
[N, X] = hist(write_strides);
bar(X, N);
xlabel('Absolute values of strides');
ylabel('Number of occurrences');
title('Successive disk write access strdes histogram at 800
xaction/sec load');
disp('press any key to view histogram of write data sizes');
pause;
hist(proxy800_write(:,2));
xlabel('Transfer size (Bytes)');
ylabel('Number of occurrences');
title('Disk data write sizes histogram at 800 xaction/sec load');
disp('press any key to plot queue sizes')
pause;
plot(proxy800_write(:,3));
xlabel('Event records');
ylabel('Disk queu size (Bytes)');
title('Disk data write queus sizes at 800 xaction/sec load');
disp('press any key to view histogram of queue sizes');
pause;
hist(proxy800_write(:,3));
xlabel('Queue size (Bytes)');
ylabel('Number of occurrences');
title('Disk data write queue size histogram at 800 xaction/sec
load');
```

Appendix D.3: Matlab script to analyze CD copying trace output.

```
% analysis CD copy writes to disk (no reads)
% reads
clear;
load cd_write.dat;
disp('histogram of address offsets');
hist(cd_write(:, 4));
xlabel('Address offsets (32-bit)');
ylabel('Number of occurrences');
title('Disk write address offset histogram for CD copying');
disp('press any key to continue to histogram of strides');
pause;
read_oset = cd_write(:, 4);
l = length(read_oset);
a = read_oset(1:l-1);
b = read_oset(2:l);
read_strides = abs(b - a);
[N, X] = hist(read_strides);
bar(X, N);
xlabel('Absolute values of strides');
ylabel('Number of occurrences');
title('Successive disk write access strdes histogram for CD
copying');
disp('press any key to view histogram of read data sizes');
pause;
hist(cd_write(:,2));
xlabel('Transfer size (Bytes)');
ylabel('Number of occurrences');
```

```
title('Disk data write sizes histogram for CD copying');
disp('press any key to plot queue sizes')
pause;
plot(cd_write(:,3));
xlabel('Event records');
ylabel('Disk queue size (Bytes)');
title('Disk data write queus sizes for CD copying');
disp('press any key to view histogram of queue sizes');
pause;
hist(cd_write(:,3));
xlabel('Queue size (Bytes)');
ylabel('Number of occurrences');
title('Disk data write queue size histogram for CD copying');
disp('press any key to view number if IO writes');
pause;
size(cd_write(:,4));
```

Appendix D.4: Sample awk script to process I/O read request traces from Tracelog

```
($1!=""){
  FS = ",";
  # Need to find DiskIoRead records for
  # disk # 2 and task_id = 0x853BF7E8
  #if((NF==13) && (substr($1, 0, 10) == "DiskIoRead")){
  if((NF==13) && ($1 == "DiskIoRead      ")){
          if (($6==0) && ($11==" 0x85868AC8")){
            print $7, $8, $9, $10
                  }
  }
}
```

Appendix D.5: Sample awk script to process I/O write request traces from Tracelog

```
($1!=""){
  FS = ",";
  # Need to find DiskIoWrite records for
  # disk # 2 and task_id = 0x85868AC8
  #if((NF==13) && (substr($1, 0, 11) == "DiskIoWrite")){
  if((NF==13) && ($1 == "DiskIoWrite      ")){
    if (($6==0) && ($11==" 0x85D634D8")){
          print $7, $8, $9, $10
                }
  }
}
```

Appendix D.6. DiskSim prefetching scheme

```
*/
extern "C"{
#include "disksim_global.h"
#include "disksim_ioface.h"
#include "disksim_pfface.h"
#include "disksim_iotrace.h"
}
#include "cachemap.h"
#include "streamid.h"
#ifdef SUPPORT_CHECKPOINTS
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#endif

// Setup the Prefetch queue
extern "C" {
#include "queue.h"
Queue *prefetchQueue = newQueue();
}
// Read command line arguments
#include "disksim_readargs.h"
extern struct arguments arguments; /* Defined in
disksim_readargs.h */
// Adaptive Prefetching Code
#include "adaptive/Block.h"
#include "adaptive/BlockContainer.h"
#include "adaptive/RSL.h"
#include <math.h>
// Demand Access Classification Code
#include "classify.h"
// Prototypes
event * getPrefetchEvent (basecache* b, ioreq_event*
demandEvent, int, int);
void adpPrefetchRead(ioreq_event* io_event, basecache* b);

disksim_t *disksim = NULL;
/* legacy hack for HPL traces... */
#define PRINT_TRACEFILE_HEADER     FALSE
/* function pointers for call backs from disksim */
void
(*external_cache_req_arrive)(my_cache_event*)=NULL;
void (*external_cache_req_done)(my_cache_event*)=NULL;
/*** Functions to allocate and deallocate empty event
structures ***/
/* Creates new, empty events (using malloc) and puts them on
the extraq. */
/* Called by getfromextraq when the queue is found to be
empty.       */
static void allocateextra ()
{
  int i;
  event *temp = NULL;
  StaticAssert (sizeof(event) == DISKSIM_EVENT_SIZE);
  if ((temp = (event *)DISKSIM_malloc(ALLOCSIZE)) ==
NULL) {
    fprintf (stderr, "Error allocating space for events\n");
    exit(0);
  }
  for (i=0; i<((ALLOCSIZE/DISKSIM_EVENT_SIZE)-1);
i++) {
    temp[i].next = &temp[i+1];
  }
 emp[((ALLOCSIZE/DISKSIM_EVENT_SIZE)-1)].next =
disksim->extraq;
  disksim->extraq = temp;
  disksim->extraqlen = ALLOCSIZE /
DISKSIM_EVENT_SIZE;
}
/* Deallocates an event structure, adding it to the extraq free
pool. */
void addtoextraq (event *temp)
{
  if (temp == NULL) {
    return;
  }
  temp->next = disksim->extraq;
  temp->prev = NULL;
  disksim->extraq = temp;
  disksim->extraqlen++;
}
/* Allocates an event structure from the extraq free pool; if
empty, */
/* calls allocateextra to create some more.                */
event * getfromextraq ()
```

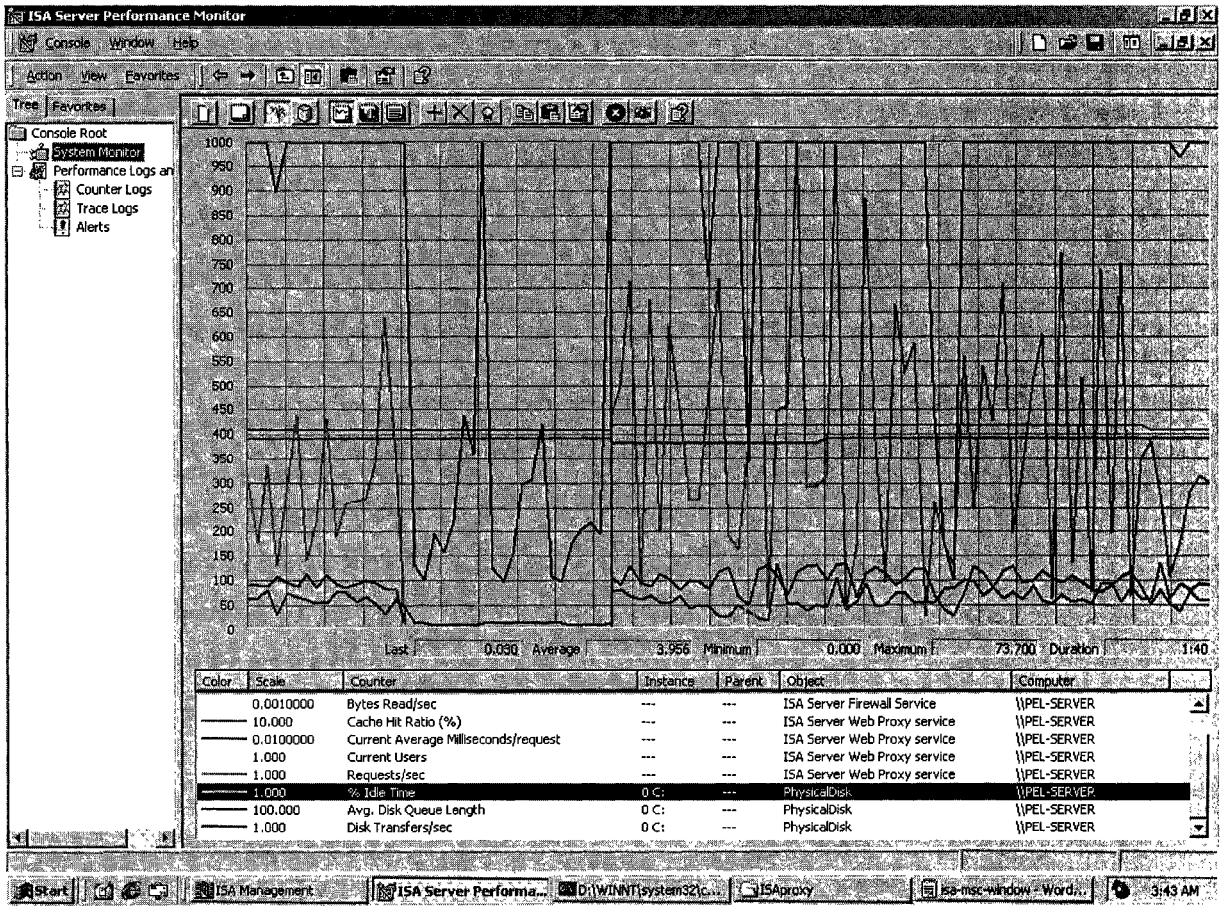
```
{
  event *temp = NULL;
  temp = disksim->extraq;
  if (disksim->extraqlen == 0) {
    allocateextra();
    temp = disksim->extraq;
    disksim->extraq = disksim->extraq->next;
  } else if (disksim->extraqlen == 1) {
    disksim->extraq = NULL;
  } else {
    disksim->extraq = disksim->extraq->next;
  }
  disksim->extraqlen--;
  temp->next = NULL;
  temp->prev = NULL;
  return(temp);
}
/* Deallocates a list of event structures to the extraq free pool.
*/
void addlisttoextraq (event **headptr)
{
  event *tmp;

  while ((tmp = *headptr)) {
    *headptr = tmp->next;
    addtoextraq(tmp);
  }
  *headptr = NULL;
}
/* Returns a pointer to a copy of event passed as a parameter.
*/
event *event_copy (event *orig)
{
  event *new1 = getfromextraq();
  memmove((char *)new1, (char *)orig,
DISKSIM_EVENT_SIZE);
/* bcopy ((char *)orig, (char *)new,
DISKSIM_EVENT_SIZE); */
  return((event *) new1);
}
/*** Functions to manipulate intq, the queue of scheduled
events ***/

/* Prints the intq to the output file, presumably for debug. */

#if 0
static void printintq ()
{
  event *tmp;
  int i = 0;
  tmp = disksim->intq;
  while (tmp != NULL) {
    i++;
    fprintf (outputfile, "Item #%d: time %f, type %d\n", i,
tmp->time, tmp->type);
    tmp = tmp->next;
  }
}
#endif
/* Add an event to the intq. The "time" field indicates when
the event is */
/* scheduled to occur, and the intq is maintained in ascending
time order. */
void addtointq (event *temp)
{
  event *run = NULL;

  if ((temp->time + DISKSIM_TIME_THRESHOLD) <
simtime) {
    fprintf(stderr, "Attempting to addtointq an event whose
time has passed\n");
    fprintf(stderr, "simtime %f, curr->time %f, type = %d\n",
simtime, temp->time, temp->type);
    exit(0);
  }
  if (disksim->intq == NULL) {
    disksim->intq = temp;
    temp->next = NULL;
    temp->prev = NULL;
  } else if (temp->time < disksim->intq->time) {
    temp->next = disksim->intq;
    disksim->intq->prev = temp;
    disksim->intq = temp;
    temp->prev = NULL;
  } else {
    run = disksim->intq;
    while (run->next != NULL) {
      if (temp->time < run->next->time) {
        break;
      }
      run = run->next;
    }
    temp->next = run->next;
    run->next = temp;
    temp->prev = run;
    if (temp->next != NULL) {
      temp->next->prev = temp;
    }
  }
}
/* Retrieves the next scheduled event from the head of the
intq. */
 static event * getfromintq ()
{
  event *temp = NULL;
  if (disksim->intq == NULL) {
    return(NULL);
  }
  temp = disksim->intq;
  disksim->intq = disksim->intq->next;
  if (disksim->intq != NULL) {
    disksim->intq->prev = NULL;
  }
  temp->next = NULL;
  temp->prev = NULL;
  return(temp);
}
/* Removes a given event from the intq, thus descheduling it.
Returns */
/* TRUE if the event was found, FALSE if it was not.
*/
int removefromintq (event *curr)
{
  event *tmp;
  tmp = disksim->intq;
  while (tmp != NULL) {
    if (tmp == curr) {
      break;
    }
    tmp = tmp->next;
  }
  if (tmp == NULL) {
    return(FALSE);
  }
  if (curr->next != NULL) {
    curr->next->prev = curr->prev;
  }
  if (curr->prev == NULL) {
    disksim->intq = curr->next;
```

```
  } else {
    curr->prev->next = curr->next;
  }
  curr->next = NULL;
  curr->prev = NULL;
  return(TRUE);
}
/***** these are the functions to handle prefetch events ****/

/* Add an event to the prefq. The "time" field indicates when
the event is */
/* scheduled to occur, and the intq is maintained in ascending
time order. */
void addtoprefq (event *temp)
{
  event *run = NULL;
  if ((temp->time + DISKSIM_TIME_THRESHOLD) <
simtime) {
    fprintf(stderr, "Attempting to addtoprefq an event whose
time has passed\n");
    fprintf(stderr, "simtime %f, curr->time %f, type = %d\n",
simtime, temp->time, temp->type);
    exit(0);
  }
  if (disksim->prefq == NULL) {
    disksim->prefq = temp;
    temp->next = NULL;
    temp->prev = NULL;
  } else if (temp->time < disksim->prefq->time) {
    temp->next = disksim->prefq;
    disksim->prefq->prev = temp;
    disksim->prefq = temp;
    temp->prev = NULL;
  } else {
    run = disksim->prefq;
    while (run->next != NULL) {
      if (temp->time <= run->next->time) {
        break;
      }
      run = run->next;
    }
    temp->next = run->next;
    run->next = temp;
    temp->prev = run;
    if (temp->next != NULL) {
      temp->next->prev = temp;
    }
  }
}
/* Retrieves the next scheduled event from the head of the
prefq. */
static event * getfromprefq ()
{
  event *temp = NULL;
  if (disksim->intq == NULL) {
    return(NULL);
  }
  temp = disksim->prefq;
  disksim->prefq = disksim->prefq->next;
  if (disksim->prefq != NULL) {
   disksim->prefq->prev = NULL;
  }
  temp->next = NULL;
  temp->prev = NULL;
  return(temp);
}

/* Removes a given event from the prefq, thus descheduling
it. Returns */
/* TRUE if the event was found, FALSE if it was not.
*/
int removefromprefq (event *curr) {
  event *tmp;
  tmp = disksim->intq;
  while (tmp != NULL) {
    if (tmp == curr) {
      break;
    }
    tmp = tmp->next;
  }
  if (tmp == NULL) {
    return(FALSE);
  }
  if (curr->next != NULL) {
    curr->next->prev = curr->prev;
  }
  if (curr->prev == NULL) {
    disksim->prefq = curr->next;
  } else {
    curr->prev->next = curr->next;
  }
  curr->next = NULL;
  curr->prev = NULL;
  return(TRUE);
}
void generatePrefetchRequests(basecache *b, ioreq_event
*newEvent) {
  prefetchBlkStr* blockStr;

  switch (arguments.prefetch_mode) {
    case 2:
      adpPrefetchRead( newEvent, b );
      break;
    case 3:
      recordOccurence( newEvent->blkno, newEvent->time );
      break;

    default:
      return;
  }
}
void issuePrefetchRequests(basecache *b, ioreq_event
*newEvent) {
  /* Inject prefetch requests into the internal Disksim queue.
*/

  prefetchBlkStr* blockStr = NULL;

  // For each prefetch request, check cache before placing
request into internal queue.
  while ( !isQueueEmpty(prefetchQueue) ) {
    blockStr = (prefetchBlkStr *) popQueue(prefetchQueue);

    // find out if the starting block/sector and
arguments.numSectorToPrefetch is in the cache
    if ( !b->queryCache( blockStr->blockNum, blockStr-
>blockCount ) ) {
      addtointq( getPrefetchEvent(b, newEvent, blockStr-
>blockNum, blockStr->blockCount ) );
      b->insert_in_cache( blockStr->blockNum, blockStr-
>blockCount );
    } else {
#ifdef PRE_DEBUG
      printf("*");
#endif
    }
    free(blockStr);
  }
  assert( isQueueEmpty(prefetchQueue) );
```

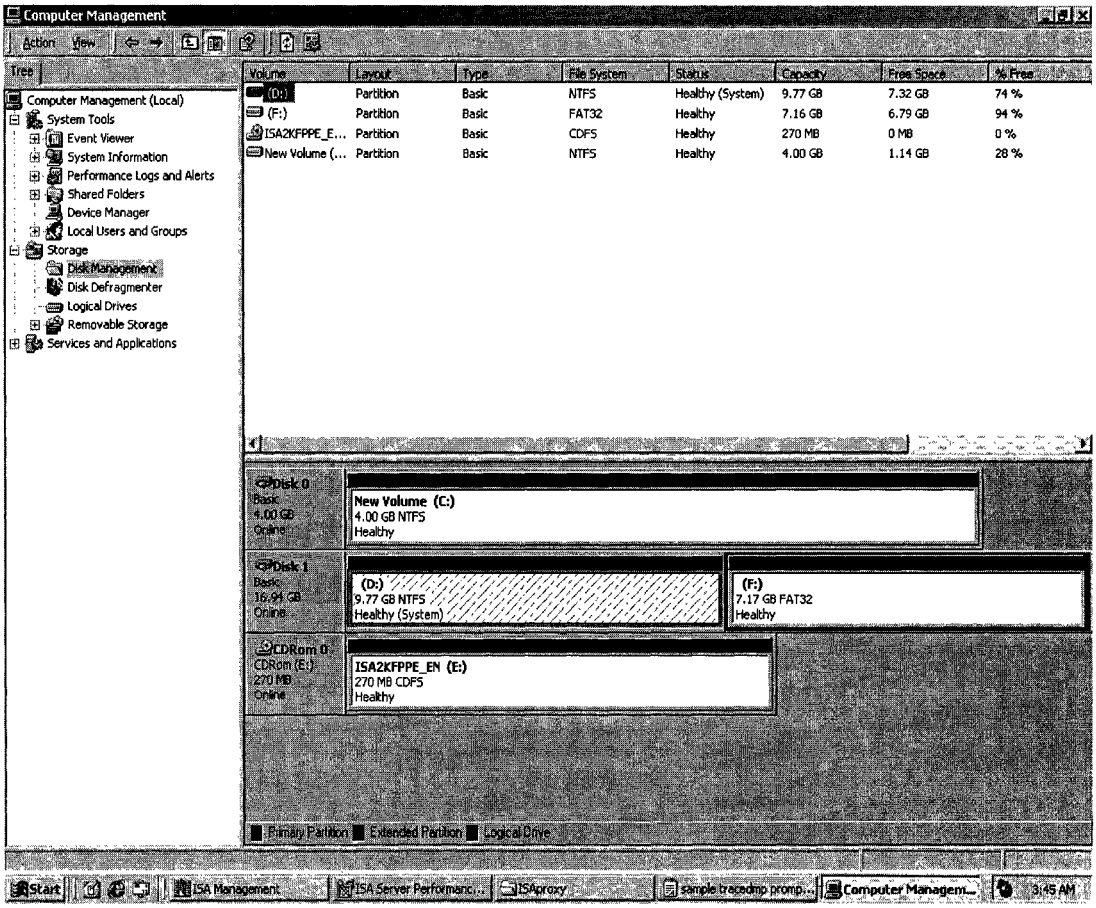
```
}
ioreq_event* getNextExternalEvent(FILE* tracefile,
basecache* b) {
  ioreq_event* newEvent = NULL;
  int isCacheHit = 1;

  while ( isCacheHit ) {
    newEvent = (ioreq_event*) io_get_next_external_event(
tracefile, b );
        /* Are there any events left? */
    if (newEvent == NULL)
      break;

    isCacheHit = b->queryUpdate(newEvent->blkno,
newEvent->bcount);
    if ( !isCacheHit )
      b->insert_in_cache( newEvent->blkno, newEvent-
>bcount );
    else {
#ifdef PRE_DEBUG
      printf(stderr, "CACHE HIT on %d + %d\n", newEvent-
>blkno, newEvent->bcount);
#endif
    }
   generatePrefetchRequests(b, newEvent);
    /* If cache miss, insert demand request in internal disksim
event queue.
     * Otherwise, discard the request. */
    if ( isCacheHit )
      addtoextraq((event *) newEvent);
    else {
      newEvent->type = NULL_EVENT;
      addtointq( (event*) newEvent );
    }
    /* Insert prefetch requests in internal disksim event queue.
*/
    issuePrefetchRequests(b, newEvent);
  }
  return newEvent;
}
void set_external_cache_req_arrive(void
(*handler)(my_cache_event*)) {
external_cache_req_arrive=handler;
 }
void set_external_cache_req_done(void
(*handler)(my_cache_event*)) {
 external_cache_req_done=handler;
}
void adpPrefetchRead(ioreq_event* io_event, basecache* b)
{
  /* Translate sector numbers into blocks number, record
occurence of
   * io_event in the RSL window, find blocks that are likely
to occur
   * soon and enqueue them in prefetchQueue.
   */
  int returnCode, i;
  prefetchBlkStr *blockStr;
  int numSectorsPrefetched = 0;
  static RSL Window( arguments.window_size,
arguments.window_type ); // Create a Recently-Seen-List
list<windowElem>::const_iterator windowElemCIt;
  list<void*>::const_iterator bContCIt;
  Block* block;
  Block* refBlock;
  bContainer* bCont;
  int sectorNum  = io_event->blkno;
  int sectorCount = io_event->bcount;
  int blockSize  = arguments.block_size;
  int sectorSize  = arguments.sector_size;

  /* Traslate sectors to blocks */
  int blockNum  = (int) floor( (double) (sectorNum *
sectorSize) / (double) blockSize );
  int blockCount = (int) ceil( (double) sectorCount / (double)
blockSize );

  /* Test Code */
  blockNum = io_event->blkno + io_event->blkno % 2;
  blockCount = io_event->bcount / 2;
  /* End Test Code */
  /* Record occurence of blocks */
  for (i = blockNum; i <= blockNum + blockCount; i += 2) {
    Window.insertBlock( i );
#ifdef PRE_DEBUG
    fprintf(stderr, "Recording %d.\n", i);
#endif
    /* Queue prefetch requests here. */
    /* Loop through each of the blocks in the RSL Window.
     * For each block, loop through the block container list.
     * For each block in the block container list, if the
probability
     * is greater than our threshold, enqueue the block in the
prefetch
     * queue.
     */
    windowElemCIt = Window.getBlockListBegin();
    assert( windowElemCIt != Window.getBlockListEnd() );

    block = windowElemCIt->blockPtr;
    while ( block->getBn() != i ) {
      windowElemCIt++;
      block = windowElemCIt->blockPtr;
    }
    bContCIt = block->getBContIt(); // Start of container
(follow) list
    while ( bContCIt != block->getBContEnd() ) {
```

APPENDIX E

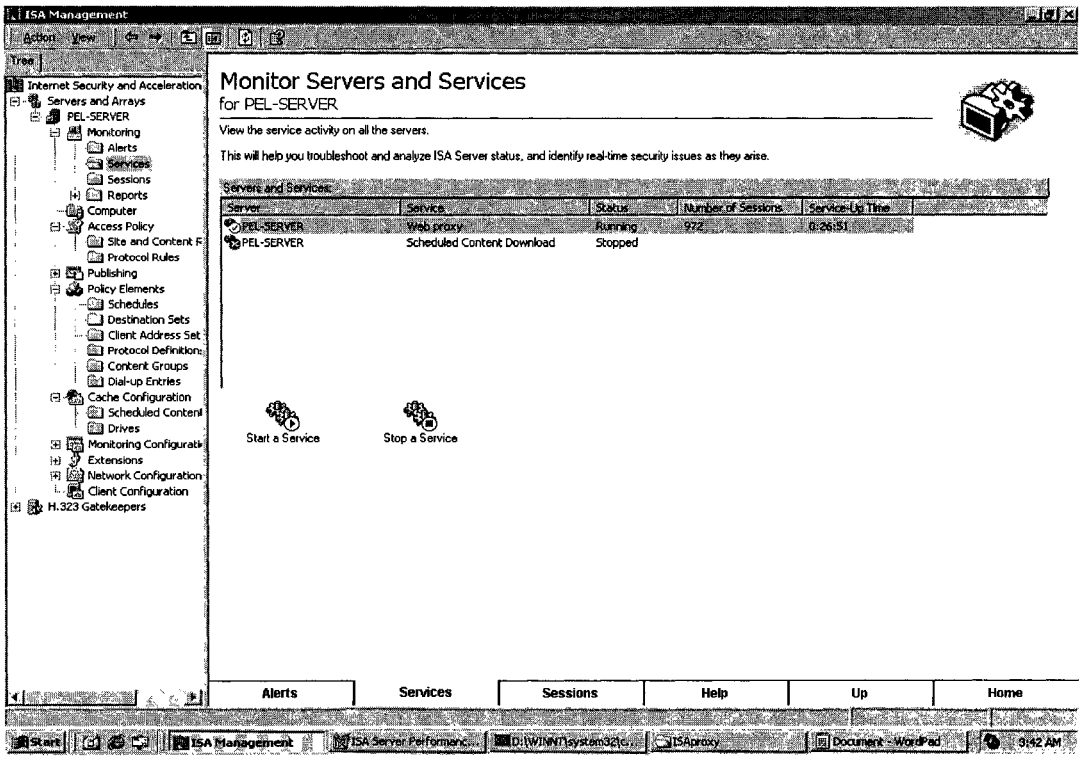
Appendix E.1 : Performance Monitor of Window 2000 server under heavy load



Appendix E.2 Disk Management Snap Short in Windows 2000 server



Appendix E.3 ISA Web Proxy Management Console



APPENDIX F

Physical specification of SCSI disk used for our experiment WDE 4360

Physical Specifications

Formatted Capacity 1 4360 MB
Interface(s) Ultra Fast (50-pin)
Ultra Fast Wide (68 and 80-pin SCA-2)
Actuator Type Rotary Voice Coil
Media Type Thin Film
Head Type Thin Film Inductive
Servo Type Embedded
Number of Disks 4
Number of Data Surfaces 8
Number of Heads 8
Bytes per Sector 512
Tracks per Inch 6150
Bits per Inch (000) 125 (ID)
Areal Density 769 MB/in2
Total Cylinders 5956
Zones per Surface 20
User Sectors per Drive (512 byte) 225 (OD), 133 (ID)
Recording Method 0,4,4 RLL 2
ECC 3 144-bit Reed Solomon

Performance Specifications

Average Seek
- Read
- Write 8 ms
9.5 ms
Track to Track Seek
- Read
- Write 1 ms
2.5 ms
Full Stroke Seek < 18 ms
Average Latency 4.17 ms
Rotational Speed 7200 RPM
Data Transfer Rate
- Media to Buffer
- Buffer to Host
81-140 Mbits/s
40 MB/s max.
Buffer Size 1 512 KB
1 MB optional
Error Rate - Unrecoverable < 1 in 1014 bits read
Spindle Start Time < 30s to ready
Spindle Stop Time < 20s
Contact Start/Stop Cycles (CSS) 2 20,000

Specifications for the WD Enterprise WDE9180 SCSI Hard Drive

Physical Specifications

Formatted Capacity 1 9150 MB
Interfaces Ultra2 LVD, Ultra2 LVD Wide
(68-pin)
(80-pin SCA-2)
Ultra Fast, Ultra Fast Wide
(68-pin)
(80-pin SCA-2)
Media Type Thin Film
Head Type MR
Servo Type Embedded

Number of Disks 3
Number of Data Surfaces 6
Number of Heads 6
Bytes per Sector 2 512
Tracks per Inch 10750
Bits per Inch (000) 208.8 (ID), 180.42 (OD)
Areal Density 2.24 Gb/in2
Capacity/Platter 3.0 GB/disk
Total Cylinders 10,601
Zones per Surface 16
Recording Method 3 0,6,8 EPR4
ECC 240-bit Reed Solomon
Performance Specifications
Data Transfer Rate
- Media to Buffer
- Buffer to Host
30 MB/s max
80 MB/s max
Average Seek
- Read
- Write
6.9 ms
7.9 ms
Track to Track Seek
- Read
- Write
0.8 ms
0.8 ms
Full Stroke Seek
- Read
- Write
16 ms
17 ms
Average Latency 4.17 ms
Rotational Speed 7200 RPM
Buffer Size 2 MB, 4 MB
Error Rate - Unrecoverable < 1 in 1014 bits read
Spindle Start Time < 20s to ready
Spindle Stop Time < 20s
Contact Start/Stop Cycles (CSS) 1 20,000

APPENDIX G:

Sample tracelog output

Event Name	TID	Clock-Time	Kern el-ms	User -ms	User Data...						IID	PIID
DiskIo Write	0x07EC	1.27E+17	45	15	1	0x00000043	1536	28376	1.08E+10	0x850955A8	0	0
DiskIo Write	0x001C	1.27E+17	210	0	1	0x00000043	4096	43111	5.26E+09	0x85E5C038	0	0
DiskIo Write	0x001C	1.27E+17	210	0	1	0x00000043	4096	6960	5.24E+09	0x85E5C038	0	0
DiskIo Write	0x001C	1.27E+17	210	0	1	0x00000043	4096	20380	5.24E+09	0x85E5C038	0	0
DiskIo Write	0x0150	1.27E+17	120	390	0	0x00000043	4096	17071	2.15E+09	0x85D634D8	0	0
DiskIo Write	0x0150	1.27E+17	120	390	0	0x00000A01	512	49708	1.42E+09	0x84E80128	0	0
FileIo	0x0150	1.27E+17	120	390	0x84E80128	"\urlcache\Dir1.cdat"	0	0				
DiskIo Write	0x0150	1.27E+17	120	390	0	0x00000A01	593920	337628	3.47E+09	0x85449128	0	0
FileIo	0x0150	1.27E+17	120	390	0x85449128	"\urlcache\Dir1.cdat"	0	0				
DiskIo Write	0x001C	1.27E+17	210	0	1	0x00000043	64512	43718	1.08E+10	0x8503BAE8	0	0

DiskIo Write	0x0150	1.27E+17	285	525	0	0x00000A01	473088	254946	3.47E+09	0x85449128	0	0
DiskIo Write	0x07BC	1.27E+17	225	60	0	0x00000A01	65536	60139	3.47E+09	0x85449128	0	0
DiskIo Write	0x001C	1.27E+17	210	0	0	0x00000043	4096	59340	2.15E+09	0x85D634D8	0	0
DiskIo Write	0x001C	1.27E+17	210	0	0	0x00000043	4096	43250	2.15E+09	0x85D634D8	0	0
DiskIo Write	0x001C	1.27E+17	210	0	0	0x00000043	4096	26958	2.15E+09	0x85D634D8	0	0

APPENDIX H

Sample output from simulator

Overall I/O System Total Requests handled: 10000
Overall I/O System Requests per second: 77.946397
Overall I/O System Completely idle time: 0.000000 0.000000
Overall I/O System Response time average: 64.133790
Overall I/O System Response time std.dev.: 5.976555
Overall I/O System Response time maximum: 87.189787
Overall I/O System Response time distribution
Overall I/O System Overlaps combined: 0 0.000000
Overall I/O System Read overlaps combined: 0 0.000000 0.000000
Overall I/O System Critical Reads: 0 0.000000
Overall I/O System Critical Read Response time average: 0.000000
Overall I/O System Critical Read Response time std.dev.: 0.000000
Overall I/O System Critical Read Response time maximum: 0
Overall I/O System Critical Read Response time distribution
Overall I/O System Non-Critical Reads: 0 0.000000
Overall I/O System Non-Critical Read Response time average: 0.000000
Overall I/O System Non-Critical Read Response time std.dev.: 0.000000
Overall I/O System Non-Critical Read Response time maximum: 0
Overall I/O System Non-Critical Read Response time distribution
Overall I/O System Critical Writes: 10000 0.999600
Overall I/O System Critical Write Response time average: 64.133790
Overall I/O System Critical Write Response time std.dev.: 5.976555
Overall I/O System Critical Write Response time maximum: 87.189787
Overall I/O System Critical Write Response time distribution
Overall I/O System Non-Critical Writes: 0 0.000000
Overall I/O System Non-Critical Write Response time average: 0.000000
Overall I/O System Non-Critical Write Response time std.dev.: 0.000000
Overall I/O System Non-Critical Write Response time maximum: 0
Overall I/O System Non-Critical Write Response time distribution
Overall I/O System Number of reads: 0 0.000000
Overall I/O System Number of writes: 10004 1.000000
Overall I/O System Sequential reads: 0 0.000000 0.000000
Overall I/O System Sequential writes: 0 0.000000 0.000000
Overall I/O System Base SPTF/SDF Different: 0 / 0 0.000000
Overall I/O System Timeout SPTF/SDF Different: 0 / 0 0.000000
Overall I/O System Priority SPTF/SDF Different: 0 / 0 0.000000
Overall I/O System runlistlen: 641466.471697
Overall I/O System runoutstanding: 0.000000
Overall I/O System simtime: 128293.294339
Overall I/O System warmuptime: 0.000000
Overall I/O System setsize: 1
Overall I/O System Average # requests: 5.000000
Overall I/O System Maximum # requests: 5
Overall I/O System End # requests: 4
Overall I/O System Average queue length: 5.000000
Overall I/O System Maximum queue length: 5
Overall I/O System End queued requests: 4
Overall I/O System Queue time average: 64.133790
Overall I/O System Queue time std.dev.: 5.976555
Overall I/O System Queue time maximum: 87.189787
Overall I/O System Queue time distribution
Overall I/O System Avg # read requests: 0.000000
Overall I/O System Max # read requests: 0
Overall I/O System Avg # write requests: 5.000000
Overall I/O System Max # write requests: 5
Overall I/O System Physical access time average: 0.000000
Overall I/O System Physical access time std.dev.: 0.000000
Overall I/O System Physical access time maximum: 0
Overall I/O System Physical access time distribution
Overall I/O System Inter-arrival time average: 12.823084
Overall I/O System Inter-arrival time std.dev.: 3.207278
Overall I/O System Inter-arrival time maximum: 23.228450
Overall I/O System Inter-arrival time distribution
Overall I/O System Read inter-arrival average: 0.000000
Overall I/O System Read inter-arrival std.dev.: 0.000000
Overall I/O System Read inter-arrival maximum: 0
Overall I/O System Read inter-arrival distribution
Overall I/O System Write inter-arrival average: 12.823084
Overall I/O System Write inter-arrival std.dev.: 3.207278
Overall I/O System Write inter-arrival maximum: 23.228450
Overall I/O System Write inter-arrival distribution
Overall I/O System Number of idle periods: 1
Overall I/O System Idle period length average: 0.000000
Overall I/O System Idle period length std.dev.: 0.000000
Overall I/O System Idle period length maximum: 0
Overall I/O System Idle period length distribution
Overall I/O System Request size average: 8.890044
Overall I/O System Request size std.dev.: 3.647106

Overall I/O System Request size maximum: 56
Overall I/O System Request size distribution
Overall I/O System Read request size average:
0.000000
Overall I/O System Read request size std.dev.:
0.000000
Overall I/O System Read request size maximum: 0
Overall I/O System Read request size distribution
Overall I/O System Write request size average:
8.890044
Overall I/O System Write request size std.dev.:
3.647106
Overall I/O System Write request size maximum: 56
Overall I/O System Write request size distribution
Overall I/O System Instantaneous queue length average:
5.000000
Overall I/O System Instantaneous queue length std.dev.:
0.000000
Overall I/O System Instantaneous queue length maximum:
5
Overall I/O System Instantaneous queue length distribution
Overall I/O System Sub-optimal mapping penalty average:
0.000000
Overall I/O System Sub-optimal mapping penalty std.dev.:
0.000000
Overall I/O System Sub-optimal mapping penalty maximum:
0
Overall I/O System Sub-optimal mapping penalty
distribution
Disk Seeks of zero distance: 0 0.000000
Disk Seek distance average: 5011.723172
Disk Seek distance std.dev.: 2432.062190
Disk Seek distance maximum: 10314
Disk Seek distance distribution
Disk Seek time average: 8.360483
Disk Seek time std.dev.: 1.946002
Disk Seek time maximum: 12.225750
s: 9999
Disk Seek time distribution
Disk Full rotation time: 4620880867666415017
Disk Zero rotate latency: 0 0.000000
Disk Rotational latency average: 4.195748
Disk Rotational latency std.dev.: 2.389641
Disk Rotational latency maximum: 8.332968
Disk Rotational latency distribution
Disk Transfer time average: 0.274292
Disk Transfer time std.dev.: 0.264849
Disk Transfer time maximum: 3.088579
Disk Transfer time distribution
Disk Positioning time average: 12.556231
Disk Positioning time std.dev.: 3.067059
Disk Positioning time maximum: 20.396728
Disk Positioning time distribution
Disk Access time average: 12.830523
Disk Access time std.dev.: 3.074026
Disk Access time maximum: 21.546692
Disk Access time distribution
Disk Number of buffer accesses: 10000
Disk Buffer hit ratio: 0 0.000000
Disk Buffer miss ratio: 10000 1.000000
Disk Buffer read hit ratio: 0 0.000000
0.000000
Disk Buffer prepend hit ratio: 0 0.000000
Disk Buffer append hit ratio: 0 0.000000
Disk Write combinations: 0 0.000000
Disk Ongoing read-ahead hit ratio: 0 0.000000
0.000000
Disk Average read-ahead hit size: 0.000000
Disk Average remaining read-ahead: 0.000000
Disk Partial read hit ratio: 0 0.000000 0.000000
Disk Average partial hit size: 0.000000
Disk Average remaining partial: 0.000000
Disk Total disk bus wait time: 0.000000
Disk Number of disk bus wait

APPENDIX I

0x00000043	65536	77435	2171158016
0x00000043	65536	27936	2171223552
0x00000043	65536	26554	2171289088
0x00000043	65536	22229	2171354624
0x00000043	65536	32443	2171420160
0x00000043	65536	8155	2171485696
0x00000043	65536	9354	2171551232
0x00000043	65536	28246	2171616768
0x00000043	65536	18868	2171682304
0x00000043	65536	9739	2171747840
0x00000043	65536	27486	2171813376
0x00000043	65536	21526	2171878912
0x00000043	65536	10591	2171944448
0x00000043	65536	9311	2172009984
0x00000043	65536	37351	2172075520
0x00000043	65536	8154	2172141056
0x00000043	65536	13987	2172206592
0x00000043	65536	62449	2172272128
0x00000043	65536	16680	2172337664
0x00000043	65536	14968	2172403200
0x00000043	65536	24611	2172468736
0x00000043	65536	29656	2172534272
0x00000043	65536	41742	2172599808
0x00000043	65536	21028	2172665344
0x00000043	65536	21143	2172730880
0x00000043	65536	25975	2172796416
0x00000043	65536	30696	2172861952
0x00000043	65536	9240	2172927488
0x00000043	65536	18676	2172993024
0x00000043	65536	9132	2173058560
0x00000043	65536	8255	2173124096
0x00000043	65536	9284	2173189632
0x00000043	65536	25329	2173255168
0x00000043	65536	16528	2173320704
0x00000043	65536	11087	2173386240
0x00000043	65536	8522	2173451776
0x00000043	65536	23293	2173517312
0x00000043	65536	10422	2173582848
0x00000043	65536	30595	2173648384
0x00000043	65536	8035	2173713920
0x00000043	65536	31672	2173779456
0x00000043	65536	14596	2173844992
0x00000043	65536	8129	2173910528
0x00000043	65536	8256	2173976064
0x00000043	65536	21791	2174041600
0x00000043	65536	31875	2174107136
0x00000043	65536	16063	2174172672
0x00000043	65536	11933	2174238208
0x00000043	65536	22623	2174303744
0x00000043	65536	31313	2174369280
0x00000043	65536	24197	2174434816

VITAE

- **Salam-Alada, Amisu Oluwakemi**
- Born in Nigeria.
- Completed Bachelor of Science (B.Sc) degree in Computer Engineering from Obafemi Awolowo University Ile-Ife, Nigeria in January 1998.
- Joined Computer Engineering Department, King Fahd University of Petroleum & Minerals (KFUPM) as a Research Assistant in February 2001.
- Completed MS in Computer Engineering from KFUPM, Dhahran, Saudi Arabia in January 2004.
- Email: aosalam@yahoo.com.