Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

A STOCHASTIC DISK I/O SIMULATION TECHNIQUE

Niki C. Thornock
Xiao-Hong Tu
J. Kelly Flanagan

Performance Evaluation Laboratory
Computer Science Department
Brigham Young University
Provo, Utah 84602, U.S.A.

ABSTRACT

In this paper, we describe a technique to construct accurate stochastic simulation models from acquired trace data. The resulting simulation models accept input trace data and return estimates of disk request service times. In addition, we describe a trace collection technique capable of collecting accurate trace data from Novell Netware servers. These traces are used to construct a simulation model of a Novell Netware I/O subsystem that is used to study the impact of disk data reorganization on I/O performance.

1- INTRODUCTION AND BACKGROUND

Processor speeds have increased considerably over the past few years while I/O speeds, particularly disk I/O, have lagged far behind. The resulting mismatch causes a bottleneck which reduces the performance of systems running I/O intensive applications. The need to model these types of systems so the performance impact of proposed alternatives can be measured and evaluated is well known.

Previous studies to increase disk I/O performance have been performed using trace-driven simulation (Ousterhout et al. 1985, Smith 1985, Floyd and Ellis 1989, Ramakrishnan et al.- 1992, Grimsrud et al. 1993). This type of modeling requires an accurate simulation model and trace data representative of the system being evaluated (Flanagan 1993). Simulation models are typically implemented using high level language descriptions of various subsystem components. An example of this type of simulator is presented in (Ruemmler 1994) in which the authors propose a simulation model requiring approximately 13,000 lines of commented C++ code. The accuracy of this type of model is dependent on the included detail and the validity of input parameters describing each component. A simulation model requires accurate parameters from each component of the system. Obviously an important component of a disk I/O subsystem is the disk drive itself. Two techniques for obtaining accurate parameters of disk drive mechanisms have been recently proposed. The first technique, proposed in (Ruemmler 1994), uses the disk parameters published by the disk drive manufacturer and regression analysis against their trace data. The second technique, proposed in (Worthington 1995), uses special tools and test vectors to drive SCSI disks and extract the desired drive parameters. Accurate models are difficult to construct and still require accurate trace data to produce meaningful results.

As previously mentioned, many studies investigating disk I/O performance have used disk trace data. Most of this trace data was collected at the file system level, before the operating system buffer cache, and does not contain high resolution timing information (Ousterhout et al. 1985, Smith 1985, Ramakrishnan et al. 1992, Floyd and Ellis 1989, Grimsrud et al. 1993). These traces are most useful for tuning and evaluating the performance of alternative operating system cache configurations or other pre-buffer cache components. They are not optimal for evaluating the remainder of the I/O subsystem: operating system device drivers, system buses (PCI, EISA, or others), disk controllers, I/O buses (SCSI, Fiber Channel, or others), caches and buffers integrated on modern drives, and drive mechanisms.

Ruemmler and Wilkes implemented and described a disk trace gathering technique that acquired trace data after operating system caches and buffers (Ruemmler and Wilkes 1993). Their trace data included timing information with a resolution of one microsecond. Their data was collected from three Hewlett-Packard workstations and was used to determine disk access characteristics of these systems. Their disk trace collection tool inspired much of this work.

This paper describes a new technique for creating disk I/O subsystem simulation models for use with trace-driven simulation. This technique requires accurate I/O traces containing timing information like

those collected by Ruemmler and Wilkes (1993). Unlike the work performed by Ruemmler and Wilkes, our work focuses on disk I/O performance of file servers running the Novell Netware operating system (Day 1993). We describe a tool to collect accurate disk I/O traces from Novell Netware servers and use these traces to construct a model of the system under test. A short case study is presented that demonstrates the usefulness of our simulation approach in determining the impact of reorganizing disk data on average I/O service times. Our simulation technique is not limited to studies involving Novell Netware servers, but is applicable to any system where disk I/O traces with timing information are available.

The remainder of this paper is organized into four sections. Section 2 describes our trace collection tool that provides us with the data necessary to construct a simulation model of a system. Section 3 discusses the construction and accuracy of our simulation model and proposes several uses for such a tool. Section 4 presents a short case study illustrating the usefulness of our simulation approach. Finally, Section 5 summarizes our work and outlines our future plans.

2- DISK TRACE COLLECTION TOOL

As previously mentioned, most disk traces that are available were collected before the operating system buffer cache and do not contain accurate timing information. Timing information is crucial for determining what real impact alternative I/O structures have on a system. Like the trace data collected by Ruemmler and Wilkes (1993), our traces contain very accurate timing information.

The remainder of this section describes our trace collection tool, the contents of acquired trace data, and the type of data that is available. All collected disk traces described in this paper are publicly available; for more information, access our web page at `http://pel.cs.byu.edu/`.

2.1- Disk Collection Tool

Our trace collection scheme relies on three basic components for data collection. They include the modification of a Novell disk device driver, a Netware Loadable Module (NLM) that intercepts and stores disk requests, and a tool that writes the buffered disk requests to secondary storage.

2.1.1- Modified Disk Device Driver

The disk I/O component of the Netware operating system can be thought of as having a hardware independent part supplied by Novell and a hardware specific part supplied by the disk drive controller manufacturer (Day 1993). The device independent portion of the operating system communicates with a disk device driver using a well defined protocol specified by Novell while the disk device driver communicates with the controller card using a controller specific set of commands.

To collect disk trace data at the disk level, a custom piece of software is placed between the Netware operating system and the disk device driver. To accomplish this, we modified existing disk device drivers. This NLM records incoming messages and passes them to the intended recipients. All disk device drivers communicate with the Netware operating system using a well defined set of interface routines. The three routines of most interest are GetRequest(), PutRequest(), and AddDiskDevice(). GetRequest is used to inform the disk drive controller that a disk read or write is requested. PutRequest is used to indicate that the requested transaction has been completed. The AddDiskDevice routine is used during system initialization to add disk drives to the system. We use this routine to map Netware drive numbers to a controller and SCSI ID pair.

Netware device drivers are dynamically loaded. This implies that the routines mentioned above have labels or tags in the binary image of the device driver that the loader uses at load time to update the operating system's calling tables. We modified the existing disk device drivers by editing the binary module and replacing the labels for GetRequest, PutRequest, and AddDiskDevice with the labels BYU_GetReq, BYU_PutReq, and BYU_AdDiskDev.

After this modification, the system is no longer functional. The operating system attempts to call GetRequest, PutRequest, and AddDiskDevice, but these routines no longer exist in the disk device driver. The next section describes our NLM that solves this problem.

2.1.2- Trace Collection NLM

The trace collection module consists of an NLM that lies between the operating system and the modified device driver described in the previous section; see Figure 1. The light gray arrows and interface routines represent the original operating system interfaces. The rest of the figure represents the new flow of information.

Our NLM, referred to as byu_disk.nlm, contains three routines, GetRequest, PutRequest, and AddDiskDevice. When these routines are called by the operating system, our NLM extracts the information we desire to store as trace data and then forwards the

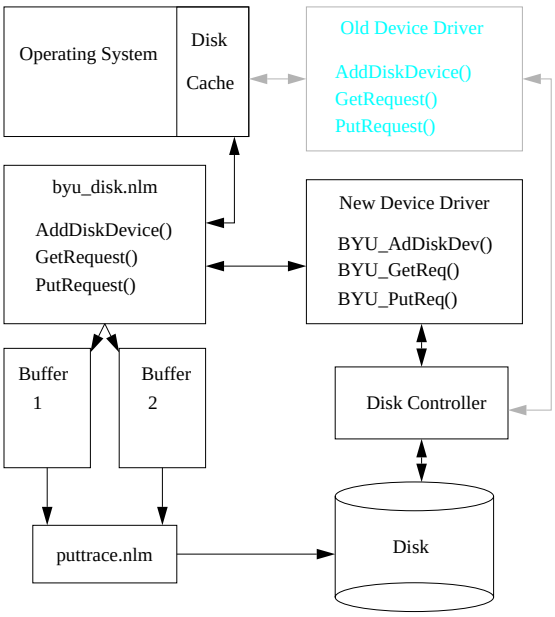

Figure 1: Block Diagram of our Disk Trace Collection Tool. The Light Gray Elements Represent the Original Flow of Information.

request to the modified disk device driver by calling BYU_GetReq, BYU_PutReq, or BYU_AdDiskDev. This results in normal server behavior.

Each request is time stamped using a precision timer built into the server's CPU. The resolution of the timer used in this work is accurate to one microsecond and is stored as a 32 bit value which results in an overflow whenever events are more than 1.2 hours apart; this is certainly adequate.

The captured disk request information is saved in an internal buffer that is globally available. When this buffer fills, a flag is set and storage shifts to a second buffer. This process continues between the two buffers until the modified disk device driver and our NLM are removed from the system under test.

2.1.3 Trace Data Extraction

We mentioned in the previous section that the trace results were temporarily stored in one of two buffers allocated in the byu_disk NLM as global variables. When either of these buffers fills, an associated global flag is asserted.

We have created a second NLM called *puttrace* that has access to both the buffers and the associated flags in the byu_disk NLM. Puttrace checks the status of the flags associated with each buffer four times each second. If a flag is set, puttrace reads the associated buffer contents, writes the trace data to disk, and resets the flag. If a flag is not asserted, puttrace goes to sleep. This process continues until puttrace is removed from the system.

2.2 Impact of Trace Collection on Server Performance

This section describes and quantifies the impact the trace collection process has on system performance. When one of the two byu_disk buffers fills it is written to disk. The request to write this buffer to disk results in two additional 4KByte disk requests that are recorded by our monitoring routines. It should be clear that if our buffers are too small, a significant amount of perturbation will result. In the standard configuration, our tool adds two requests for every thousand collected.

Performance degradation is caused by both the byu_disk and puttrace NLMs. When the byu_disk NLM is loaded into the Netware server it formats and places trace records into the internal buffers. This process increases the time required to perform disk accesses. In order to see how much our tool affected system performance, we compared the average run times from five runs of the Ziff-Davis Winstone 95 benchmark suite for three different cases. The first case tested the average time needed to run the suite without any of our instrumentation present. The average execution time was 3385 seconds. The second case investigated the time required to perform the same tasks with the byu_disk NLM installed, but without puttrace saving the collected data to disk. This instrumentation increased the average execution time by one second.

With the byu_disk NLM installed and saving trace records to its internal buffers, puttrace was installed to save the collected trace data to disk. This instrumentation consumed no measurable additional time. Overall, the trace collection tool slows the system down by one second out of 3385 seconds or 0.03%. This perturbation is extremely minor and indicates that the trace data collected using this system is accurate and useful for trace-driven simulation.

2.3 Server and Workload Characteristics

This section describes the systems where our trace collection tool was installed and the trace data that was collected and used for this work.

Trace data was collected from two servers used in our department. The first system (CSL) is used to serve hundreds of undergraduate students performing programming assignments for various classes. The second server (PEL) is a research server located in the Performance Evaluation Laboratory in our department.

2.3.1- Computer Science Laboratories Server

The CSL server is an Intel Pentium based system containing 80 Mbytes of memory, eight ethernet connections, and 20 Gbytes of disk storage. The system runs version 4.1 of the Novell Netware operating system with a 250 user license. Approximately 100 client PCs are connected to the server. The server contains five Seagate ST15150W drives. Each of these disks holds approximately four gigabytes of data and has a spindle speed of 7,200 RPM. The average access time for a read or write is 8.0 or 9.0 milliseconds respectively. Disk 0 contains the operating system and applications; disk 4 mirrors it. Disks 1, 2, and 3 are striped together to form a 12 gigabyte virtual disk for user space. Any trace data collected from this server includes references from all five disks, after the references have passed through a 60 megabyte Netware disk cache.

This server is mainly used by undergraduate students in lower division computer science courses. The most widely used applications are Microsoft Windows, various programming tools, user written applications to fulfill assignments, applications for reading and sending email, and WWW browsers. As previously mentioned, these applications are stored on disk 0 and mirrored on disk 4.

2.3.2- Performance- Evaluation- Laboratory Server

Traces were also collected from our experimental server in the Performance Evaluation Laboratory. This server uses an Intel Pentium processor, has 2 gigabytes of disk space, 32 megabytes of RAM, and a single network connection. It also runs Novell Netware 4.1, and has a 5 user license. This server supports research faculty and graduate students performing experiments using the Ziff-Davis and BAPCO benchmarks, programming Netware Loadable Modules, and preparing documents using Microsoft Windows based applications.

2.4- Collected Trace Data

The byu_disk NLM stores twelve bytes of data for each request generated by the system. This data is organized into a structure consisting of six elements:

1. request type or operation, one byte
2. disk controller number, one byte
3. SCSI ID of disk, one byte
4. request length in sectors, one byte
5. sector or block number, four bytes
6. time stamp in microseconds since last request, four bytes

Possible request types include read, write, and done. A done request indicates that a previous read or write request has been completed. The disk controller number and SCSI ID aid in identifying which disk responded to a request. All read requests are used to fill a disk cache line and are eight sectors long. Write requests vary in length from one to eight sectors. The sector or block number identifies the logical disk block or sector where the request begins. The time stamp is the time in microseconds since the last read, write, or done request. By matching a done record with its corresponding read or write, it's possible to determine how long it took to service the request. These records, stored in sequential order, make up a trace.

While many traces have been collected from both the CSL and PEL servers, two are used for the analysis presented in this paper. The first trace contains the disk requests made by the PEL server in response to a single client executing the Ziff-Davis Winstone 95 benchmark suite. This trace, referred to as PEL-ZD, has the characteristics shown in Table 1.

The second trace (CSL-29) was collected on the CSL server over a 29 day period. This trace is representative of the type of activity observed in our instructional laboratories. The trace was begun in the middle of October and ended near the middle of November. The characteristics of this trace can also be seen in Table 1.

The trace collection tools and collected trace data are available to interested parties. The availability of these tools should increase the quantity and quality of available trace data.

3- SIMULATION MODEL

This section describes our simulation methodology. We first describe our simulation tool and present an example using the trace data described in the previous section. We then evaluate the accuracy of this model and present several uses for this type of tool.

3.1- Simulation Tool

The goal of our simulator is to return the estimated service time for each request. To accomplish this, we create tables with a row for each service time and a column for each seek distance. Each entry of the table contains the probability of the associated seek distance and service time acquired from the trace data for a given system. Each column of this table is a probability distribution of service times for a particular seek distance. This table can be thought of as a collection of curves with axes of service time and probability and indexed by seek distance.

Table 1: Characteristics of Disk Traces Selected For This Research.

Name	Requests	Reads	Writes	Sectors Read	Sectors Written	Length
PEL-ZD	117,759	34,095	83,664	272,760	547,629	0.94 hours
CSL-29	23,650,978	6,678,672	16,972,306	53,429,376	91,181,192	704.19 hours

To create a simulation model for a given I/O subsystem, the surfaces for that system must be created. The first component of our simulation tool reads trace data from the system of interest and processes it to obtain the appropriate surfaces. There are surfaces associated with each request length, read and write requests, numerous queue lengths, and various disks in the system. For example, the CSL server has five disk drives, eight possible write request lengths, eight possible read request lengths, and fifteen considered queue lengths requiring 1200 surfaces to obtain a complete simulation model.

The second component of our simulation tool reads the surfaces describing a particular system as well as input trace data. The requests found in the input trace data are used to compute a seek distance, request type, request length, and current queue length. These values are used to access the correct surface and retrieve the probability distribution for the associated seek distance which is used to estimate the service time. This process is repeated until the input trace data is exhausted.

For example, assume that a request is made for block 1000 and that the previous request was for block 2000. This results in a distance of -1000 blocks. This distance is used to index into the table previously described and return the associated probability distribution of service times. Assume that at this distance, the service times are uniformly distributed between 4 and 21 milliseconds. A random number with this distribution is generated and returned as the estimated service time. This process is repeated for each request.

The shortcoming of previous approaches is that they accurately model the disk mechanism, but provide little information about other parts of the disk I/O subsystem. Other researchers have attempted to model the rest of the I/O subsystem (Ruemmler and Wilkes 1994), but accurate models of disk device drivers, system buses, disk controllers, and I/O buses are difficult to create or obtain. Our trace data contains timing information contributed by each of these components which results in a simulation model which accurately represents the entire I/O subsystem.

3.2- CSL Server Simulator

This section describes the process of creating a simulator for the CSL server. The accuracy of this simulator is then evaluated and quantified.

3.2.1- Creating the CSL Server Model

This section describes the process of generating the seek distance versus service time tables used to simulate the CSL server. In addition, a representative probability surface is presented.

The first step in generating a simulation model is to collect a disk trace containing block requests and associated service times. For this example, we use the CSL-29 trace described in the previous section. We use the first seven days of this trace to construct a simulation model.

The first component of our tool is used to create the tables necessary for simulation from the first seven days of the CSL-29 trace. Its output is a file that contains 1200 tables in a format readable by the second process. These tables are used to estimate the service time of each request during simulation.

3.2.2- CSL Simulator Evaluation

In this section, we evaluate the simulation model by comparing the predicted service times to those actually produced by the system under test. We perform this evaluation by comparing the probability density function of the original CSL-29 trace with the estimated density function. To accomplish this we performed three tasks:

1. We built a simulation model using the first seven days of the CSL-29 trace.
2. We drove this model with the requests from the complete CSL-29 trace and obtained an estimate for the service time of each request.
3. We compared the estimated service time density function with the service time density function obtained from the original trace.

To compare the density functions, we use a mean square error technique proposed in (Ruemmler and Wilkes 1994). We compute the root mean square of the horizontal distance between the two density functions. This absolute error indicates the service time

difference between the real and simulated systems. We use the absolute error figure and the total of all service times in the original trace data to compute a percentage error.

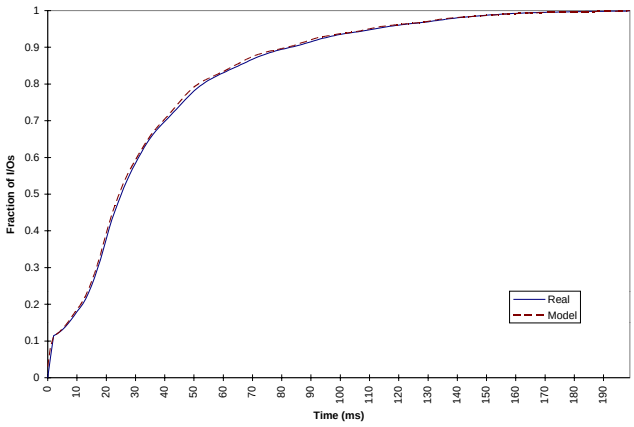

Figure 2: Service Time Density Functions for the Detailed Simulation Model and the Original CSL-29 Trace Data. The Absolute Mean Square Error is 0.125 Milliseconds and the Percentage Error is 0.35%.

Figure 2 presents the estimated and original service time density functions. As can be seen in the figure, the curves are extremely similar and difficult to differentiate in most locations. The absolute and percentage mean square error between the original and estimated density functions are 0.125 milliseconds and 0.35% respectively. The same process was repeated for the PEL-ZD data and the absolute and percentage mean square errors are 0.200 milliseconds and 0.31%. These figures illustrate the accuracy of these models. The next section describes our use of this model to improve I/O performance by reorganizing disk data.

4- DISK DATA REORGANIZATION

In this section we describe our simulation experiments used to determine the impact disk data rearrangement has on disk I/O service times in server environments.

4.1- Reorganizing Disk Data

There are two ways to determine how reorganizing disk data will impact disk I/O service times. The first approach involves physically moving the data from one disk sector to another. This approach requires accurate low-level tools to manipulate disk data and has the potential of causing permanent damage. Once the data is moved, a benchmark program can be run to evaluate the new disk organization as compared to the original disk layout. The second approach involves simulating the entire system and moving the disk data on the simulated disk drive. This approach is safer, but requires the creation of accurate simulation models. We chose the second approach due to its safety and the ease of creating an accurate simulation model using our approach.

With our approach, there is no simulated disk drive, and therefore no disk data to move, but we can simulate the movement of data by remapping the input trace data used to drive the simulation model. For example, if we desire to swap sector 5000 with sector 2 we simply change all references to sector 5000 in the trace to requests for sector 2 and all references for sector 2 into requests for sector 5000. If this swapping causes the seek distance to increase it is likely that the simulation model will return a larger estimated service time.

4.2- Reorganizing the CSL Servers Drives

We used the data movement technique described above and our simulation model of the CSL server to determine the impact of several reorganization techniques on disk I/O service times. Due to space limitations, we only present the results obtained using our *hot block* approach. The other approaches we investigated were less successful.

We define a *hot block* to be a group of eight sectors that are heavily used. The hottest block on the disk is the group of eight sectors that has the most requests. Our approach is simple: move the hottest N blocks to the outside edge of the disk starting at sector zero and ending at sector $N-1$ in order from hottest to coolest.

We measured the total service time in hours for both the original and reorganized disk. On the reorganized disk, 1000 of the hottest blocks were moved to the outside edge of the disk. We observed that this approach reduces service time on each of the days, but the improvement is quite small. The average improvement over the entire 29 day period is 7.84%. Through analysis of the trace data we determine that our approach reduced the total seek distance by 31% and increased the number of requests that did not require any head movement by 14%. With these figures one would expect a larger reduction in overall service time, but the limiting factor for increasing performance on this server turns out to be queue length. In other words, on this server a large portion of a disk request's service time is time waiting in the queue. If the disk seek time is only a small fraction of the overall waiting time, improving this figure by reorganizing disk data will not yield a significantly smaller

average service time.

Although a 7.84% decrease in average service time will not significantly increase overall system performance, this study does demonstrate the usefulness of our simulation technique. It is not possible while using this technique to determine what percentage of the service time is contributed by each component of the I/O subsystem; but this detail is not necessary for this sort of study.

5- CONCLUSIONS AND FUTURE WORK

This work has presented techniques for developing simulation models, collecting accurate trace data to aid in conducting I/O subsystem studies designed to enhance performance, and a simple case study demonstrating the usefulness of our technique. This section summarizes our work and discusses future directions.

Our methodology for generating system simulators has several advantages over traditional approaches:

1. A system simulator is automatically generated by tools using representative trace data as input.
2. No input parameters describing the system to be modeled are required other than input trace data.
3. Simulation consists of repeated table lookup operations, resulting in fast execution time.
4. Simulators can be quickly constructed for any part of the system for which trace data can be collected.

On the other hand, a potential disadvantage is that the performance contributions of components in a complex system are hard to distinguish. For example, the simulator described in Section 3 accurately models the device driver, system bus, controller, I/O bus, and disk mechanism, but it is nearly impossible to determine what fraction of a particular service time result came from which component in this hierarchy. For some studies, this is a serious drawback; for others, it is not important.

The point of this paper is to demonstrate that this methodology results in accurate simulation results applicable to a useful set of problems like the one described in the previous section. We plan to investigate several areas of research that can benefit from this type of simulation model:

- prefetching of I/O data
- hierarchical file storage on both server and client file systems; optimizing clients and server as a single unit
- organization of disk controller caches
- usefulness of victim caches on disk drive controllers
- quantifying the performance enhancement or degradation due to RAID systems

Our simulation methodology is well suited to these kinds of studies because we are not interested in which components of the I/O hierarchy contribute to the service time, but only what the service time is for a particular request. Each study may require trace data collected at a different level in the hierarchy; but, with the availability of trace data, the methodology is the same. Other simulation techniques could be used to perform these studies, but they would require much more effort for an equivalent level of accuracy. In addition, other techniques are likely to require significantly more time to obtain results, due to the complexity of the simulation code.

In summary, we have described a new technique to model disk I/O subsystems using trace-driven simulation that is applicable to any environment where accurate trace data containing timing information is available. In addition, a tool to collect accurate disk I/O traces from Novell Netware servers has been presented and used to construct a model of an example system.

Finally, we have demonstrated that this methodology results in highly accurate disk I/O subsystem simulators and that these simulators are useful for conducting performance evaluation studies. The collected trace data, trace collection tools, and simulation tools discussed in this paper are available to interested parties.

REFERENCES

Day, M. 1993. *Netware for NLM Programming.* Novell Press.

Flanagan, J. K., B. E. Nelson, J. K. Archibald, and K. Grimsrud. 1993. Incomplete trace data and trace driven simulation. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems MASCOTS.* 203–209. SCS.

Floyd, R., and C. Ellis.- 1989.- Directory reference patterns in hierarchical file systems. *IEEE Transactions on Knowledge and Data Engineering.* 1(2):238–247.

Grimsrud, K., J. K. Archibald, and B. Nelson. 1993. Multiple prefetch adaptive disk caching. *IEEE Transactions on Knowledge and Data Engineering.* 5(1):88-103.

Ousterhout, J., H. D. Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. 1985. A trace

driven analysis of the UNIX 4.2 BSD file system. In *Proceedings of the 10th ACM Symposium on Operating System Principles*. 15–24.

Ramakrishnan, K., P. Biswas, and R. Karedla. 1992. Analysis of file I/O traces in commercial computing environments. In *Proceedings of 1992 ACM Sigmetrics and PERFORMANCE92 International Conference on Measurement and Modeling of Computer Systems*. 78-90. ACM.

Ruemmler, C., and J. Wilkes. 1993. UNIX disk access patterns. In *USENIX Winter 1993 Technical Conference Proceedings*. 405–420.

Ruemmler, C., and J. Wilkes. 1994. An introduction to disk drive modeling. *IEEE Computer*. 27(3):17–28.

Smith, A. J. 1985. Disk cache-miss ratio analysis and design considerations. *ACM Transactions on Computer Systems*. 3(3):161–203.

Worthington, B., G. Ganger, Y. Patt, and J. Wilkes. 1995. On-line extraction of SCSI disk drive parameters. In *Proceedings of 1995 ACM Sigmetrics*. 146–156. ACM.

ACKNOWLEDGMENT

This research was partially supported by a grant from Intel Corporation entitled "Increasing Disk I/O Performance in a Novell Netware Environment".

AUTHOR BIOGRAPHIES

NIKI C. THORNOCK is working on her M.S. degree at Brigham Young University. She received her B.S. degree from Brigham Young University in August, 1995. Her academic interests include computer architecture and performance evaluation studies.

XIAO-HONG TU completed her M.S. degree at Brigham Young University in August, 1997. Her main interests are computer architecture and performance evaluation of I/O subsystems. She is currently working in the LANDest Management Suite department of Intel Corporation in Utah.

J. KELLY FLANAGAN is Director of the Performance Evaluation Laboratory at Brigham Young University where he is an Assistant Professor of Computer Science. His research interests include computer system performance evaluation and computer architecture. He received his PhD in Electrical Engineering from Brigham Young University. Before joining the faculty of the Computer Science Department at BYU, he spent a year at Intel Corporation in Oregon and taught a graduate architecture course for Oregon State University. He is a member of the Association for Computing Machinery and the IEEE.