A Stochastic Disk I/O Simulation Technique*

Niki Crockett, Xiao-Hong Tu, J. Kelly Flanagan, and Frank Sorenson
Performance Evaluation Lab
Computer Science Department
Brigham Young University
Provo, Utah 84602

April 12, 1996

Abstract

Processor speeds continue to increase while disk I/O speeds lag far behind. This mismatch causes an I/O bottleneck reducing the performance of systems running I/O intensive applications. There is a need to model these types of systems to determine and quantify the performance impact of alternative designs.

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. This simulator is evaluated and found to be an accurate predictor of disk service times.

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. A significant amount of research has taken place to increase the performance of memory hierarchies while far less effort has been made to model, simulate, and study I/O subsystems.

Previous studies to increase disk I/O performance have been performed using trace-driven simulation [1, 2, 3, 4, 5, 6]. This type of modeling requires an accurate simulation model and trace data representative of the system being evaluated [7]. Simulation models are typically implemented

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

using high level language descriptions of various subsystem components. Each component is described in detail and is tuned using input parameters acquired from physical devices [8]. These parameters may be obtained through direct measurement [9] or from data sheets provided by device manufacturers [8]. 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 equivalent of an operating system buffer cache and does not contain high resolution timing information [1, 2, 3, 4, 5, 6]. 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 [10] implemented and described a disk trace gathering technique that acquired trace data after operating system caches and buffers. In addition, 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.

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 [11]. This paper describes a new technique for creating disk I/O subsystem simulation models for use with trace-driven simulation. In addition, a tool to collect accurate disk I/O traces from Novell Netware servers is presented and used to construct a model of the system under test. This example is used to demonstrate the feasibility of our approach. We believe this is an important area of study because PCs have proliferated tremendously throughout the world, and a large number of them are running Novell

Netware. To our knowledge, no one has addressed the topic of disk I/O performance in a Novell Netware environment.

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

2 Simulation Methodology

Trace driven simulation is a common modeling technique used to evaluate the performance of proposed system configurations. To acquire accurate results using this methodology, two items are required: an accurate simulation model of the system being investigated and representative trace data. This section discusses techniques for creating accurate disk I/O simulation models.

2.1 Previous Simulation Models and Parameter Generation

One obvious technique for generating accurate simulation models is to write a detailed description of the system in a high level language. For a disk I/O subsystem model, this would require a detailed description of several key components, including device drivers, system buses, disk controllers, I/O buses, and disk mechanisms. An example of this type of simulator is presented in [8] 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. In other words, if great care is taken to understand the target system and to obtain accurate component parameters, this technique will result in very accurate simulation models.

A detailed 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 [8], uses the disk parameters published by the disk drive manufacturer and regression analysis against their trace data. The second technique, proposed in [9], uses special tools and test vectors to drive SCSI disks and extract the desired drive parameters. Both of these techniques result in accurate parameters that can be used to construct simulation models.

We have developed a new approach for extracting disk drive parameters. Our technique requires a high resolution timer and the ability to access the disk drive directly. Our technique is quite simple. The disk drive under test receives requests to read data from a logical block a random distance from the current location. The distance may be either positive or negative and is always a power of two. It is important to model both negative and positive seek distances because the performance in each direction may not be the same. The constraint of making each random distance a power of two reduces the number of required random events without significantly decreasing the accuracy of the extracted results. Before a request is made, a value is read from a high resolution timer. When the request is complete, the timer is read again and a service time is computed. This time includes three components: disk seek time, rotational latency, and data transfer time. This process is repeated until a generous number of requests have been generated for each possible seek distance.

The data resulting from this process is a list of seek distances measured in logical blocks. We measure seek distance in logical blocks because operating systems make requests to disk I/O subsystems in terms of logical blocks; we are interested in the system's response to logical block requests. When this data is plotted on a three dimensional graph with axes of seek distance, service time, and probability, Figure 1 results.

In the left figure, it can be seen that the top of the surface is relatively flat and that the width of the surface on the service time axis is equal to the rotational period of the drive under test. This is true because a seek to a particular track has an equal probability of finding the requested sector immediately or having to wait for one or any fraction of a rotational period. For example, the surface extends from approximately 10 to 26 milliseconds on the service time axis. This time,

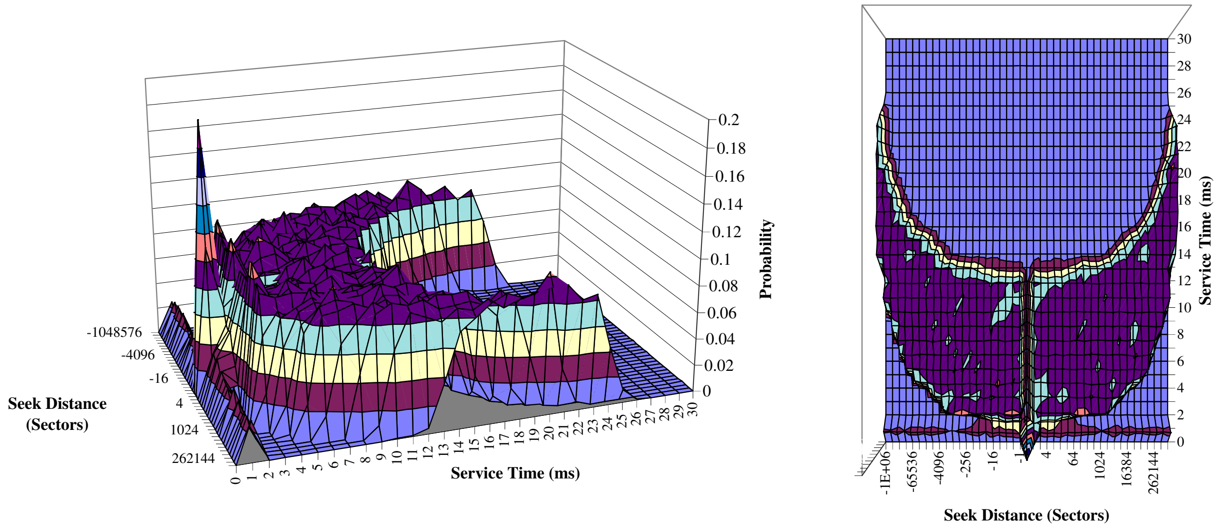

Figure 1: Seek distance versus service time for a 1 GByte Hewlett-Packard SCSI disk drive. The left figure illustrates the performance of the integrated cache and the rotation period. The seek profile is the lower edge of the curved surface In the right figure. The peak at seek distance zero extends to 0.91, but is truncated for clarity.

approximately 16 milliseconds, is the rotation period of the drive under test. Also note that along the seek distance axis, there is some probability of finding the requested sector in the disk cache or buffer which results in a short service time. The highest peak in the figure is at a seek distance of zero. If you request the same sector twice there is a very high probability (0.91) that the request will be serviced within a couple of milliseconds.

Of most importance is the lower curve of the surface shown in the right figure. This curve represents the seek time of the drive for various seek distances with no rotational delays. This type of relationship is a very important parameter in most disk drive simulators. These two curves were created using a read request of a single sector. This yields little insight into the response of the drive to larger requests. This limitation can be overcome by choosing random request lengths and constructing multiple surfaces or a single four dimensional graph.

Notice that this type of analysis yields the parameters necessary for traditional simulation models: rotation period, seek profile, and disk transfer rates. The next section describes a technique that uses this information for simulation directly rather than simply providing the parameters for other tools.

2.2 Stochastic Simulator

Our disk drive simulation model is extremely simple and requires no input parameters. The goal of our simulator is to return the estimated service time for each request. This is accomplished by first calculating the distance from the previous request to the current request in terms of logical blocks. This distance is used to retrieve the probability distribution for this seek distance from the surface shown in Figure 1. Finally, the simulator returns an estimated service time that will result in a distribution of service times equal to that of the retrieved distribution.

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 surface described in the previous section along the seek distance axis 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 between 4 and 21 milliseconds and returned as the estimated service time. This process is repeated for each request.

The shortcoming of this and previous approaches is that they accurately model the disk mechanism, but provide little information about other parts of the disk I/O subsystem. Certainly other researchers have attempted to model the rest of the I/O subsystem [8], but accurate models of disk device drivers, system buses, disk controllers, and I/O buses are difficult to create or obtain. Our technique can be extended to model these components. In the previous section, the surface described was constructed by driving the hard disk mechanism with requests for random seek distances. This was done to obtain a large number of events at all possible distances with a minimum number of requests.

If real (non-random) disk access patterns were used to drive the disk mechanism, a much larger number of requests would be required to obtain a large number of samples at each possible seek distance. If these requests and timing information were monitored at the disk level, the surface previously described would eventually result. However, if these requests and timing were monitored just after the disk controller, but before the SCSI I/O bus, the resulting surface would contain service times with time contributions made by the disk drive and bus. This surface could be used to model these two components. This process can be used at any level in the hierarchy.

This simulation methodology is useful for simply and accurately predicting the cost of events in certain portions of the I/O hierarchy. It relieves the researcher of the burden of creating detailed models and requires no input parameters. However, this technique requires input data containing I/O requests and associated service times. This information can be obtained from trace data collected at the appropriate point in the I/O hierarchy. Ruemmler and Wilkes [10] discuss a technique to collect this type of data from Hewlett-Packard hp-ux based workstations. Their data is collected after the operating system buffer cache. The next section describes a tool we've constructed for collecting similar data from a Novell Netware based file server.

3 Disk Trace Collection Tool and Collected Trace Data

A problem that frequently confronts researchers is a lack of disk trace data. 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, 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 [deleted for review].

3.1 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.

3.1.1 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 [11]. 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. The NLM records incoming messages and passes them to the intended recipients. To accomplish this, we modified existing disk device drivers. 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 NNN_GetReq, NNN_PutReq, and NNN_AdDiskDev. It is important to note that since we are editing a binary file that will eventually be executed, we don't want to disturb any relative offsets associated with branch instructions. We avoid this problem by replacing the routine names with new names of equal length.

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.

3.1.2 Trace Collection NLM

The trace collection module consists of a memory resident program, known as a Netware Loadable Module (NLM), that lies between the operating system and the modified device driver described in the previous section; see Figure 2. The light gray arrows and interface routines represent the original operating system interfaces. The rest of the figure represents the new flow of information.

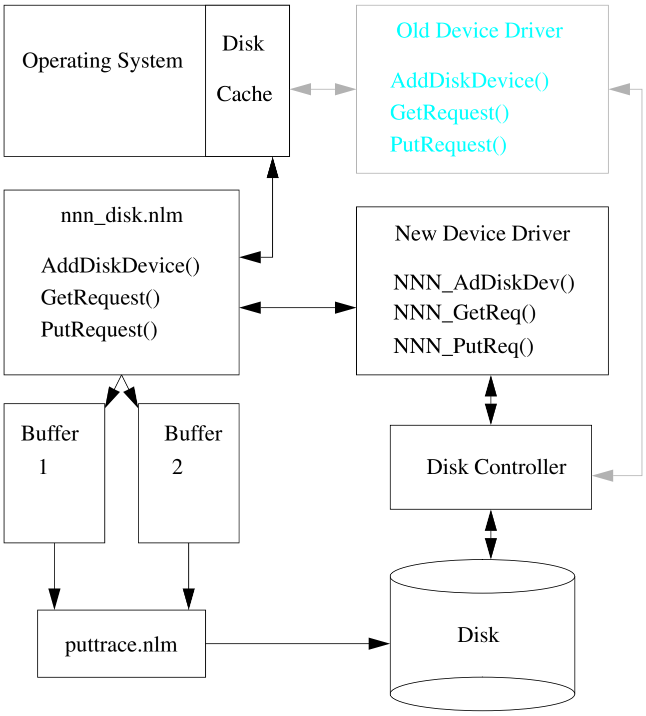

Figure 2: Block diagram of our disk trace collection tool. The light gray elements represent the original flow of information.

Our NLM, referred to as nnn_disk.nlm, contains three routines, GetRequest, PutRequest, and AddDiskDevice, which the operating system expects to find in a standard disk device driver module. 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 request to the modified disk device driver by calling NNN_GetReq, NNN_PutReq, or NNN_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 a single CPU cycle: approximately 11 nanoseconds on a 90 MHz system. The time stamped on each request is a 32 bit value that represents the number of time units since the last event. To extend the time storable as a 32 bit value, we reduce the resolution from CPU cycles to microseconds 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.

3.1.3 Trace Data Extraction

We mentioned in the previous section that the trace results were temporarily stored in one of two buffers allocated in the nnn_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 nnn_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. It is important to note that puttrace can be loaded and unloaded without rebooting the server.

3.2 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 nnn_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 nnn_disk and puttrace NLMs. When the nnn_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. The bars in Figure 3 illustrate what little effect our tool has on system performance. Each bar represents the average run time from five runs of the Ziff-Davis Winstone 95 benchmark suite. The leftmost bar in this figure is the average time needed to run the suite without any of our instrumentation present in the Netware kernel. The center bar represents the time required to perform the same tasks with the nnn_disk NLM installed, but without puttrace saving the collected data to disk. As can be seen in the figure, this instrumentation increases the execution time by one second.

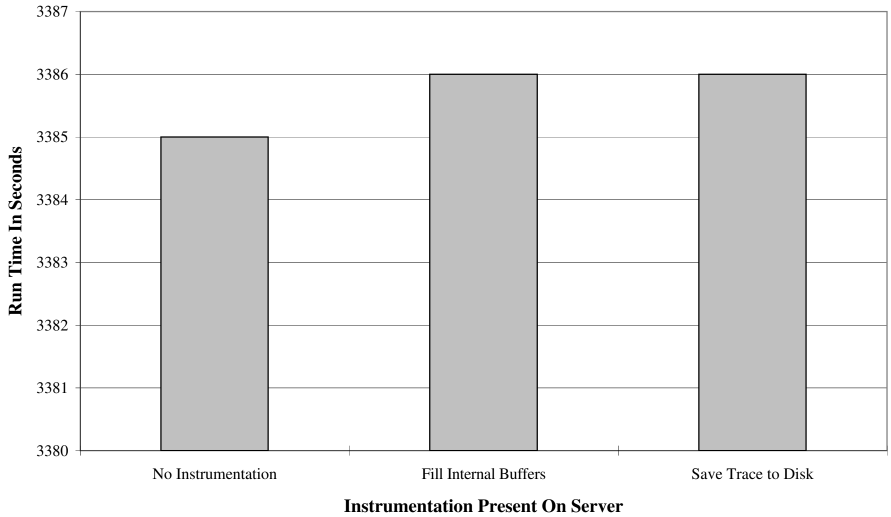

Figure 3: Average run time of the Ziff-Davis Winstone 95 benchmark suite without any server modifications (left), with nnn_disk.nlm added (center), and with both nnn_disk.nlm and puttrace.nlm added (right).

With the nnn_disk NLM installed and saving trace records to its internal buffers, puttrace is installed to save the collected trace data to disk. This instrumentation consumes no measurable additional time, as can be seen from the rightmost bar of Figure 3. 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.

3.3 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.

3.3.1 Server Configurations

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.

Computer Science Laboratory (CSL) 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.

Performance Evaluation Laboratory (PEL) 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.

3.4 Collected Trace Data

The nnn_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.

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

Table 1: Characteristics of disk traces selected for this research.

While many traces have been collected from both the CSL and PEL servers, two are used for the analysis presented in this paper. Other traces have similar characteristics to those presented here. 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. This is in the middle of our academic semester and therefore does not contain inactive startup periods or final project congestion. The characteristics of this trace can also be seen in Table 1.

We intend to collect more traces in the future from experimental, instructional, and administrative servers on and off campus. 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.

4 Simulation Model

This section describes an implementation of the simulation methodology described in Section 2. 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.

4.1 Simulation Tool

This section describes our simulation tool which consists of two components. The first component reads trace data and generates the probability surfaces described in Section 2.2. The second component reads probability surfaces and disk requests and returns estimated service times.

To create a simulation model for a given I/O subsystem, the probability 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.

The procedure described in Section 2.2 is a simplistic description of this process. Instead of a single three dimensional surface with axes of seek distance, service time, and probability, accurate simulation models consist of numerous two-dimensional surfaces or tables. For example, there are tables associated with each request length, read and write requests, and various disks in the system. To illustrate this situation, consider a system similar to the CSL server described in the previous section. This server has five disk drives and would require 80 tables to obtain a complete simulation model. There are eight tables for each request length between one and eight sectors for each of the five disk drives; these are duplicated for both read and write requests.

The second component of our simulation tool reads the tables 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 and request type. These values are used to access the correct table which returns an estimated service time. This process is repeated until the input trace data is exhausted.

4.2 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.

4.2.1 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.

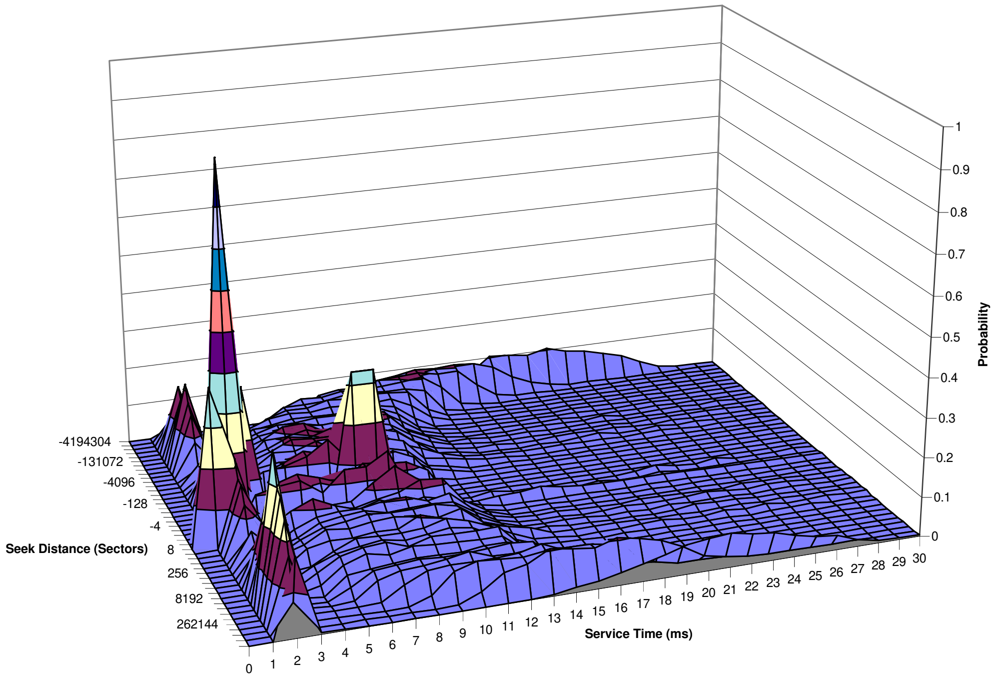

Figure 4: Seek distance versus service time for read requests to drive 0 obtained from the first seven days of the CSL-29 trace. This is a graphical representation of a table used by our simulator

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 80 tables in a format readable by the second process. By plotting the seek distance versus service time for read requests to disk 0 during the first seven days of the CSL-29 trace, we obtain a surface that represents a single table of this model. This surface is illustrated in Figure 4.

The axes on this figure are the seek distances measured in logical blocks and the service times measured in milliseconds. Note that the service time axis extends from zero to 30 milliseconds in the figure, but the real tables extend to 200 milliseconds. The upper limit is the smallest value that encompasses nearly all disk activity, and was determined by evaluating the collected data.

4.2.2 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. In other words, we built a simulation model using the first seven days of the CSL-29 trace and used that to estimate the service times for all requests made over the 29 day period. We compare the estimated service time distribution with the service time distribution obtained from the original trace.

To compare the density functions, we use a mean square error technique proposed in [8]. 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.

Our model is capable of eight levels of detail. The simplest model makes decisions based on seek distance only while the most detailed model takes into account the request type, length, and destination disk. It is possible to add each of these decision making criteria individually or together. Due to lack of space, we only present the results for the most detailed model.

Figure 5 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 possible uses for this type of simulation model.

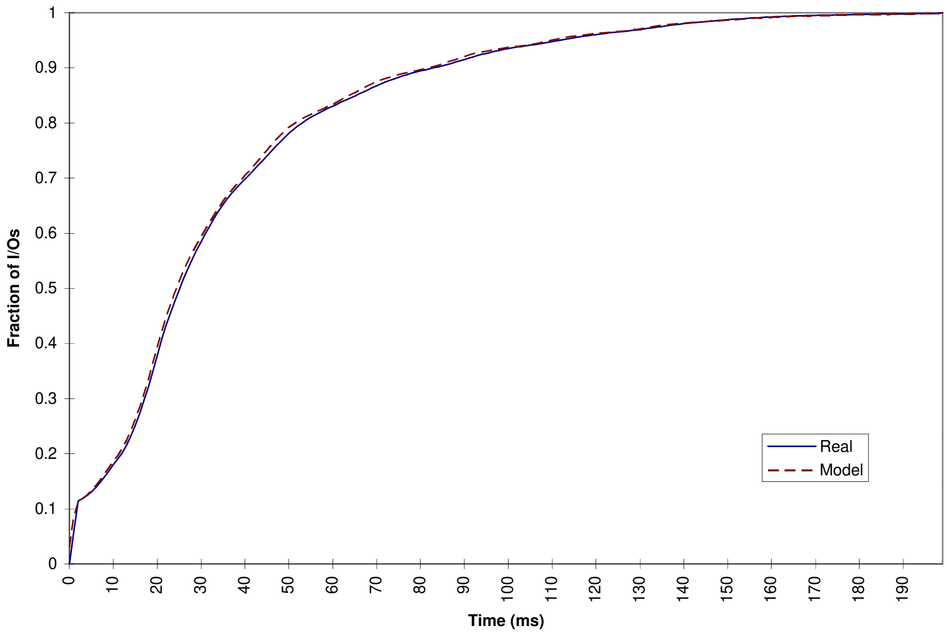

Figure 5: Service time density functions for the detailed simulation model and the original CSL-29 trace data. The absolute mean square error is 0.125 milliseconds and the percentage error is 0.35%.

5 Future Research and Conclusions

Disk I/O speeds have lagged far behind processor speeds for a number of years. The resulting mismatch causes an I/O bottleneck which reduces the performance of systems running I/O intensive applications. This work has presented techniques for developing simulation models and collecting accurate trace data to aid in conducting I/O subsystem studies designed to enhance performance. This section describes some of the advantages and disadvantages of this technique, discusses how this type of model is being used in current research projects, and draws conclusions.

5.1 Future Work

The proposed technique 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. The only constraint on the input trace data is that it must exercise all parts of the system that will be used during future simulation runs to achieve accurate results.
2. No input parameters describing the system to be modeled are required.

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 the previous section 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. We are currently involved with several projects that use this type of simulation model:

- disk data reorganization
- 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.

5.2 Conclusions

We have described a new technique to model disk I/O subsystems using trace-driven simulation. 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

[1] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson, A trace driven analysis of the UNIX 4.2 BSD file system, In *Proc. of the 10th ACM Symposium on Operating System Principles*, pages 15–24, December 1985.

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

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

[4] C. Staelin and H. Garcia-Molina, Smart filesystems, In *USENIX Winter 1991 Technical Conference Proceedings*, pages 45–51, 1991.

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

[6] K. Grimsrud, J. Archibald, and B. Nelson, Multiple prefetch adaptive disk caching, *IEEE Transactions on Knowledge and Data Engineering*, February 1993.

[7] J. Kelly Flanagan, Brent E. Nelson, James K Archibald, and Knut Grimsrud, Incomplete trace data and trace driven simulation, In *Proc. of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems MASCOTS*, pages 203–209. SCS, 1993.

[8] Chris Ruemmler and John Wilkes, An introduction to disk drive modeling, *IEEE Computer*, pages 17–28, March 1994.

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

[10] Chris Ruemmler and John Wilkes, UNIX disk access patterns, In *USENIX Winter 1993 Technical Conference Proceedings*, pages 405–420, 1993.

[11] Michael Day, *Netware for NLM Programming*, Novell Press, 1993.