A System-Assisted Disk I/O Simulation Technique

Franklin E. Sorenson *sorenson@byu.edu*
Elizabeth S. Sorenson *leb@pel.cs.byu.edu*
J. Kelly Flanagan *kelly@cs.byu.edu*
Heng Zhou *heng@pel.cs.byu.edu*
Performance Evaluation Laboratory
Brigham Young University
Provo, UT 84604

Abstract

Simulation and modeling are useful tools in prototyping advances in many areas of technology. The performance of modern processors advances much faster than that of storage components making disk systems prime candidates for simulation. Simulating disk systems allows researchers to determine and quantify the performance impact of alternative file system designs and disk data rearrangement techniques. This paper describes a System Assisted disk I/O Simulation Technique (SAST). This technique uses a real system to estimate disk request service times, relieving the researcher from the burden of modeling the target system. The system accepts disk trace data as input, and accurately predicts disk service times. In addition, we describe a disk trace collection technique capable of collecting accurate trace data from the Windows 95 operating system.

1. Introduction

Processor speed has increased much faster than disk I/O performance over the past few years. VLSI performance has improved at a rate of 40-60% per year while disk drive performance has improved approximately 7% per year [1], and this trend continues. The increasing performance gap between processors and disk subsystems limits overall system performance, and has drawn more and more attention. A significant amount of research has been performed to improve disk performance, including new file systems [2, 3], more efficient disk buffer strategies [4, 5], I/O scheduling [6], and redundant arrays of inexpensive disks [7].

Simulation is commonly used to evaluate the performance of proposed system configurations. Previous studies to improve disk I/O performance have been performed using trace-driven simulation [8, 9, 10]. The accuracy of trace-driven simulation depends on the quality of the simulation model, and the quality of the input trace data [11].

In this paper, we propose a new, simple, and accurate simulation technique: a System-Assisted disk I/O Simulation Technique (SAST). In addition, we present a technique that collects accurate disk I/O traces that contain precise timing information from a Windows 95 system.

1.1. Previous disk subsystem simulation techniques

Previous research has generated three types of disk subsystem simulation models: abstract simulation models, detailed simulation models, and stochastic techniques.

Abstract simulation model. Abstract models were first used to answer simple performance questions [9, 10]. These simplistic models ignore the system details; they typically assume that every disk request requires a constant amount of time, ignoring the current disk head position and averaging the seek time and rotation delay. Slightly more elaborate models [10] acknowledge that disk I/O service times vary, and consist of seek time, rotational latency, and transfer time. The disk seek time is usually assumed to be linear with seek distance, without consideration for head acceleration and settling. Although simulation with abstract models is quite simple, these models often produce inaccurate results. These simple techniques are not sufficient for the study of subtle details in disk I/O subsystems.

Detailed simulation model. Detailed simulation models have been used to produce more accurate results. These techniques model and parameterize the components of disk I/O subsystems in detail, and then simulate their behavior with software written in high-level programming languages [12, 13, 14]. The accuracy of these models depends on included details and the validity of the input parameters describing each component. If the target system is well understood and accurate parameters are obtained, detailed simulation models can yield very accurate results. To model the disk I/O subsystem, several key components,

including device drivers, system buses, I/O buses, disk controllers, and disk mechanisms need to be analyzed. It is not trivial to model all the components and obtain their input parameters. In the simulator presented in [12], the disk drive modeling alone requires approximately 13,000 lines of commented C++ code.

Detailed simulation models require correct input parameters describing each component in the target system. Though disk drive technical reference manuals can often be obtained directly from the manufacturer, the manuals are usually incomplete, and their accuracy is questionable. The on-line extraction of SCSI disk drive parameters, proposed in [15], uses special tools and test vectors to drive SCSI disks and extracts the desired drive parameters. This technique requires a large amount of work to find the necessary details, and does not work for other I/O protocols such as IDE disks.

Stochastic simulation technique. The stochastic disk I/O simulation technique proposed in [16] models the target system by generating the system's service time distribution from long trace data. For each possible combination of disk seek distance, request length, and disk operation (read/write), the probability distribution of disk service times is obtained. Accordingly, the simulator estimates the service time for an I/O request, resulting in a distribution of service times equal to the retrieved distribution.

The stochastic simulation technique abstracts the system, relieving the researchers from the burden of obtaining specific input parameters, and creating detailed models. However, to model disk subsystems accurately, this technique requires trace data which exhausts all disk access patterns, and generates a disk service time probability distribution. Since the disk service time for an I/O request is randomly estimated according to the retrieved distribution, the simulation results are accurate for a large number of I/O requests, but not necessarily accurate for individual requests.

1.2. Previous workload generation techniques

In addition to simulation models, representative workloads are also critical to the accuracy of trace-driven simulation. Two common input workloads are synthetic workloads and traces.

Synthetic workloads. Some previous disk subsystem research was performed using synthetic workloads [9]. Synthetic workloads are generated on-the-fly, and consist of a series of I/O requests which may represent an average workload. Synthetic workloads are more flexible than traces, and do not require significant storage, however, they are specific to a single environment, and generally do not make representative disk references [19]. In addition, synthetic workloads are difficult to parameterize [13]. For example, an NFS-workload generator described in [17] uses 24 parameters to describe the workload.

Traces. Even though much can be learned from synthetic workloads, traces collected from various real-world systems often yield more accurate results. Most available trace data is collected at the file-system level, before the operating system disk cache, but does not contain high resolution timing information. This kind of trace data is useful for evaluating disk cache strategies, but is not accurate for investigating low-level disk subsystems.

The disk trace collection technique proposed in [1] collects the disk requests after the operating system disk cache, and contains the requests that are actually made to the hard disk. The service time contained in the trace data has the resolution of one microsecond, and is more accurate than most file-system level traces. This technique was implemented in release 8 of the HP-UX operating system. A similar technique was used to collect trace data from a Novell Netware based system [16].

2. Disk I/O trace collection technique

In this section, we describe a disk trace collection technique for the Windows 95 operating system. Our trace collection technique gathers disk requests after the operating system disk buffer and includes detailed timing information.

2.1. Disk trace collection tool

Our disk trace collection tool consists of two components. One component is a virtual device driver (VxD) which intercepts and stores all disk requests. The other component is a Windows application by which we can start and stop tracing and log the buffered disk requests to secondary storage. Figure 1 shows the structure of our disk trace collection tool.

2.1.1. Disk trace collection VxD. A layered virtual device driver model characterizes the disk subsystem in Windows 95. The Input/Output Supervisor (IOS) coordinates the layered device drivers and VxD's, allowing a hardware or software vendor to implement nonstandard, proprietary, or additional functionality. The IOS issues I/O requests and returns results, acting as an interface between clients such as file system drivers and the page swapper [18]. We used the functionality provided by this layered approach to create a VxD, allowing us to incorporate the trace collection function into the Windows 95 I/O subsystem.

Our trace collection VxD registers itself with the IOS, and inserts functions into the list of functions called when issuing I/O requests. As I/O requests pass through the function list, we store disk request information into a

buffer. When I/O requests complete, another function in the VxD stores the completion information.

As a result of these functions in the VxD, the information for each request is stored in the buffer when a request starts and when it completes. Each trace entry is time stamped to the nearest microsecond using a precise timer built into the CPU, which is accurate to a single CPU cycle. In our 450 MHz test system, this is 2.22 nanoseconds. The time stamp for each entry in the trace represents the elapsed time since the previous entry.

Two buffers are used to store disk requests. When one buffer is full, storage shifts to the other buffer, and the VxD informs the Windows application to retrieve the trace data, thereby emptying the first buffer.

2.1.2. Trace collection control application. The control application for our trace tool is a Windows 95 application that controls the trace collection VxD. It starts and stops tracing and empties buffered disk requests by writing them to secondary storage when the VxD informs the Windows application that its buffer is full. Windows 95 provides the communication mechanism between the VxD and the Windows application**.**

2.2. Trace data format

The recorded information for each disk request is organized into a six-element, twelve-byte structure:

1 Request type, one byte.
2 Major device number, one byte.
3 Minor device number, one byte.
4 Starting sector number, four bytes.
5 Request length in sectors, one byte.
6 Elapsed time 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 major and minor device number fields identify the device as a hard disk, and which disk is involved in the disk request. The sector number identifies the physical sector number where the request

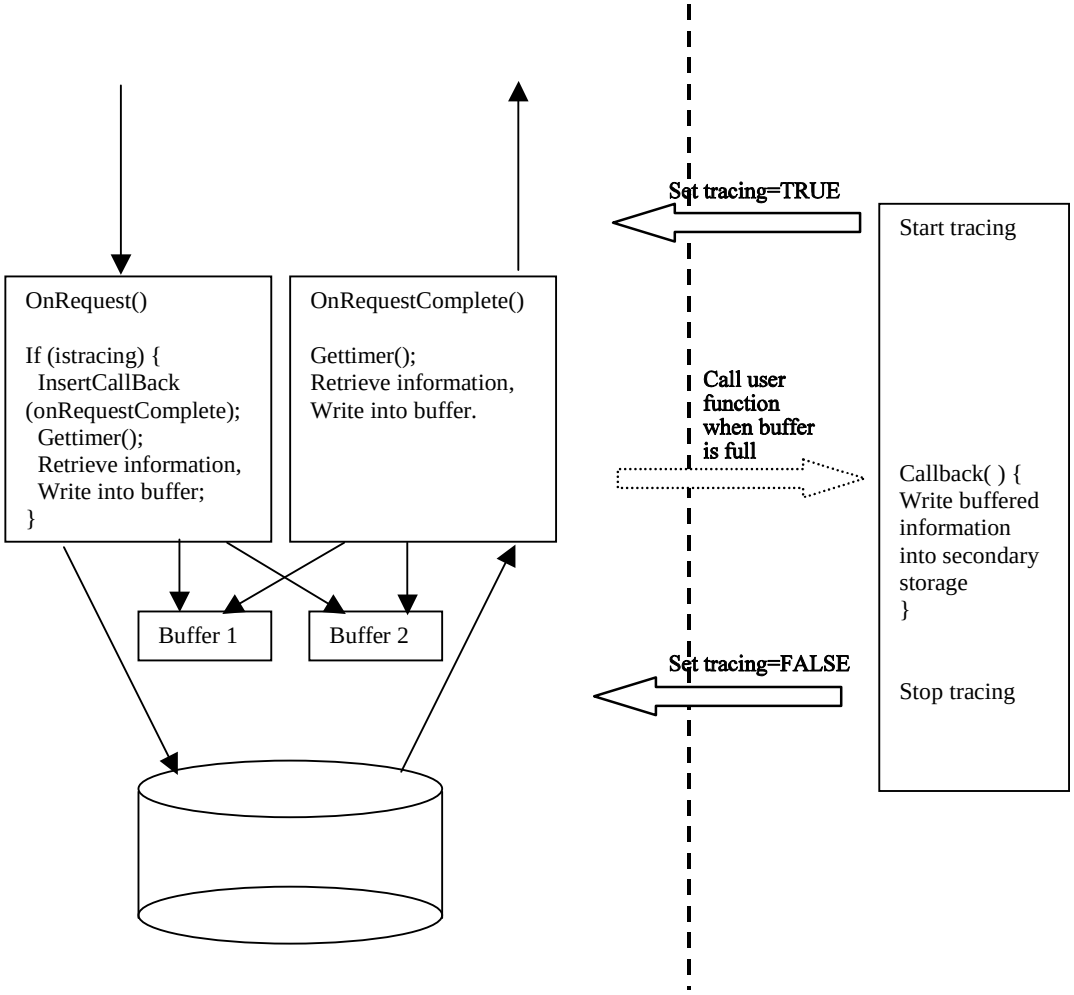


Figure 1. The structure of our disk trace collection tool.

begins. The time stamp is the elapsed time since the previous entry in the trace (in microseconds). By matching a done request with its corresponding read or write request, it is possible to determine how long it took to process the request. These records, stored in sequential order, make up a trace.

2.3. Evaluation of our disk trace collection tool

Traces gathered using our trace collection tool are complete and accurate; all the disk requests are collected, and every entry collected is valid. Precise time stamps are contained in our traces. This section describes and quantifies the impact of the trace collection process on system performance.

Performance degrades in two aspects. First, as previously described, the disk trace collection VxD introduces two functions. The two functions are inserted into the calldown function list and callback function list for disk requests respectively. Each disk request needs to go through these two functions to allow information gathering, increasing the service time. Secondly, when one of the two buffers in the disk trace collection VxD is full, an asynchronous procedure call is scheduled to have the Windows application store the buffered requests to disk. Additional disk requests are generated. In the standard configuration, our tool adds three I/O requests for every thousand normal requests.

The WinBench97 benchmark is a subsystem-level benchmark that measures the performance of a PC's major subsystems, such as graphics, disk, processor, video, and CD-ROM subsystems in a Windows95/NT based environment. The disk test suite of the WinBench97 benchmark reflects the performance of a PC's disk subsystem in Windows. The Business Disk WinMark97 test represents the disk activity of popular business applications.

Figure 2 illustrates the effect our tool has on system performance. Each bar represents the average I/O throughput from five runs of the Business Disk WinMark97 suite. The leftmost bar is the average I/O throughput while running the suite with both the trace collection VxD and the control application disabled. The center bar represents the average I/O throughput while running the suite with the VxD enabled, but with the application disabled. The rightmost bar is the average I/O throughput while running the suite with both the VxD and the application enabled. From Figure 2, we can see that the trace collection VxD alone slows the system down by 1.56%. The trace collection VxD together with the control

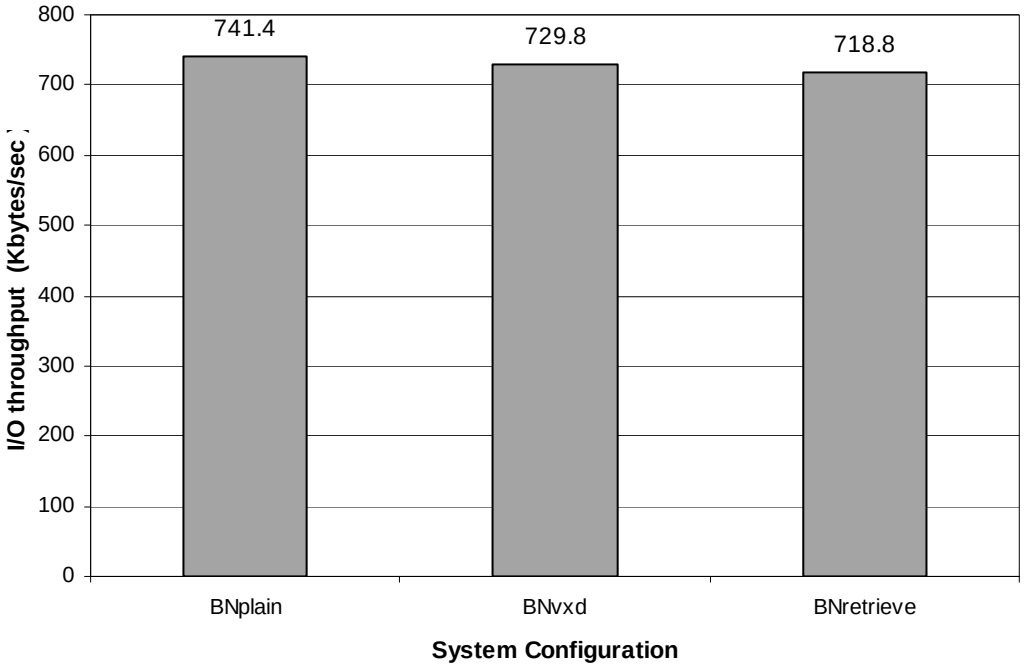


Figure 2. The I/O throughput of the business disk WinMark97 suite with both the disk trace collection VxD and the control application disabled (left), with only the VxD enabled (center), and with both the VxD and the application enabled (right).

application slows the system down by 3%.

In summary, our trace collection tool consists of a VxD and a control application. It acquires complete and accurate traces with precise time stamps, with a dilation of only 3%.

3. System-assisted disk simulation technique (SAST)

3.1. Overview

The basic concept behind SAST is to use an actual hard disk to return the service times of I/O requests. Disk I/O trace data is submitted as I/O requests to a dedicated physical disk on a real system. The returned service times are the output of SAST.

SAST is not intended for those wishing to investigate new disk drive technology that currently does not exist, but is intended for use by those evaluating system software such as file systems, disk data rearrangement techniques, etc.

SAST requires several things to return useful and accurate results. First, all disk requests are made to a dedicated secondary disk where no post operating system or user program requests will perturb the results. This also ensures that arbitrary write requests can be performed without fear of corrupting any file-system related information. Second, direct access to the disk drive is required. This allows SAST to ignore effects caused by the operating system, such as operating system disk cache or prefetch buffers. Third, SAST requires a high-resolution timer. The timer is read at the beginning of the request, and again at its completion. This allows the simulator to return precise results for service times and also gives the simulator an accurate timer to determine when to issue requests to the disk.

3.2. Methodology

In order to accurately model a disk I/O subsystem, we must take into account the inter-arrival rate of I/O requests. The inter-arrival rate may affect the behavior of the drive mechanism, and the time each request spends in the I/O queue. Since traces reflect the characteristics of the system from which they are collected, we must adapt trace data to reflect the characteristics of the system of interest.

User applications can be thought of as consisting of two segments of time: disk I/O time and CPU processing time preparing for more I/O. To allow for differences in the performance of various disk mechanisms, as well as to accurately model the CPU time required for processing, our simulator issues I/O requests asynchronously. We also chose to hold constant the time attributed to the CPU, removing differences in processing power from the model. This is not an issue in this work since the entire system remains constant other than the disk mechanism.

The behavior of the I/O queue is directly simulated by issuing requests when the number of I/O requests completed matches the number completed at the same point in the input trace. SAST must determine when to issue each Read or Write request. To determine how long to wait before issuing a Read or Write request, SAST considers whether the previous trace entry was a request (Read or Write) or a report of completion (Done).

If the previous trace entry was a request, the elapsed time in the trace represents CPU processing time. The simulator models the behavior of the CPU by waiting the elapsed time from the issue of the previous request to the next request, then issuing the request. If the previous entry was a Done, the elapsed time includes time waiting for disk I/O as well as CPU processing time. This behavior is modeled by waiting until the length of the I/O queue matches the length indicated in the trace, then waiting until the required time has passed since the previous request.

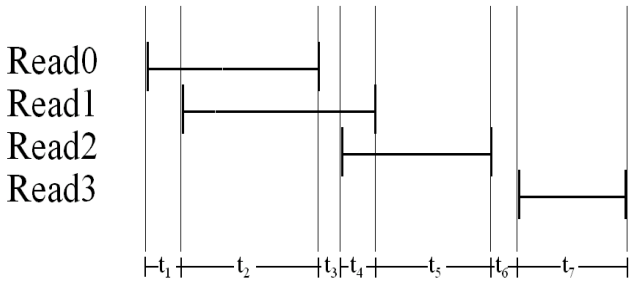
For example, consider the disk requests and trace data shown in Figure 3. SAST begins by issuing the first request, Read0. Since the next request, Read1, is issued before Read0 is completed, t_1 represents CPU processing time. Therefore, SAST waits time t_1 before issuing Read1. The next request, Read2, is issued after Read0 is complete, but before Read1 is complete. In this case, t_2 represents time spent waiting for disk I/O, and t_3 represents CPU processing time. So SAST first waits until the length of the disk queue equals 1, because the length of the queue in the trace equaled 1 just prior to Read2. SAST next waits time t_3, and then issues Read2. The final request in this example, Read3, is similar to Read2. SAST first waits until the disk queue length equals 0, then waits time t_6, and then issues Read3.

This method attempts to simulate the behavior of the I/O queue. The length of the queue characterizes the interaction of workloads and I/O mechanisms better than the inter-arrival times alone, since the processor depends on data received in previous requests, and issues the next requests based on what information it computes. Also, since this method is independent of disk speed, we can apply this method to various disks.

3.3. SAST for Windows 95

To demonstrate the feasibility and efficiency of SAST, we implemented and evaluated SAST using a Windows 95 system. We used a Pentium II 450, giving a timer resolution of 2.22 nanoseconds. The simulation system we developed for Windows 95 relies on two components.

The first component is a virtual device driver (VxD).



TRACE ENTRIES

Read0	t_0	← queue = 1
Read1	t_1	← queue = 2
Done0	t_2	← queue = 1
Read2	t_3	← queue = 2
Done1	t_4	← queue = 1

Figure 3. Sample disk requests and associated trace.

This VxD is a file system driver (FSD), which has direct access to the I/O Subsystem. The second component is a Win32 application which drives the VxD to simulate I/O requests acquired from trace data. This application uses traces collected using the disk trace collection tool previously described.

3.4. Evaluation of SAST

This section will demonstrate the accuracy of SAST by comparing the service time distribution from the input trace data with the service time distribution output by SAST. We will first describe our evaluation process in general and then give a specific example.

First, we collect trace data from disk **A** running our workload, resulting in trace T_A. We then simulate disk **A** using SAST with input trace T_A. Clearly the service time distribution of T_A and the service time distribution output by SAST should be identical.

Then, using a second drive **B**, we collect a new trace T_B from the same workload used to create T_A. Though T_A and T_B contain the same requests, they have vastly different service and inter-arrival times. We simulate drive **A** using SAST with input trace T_B. Again, the service time distribution of T_A and the service time distribution output by SAST should be the same because SAST should remove the mechanism specific features such as service times from T_B. This demonstrates that any trace T_X of a workload acquired on disk **X** can be used as input to SAST to simulate a disk **Y**'s response to the workload that produced T_X.

3.4.1. Workload selection. Evaluation of SAST requires a workload which generates a repeatable disk request sequence. When applications load from disk, they tend to request nearly the same disk blocks in nearly the same order. Since application startup provides an almost deterministic disk request sequence, we chose to launch a series of eight applications, with a set time between each application start.

We used Microsoft Visual C++ 6.0, Microsoft Word 97, WordPerfect 8, Microsoft Excel 97, Quattro Pro 8, Internet Explorer 4, Netscape Communicator, and Corel Presentations 8. After launching each application, our evaluation program slept for long enough to allow the application to load, then proceeded with the next application. This process generated about 4700 disk requests, in a simple repeatable series.

When selecting traces for input to SAST, we attempted to acquire an average trace of our workload. We define average as the trace with the total service time closest to the mean of the total service times of all collected traces. To determine how many traces were required to achieve confidence that the mean total service time was accurate to within 0.01 ms, we used the following formula:

$$n=\left(\frac{100ts}{r\bar{x}}\right)^2,$$

where n is the number of traces required, t is the Student's t distribution value, s is the standard deviation of the total service times of the traces, r is the desired accuracy, and $\bar{x}$ is the mean of the total service times of the traces. For disk **A**, we used an 8 Gigabit Western Digital drive. Using this method, we found that we required 8 traces collected from disk **A** to achieve a 90% confidence that the mean of the total service times was 23.48 seconds. We chose the trace with the total service time closest to the mean of the total service times and refer to it as T_A.

3.4.2. Two specific examples. Using T_A as input to SAST, we again used the above equation to determine the number of simulations we needed to achieve a 90% confidence that the mean of the total service time output by SAST was within a 0.01 ms accuracy. Only two simulations were

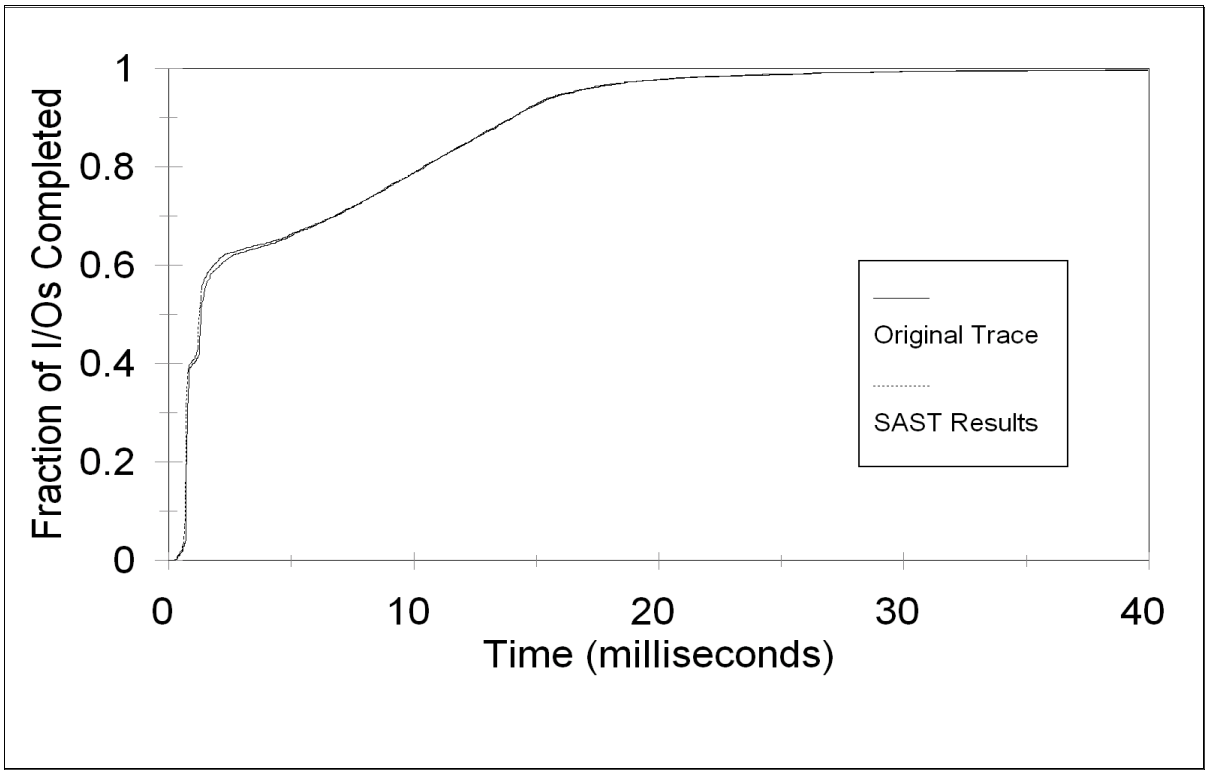


Figure 4. The service time distribution of T_A with the service time distribution of SAST simulating disk A using T_A as its input.

necessary to provide this accuracy. Figure 4 shows the service time distribution of T_A and the output of SAST with T_A as input; the curves are nearly identical.

To compare the two distributions, we use the root mean square error technique proposed in [12]. This technique computes the root mean square of the horizontal distance between the two curves. We used this absolute error, along with the average service time in the original trace to compute a percentage error.

We found that the absolute root mean square error between the two distributions is 0.178 ms, and the average service time of the original trace is 5.03 ms, resulting in an error of 3.54%. In addition, the 90% confidence interval of the average service time from trace T_A is 5.03 ms ± 0.168 ms, and the simulation output yielded a confidence interval of 4.93 ms ± 0.161 ms. Notice that each 90% confidence interval includes the mean of the other interval. Therefore the two means are not statistically different from each other at the 90% confidence level. From this, we can see that there is not a significant difference between the original disk drive and the simulated disk mechanism.

For the second part of the SAST validation, we picked a 2 GB Western Digital drive as our drive **B**. This time, we took 16 traces to satisfy the previously cited statistical equation. The mean total service time for disk **B** was 50.52 seconds. Again we chose the trace, T_B, with the total service time closest to the mean of the total service times. Using T_B as input, we simulated disk **A** using SAST. Figure 5 shows the service time distribution of T_A and the output of SAST driven with T_B. Again, notice how close the two distributions are. The absolute root mean square error between the two distributions is 0.411 ms, or a percentage error of 8.17%. This shows that SAST can closely predict disk I/O performance of an input trace collected from other disk subsystems.

SAST can also be used to accurately simulate a set of disk drives using a single input trace. This can be useful for evaluating the performance of various disks for a particular workload of interest. Given a benchmark trace, several drives can be simulated to determine which performs best on the given workload.

4. Conclusions and future work

4.1. Conclusion

We designed a simple and accurate system to simulate disk activity with the aid of a real-world system. In addition, we developed a disk trace collection tool to collect disk traces in a Windows 95 environment. The

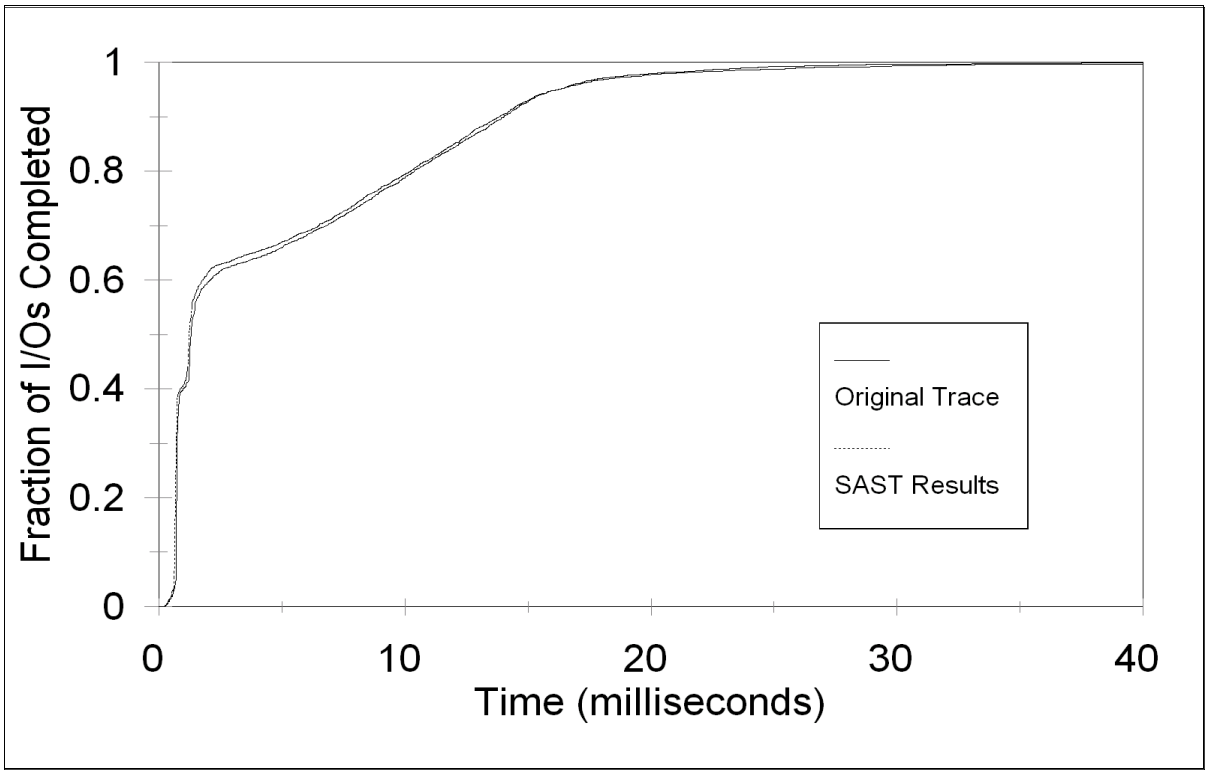


Figure 5. The service time distribution of T_A with the service time distribution of SAST simulating disk A using T_B as input.

collected trace data includes precise timing information, which we used as input to our simulator. The high quality of our trace data and simulation model led to accurate simulation results.

SAST provides a fast way to prototype and evaluate changes to operating system cache or prefetch algorithms, file system changes such as block size, or future performance evaluation techniques, such as reorganization techniques or dedicating part of the hard disk as a cache for frequently accessed blocks.

4.2. Advantages of SAST

The system-assisted disk simulation technique has several advantages over previous approaches:

1. SAST is easy to implement. It eliminates the need to create a detailed model of the disk subsystem and obtain input parameters. Since the real-world system is available, we can use a real disk to provide actual service times.

2. The simulation results using SAST are accurate within acceptable ranges for both large and small numbers of requests.

3. SAST is easily portable to different systems and disk subsystems as long as we can gain direct access to disks. SAST can also be used to model more complex systems, such as RAID.

4. Using SAST, it is possible to do research in a completely file-system independent manner. This allows researchers to prototype changes to file-systems or research performance optimizing techniques without writing file-system tools for each environment.

A potential disadvantage of SAST is that the service time contributions of individual disk subsystem components are hard to distinguish. The estimated service time includes device driver time, bus transfer time, disk controller overhead, and disk access time, so it is difficult to determine what fraction of time each portion takes. For some studies, this is insignificant, but in others, it is a drawback.

Another disadvantage of SAST is that the simulator can not be used in the study of hypothetical disk mechanisms. If no real-world system exists to assist the simulation, detailed simulation models must be used. However, though SAST can't model hypothetical disk mechanisms, SAST can be used in conjunction with a simulation model in a hybrid approach. This approach could be used in evaluating RAID systems, where SAST provides the service time distribution for the disks, and a detailed simulation model models the remaining portions of the system.

4.3. Future work

We plan to implement and evaluate SAST with other operating systems besides Windows 95. Our studies of the Windows NT and Linux systems suggest that it would be easy to implement SAST in these systems.

Acknowledgments

We would like to thank Knut Grimsrud of Intel Corporation for his valuable insights and ideas which motivated this work.

References

[1] Chris Ruemmler and John Wilkes, "UNIX disk access patterns", *USENIX Winter 1993 Technical Conference Proceedings*, 1993, pp. 405-420.

[2] Mendel Rosenblum and John K. Ousterhout, "The design and implementation of a log-structured file system", *Proceedings of 13th ACM Symposium on Operating Systems Principles*, Association for Computing Machinery SIGOPS, October 1991, pp. 1-15.

[3] P. Cao, E. W. Felten, A. Karlin, and K. Li, "Implementation and performance of integrated application-controlled file caching, prefetching, and disk scheduling", *ACM Transactions on Computer System*, 14(4):311-343, November 1996.

[4] Tracy Kimbrel, Andrew Tomkins, R. Hugo Patterson, Brian Bershad, Pei Cao, Edward Felten, Garth Gibson, Anna R. Karlin, and Kai Li, "A trace-driven comparison of algorithms for parallel prefetching and caching", *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation*, pp. 19-34, USENIX Association, October 1996.

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

[6] Bruce L. Worthington, Gregory R. Ganger, and Yale N. Patt, "Scheduling algorithms for modern disk drives", *Proceedings of the Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 241-251, ACM Press, May 1994.

[7] David Patterson, Garth Gibson, and Randy Katz, "A case for redundant arrays of inexpensive disks (RAID)", *ACM SIGMOD 88*, pp. 109-116, June 1988.

[8] 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", *Proceedings of the 10th ACM Symposium on Operating System Principles*, pp. 15-24, December 1985.

[9] P.M. Chen and D.A. Patterson, "Maximizing performance in a striped disk array", *Proceedings of the 17th Annual Symposium on Computer Architecture*, pp.322-331, May 1990.

[10] Richard R. Muntz and John C.S. Lui, "Performance analysis of disk arrays under failure", *Proceedings of the 16th Conference on Very Large Databases*, pp. 162-173, 1990.

[11] J. Kelly Flanagan, Brent E. Nelson, James K. Archibald, and Knut S. Grimsrud, "Incomplete Trace Data and Trace Driven Simulation", *Proceedings Of the International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* MASCOTS, pp. 203-209, SCS, 1993.

[12] Chris Ruemmler and John Wilkes, "An Introduction to Disk Drive Modeling", *IEEE Computer*, pp. 17-28, March 1994.

[13] C. A. Thekkath, J. Wilkes, and E. D. Lazowska, "Techniques for File System Simulation", *Technique reports HPL-92-131 and 92-09-08*, Hewlett-Packard Laboratories, Palo Alto, Calif., and Dept of Computer Science and Eng., Univ. of Washington, Seattle, Washington, Oct. 1992.

[14] David Kotz, Song Bac Toh, and Sriram Radhakrishnan, "A Detailed Simulation Model of the HP 97560 Disk Drive", *Technique Report PCS-TR94-220*, Department of Computer Science, Dartmouth College, July 1994.

[15] B. Worthington, G. Ganger, Y. Pratt, and J. Wilkes, "On-line Extraction of SCSI Disk Drive Parameters", *Proceedings of 1995 ACM SIGMETRICS*, Pages 146-156, ACM, 1995.

[16] Niki C. Thornock, Xiao-Hong Tu, and J. Kelly Flanagan, "A Stochastic Disk I/O Simulation Technique", *Proceedings of the 1997 Winter Simulation Conference*, Pages 1079-1086, 1997.

[17] Roberta A. Bodnarchuk and Richard B. Bunt, "A Synthetic Workload Model for a Distributed System File Server", *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Pages 50-59, May 1991.

[18] Walter Oney, "Systems Programming for Windows 95", Microsoft Press, 1996.

[19] Raj Jain, "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling", John Wiley & Sons, Inc., 1991.