File Access Performance of Diskless Workstations

EDWARD D. LAZOWSKA and JOHN ZAHORJAN
University of Washington

DAVID R. CHERITON
Stanford University

and

WILLY ZWAENEPOEL
Rice University

This paper studies the performance of single-user workstations that access files remotely over a local area network. From the environmental, economic, and administrative points of view, workstations that are diskless or that have limited secondary storage are desirable at the present time. Even with changing technology, access to shared data will continue to be important. It is likely that some performance penalty must be paid for remote rather than local file access. Our objectives are to assess this penalty and to explore a number of design alternatives that can serve to minimize it.

Our approach is to use the results of measurement experiments to parameterize queuing network performance models. These models then are used to assess performance under load and to evaluate design alternatives. The major conclusions of our study are: (1) A system of diskless workstations with a shared file server can have satisfactory performance. By this, we mean performance comparable to that of a local disk in the lightly loaded case, and the ability to support substantial numbers of client workstations without significant degradation. As with any shared facility, good design is necessary to minimize queuing delays under high load. (2) The key to efficiency is protocols that allow volume transfers at every interface (e.g., between client and server, and between disk and memory at the server) and at every level (e.g., between client and server at the level of logical request/response and at the level of local area network packet size). However, the benefits of volume transfers are limited to moderate sizes (8–16 kbytes) by several factors. (3) From a performance point of view, augmenting the capabilities of the shared file server may be more cost effective than augmenting the capabilities of the client workstations. (4) Network contention should not be a performance problem for a 10-Mbit network and 100 active workstations in a software development environment.

This material is based upon work supported by the National Science Foundation under grants DCR 8352098, MCS 8302383, and MCS 8004111, and by the Defense Advanced Research Projects Agency under contract N00039-83-K-0431.

Authors' addresses: E. D. Lazowska and J. Zahorjan, Department of Computer Science FR-35, University of Washington, Seattle, WA 98195; D. R. Cheriton, Computer Science Department, Stanford University, Stanford, CA 94305; W. Zwaenepoel, Computer Science Department, Rice University, Houston, TX 77001.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*network operating systems*; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*; D.4.8 [**Operating Systems**]: Performance—*measurements*; *modeling and prediction*

General Terms: Design, Performance

Additional Key Words and Phrases: Computer system performance analysis, diskless workstations, distributed systems, file servers, local area networks

1. INTRODUCTION

This paper studies the performance of single-user workstations that access files remotely over a local area network. Three examples of such systems are Berkeley UNIX[1] hosted on SUN Microsystems' MC68000-based workstations, the V system (also implemented on SUN workstations) [1, 2], and the DOMAIN software for Apollo Computer's MC68000-based workstations [5].

Previous work on this subject has concentrated on measurements of elapsed time and service demands (client CPU, disk, server CPU, and network costs) for file access on an otherwise idle system [2, 5, 7, 9, 10, 13]. We also report the results of a number of measurement experiments. We have characterized the hardware and software environment by measuring elapsed times and service demands for local and remote file access on each of the three systems mentioned in the first paragraph, as well as the changes in these values that result when certain key parameters (e.g., the disk transfer block size) are varied. We have characterized the workload by measuring the rate at which "average" users and certain specific applications (e.g., compilers and linkers) generate file accesses, the amount of local processing that takes place per file access, and the distribution of file sizes weighted by frequency of access.

Although measurements such as these yield a number of insights, they have two serious limitations. First, from measurements of a single activity on an otherwise idle system, it is difficult to project how performance will degrade under load. (Of the papers noted above, only two [10, 13] attempt to assess performance under load, in the former case by measuring the elapsed time for certain operations with "<4 clients" and with ">8 clients," and in the latter case by having one, two, and three users exercise the system from scripts.) Second, from measurements of a specific design or configuration it is difficult to extrapolate the performance of a modified design or configuration. (Of the papers noted above, only one [7] attempts to assess the performance implications of design alternatives, by comparing the elapsed times of five specific operations on two specific transaction-oriented file servers.)

Our approach is to use the results of our measurement experiments to parameterize queuing network performance models. These models then are used to assess performance under load and to evaluate design alternatives. Our objective is not to build a "capacity planning model" of a specific system. Rather, our objective is to use queuing network performance models to enhance our under-

[1] UNIX is a trademark of Bell Laboratories.

standing of the various issues arising in the design of diskless workstations and their file servers. Among the factors that we investigate are the effect of local "user mode" processing on the perceived degradation resulting from diskless operation, the buffering of partial disk blocks at the file server, the disk transfer block size, the efficiency of the design and implementation of file access protocols, the local area network packet size, the location of disk blocks to minimize seek time, the prefetching of disk blocks, the use of caching at the file server and at the client, the use of limited local secondary file storage, the use of multiple file servers, and the use of a high-performance CPU at the file server. We study remote file access rather than remote paging, assuming either that workstations have a large amount of local primary memory, or that a local disk is used for this purpose. In general, we consider a program development environment characterized by many small files in a hierarchical directory structure.

Section 2 introduces the queuing network performance models that are the principal tools in the remainder of the paper, and discusses the measurements used in building these models. Section 3 illustrates the costs of local versus remote file access using the SUN/UNIX environment. The objective is to convey some basic intuition that is applicable to other systems as well. Section 4 considers design alternatives such as those enumerated above. Section 5 compares the remote file access performance of the SUN, Apollo, and V systems, explaining the differences in the light of earlier sections.

2. THE APPROACH

As noted, our approach is to leverage our measurement data by using them to build queuing network performance models. In this section we describe our models, their inputs, and the experiments conducted to obtain these inputs.

2.1 Queuing Network Models

Since the input requirements of our models dictate the quantities that must be measured, we begin with a description of these models. We have chosen queuing network performance models because they possess an attractive combination of efficiency and accuracy. In recent years they have become the tool of choice in a wide variety of computer system design and analysis applications.

There are three components to the specification of a queuing network model: the *service center description*, the *customer description*, and the *service demands*:

—The service center description identifies the resources of the system that will be represented in the model: disks, CPUs, communication networks, etc.

—The customer description indicates the workload intensity (or offered load): the average number of requests in the system, or the average rate at which requests arrive to the system, or the number of users and the average time that a user pauses between the receipt of a response and the initiation of another request.

—The service demands state the average amount of service that each request requires at each service center.

Once these inputs have been specified, the model can be *evaluated* using efficient numerical algorithms, yielding performance measures such as utiliza-

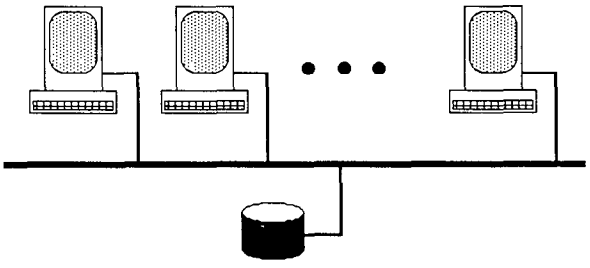

Figure 1

tions, residence times, queue lengths, and throughputs. In essence, the evaluation algorithm calculates the effect of the interference or queuing that results when customers having certain service demands share a system at a particular workload intensity.

Once created, the model can be used to project the performance of the system under various modifications. System modifications often have straightforward representations as modifications to the model inputs. For example, the substitution of a faster device can be represented by decreasing the service demand of customers at the corresponding service center. This fact, coupled with the efficiency with which queuing network models can be evaluated, makes them valuable performance projection tools.

For a comprehensive introduction to computer system analysis using queuing network models, see [4].

2.2 Models of Systems of Diskless Workstations

Figure 1 illustrates the "canonical system" studied in this paper. A number of diskless single-user workstations, the "clients," share a file server over a local area network. In the specific case of the SUN system considered in Section 3, the client workstations incorporate Multibus-based MC68010 processors and high-resolution bit-map displays, the file server is a workstation equipped with a Fujitsu Eagle Winchester disk, the network is a 10-Mbit Ethernet using 3Com programmed I/O interfaces, and all machines run the UNIX 4.2 BSD operating system.

Figure 2 illustrates the "canonical model" used in our study. The file server is represented by two service centers, corresponding to the CPU and the disk. The network is represented by a single *load-dependent* service center, since the efficiency of an Ethernet depends on the number of workstations simultaneously attempting to assert data. (The technique we use to represent the Ethernet is discussed in [4]; we have extended this technique to represent the variability in packet sizes encountered in a remote file access environment.) Each client workstation is represented by a single service center corresponding to its CPU.

The model includes one "token" or "customer" corresponding to each client workstation. Each customer cycles between its workstation and the file server via the network, accumulating service and encountering queuing delays due to competition from other customers (workstations). (Obviously, contention occurs only at the network and at the file server CPU and disk.) After some number of

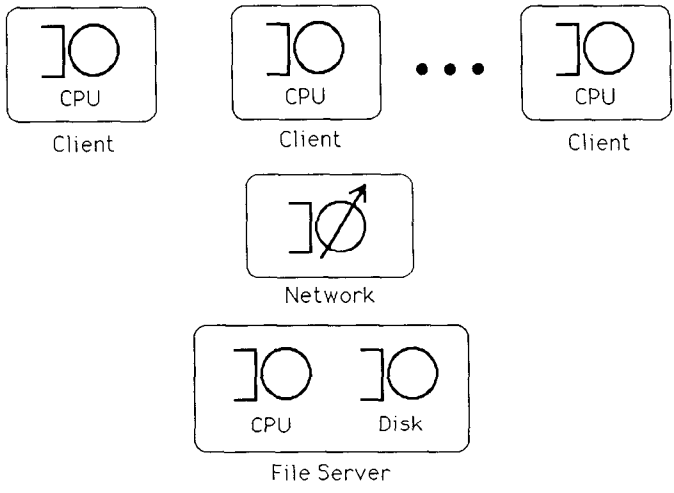

Figure 2

cycles, the customer may pause for a period of time before initiating another sequence of cycles.

Figure 2 provides the basis of the service center description and the customer description for our baseline model. (We have not yet specified the number of client workstations; this is the parameter to be varied in assessing performance under load. We also have not yet specified the "pause" characteristics of the customers; this is discussed shortly.) What remains is to specify the service demands at the various service centers. It is here that measurements of existing systems play a crucial role.

2.3 Customer Characterization

As noted, the service demands state the average amount of service that each request requires at each service center. We must begin by selecting a definition of "request." Suppose that "a compilation" were chosen as the definition of a request. Then, as service demands, we would provide (presumably from measurements) the total service required by a compilation at the client CPU, the network, the file server CPU, and the file server disk. The model would calculate response times and throughputs on a per-compilation basis.

For our study we have chosen to take a lower level view. A *request* is defined as the transfer of 4 kbytes of data, plus the average amount of "user mode" processing that occurs per 4K transferred. The service demand at the client CPU is the user mode processing, plus the overhead processing required to transfer 4K. The service demand at the network is the total transfer time of the multiple packets needed to carry out the transfer. The service demand at the file server CPU and disk are the times required at those devices to carry out the transfer. (The measurement of these service demands is discussed shortly.)

Clearly, no user computation (e.g., a compilation, or an editing session, or a file transfer, or the "average" behavior that characterizes a user during an extended period of activity) corresponds directly to the request just defined.

However, each user computation has a particular mix of file access and user mode processing. Thus, by suitably setting the parameter that describes the user mode processing occurring per 4K transferred, we can tailor our request to represent accurately a fixed portion of any computation. (The I/O-related portions of the service demands are intrinsic; their only dependence on the nature of the computation is through efficiencies that may arise in the case of sequential versus random access.)

As one example, we have measured compile/assemble/link sequences for several different compilers and several different source programs, running on otherwise idle diskless SUN/UNIX systems. A representative sequence transferred 2160 kbytes and consumed 156 seconds of user mode CPU processing. Thus, the user mode processing per 4K transferred was 156/(2160/4) = 289 milliseconds (ms). Given this information plus measurements of the intrinsic service demands at each device for transferring 4K, it is possible to parameterize the model. The response time calculated by this model will be "per cycle," where a cycle represents 1/540-th of the compile/assemble/link sequence (because 540 × 4K = 2160 kbytes are transferred).

As another example, we have monitored a number of highly interactive users engaged in software development on SUN/UNIX systems. On the average, these users consumed 106 ms of user mode processing per 4K transferred. In contrast to a compilation, a human does not have a "100 percent duty cycle." We account for this by using a pause in the model. The length of the pause is determined on the basis of further measurements showing that, on the average, our software developers transferred approximately 4K per second. We add the measured 106 ms of user mode computing that accompanies each 4K transferred to the measured elapsed time of such a transfer on an otherwise idle system and subtract this result from one second. The result is a value for the average pause that accompanies a cycle of file access and user mode processing. (Obviously the user actually completes a number of cycles of file access and user mode processing, then pauses for an extended period of time. These two points of view yield identical results from our models.) Given this information plus measurements of the intrinsic service demand at each device for transferring 4K, it is possible to parameterize the model. The response time calculated by this model will be "per cycle," where a cycle represents a "slice" of typical user activity. (Obviously, the response time reported by the model excludes the pause period.) As an absolute value it conveys little intuition, but as a relative value it is appropriate for comparing the effects of changes in workload intensity or system design.

These examples are meant to be illustrative. However, it also is the case that the second example—highly interactive users engaged in software development—will be the source of the customer characterization used in the bulk of our study. Thus, further discussion is warranted.

The objective of our work is to use modeling to investigate design alternatives, not to present a comprehensive measurement study of an existing system. Of course, measurements are necessary to parameterize our models. At the time our work was conducted, no suitable measurement study of UNIX systems had been reported. We undertook the minimal study that would meet our needs.

One element of our customer characterization is the user mode processing per 4K transferred: 106 ms. This number, of course, is dependent on the speed of the

user CPU and the nature of the user activity we were measuring. However, we show in Section 4 that most of the conclusions we reach are relatively insensitive to this number.

The other element of our customer characterization is the number of bytes transferred per unit of elapsed time: 4K per second. This number again is dependent on the nature of the user activity we were measuring. Although our results concerning the number of workstations that can be supported by a well-designed file server do depend on this number, we would claim that it is conservative. In an excellent measurement study of the UNIX 4.2 BSD file system, conducted on multiuser systems after our work first appeared in technical report form, Ousterhout et al. [8] found that when an "active" user was defined as one who had caused any file I/O in a 10-minute interval, then an average active user transferred 300–600 bytes per second, but that when an "active" user was defined as one who had caused any file I/O in a 10-second interval then an average active user transferred several thousand bytes per second. Our definition of "active" was more stringent but less formal: Software developers on workstations were told to "work hard" for 30-minute intervals, during which they were supervised and measured.

2.4 Measuring Service Demands

We see that the key parameters whose values are required are the service demands at the client CPU, the disk, the server CPU, and the network, for transferring 4 kbytes of data.

We begin by noting that although the definition of a "request" is a 4K transfer plus associated user mode processing, we are *not* assuming that file access is carried out 4K at a time. As an example, we measured the V system for file access (both client request and disk transfer) performed in units of both 1K and 8K. For the 1K case, service times *per access* are 3 ms at the client CPU, 26 ms at the disk, 17 ms at the server CPU, and 1 ms at the network. The service demands for our model are $4 \times 3 = 12$ ms at the client CPU, $4 \times 26 = 104$ ms at the disk, $4 \times 17 = 68$ ms at the server CPU, and $4 \times 1 = 4$ ms at the network. For the 8K case, service times *per access* are 20 ms at the client CPU, 30 ms at the disk, 42 ms at the server CPU, and 7 ms at the network. The service demands for our model are $20/2 = 10$ ms at the client CPU, $30/2 = 15$ ms at the disk, $42/2 = 21$ ms at the server CPU, and $7/2 = 3.5$ ms at the network.

Service demands were measured in a series of controlled experiments that transferred large numbers of blocks using the block size of the system under study. These experiments were repeated to ensure reliability. A key decision was whether to consider sequential or random access. The client and server CPU service demands are not dependent on the mode of access in any substantial way. The disk service demands are, however, because of additional seek and latency costs in the case of random access. (Elapsed times also are dependent on the mode of access, because of the dependency of disk service demands and because the server may prefetch in the sequential case. But elapsed time is an output of our models, not an input to them.) We have based our measurements on the case in which a seek is required, on average, once per two 4K transfers. (What this means is that, in the controlled experiments used to measure service demands, blocks were hand-placed on disk to achieve this behavior.) This seems at first

like a pessimistic assumption, given that the majority of client file accesses are "logically sequential." (Measurements by Ousterhout et al. [8] show that over 90 percent of all files are processed sequentially.) However, we believe that it is a reasonable starting point for our study, for three reasons. First, the fact that a client's accesses are logically sequential does not mean that they are physically sequential. Second, measurements to be discussed in Section 4 indicate that the average file size, weighted by frequency of reference, is only 11K—in other words, seeks will be frequent even assuming physical contiguity of logically sequential blocks. Third, when the file server is shared among a number of clients there will be head contention. In any event, the sensitivity of our results to this assumption is shown to be negligible.

We measured CPU consumption at the client and the server by using a background process on each machine that was continuously incrementing a counter; we compared the rate at which it progressed during a measurement interval to its rate when running alone.[2]

The measurements that we have described are similar to those reported in earlier studies of remote file access. They are interesting in their own right, and they allow us to construct baseline models of systems for which they can be obtained, which in turn allow us to investigate the effect of variations in workload intensity (in Section 3) and to compare the performance of specific systems (in Section 5). A key objective of this paper, however, is to investigate the effects on performance of various design alternatives. Many of these alternatives cannot be measured, and if we are to be able to determine their impact on the service demands accurately, then we need to carry our measurements of baseline systems one step further. We discuss this next.

2.5 Measurements to Support the Investigation of Design Alternatives

Examples of the system parameters that we wish to investigate are the client request size, the disk transfer block size, the use of prefetch, and the CPU cost of remote file access. In order to use our model, we must express the effect of these system parameters on the service demands. The model than can calculate the effect on performance measures, such as response times.

To do this with accuracy requires understanding not only the service demands of the baseline case, but also the contribution to these service demands of various lower level operations that take place. This is most easily understood by means of an example.

For V, we were able to measure the server CPU cost of responding to a single request to read 1K of data and to a single request to read 8K of data, where in each case the data were resident in memory so that no disk activity was required. The 1K case required a request message and a response message; the server CPU cost was 5 ms. The 8K case required a request message and eight response messages (owing to Ethernet packet size restrictions); the server CPU cost was 23 ms. By solving a set of two linear equations one can infer that the cost of

[2] We adopted this approach because the reporting of per-process CPU consumption by most operating systems is unreliable. We observed as much as a factor-of-5 discrepancy between reported values and values obtained using an auxiliary process. Among the contributing factors were erroneous attribution of interrupt-level processing and failure to capture processor degradation due to DMA memory interference.

sending a response message is 2.6 ms and that the cost of processing a request message is 2.4 ms. (This difference is inconsequential, but it is only through doing such things that one can know that they did not need to be done!)

We also were able to measure the server CPU cost of responding to a single request to read 8K of data when the data was disk-resident: 42 ms. Since the single disk access was the only difference between this case and the 8K case in the previous paragraph, the CPU cost of an 8K disk operation must be 19 ms.

We now are in a position to estimate accurately the server CPU costs for a protocol that was not measured. Suppose that clients request and receive data in units of 1K, but that disk transfers take place 8K at a time. (The 1K client interface reduces file access latency under light loads; the 8K disk transfer block size amortizes disk seek and latency costs across a number of client requests.) To transfer 1K using this protocol requires one request message, $\frac{1}{8}$th of an 8K disk access, and one response message, for a total of $2.4 + 19/8 + 2.6 = 7.375$ ms of file server CPU activity.

The performance projections undertaken in Section 4 take advantage of a large number of experiments such as this.

3. THE BASELINE CASE

In this section we use measurement data from a SUN/UNIX environment to illustrate the costs of local versus remote file access. This is merely a prelude to the study of design alternatives that we present in Section 4; our objective here is to introduce our approaches and techniques, and to convey some basic intuition about the performance of diskless workstations—intuition that is applicable to systems other than SUN/UNIX. Note that our measurements predate the recent introduction of SUN's Network File System (NFS).

3.1 Comparison of Service Demands

In Table I we compare the measured *service demands* for single 4K reads and writes, for the local and the remote case. We find that there are significant additional costs in the remote case.

File access in SUN/UNIX is accomplished 4 kbytes at a time: this is the unit in which the I/O library buffers data; it is also the disk blocking factor. Thus, a single 4K access is the natural basis for comparison. For the local case we consider the CPU and the disk; for the remote case we consider the client CPU, the disk, the server CPU, and the network. (The *total* and *elapsed* entries in the table are discussed in Section 3.2.)

The service demands, which are the principal inputs to the queuing network performance models used in later sections, tell us a considerable amount about the "cost of doing business" in the local and remote cases. We make the following observations:

—The disk service demand is, not surprisingly, the same in every case.
—In each remote case
  —the network service demand is negligible;
  —the CPU service demand at the client is more than double the CPU service demand in the local case;

Table I. Measured Service Demands (milliseconds) for Single 4K Transfers

	Local	Remote
Read		
Client CPU	14.6	43.8
Disk	18.8	18.8
Server CPU		41.0
Network		3.6
Total	33.3	107.1
Elapsed	28.6	69.4
Write		
Client CPU	20.0	47.5
Disk	18.8	18.8
Server CPU		36.8
Network		3.6
Total	38.8	106.6
Elapsed	23.8	47.0

—the CPU service demand at the server is significantly greater than the CPU service demand in the local case;
—the CPU service demand at the server dominates among the shared resources.

—Overall, the service demands for reads and writes are quite similar to one another, in both the local case and the remote case.

3.2 Total Service Demands versus Elapsed Times

Figure 3 compares the total service demands for local and remote reads and writes with the corresponding measured elapsed times. (Data come from the *total* and *elapsed* entries of Table I. Total service demand is the sum of the various components; elapsed time in the remote case was measured on an otherwise idle network.) We find that there is considerable concurrency, that is, that the elapsed times are considerably shorter than the total service demands. We also find that the degree of concurrency is noticeably greater for remote than for local access. Thus, although there is an elapsed time penalty for remote file access, it is not as great as the discrepancies in service demands would suggest. In more detail:

—For reads, the ratio of total service demand in the remote case to total service demand in the local case is 3.2:1. For writes, this ratio is 2.75:1.
—For reads, the ratio of elapsed time in the remote case to elapsed time in the local case is 2.4:1. For writes, this ratio is 2.0:1.
—For local reads, elapsed time is 86 percent of total service demand. For local writes, elapsed time is 61 percent of total service demand. Some of the overlapped activity is accounted for by direct memory access (DMA).[3] The remainder must be actual processing that takes place concurrently with disk activity.

[3] DMA contributes to the CPU service demand because cycle stealing impedes the processing rate; this effect is, properly, captured by our measurement technique.

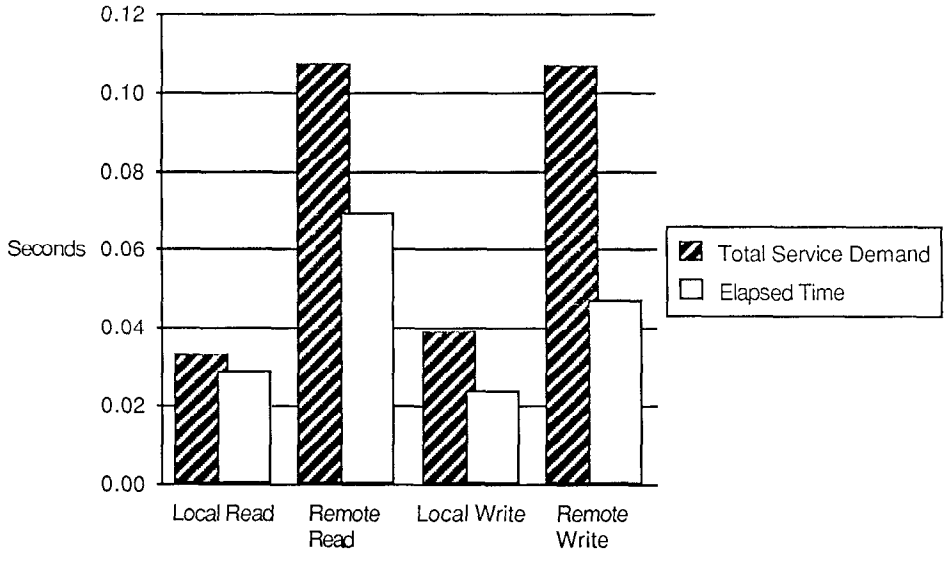

Figure 3

—For remote reads, elapsed time is 65 percent of total service demand. The additional concurrency over the local case can be accounted for by the multiple packet exchanges on the Ethernet. For remote writes, elapsed time is 44 percent of total service demand. The additional concurrency over remote reads can be accounted for by the fact that remote writes are largely asynchronous.

If we assume a 3:1 ratio of reads to writes,[4] then a general conclusion is that the factor by which the elapsed time of the I/O portion of workstation activity is degraded by remote rather than local file access is 2.3:1, ignoring any delay in file server access due to competition from other workstations.

3.3 Consideration of Local Processing

The elapsed time ratio of 2.3:1 between the remote and local cases is substantial and seems to indicate that response times on diskless workstations will be unacceptably worse than those on workstations with local disks. Such a conclusion, though, ignores the fact that I/O comprises only a portion of any computation; user mode CPU processing is unaffected by the location of the disk. The effect of considering this processing is to reduce the effective response time penalty of diskless operation dramatically.

To illustrate this, we employ the two customer characterizations used as examples in Section 2.3. Measurements of a live load at the University of Washington found that highly interactive users engaged in software development on SUN/UNIX systems averaged 106 ms of local user mode processing per 4K

[4] Although the ratio of reads to writes is quite sensitive to workload, our "general conclusion" is not highly sensitive to the value of the ratio that is chosen, because the remote-to-local elapsed time ratios for reads and writes do not differ dramatically.

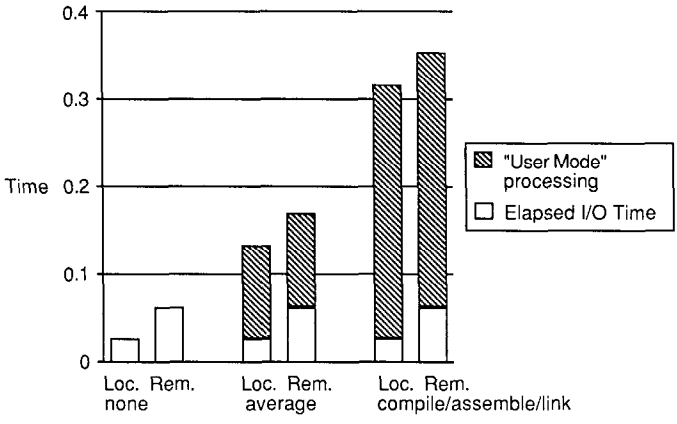

Figure 4

data transfer. During a compile/assemble/link sequence, approximately 289 ms of local user mode processing took place per 4K transfer.

Figure 4 illustrates each of these cases, as well as the baseline case, in which user mode processing is ignored. The columns in Figure 4 should be considered in pairs: The remote versus the local case for no user mode processing, for "average" activity (the software development characterization, to which we refer as "average" because it represents a composite of activities, rather than, say, purely editing or purely compilation), and for the compile/assemble/link sequence. Each column consists of two components: elapsed time for I/O (assuming a 3:1 ratio of reads to writes) and local processing. The overall height of each column can be interpreted as *relative response time.*

When only I/O activity is considered, the response time ratio of the remote case to the local case is 2.3:1. This ratio drops to 1.3:1 for "average" activity (software development, with 106 ms of user mode computing per 4K I/O) and to 1.1:1 for the compile/assemble/link sequence (289 ms of user mode computing per 4K I/O).

Our conclusion is a fairly strong one. The SUN/UNIX implementation of remote file access (again, our measurements predate NFS) is not particularly efficient. Nonetheless, when user mode activity is considered, the penalty for remote rather than local file access is negligible in the single-user case.

3.4 The Effect of Congestion

Unlike a dedicated disk on a single-user workstation, the file server and local area network are shared facilities, and access to them is subject to delay owing to competition from other workstations. In other words, the relative response time perceived by a workstation user can be thought of as having three components: User mode processing (which is the same regardless of whether file access is local or remote), basic elapsed I/O time assuming no congestion (which is lower for local than for remote access), and queuing delay due to congestion (which exists only in the remote case, and grows with the number of workstations sharing the file server). The effect of this delay is to increase the remote penalty.

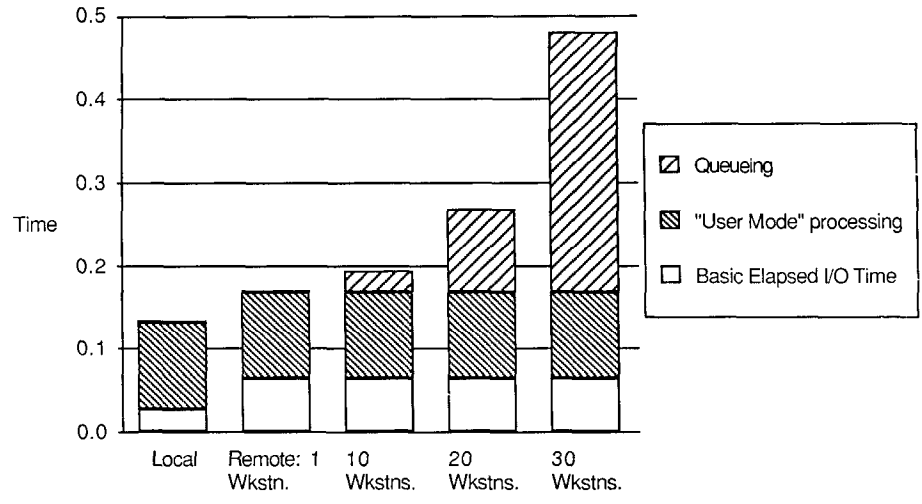

Figure 5

Figure 5 illustrates this for our "average" (software development) customer characterization: 106 ms of user mode processing per 4K I/O, and 4K of I/O per second of elapsed time. The first two columns reproduce the middle pair of columns in Figure 4: For the local and the single-user remote case, respectively, they show relative response time broken down into two components—basic elapsed I/O time and user mode processing. In the next three columns, we add the queuing delay incurred when 10, 20, and 30 "average" users, respectively, are sharing the server and network. These values were calculated using the queuing network performance model discussed in Section 2.

We see from Figure 5 that queuing delay can be significant, even for 20 workstations. For "average" activity in the single-user case, the remote-to-local response time ratio is 1.3:1. In the case of 10 workstations, this ratio increases to 1.5:1. With 20 workstations, the ratio is 2.0:1; 30 workstations yields a ratio of 3.6:1.

The queuing delays evident in Figure 5 are due to congestion at shared resources. Figure 6 graphs the utilization of the server CPU, the disk, and the network, as the number of workstations increases. Several key points are clearly illustrated by Figures 5 and 6 taken together:

—At light loads (small numbers of workstations), a file server design that minimizes file access latency in the absence of congestion (e.g., by maximizing overlapped activity) is of some value.

—However, even a design that is mediocre from the standpoint of light-load file access latency probably is acceptable when elapsed I/O time is considered in the context of user mode processing.

—At heavy loads (large numbers of workstations), performance is governed by the most heavily utilized device, which limits throughput and thus inflates response times.

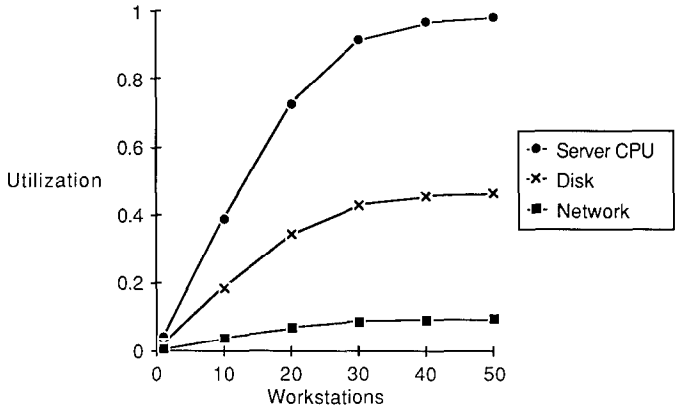

Figure 6

—It is clear that a single Ethernet can accommodate several file servers and their associated workstations before network contention becomes a serious issue.

—Finally, we note that our assumption that a seek occurs on average once per two 4K transfers has a negligible effect on our results, because the file server CPU cost of the SUN/UNIX system is so great.

The response time degradation due to congestion is much more pronounced than the savings due to improved light-load file access latency, especially when elapsed I/O time is considered in the context of user mode processing. Thus, a design that reduces light-load file access latency at the expense of increased service demands would appear to be inappropriate. Conversely, a design that reduces service demands, even at the expense of some increase in light-load file access latency, would appear to be desirable. It is trade-offs such as these that are explored in the next section.

4. DESIGN ALTERNATIVES

In this section we use the performance model developed in Sections 2 and 3 for the SUN/UNIX environment to investigate a number of system design alternatives intended to improve performance relative to this baseline.

As noted in Sections 2 and 3, light-load performance can be improved by reducing the service demand at any resource or by increasing the concurrency in processing a single request. In contrast, high-load performance can be improved only by reducing the service demand at the most heavily utilized device. In each of the three systems that we have measured, this bottleneck has been the file server CPU by a wide margin. For example, in the SUN/UNIX system the service demand at the file server CPU is more than twice as great as the service demand at the next most heavily utilized shared device, the file server disk. Thus, we expect design alternatives that reduce file server CPU demand to exhibit proportionately improved performance and other modifications to show little or no improvement.

Most of the design alternatives to be considered reduce service demands in one of two ways: amortization of overhead or enhancement of the configuration. The former actually reduces the total amount of work done by the system, whereas the latter spreads the existing work among more devices.

With amortization, the goal is to reduce the fixed overhead costs that are incurred per operation (rather than per byte) by increasing the number of bytes processed in each operation, thus reducing the total number of operations required. For example, increasing the disk block size decreases the number of I/O operations required to read a file and consequently reduces the amount of time spent in seek, latency, and device driver communication.

With enhancement, the major questions concern which resources to add and whether to add them to the server or to the clients. For example, if additional disks were to be added to the system, should they be used to augment the existing file server, to form (part of) another file server, or to provide (at least some) clients with local disks?

In evaluating each of the alternatives, our approach is the same: We modify the parameters of the queuing network performance model of the baseline system, then evaluate the resulting model to obtain performance projections. These analyses are intended to provide a sense of the relative merits of the various alternatives, rather than to present absolute measures of their performance. We have chosen an essentially arbitrary system (one conveniently available at the University of Washington) for our baseline, with the workload composed of "average" users from that system (as described in Sections 2 and 3). Although other systems would demonstrate differences both in absolute response times and in precise user behavior, the lessons learned from our models indicate what should be expected of any system in which similar modifications are undertaken.

As described in Sections 2 and 3, some assumptions concerning file access behavior that are used in our models are that

—half the disk operations require no seek,
—the ratio of reads to writes is 3:1,
—files are accessed sequentially.

These assumptions are discussed where relevant in the subsections that follow.

We note that queuing network models do not naturally allow the representation of overlapped processing of a single request, as that which occurs among the client CPU, the server CPU, and the disk in the systems we are considering (see Figure 3). Although sophisticated approaches to this problem exist (e.g., [3]), we have chosen to use a very simple technique. For each modification, we estimate the expected amount of overlapped processing under light loads by extrapolating from the measured overlap in the baseline system. We subtract this time from the response times projected by the model for all loads.[5]

[5] This technique introduces some error because we subtract a fixed amount of time for all loads, whereas in practice, because of queuing delays, the amount of overlap for high loads must be less than for light loads. Nonetheless, this simple technique is appropriate. First, the error introduced is applied uniformly to all design alternatives, so it does not affect projections of relative performance. Second, while the error increases in absolute value with increasing numbers of workstations, the percentage error decreases to a negligible value quickly, as total response times grow owing to congestion. Finally, the intent of this section is to evaluate system design issues; the use of a more complex modeling technique would distract from and perhaps obscure this thrust.

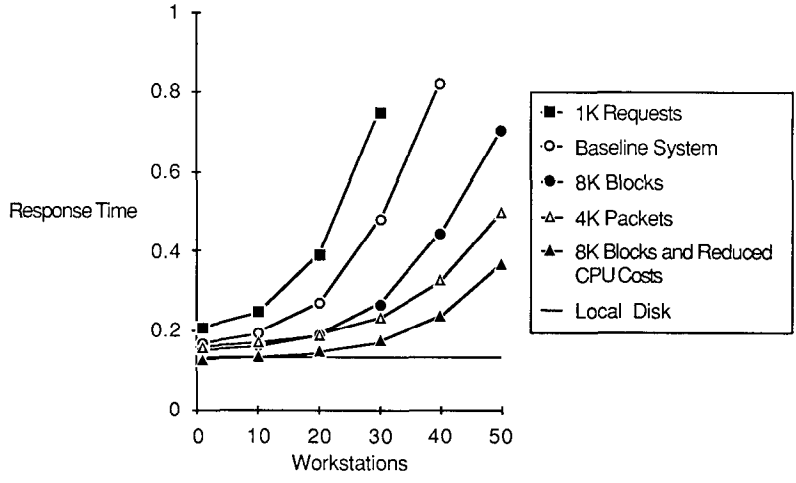

Figure 7

4.1 Reduced Data Access Latency

We begin by considering a scheme for reducing data access latency by buffering partial disk blocks at the file server. In the baseline system the client requests 4K blocks from the server, which replies with four 1K packets. An alternative is for the client to request 1K blocks. The intent is to reduce the time between the request and the resumption of processing by the client: Instead of waiting for 4K, the client can resume execution after 1K. To preserve efficiency in physical disk accesses, the server continues to transfer 4K disk blocks, buffering them in anticipation of subsequent requests.

These changes were reflected in the model by increasing the client and server CPU service demands to account for the increased number of request packets exchanged. Additionally, we eliminate the portion of the overlapped service of the baseline system corresponding to preparation of data packets by the server (or client) while earlier packets are being moved into memory by the client (server).

Figure 7 shows relative client response time as a function of the number of workstations for several design alternatives. (The line marked "Baseline System" corresponds to the data presented in Figure 5.) The latency reduction scheme causes a degradation in performance at both low and high loads. This degradation is due to the increased file server CPU service demands resulting from protocol overhead. In our modified system the file server must process one incoming and one outgoing message for each 1K transferred. In the baseline system, one incoming message was required for each 4K transferred. At low loads, this additional work more than offsets the decreased latency experienced by the client. At high loads, this additional work exacerbates the congestion at the file server CPU, which is the system bottleneck.

The negative impact of this modification illustrates the importance of overhead amortization. In this case the decrease in the user request size resulted in an increase in the number of overhead operations. Consequently, the CPU cost per byte increased. This suggests a guideline for system implementation: High-level

protocols should allow for large requests even if lower level protocols do not. In the specific case of our example, it was best to allow the client to request 4K at a time, even though the Ethernet restricts packets to at most 1K.

4.2 Increased Disk Block Size

The next modification considered is an increase in the disk block size from 4 to 8 kbytes, and a corresponding increase in the size of user requests to 8K (eight 1K packets are returned for each client request). This change has two primary effects: the effective disk access time is reduced since seek and latency costs are amortized over 8 kbytes rather than over 4, and the server CPU time is reduced since only one disk I/O operation and one user request packet are required per 8K transferred.

In representing this modification in the model, we assume that, as in the baseline system, half of the disk operations do not require a seek. Further, since file access is sequential, there is no penalty associated with "fragmentation," that is, retrieving unneeded information with the additional 4K per access. Although this is an optimistic assumption, its performance impact probably is small. In Section 4.5 we see that a number of factors militate against an even larger block size, where fragmentation would become a serious issue.

As shown in Figure 7, the system with 8K disk blocks performs much better than the baseline system.

4.3 Reduced CPU Costs

Although the increase in block size reduces the file server CPU service demand, that resource still is the system bottleneck. Thus, further improvements in performance are possible by further reductions in the file server CPU service demand. To illustrate this possibility, Figure 7 includes performance projections using parameter values taken from measurements of the V system with 8K disk blocks and user request sizes. The file server CPU service demand under V is only two-thirds that of the baseline system. Some of this improvement is due to the increased block sizes (for reasons noted in the previous subsection). The remainder is due to more efficient design and implementation of protocols—in particular, to a careful effort to minimize data copying. As shown in the figure, these latter improvements result in a further reduction in response times of roughly the same magnitude as that obtained from the block size increase alone.

4.4 Increased Local Area Network Packet Size

It is clear from the results of the previous examples that reductions in file server CPU service demand can be extremely effective in improving performance. Another way in which CPU service demands might be reduced is to decrease the number of data-carrying messages (as opposed to request messages, which were considered in Section 4.1) passed between client and server. Thus, the next modification considered is an increase in the maximum network packet size from 1 to 4 kbytes. The effect of this modification is to permit 4 kbytes of data to be exchanged with a single data packet, rather than with four. This reduces both client and server CPU overhead. The performance of the baseline system with this modification is shown in Figure 7.

4.5 The Limits to Amortization

One obvious conclusion to be drawn from our results thus far is that fixed overheads (such as disk access time and protocol overhead) should be amortized over large volumes of data. Given these results, it is natural to consider using block sizes even larger than 8K in an attempt to reduce both CPU processing time and effective disk service time per kilobyte. In fact, it appears that the block size should be made as large as possible. There are several moderating considerations, however:

—If blocks were very large, the file server and network would be busy for long "bursts," a situation that would cause both the mean and the variance of response times to increase.
—Disk and network buffering capacity is limited, and its effectiveness is reduced as transfer sizes increase.
—Because some costs are proportional to the amount of data processed, the marginal benefit of amortization decreases.
—Once the block size exceeds the size of a particular file, no further reduction in access time to that file can be obtained by further increases in block size.

To illustrate the third of these four points, Figure 8 shows disk time per kilobyte transferred as a function of disk block size, while Figure 9 shows CPU time per kilobyte transferred as a function of disk block size.

The disk times shown in Figure 8 were calculated from device characteristics. Since data transfer time is independent of block size, disk time per kilobyte has a nonzero limit as block size increases. Considering only disk time, there is little advantage in block sizes greater than 16 kbytes.

The CPU times shown in Figure 9 were measured on the Apollo system. (Some values were calculated from other measurements, rather than measured directly.) Disk transfer time (which ties up the CPU because of DMA bus contention) is independent of block size, as is the cost of sending data packets over the network. In addition, the Apollo system links 1K physical data blocks on disk to form larger "logical" blocks, so there is a small fixed processing cost per kilobyte transferred. Considering only CPU time, there is little advantage in block sizes greater than 8 kbytes.

To illustrate the fourth point, we measured the number of file CLOSEs broken down by file size—in other words, the distribution of file sizes weighted by frequency of use—on several UNIX systems used for software development.[6] Nearly half of all files were 1 kbyte or less in size, and nearly three quarters were 4 kbytes or less. The mean file size was roughly 11 kbytes. (These results are

[6] These measurements include both user file accesses and program loads. (There were approximately five times as many file accesses as program loads. Of the program loads, approximately half were accomplished via faulting rather than reading and are omitted from our calculations.) The measurements include directory references by application programs, but *not* references made by the system in resolving path names. We attempted to "filter out" those file CLOSEs that represented the periodic appending of a block to a log file. Some of these remain in the data, however. The net result is that our measurements probably *overestimate* the mean file size weighted by frequency of use in our environment, strengthening our conclusion concerning the diminishing benefit of large block sizes.

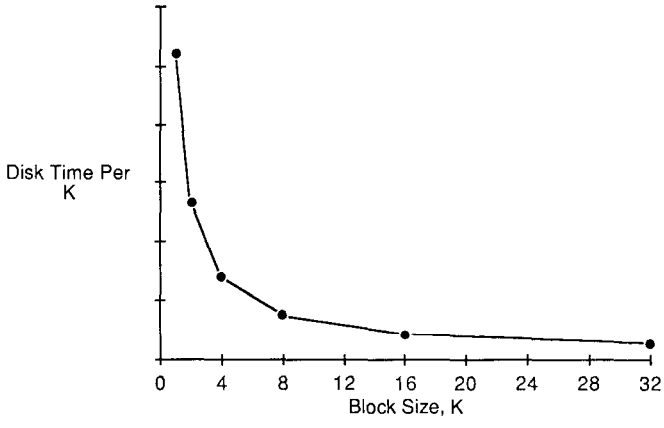

Figure 8

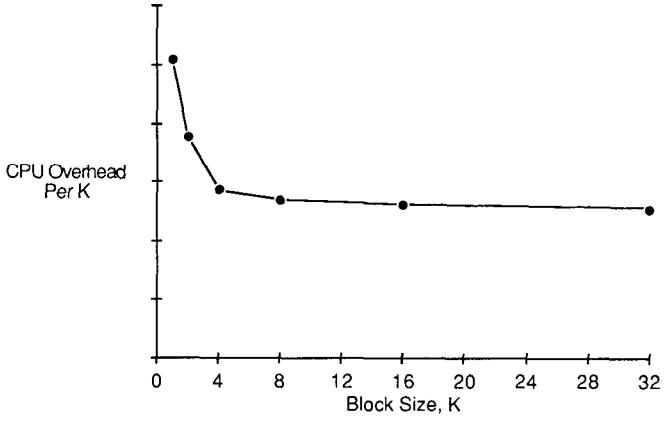

Figure 9

entirely consistent with the thorough measurements of the UNIX 4.2 BSD file system reported by Ousterhout et al. [8] after our study was conducted.) Such a distribution of file sizes serves to further accentuate the decreasing marginal benefit of increased block sizes that is evident in Figures 8 and 9. Consider "disk accesses" as the basic unit of cost. If all files were small, then there would be no benefit to increased block sizes. If all files were large, then there would be a geometrically decreasing marginal benefit: Doubling the block size would always halve the number of disk accesses. (This is the optimistic assumption underlying Figure 8.) With the observed file size distribution, the marginal benefit of increased block size is greater than in the case of uniformly small files, but less than in the case of uniformly large files. The asymptotic value for the number of disk accesses per kilobyte as the block size increases is the inverse of the mean file size. This is illustrated in Figure 10.

We note that a hierarchical directory structure encourages the use of many small files, both because they can be kept track of and because the directories

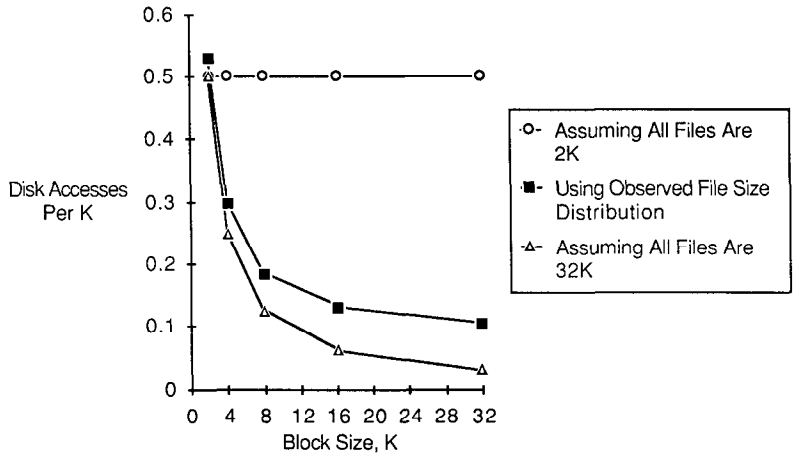

Figure 10

themselves are small. Thus, our distribution is considerably different than that observed for IBM systems by Smith [12]. Smith noted substantial differences between the "unweighted" file size distribution (that which would be observed by scanning the disk) and the distribution weighted by frequency of use. Thus, our distribution is considerably different from the "unweighted" distribution measured by Satyanarayanan [11] for a different system with a hierarchical directory structure.

We observe that the relatively small mean file size, coupled with competition among workstations, means that seeking is likely to be frequent in a shared file server environment, even if contiguous allocation of logically sequential blocks is achieved. Rating file system efficiency by the speed with which large files can be streamed off the disk [6] ignores the "average" case and can be misleading if the CPU costs of this streaming are ignored.

In summary, device characteristics (Figure 8), overhead characteristics (Figure 9), and file size characteristics (Figure 10) all indicate that 8–16 kbytes is a reasonable choice of block size.

4.6 Cylinder Loading: Prefetch

Two modifications that we do not consider in detail are to locate all blocks of a single file on the same cylinder in an attempt to reduce seek time, and to prefetch the logical successor of each block requested by the client in anticipation of the next request.

The first of these has the potential to decrease low-load response times (by reducing the disk access time), although not high-load response times (since the CPU remains the primary bottleneck). However, as was just argued, most files consist of only a few 4K blocks. Thus, there is limited potential to reduce seeks in this manner.

The second modification, prefetch, does not significantly reduce service demands at either the server CPU or the disk. Although file server CPU overhead may be reduced somewhat, the communication costs remain constant. Thus, the

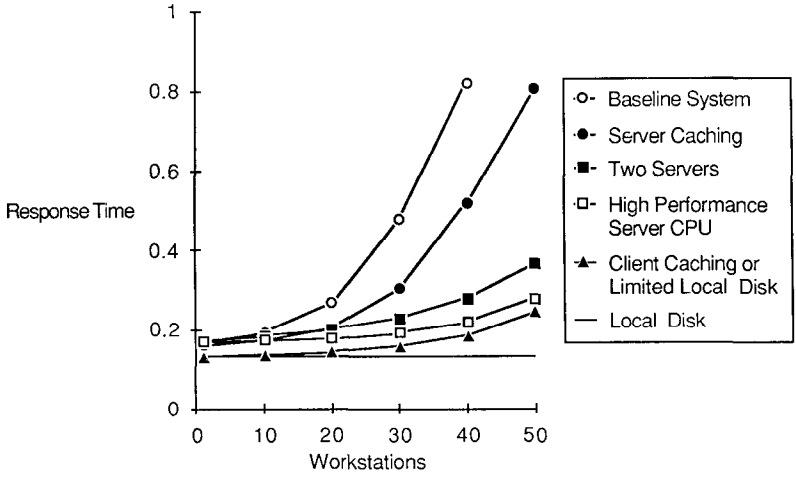

Figure 11

bottleneck service demand is largely unaffected, so this modification cannot significantly improve high-load performance. At low loads there is some increased concurrency between file server and client, so a limited improvement might result.

4.7 File Block Caching

In a caching scheme, file blocks are saved in main memory so that subsequent requests for these blocks can be satisfied without a disk operation. This is different from the buffering scheme studied in Section 4.1 in that an attempt is made to accommodate nonsequential references.

The success of caching depends on the cache hit ratio, which in turn depends on the cache management policy, the cache size, and the behavior of the workload. We have arbitrarily assumed a 50 percent cache hit ratio, which, based on studies of file caches in centralized systems, is quite conservative. (A cache hit ratio of 90 percent for reading has been reported for a distributed system [10], which with a 3:1 ratio of reads to writes gives a hit ratio of at least 67 percent.) Cache hits result from activities such as multipass compilers, which repeatedly access the source file and the temporary files they create. The first access of these files brings them into the cache; subsequent accesses result in cache hits.

In a file-server environment, file block caching can be done either at the file server or at the client workstations. As shown in Figure 11, these two alternatives yield significantly different improvements when introduced to the baseline system. It is this observation, rather than any more detailed conclusion, that is the import of this aspect of our work. File cache performance is considered in considerably greater detail in [8].

Caching at the file server reduces the service demand at the bottleneck (the file server CPU) only modestly, eliminating 50 percent of the CPU overhead due to disk access (because of the assumed 50 percent hit ratio) but having no effect on communication costs. Thus, the improvement is quite small. (Note that in

fact we have overstated the performance of this scheme, since we have not attempted to account for the CPU cost of managing the cache.)

Caching at the client yields a much greater improvement, since this approach eliminates 50 percent of the communication with the file server in addition to 50 percent of the CPU overhead due to disk access. (The CPU costs of cache maintenance are again ignored, but these costs now are borne at the client workstations, so do not contribute to queuing delay under load.)

Of course, there is a further cost associated with caching: Additional main memory is probably required. In the server caching scheme a single cache can be shared among all client workstations. In the client caching scheme, independent caches are required. It seems likley that the latter scheme would require more resources than the former to achieve the same hit ratio.

4.8 Limited Local Secondary Storage

In this design alternative, a small local disk is added to each workstation, to be used for paging, for temporary and nonshared files, and perhaps for caching shared files. The rationale for this alternative is the same as that of the client caching scheme just considered: To decrease the service demand at the bottleneck (the file server CPU) by reducing file server disk accesses and client/server communication.

If we assume a 50 percent hit ratio for the local disks, then at high loads the performance of this alternative will be identical to that of the client caching scheme. At light loads the client caching scheme offers a slight advantage, since the cache memory can be accessed faster than the local disk. However, the difference is extremely small, and to avoid clutter these alternatives are represented by a single line in Figure 11.

4.9 A Second File Server

We have considered two enhancements to the client workstations that yielded significant performance improvements. In this and the next subsection we consider two enhancements to the shared file service that provide comparable gains.

The most obvious of these, considered in this subsection, is to acquire a second file server to share the load imposed by the client workstations. Because the effect of this is to halve the service demand at the bottleneck, performance at high loads is extremely similar to that of both client caching and limited local secondary storage. At light loads the two client enhancements are somewhat superior, since the costs of remote access are avoided 50 percent of the time. This, however, is an artifact of considering this design alternative in the context of the baseline system, rather than in combination with increased block sizes and reduced CPU costs, which, as shown in Figure 7, yield performance comparable to a local disk in the lightly loaded case.

4.10 A High-Performance File Server CPU

The final alternative that we consider is the replacement of the file server CPU with one that is faster by a factor of 5, corresponding very roughly to the power of a Xerox Dorado.

The resulting file server CPU service demand is much smaller than that of any other alternative considered. However, the performance of this alternative is no better than that of the best of the previous modifications. The reason is that the file server disk (with a service demand roughly half that of the file server CPU in the baseline system) has become the bottleneck. Thus, once the CPU service demand has been halved, further improvements at high loads require roughly equal reductions in both CPU and disk service demands.

To exploit more fully the power of this CPU, the simplest change probably would be to add a second disk to the file server.

5. A COMPARISON OF THREE SYSTEMS

In this section we briefly examine the relative performance of three systems in the light of the design alternatives investigated in Section 4.

Figure 12 shows relative response time as a function of load for SUN/UNIX, Apollo DOMAIN, and V. For V, three different file access protocols are shown: Client requests and disk I/O in 1K blocks, client requests and disk I/O in 8K blocks, and client requests in 1K blocks but disk I/O in 8K blocks. As in earlier sections, a 3:1 ratio of reads to writes is used, and the measured "average" activity is taking place on each workstation. All workstations are running diskless; all file servers have equivalent disks. Measured CPU service demands are adjusted to compensate for minor differences in CPU capability (e.g., 8-MHz versus 10 MHz 68000).

Despite these efforts at "equating," it is important to acknowledge that to some extent we are comparing apples and oranges. Consider the processing required on the client end prior to the actual sending of a message to the file server. V was designed from the outset with the objective of diskless operation using a fast network IPC; very little client processing is required. Apollo DOMAIN handles file I/O through the paging system, so the overhead is roughly equivalent to that of a page fault. SUN/UNIX requires substantial file system processing prior to initiating a remote request. Note, however, that congestion on the file server end, rather than on the client end, is responsible for the differences in performance illustrated in Figure 12, and that any of these systems could in principle be supported by any of the file server architectures. Thus, the comparisons are meaningful, although by no means do they present a complete picture of the relative performance of these systems. We briefly discuss the results system by system, from the worst performer to the best:

V, 1K client requests, 1K disk transfers. In V, the CPU cost of the primitive operations is extremely low. The poor performance demonstrated in the figure indicates the substantial penalty that must be paid for interacting with the file server and with the disk on a 1K basis. This cost has two components: The primitive operations, although efficient, are executed many times, and disk overhead (seek and latency) cannot be amortized.

SUN/UNIX. Because file accesses in SUN/UNIX involve 4K blocks, it is not surprising that the performance of this system falls between that of V with 1K blocks and the various systems employing 8K blocks. It is worth noting, though, that the CPU costs of remote file access in SUN/UNIX are disproportionately

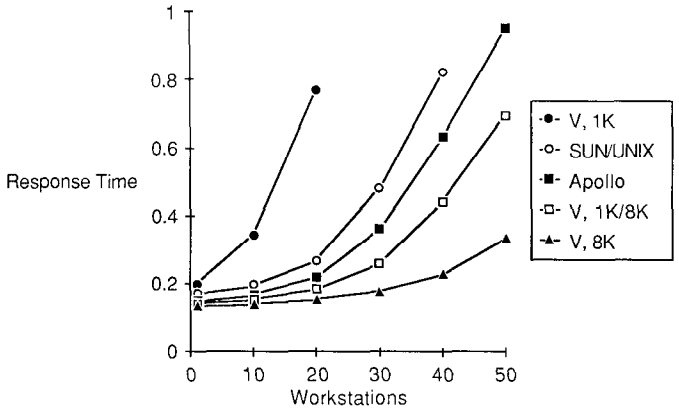

Figure 12

high: The projected performance of V or Apollo DOMAIN operating at a 4K block size is noticeably better than that of SUN/UNIX. (We are reporting SUN/UNIX performance prior to the recent introduction of SUN's Network File System.)

Apollo DOMAIN. The Apollo system differs from both V and SUN/UNIX in that reads and writes have substantially different costs. Before discussing its performance relative to these systems we consider Figure 13, which shows the performance of reads, the performance of writes, and the performance of a 3:1 ratio as graphed in Figure 12.

Performance for reads is equivalent to that of V with 8K client requests and 8K disk transfers (the axes on Figures 12 and 13 are identical)—a highly efficient system. The fact that data are transferred in units of 8K, both between the disk and the server and between the server and the client, reduces both I/O and CPU service demands. (In fact, the disk block size is 1K, but logically contiguous blocks are allocated to physically contiguous sectors, and the driver transfers eight blocks at once.) The CPU service demand is further reduced by two factors. First, the network interface does DMA directly to/from process address spaces. (This reduces both "direct" CPU cost and also overhead.) Second, files are mapped into process address spaces, so file I/O is handled by the paging system rather than by a layered file system.

Performance for writes is equivalent to that of V with 1K client requests and 1K disk transfers—a highly *in*efficient system. CPU costs are increased by the fact that every page must be handled multiple times. On each client, a process called the *purifier* detects "dirty" pages, clusters them into groups of eight (memory pages, like disk pages, are 1K), and delivers them to the file server. These pages then are mapped into the address space of the file server. There, *another* purifier detects dirty pages, clusters them into groups of 20, and delivers them to the disk driver. Disk costs are increased by the fact that the 20 pages delivered by the file server purifier to the disk driver will not be logically or physically contiguous (the purifier scans page frames in order by physical ad-

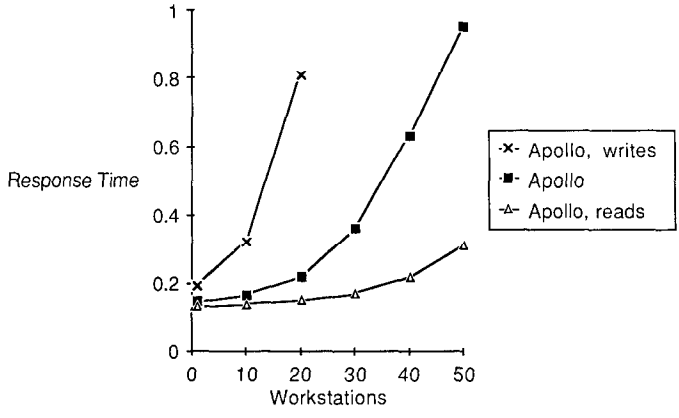

Figure 13

dress). Although "scan" scheduling is employed at the disk, both seek and latency costs are high. In other words, the "clustering" of a number of 1K disk pages is not effective when writing.

It is a tribute to the efficient implementation of file reading that overall performance (assuming a 3:1 ratio of reads to writes) is relatively good. Our results probably do not do full justice to the performance of this system, since it employs a two-level caching scheme that may reduce total disk I/O in the case of temporary files.

V, 1K Client Requests, 8K Disk Transfers. In this protocol, disk seek and latency are effectively amortized, although clients continue to interact with the server using a 1K block size. The amortization makes performance considerably better than that of V with 1K disk transfers, while the overhead of 1K client interactions makes performance somewhat worse than that of V with 8K client requests.

Note that the entire improvement observed here relative to V with 1K disk transfers is attributable to the factor of eight reduction in interactions with the disk.

V, 8K Client Requests, 8K Disk Transfers. Since the underlying Ethernet protocol imposes a 1K limit on packet size, the entire improvement observed here relative to V with 1K client requests is attributable to the factor-of-8 reduction in client request messages. In other words, allowing the high-level protocol to interact in large block sizes, even if the low-level protocol cannot, is of significant benefit.

6. CONCLUDING REMARKS

Environmental, economic, and administrative considerations are leading to the increased use of workstations that are essentially diskless, relying on shared servers for file storage. In this paper we have used measurement and modeling to study various factors governing the performance of such systems.

The first subsection of these concluding remarks states and elaborates the major conclusions of our study. It is important to bear in mind the environment

that we have considered: processors similar in power to the MC68000 at the clients and the file server, remote file access but not paging, and user behavior as observed in a UNIX software development environment. The extent to which our conclusions rely on these or other assumptions is noted.

The second subsection discusses several other issues raised in the course of our study, while the third suggests some interesting further questions.

6.1 Major Conclusions

Conclusion 1. A system of diskless workstations with a shared file server can have satisfactory performance. By this, we mean performance comparable to that of a local disk in the lightly loaded case, and the ability to support substantial numbers of client workstations without significant degradation. As with any shared facility, good design is necessary to minimize queuing delays under high load.

Elaboration. When viewed in the context of the user mode processing that also occurs, the cost of remote file access is reasonable, assuming that the load on the file server is light enough that queuing delays are negligible. This is true even for a file server and a client/server interface that are not especially well designed.

As the load on the file server increases, queuing delays cause performance to degrade. It is here that differences in design become important. With a well-designed file server and client/server interface, the cost of remote file access is reasonable even for substantial numbers of client workstations.

The performance of the system under load is determined by the service demand at the most heavily utilized device. Based on measurements of three existing systems and on the results of our modeling experiments, the file-server CPU is the critical resource when disk block sizes are 4 kbytes or greater.[7] Network congestion is not an issue for a small number (<5) of server/client suites (assuming capacity equivalent to a 10-Mbit Ethernet).

Because service demand determines performance under load, design decisions that attempt to minimize file access latency at light loads at the cost of even slight increases in service demands at higher loads probably are inappropriate. (A small client request size is an example of such a design decision.)

Neither this nor our other conclusions rely heavily on the characterization of "average" user behavior that is employed. The amount of user mode processing per 4K transferred (106 ms) affects the ratio of remote to local response times in the single-user case, but not the growth of this ratio due to congestion. The rate of data transfer (4K per second) affects the labeling of the x axis in our graphs of response time, but not the relative performance of design alternatives.

Nor are our conclusions affected as systems evolve to use faster processors not only at the first server, but also at the client workstations. Although performance clearly will improve in absolute terms, a natural concern is that the *relative* performance of remote file access—the response time ratio of the remote case to

[7] A significantly faster file server CPU would expose the disk as a bottleneck. Of course, the addition of a second disk to the file server would address this problem; such an addition might well be required for reasons of file storage capacity in any case.

the local case considered in Section 3—will degrade because of the dramatic reduction in the amount of user mode processing that takes place per file access. In fact, just the opposite occurs. If both client and server are equipped with processors five times as fast as in the baseline system, we project that the response time ratio of the remote case to the local case will drop from 2.3:1 (for the baseline system) to 1.5:1 ignoring user mode processing entirely, and from 1.3:1 to 1.2:1 considering "average" user mode processing. The reason is that as CPU power increases, communication overhead becomes relatively insignificant, while the fixed costs of I/O (e.g., disk access time) play an increasingly dominant role.

Conclusion 2. The key to efficiency is protocols that allow volume transfers at every interface (e.g., between client and server, and between disk and memory at the server) and at every level (e.g., between client and server at the level of logical request/response and at the level of local area network packet size). However, the benefits of volume transfers are limited to moderate sizes (8–16 kbytes) by several factors.

Elaboration. Large disk transfers are advantageous because they allow the fixed overheads of disk access (seek and latency) to be amortized over a large number of bytes. Of equal importance, the CPU service demand incurred in initiating and terminating disk data transfers also can be amortized.

Similarly, provision for large client request sizes reduces software overheads. The effect is especially pronounced if lower level protocols (e.g., local area network packet size) support these large requests. However, provision for large client request sizes is important even if lower level protocols require decomposing each request into a number of smaller operations.

The benefits of amortization through volume transfers are well known. Our contribution is to demonstrate that this is the *key* performance issue, and to quantify its effect.

Among the factors that limit the benefits of volume transfers to moderate sizes are the undesirability of long data transfer bursts, limitations on the size and effectiveness of buffers, the decreasing marginal benefit of overhead amortization, and the relatively small mean file size observed in such systems.

Large service bursts contribute to high means and high variability in response times, an undesirable situation.

Disk and network buffering capacity is limited, and its effectiveness is reduced as transfer sizes increase. Clearly, large disk buffers reduce the number of buffers available in a given amount of memory, delaying client requests or disk prefetching under multiple client load. Less obviously, inadequate buffering for network traffic can lead to packet loss and the attendant overhead of time-out and retransmission processing.

The marginal benefit of amortization decreases with increased volume. Data transfer time becomes an increasingly significant component of the overall cost of remote file access. Some components of CPU processing may grow linearly with volume.

Finally, our measurements indicate that when file sizes are weighted by frequency of use, roughly 50 percent are 1 kbyte or less in size, roughly 75 percent

are 4 kbytes or less in size, and the mean size is roughly 11 kbytes. This further limits the benefits of large volume transfers.

Conclusion 3. From a performance point of view, augmenting the capabilities of the shared file server may be more cost effective than augmenting the capabilities of the client workstations.

Elaboration. The shared file service can be enhanced by adding additional file servers or by adding disks, memory, or processing capability to existing file servers. The file access performance of individual clients can be enhanced by adding memory or local secondary storage. Both approaches contribute to the robustness of the system with respect to peaks in load—the response time curve is flat over a relatively wide range—and also yield improved "average" performance, by reducing the load on the shared file service.

A comparative cost-benefit analysis naturally depends on current economics and technology. As one example, consider a system of 15 diskless workstations sharing a single file server. (Our results indicate that this is well within the realm of feasibility for a good design.) This system could be expanded to 30 workstations without affecting response times by adding a second file server, or by equipping each of the 30 workstations with a local disk (assuming roughly a 50 percent hit rate at the local disk). At this point in time, a local 42-Mbyte disk for a workstation costs roughly $6000, for a total cost of $180,000 for 30 workstations. A file server with the same total disk capacity would cost roughly $50,000. Augmenting the shared file service is further supported by lower software, environmental, and administrative costs.

Of course, various factors may justify the enhancement of specific client workstations. An individual may require (and be willing to pay for) better than average performance, justifying the addition of a local disk or the expansion of the local file cache. The use of a local disk for paging and for temporary files can yield benefits without incurring the disadvantages of relying entirely on local disks: Such use does not require large amounts of local disk space, backup of the local disk, or file migration. (Of course, local disks used in this way do not eliminate the need for high-performance transparent access to large-capacity shared file systems.) Caching can be effective in several situations where I/O is actually reduced: directory entries, which may be referenced many times, and temporary files such as those produced during compilation, which may never need to reach disk. The fact that a large proportion of references are to extremely small files suggests that a relatively small cache may be effective.

Desire for occasional node autonomy and for local storage of private data may militate in favor of local secondary storage. These applications increase the need for local disk space, backup, and migration, while still not eliminating the need for high-performance transparent access to large-capacity shared file systems.

Thus, workstation systems should be designed such that a local disk may optionally be configured with any individual workstation, in a way that is transparent to most of the operating software and to all applications. For example, the V system's transparent message-based I/O allows file access to be local, remote, or both, transparent to applications and to server processes.

Conclusion 4. Network contention should not be a performance problem for a 10-Mbit network and 100 active workstations in a software development environment.

Elaboration. Measurements of UNIX software development environments report average file transfer rates ranging from 300 bytes to 4 kbytes per active user per second, depending on the definition of an "active" user. Even at the high end of this range, 150 active workstations would be required to achieve an average utilization of 50 percent on a 10-Mbit network.

6.2 Other Issues

System Measurement. In measuring our baseline systems, we found much greater than expected errors in the process CPU time reported by the operating system. Among the contributing factors were erroneous attribution of interrupt-level processing and failure to capture processor degradation due to DMA memory interference. Discrepancies as great as a factor of 5 were observed between reported values and values obtained using an auxiliary process that "counted" available processor cycles. A system design issue that this emphasizes is the fact that DMA is not "free" and certainly is not a panacea for performance problems. In fact, measurements of the V file server reveal significantly worst performance using a DMA Ethernet interface than using a programmed interface.

Accommodating a Wide Range of Loads. Considerable care is required both in system design and in system modeling when a wide range of loads must be accommodated. As one example, we already have noted that design decisions that attempt to minimize file access latency at light loads at the cost of even slight increases in service demands at higher loads probably are inappropriate. As another example, various factors may lead to more efficient processing under heavy loads than would be extrapolated from light-load measurements. For instance, multiple concurrent compilations may access the same compiler binary files and the same header files, resulting in improved cache performance. As a third example, other factors may have the opposite effect: contention for packet buffers, disk buffers, disk heads, etc. Our study has incorporated these influences to a large degree, but system and model parameters must be chosen carefully with these factors in mind.

Secondary Implications of Volume Transfers. The use of large data request and transfer units has secondary implications on design and configuration. Large network request and transfer units currently require additional dedicated packet buffering to avoid packet loss. Large disk transfer units require large individual buffers or scatter/gather capability in the disk controller. Maximum efficiency is gained when the file layout on disk matches the transfer unit. This can be accomplished with either large disk pages or highly contiguous allocation of smaller pages as used in the Apollo and V servers. The latter approach preserves the efficiency of large pages while reducing disk space wastage due to fragmentation. (Note that control information (e.g., links) must be separated from data to achieve maximum efficiency in this case. V does so; Apollo does not.)

6.3 Additional Questions

High Performance CPUs. We have modeled the use of high performance CPUs (five times the power of an MC68000, roughly equivalent to a Xerox Dorado) at the file server. We also have briefly discussed the implications for remote file access of the use of such CPUs at client workstations. A more detailed consideration of this coming technology certainly would be worthwhile.

The Effect of Paging. Our study has considered file access but not paging. We had a number of reasons for this choice. On the diskless workstation systems that we studied, the number of bytes transferred owing to file access dominates the number of bytes transferred owing to paging. (E.g., in the SUN/UNIX software development environment at the University of Washington, where SUN workstations are equipped with 2 Mbytes of primary memory, file access dominates paging by a ratio of 4 : 1.) The dominance of file access can be expected to increase as the primary memory sizes on workstations continue to grow. The use of local disks for paging presents far fewer challenges than the use of local disks for file access. Finally, paging seems to have received much more attention than file access in the literature. Nonetheless, further consideration of remote paging is important. It seems beneficial to use different block sizes for paging activity and sequential file activity, as done for example by the Apollo DOMAIN software, which transfers blocks of two 1K pages for paging and eight 1K pages for file access.

Other Application Areas. To a considerable extent, our results are dependent on the distribution of file sizes observed in the UNIX software development environment. Even in this environment, a network file system will want to handle in a reasonable way the occasional enormous file. In application domains such as computer imaging, though, such files are the exception rather than the rule, and further study is necessary.

ACKNOWLEDGMENT

Norm Hutchinson and Jan Sanislo of the University of Washington contributed substantially to our work by instrumenting various SUN and VAX systems and by conducting measurement experiments. Paul Leach of Apollo Computer conducted measurement experiments on the Apollo system. Among the others who lent us their insights were Andrew Birrell, Mike Powell, and Mike Schroeder of Digital Equipment Corporation, Jim Hamilton of Apollo Computer, and Alan Jay Smith of Berkeley. Norm Hutchinson and Jerre Noe of the University of Washington, along with the TOCS referees and editors, provided many constructive suggestions. Paul Roy of Stanford is responsible for the implementation of the V file server.

REFERENCES

1. CHERITON, D. R. The V kernel: A software base for distributed systems. *IEEE Softw. 1*, 2 (Apr. 1984), 19–42.
2. CHERITON, D. R., AND ZWAENEPOEL, W. The distributed V kernel and its performance for diskless workstations. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles* (Bretton Woods, N.H., Oct. 10–13). ACM, New York, 1983, pp. 128–140.

3. JACOBSON, P. A., AND LAZOWSKA, E. D. A reduction technique for evaluating queueing networks with serialization delays. In *Proceedings of the 9th International Symposium on Computer Performance Modelling, Measurement, and Evaluation* (May). North Holland, Amsterdam, 1983, pp. 45–59.
4. LAZOWSKA, E. D., ZAHORJAN, J., GRAHAM, G. S., AND SEVCIK, K. C. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Englewood Cliffs, N.J., 1984.
5. LEACH, P. J., LEVINE, P. H., DOUROS, B. P., HAMILTON, J. A., NELSON, D. L., AND STUMPF, B. L. The architecture of an integrated local network. *IEEE J. Selected Areas Commun. SAC-1*, 5 (Nov. 1983), 842–857.
6. MCKUSICK, M. K., JOY, W. N., LEFFLER, S. J., AND FABRY, R. S. A fast file system for UNIX. *ACM Trans. Comput. Syst. 2*, 3 (Aug. 1984), 181–197.
7. MITCHELL, J. G., AND DION, J. A comparison of two network-based file servers. *Commun. ACM 25*, 4 (Apr. 1982), 233–245.
8. OUSTERHOUT, J. K., DA COSTA, H., HARRISON, D., KUNZE, J. A., KUPFER, M., AND THOMPSON, J. G. A trace-driven analysis of the UNIX 4.2 BSD file system. In *Proceedings of the 10th ACM Symposium on Operating Systems Principles* (Orcas Island, Wash., Dec. 1–4). ACM, New York, 1985, pp. 15–24.
9. POPEK, G., WALKER, B., CHOW, J., EDWARDS, D., KLINE, C., RUDISIN, G., AND THIEL, G. LOCUS: A network transparent, high reliability distributed system. In *Proceedings of the 8th ACM Symposium on Operating Systems Principles* (Pacific Grove, Calif., Dec., 14–16). ACM New York, 1981, pp. 169–177.
10. RICHARDSON, M. F., AND NEEDHAM, R. M. The TRIPOS filing machine, a front end to a file server. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles* (Bretton Woods, N.H., Oct. 10–13). ACM, New York, 1983, pp. 119–128.
11. SATYANARAYANA, M. A study of file sizes and functional lifetimes. In *Proceedings of the 8th ACM Symposium on Operating Systems Principles* (Pacific Grove, Calif., Dec. 14–16). ACM, New York, 1981, pp. 96–108.
12. SMITH, A. J. Analysis of long term file reference patterns for application to file migration algorithms. *IEEE Trans. Softw. Eng. SE-7*, 4 (July 1981), 403–417.
13. SWINEHART, D., MCDANIEL, G., AND BOGGS, D. WFS: A simple shared file system for a distributed environment. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles* (Pacific Grove, Calif., Dec. 10–12). ACM, New York, 1979, pp. 9–17.

Received July 1984; revised October 1985.