Performance Benefits of Non-Volatile Caches in Distributed File Systems

Prabuddha Biswas
Office and Teamware Engineering
Digital Equipment Corporation
110 Spitbrook Road, Nashua, NH.
pbiswas@star.enet.dec.com

K. K. Ramakrishnan
Distributed Systems Architecture and Performance
Digital Equipment Corporation
550 King Street, Littleton, MA.
rama@erlang.enet.dec.com

Don Towsley
Computer Science Department
University of Massachusetts
Amherst, MA.
towsley@cs.umass.edu

C. M. Krishna
Electrical & Computer Engg. Dept.
University of Massachusetts
Amherst, MA.
krishna@ecs.umass.edu

Abstract

We study the use of non-volatile memory for caching in distributed file systems. This provides an advantage over traditional distributed file systems in that the load is reduced at the server without making the data vulnerable to failures. We propose the use of a small non-volatile cache for writes, at the client and the file server, together with a larger volatile read cache to keep the cost of the caches reasonable.

We use a synthetic workload developed from analysis of file I/O traces from commercial production systems and use a detailed simulation of the distributed environment. The service times for the resources of the system were derived from measurements performed on a typical workstation.

We show that non-volatile write caches at the clients and the file-server reduce the write response time and the load on the file server dramatically, thus improving the scalability of the system. We examine the comparative benefits of two alternative writeback policies for the non-volatile write cache. We show that a proposed threshold based writeback policy is more effective than a periodic writeback policy under heavy load. We also investigate the effect of varying the write cache size and show that introducing a small non-volatile cache at the client in conjunction with a moderate sized non-volatile server write cache improves the write response time by a factor of four at all load levels.

1 INTRODUCTION

Due to dramatic improvements in processor speeds, the I/O system has become the bottleneck in current systems. In distributed file systems, this has led to the placement of caches at the file server, and often at the clients, in order to increase the performance of the I/O subsystem. This has traditionally reduced read response times and increased the throughput as seen by user applications. As a consequence, the workload to the I/O system, particularly on the file server, consists mostly of writes. Further performance improvements have been achieved by allowing either the client or server to delay committing writes to stable storage (the disk at the file server) in order to take advantage of overwriting of the same file's contents [6, 12, 17]. Such delayed writes are termed writebacks. Writebacks are often performed periodically. This is typical of an operating system such as Unix and a distributed file system such as Sprite [12], even though it exposes data to loss due to failure for short intervals of time.

The exposure of data to loss through the use of a writeback mechanism cannot be tolerated in many commercial applications, in spite of the performance benefits. In this paper, we consider a distributed file system in which a small amount of non-volatile random access memory (NV-RAM) is introduced as a write cache in order to remove the vulnerability to failure and, at the same time, obtain the performance benefits that can be achieved using a writeback policy. NV-RAM is relatively inexpensive in small quantities and easily incorporated into existing file systems. We show that a small amount of NV-RAM as the write cache, in addition to a volatile RAM read cache, located at the server and/or the clients provides most of the performance achievable through a writeback scheme using a single large cache at the server and/or clients. As part of this study,

- we examine two different writeback policies for the NV-RAM write cache, one a *Periodic policy*, the other a *Threshold-based policy*, and observe that the latter provides better performance under heavy load and is also insensitive to a range of threshold values.
- we examine the effect on performance of placing different amounts of NV-RAM at the server or client.
- we study the effects of different parameters such as threshold values, frequency of periodic writebacks and NV-RAM cache size on I/O performance.
- we study the effects of the NV-RAM write caches has on the behavior of the policies that are required to maintain consistency among the client caches [12, 17].

Our evaluations are based on simulations using synthetic workloads derived from file I/O traces collected on operational production systems [4, 15]. The synthetic workloads were derived from an extensive characterization of these traces and are of independent interest. These workloads account for the presence of sequential and random access files, sharing of files by clients, and whether or not files are exclusively read, exclusively written, or both read and written. This methodology was chosen over analytic modeling [9,14] because it allows us to model cache management policies more closely. We also felt it was better to use this methodology over trace driven simulations [1,7,12,17] because it provides us with the ability to modify different workload parameters in order to examine their effects on I/O system performance.

Although there is currently a product commercially available [11] which uses NV storage to allow for writebacks of data at the server, there has not been any study of the quantitative benefit of using NV storage in distributed file systems, particularly at both the client and server. Several

recent efforts have examined quantitatively the performance benefits to the I/O subsystem that are gained with the introduction of NV write caches at the disk controller [3,10]. The threshold based writeback policy was first suggested in this context in [3].

In section 2 we present the traditional caching policies and also explain how an NV write cache can be used together with a volatile read cache. In the following section, we discuss the performance metrics used in this study. Section 4 includes a description of the traces and a discussion on how they are used for characterizing the workload. Section 5 presents the details of the simulation model and section 6 contains the results of the study. Finally, section 7 concludes with a summary of the major observations.

2 CACHING POLICIES

We consider a distributed file system where a certain number of client workstations access all files, including system executables, from a common file server. The clients are always assumed to have volatile file block caches. We study the effect of using non-volatile (NV) caches at the server and/or the clients to buffer writes coupled with different cache replacement algorithms. File blocks that are accessed for read are loaded into the volatile cache and the ''least recently used'' (LRU) replacement policy is used to replace these blocks when the cache is full. When a single volatile file cache is used for both reads and writes, three alternative policies are traditionally used to handle write operations:

- write through
- writeback
- periodic writeback

In the write through scheme, the updated file block is inserted into the volatile cache at the same time that the write request is sent to the server. The write is considered to be completed by the application at the client when a successful response is received from the server. This is done so that the file server always has the most up to date version of the files and no special policies are required to maintain consistency of data at the clients and the server. When a writeback client caching scheme is used, the updated file block is inserted into the volatile cache but not immediately sent to the server. An LRU replacement policy is used and a ''dirty'' block is written back to the server only when it is displaced from the cache. This scheme results in fewer writes going to the server but leads to the situation where the file server may not have up to date information. The most recent version of a file block may be in a client's cache and the server requires a policy to ensure a consistent view to all clients accessing a shared file. If a client crashes, the modified file blocks in its volatile cache would be lost. This could potentially leave the file system in an inconsistent state. As a result, write back with only a volatile cache at the client or server is rarely used. However, a compromise policy is one where dirty blocks in client caches are periodically written back to the server. In such a scheme, the updated blocks are written to the volatile cache and are not immediately written through to the server as in a traditional write through scheme. However, in order to narrow down the window of time during which the file is vulnerable to corruption, the client cache is periodically checked and modified blocks older than a cer-

tain time are written back to the server. The Sprite distributed file system [12] uses this scheme where the cache is checked every 5 seconds to flush modified blocks that are more than 30 seconds old.

2.1 Cache Consistency Policies

With any form of writeback-based client cache management policy the problem of data consistency will continue to exist. When it is important to provide ''strict'' consistency (i.e. each client will always get the most up to date version of a file block it requests), a client cache consistency policy similar to that used in the Sprite distributed file system may be used. In this scheme all file open and close operations have to be performed at the file server and the file server keeps track of the last client that opened the file with the intent to update it. When another client opens a file that was updated and closed by another client, the server recalls the ''dirty'' blocks from the last writer. If the server detects that a file is concurrently opened by multiple clients and at least one client has the file open for writing, then it disables client caching for that file at all the clients. To ensure that the client cache contains the most up to date information, the clients and the servers maintain a version number for each file. When a client opens a file and finds blocks from that file in its cache, it checks the version number of the file at the server to ensure that it has the latest version. If it does not, it removes the old file blocks from the cache and issues a request to the server so that the up-to-date blocks are brought into the client cache. These policies are explained in greater detail in [12] and improvements to them have been investigated in [1,17].

2.2 Non Volatile Write Cache

File system measurement studies have shown that writes will be the primary load on the file server when large write-through client file caches are used. Even when a periodic client cache writeback policy is used, most of the writes go through to the server in order to ensure data integrity and not as a consequence of cache size restrictions [2]. This has motivated the use of non-volatile caches to buffer writes so that a writeback policy can be used without the fear of losing modified file blocks. There are practical issues regarding how the NV cache of a crashed client can be accessed. These however, are outside the scope of this paper and will not be considered further.

We suggest a modified cache management policy to take advantage of the NV write cache. An updated file block is inserted into the NV write cache and the block is removed from the volatile read cache if it is present there. This avoids maintaining duplicate blocks in both caches. The volatile read cache is first searched when a file block is requested for read. If it is not found, then the write cache is searched. If the block is found in the write cache and it is found to be ''clean'' (i.e., an up to date version is available at the file server) then the block is removed from the write cache. A block in the write cache may become ''clean'' if one of the cache consistency or replacement algorithms causes it to be written back to the server. When the write cache is full, the LRU block is replaced and is written back to the server only if it is ''dirty''. One can writeback dirty blocks early from the cache in order to avoid the scenario where an operations is stalled

because the cache is full and a dirty block has to be committed to the server to make room. Such writebacks could be governed by a threshold policy [3] or use a periodic writeback policy.

Under a threshold based policy, the number of dirty blocks in the cache is tracked and the least recently used dirty blocks are purged out of the NV cache as soon as the percentage of dirty blocks in the NV cache crosses a predetermined high-limit threshold. Purging of dirty blocks from the client NV caches to the server continues till the fraction of the dirty blocks in the cache drops below the low-limit threshold. We observed the benefits of such a policy in the context of a disk cache where write cache purging occurs while the disk is idle [3]. In the DFS environment it is not possible for the client to know when the server is idle and therefore the client will purge its NV write cache when it deems necessary.

In the periodic writeback policy, each client checks its NV cache periodically (say 5 seconds) and purges dirty blocks residing in the cache for a period longer than a certain interval (say, 30 seconds). A possible advantage of this method is that such a policy might reduce the file open time by reducing the cache consistency traffic resulting from the server having to check and flush the contents of the cache of the last writer of the file. The disadvantage of the periodic writeback policy is that file blocks may be prematurely written back to the server when they might be overwritten or deleted by the same client before it is ever accessed by any other client.

3 PERFORMANCE METRICS

The metric that is of primary importance from the user/application perspective is the response time of read and write requests. When a request is satisfied in the cache locally at the client, then the read or write response time does not include a transaction with the server. When the request has to go to the server, the response time includes the server execution time and, if there is a miss at the server cache, an access to the server disk subsystem. Since we are evaluating the impact of introducing NV storage to improve write response time, we focus on the effects of different NV storage management policies on the write response time. Since writeback caching policies at the clients can impact not only writes but also file open operations as a result of cache consistency policies, we also evaluate the effect of the policies on the file open response time when client caching is used. We look at the read response time to ensure that it is not adversely affected by the cache writeback policies.

We also look at the utilization of resources in the system to identify the bottleneck resource and to understand the capacity of the system to support a large number of clients. Since the server CPU is involved in the execution of different file system operations, including file opens and possibly the resultant cache consistency mechanisms, handling network traffic and moving data, it has often been the bottleneck in the past [9,14]. With the increasing use of client caches and faster server CPUs, we anticipate the server disk subsystem to be the future bottleneck. We examine the utilization of the server CPU and disks while evaluating the different policies.

Finally, in order to understand the different caching policies and the reason for their effectiveness, we also look at the actual numbers of different operations going from the client to the server and at the server to its disks. For example, write response time is seriously impacted if a

write operation finds the NV write cache to be completely full with dirty blocks. The write cache has to ''immediately'' replace one dirty block to make room for the incoming block. Consequently, we present the number of these ''immediate'' write operations wherever applicable. In a distributed file system, the client-server interactions are not just affected by the client/user workload, but are also affected by the mechanisms used to maintain consistency of data in the distributed environment. We therefore look at the contribution of the consistency mechanisms in detail by examining the actual number of reads and writes to the server, the cache hit ratios at the server and client and, in particular, the number of server disk operations for the different policies.

4 WORKLOAD AND PARAMETERIZATION

In this section we describe the characteristics of the workload derived primarily from the analysis of file I/O activity observed at two operational, commercial sites. We attempt to arrive at a synthetic model for the file system characteristics to use in the simulation of the distributed file system. In particular, we arrive at a model for the file access behavior, read/write characteristics, and find distributional fits for the file open-close times and file sizes. The traces that we use to come up with a synthetic model were collected by instrumenting the I/O processing routines of the operating system. The instrumentation captured every I/O request and recorded detailed information about the system, storage device, file and process identification and disk block request information. A detailed description of the tracing mechanism, the traces and the characteristics of file I/O activity can be found in [4,15].

The functionality offered by a distributed file service is often identical to that offered at the local file system in order to allow applications to use it transparently. We extrapolate the results of the analysis of the traces from two time sharing environments for use in the simulation of the distributed file system:

1. Scientific and Office (*Sci-Off*): This trace was collected at a Fortune 100 chemical company site and the workload constituted a mixture of scientific and office automation applications. The system was a VAXcluster with four processors (15 SPECmarks) and 19 disk drives. The trace data analyzed was from 7 a.m. to 5 p.m., the usual work day for this site.
2. Office Applications and Decision Support (*Off-Day*): The trace was collected at the headquarters of a large Fortune 100 corporation. The installation was a VAXcluster system of 3 processors (~20 SPECmarks) with 51 disk drives. There were about 100 active users primarily running office applications and decision support software. Three consecutive days were traced. In this study we used a trace from the ''prime time'' (7 a.m. to 5 p.m.) period of one day as the access pattern was found to be similar across the three days [15].

The environment we consider here is of clients that access a file server for mass storage. Although it is common for clients to have local storage, particularly for maintaining the operating system executables, we assume here that the local disks are available for paging and swapping only. All access to data and operating system executables are done from the file server. This allows us to translate the file system trace data directly to the distributed environment.

4.1 File Access Characteristics

It is important to distinguish files into different categories that have distinct access and sharing properties. We show that the way a file is opened and the way the file is organized determines many of the important access characteristics.

We first examine the trace data by categorizing the files by the way they are opened. Each file is classified into one of the following three file open modes:

- Opened for read only access (R_ONLY)
- Opened for write only access (W_ONLY)
- Opened for both read and write access (RW)

Figure 4.1 shows that the active files are almost evenly distributed among the three file open mode categories for the traces used in this study.

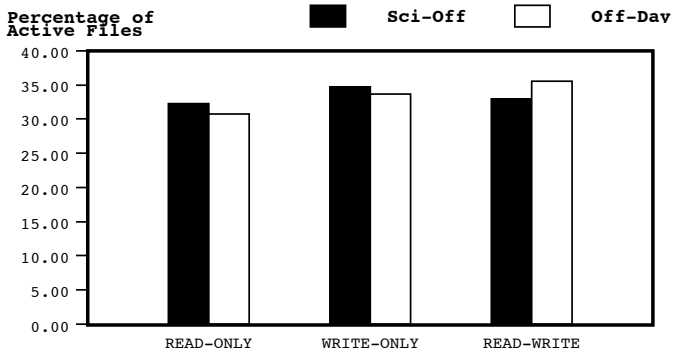


Figure 4.1: Distribution of active files in open-mode categories

When a file is accessed by two or more different clients which do not concurrently access the file at any time during the trace interval, the file is said to be sequentially shared. If multiple clients open a file at the same time, then that file is simultaneously/concurrently shared. Our method of identifying the different clients that open a file at any time during the trace interval captures both simultaneous and sequential sharing. We also observe that the sharing characteristics of a file, an

important parameter in the distributed environment, is strongly dependent on its open mode.

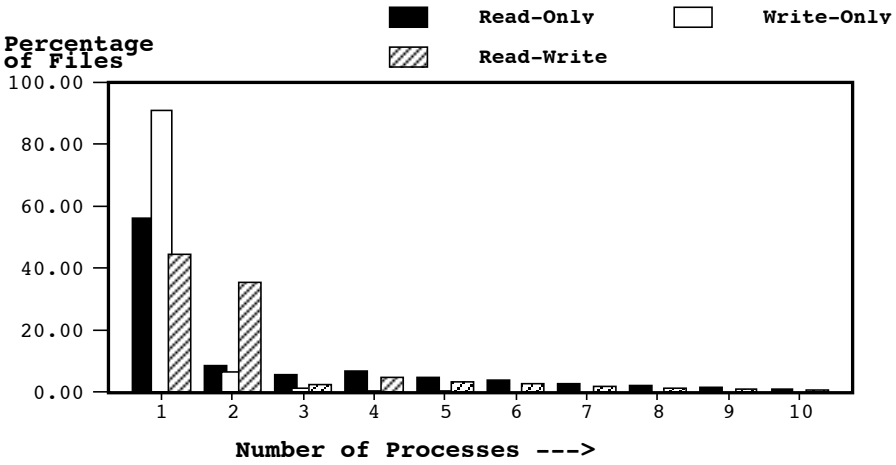


Figure 4.2: Sharing Distribution for the Sci-Off trace

From Figures 4.2 and 4.3 we observe, across both traces, that the files opened for write-only (W-ONLY) are least likely to be shared. The files opened for both read and write (RW) are shared the most but are predominantly shared between two processes. The files opened for read-only (R-ONLY) have characteristics that lie in-between those of W-ONLY and RW files.

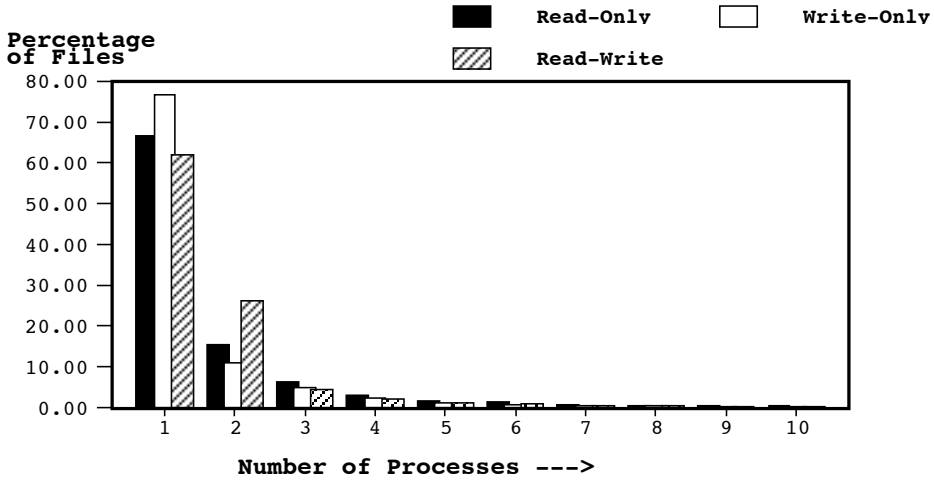


Figure 4.3: Sharing Distribution for the Off-Day trace

One could classify the active files from the traces in another way based on the way the blocks in the file are logically organized (accessed). These typically are:

- sequential files, and

- random access files.

	Sci-Off		Off-Day	
	Sequential	Random Access	Sequential	Random Access
Percent of Active Files	91.86%	8.14%	87.16%	13.18%
Mean File Size (Kbytes)	31.75	108.46	65.52	148.26
Mean Opens/File	9.71	32.25	6.37	10.01
Mean Reads/File	84.37	386.69	64.49	85.64
Mean Writes/File	21.10	57.02	22.63	34.53

Table 4.1: File Characteristics for Sequential and Random Access Files

From our traces, we observe that the file organization has a strong influence on the file access pattern. The file size distribution and the intensity of access (open/read/write operations per sec) are also dependent on the file organization. We see that approximately 90% of the active files are sequentially accessed (Table 4.1), and the remaining are randomly accessed. On the other hand, random access files are, on an average, about three times larger in size than sequential files and account for a larger amount of file system activity.

4.2 Workload Parameters

A typical client interaction with the server involves a file open, several reads or writes and a file close. The number of reads and/or writes to a file depends on its access characteristics (sequential or random) and it's size. We look at the different workload components, such as the temporal characteristics of the file open-close information and the characteristics of the open operations. We then estimate the read-write operations within an open-close session. We also look at the file size characteristics within the 3 different categories of files (R_ONLY, W_ONLY and RW). We find distributional fits for the open-close times as well as for the file sizes, and find that a multi-stage hyper-exponential fit is quite good. We believe that these distributions are of value in their own right, as these synthetic models may be relatively easily used in other studies.

The number of active users (each of which is represented in our simulation as a distinct client) was obtained by identifying login sessions of distinct users. The ratio of the number of active users to the total number of users is found to be in the 1:5 to 1:6 range. This was calculated by taking the ratio of the average number of distinct users in a 1-hour interval to the total number of distinct users observed in a prime time period. The prime time period was 9 hours (8 a.m. to 5 p.m.) and 10 hours (7 a.m. to 5 p.m.) for the Sci-Off and Off-Day scenarios respectively. The number of active users for the environment was then used to derive several of the workload pa-

rameters, knowing the total number of users seen in the trace.

Inter-Open Time: The inter-open time is defined to be the interval of time between successive file opens by one user. The mean inter-open time is calculated from the the number of active users found in the trace, the total number of opens and the length of the trace interval. All of these measures were available from the file I/O trace data. We calculate the mean inter-open time using the following formula.

 # opens/sec per user = ((total# opens)/(total trace time)) * (1/# active users)

mean inter-open time = (1/#opens/sec per user)

We assume that the inter-open time is exponentially distributed, with the above mean.

Open-Close Time: The statistics of the time between when a file is opened and closed is derived from the trace data. For example, for the Sci-Off trace we observe that about 90% of the open-close intervals are less than 5 seconds in length but that the remaining 10% of the open-close intervals are distributed over a wide range resulting in a large coefficient of variation. We first attempted to fit a simple two stage hyper-exponential distribution to the open-close time distribution derived from the trace for use as a synthetic model in the simulation. The 2-stage hyper-exponential distribution has two parallel stages of exponential distributions of different means, and a probability of branching to either of these stages as shown in Figure 4.4.

For the 2-stage hyper-exponential model, we attempt to fit to the data by varying the branching probability *p*. The mean values for each of the stages need to be adjusted for each value of *p*, so that we match these values to the mean and variance of the original distribution.

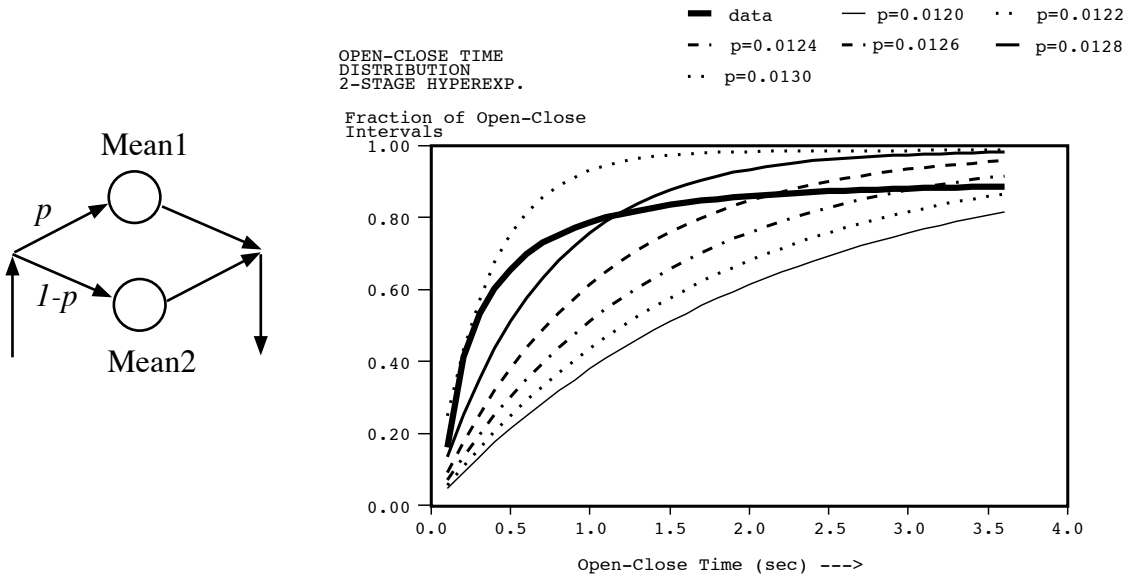
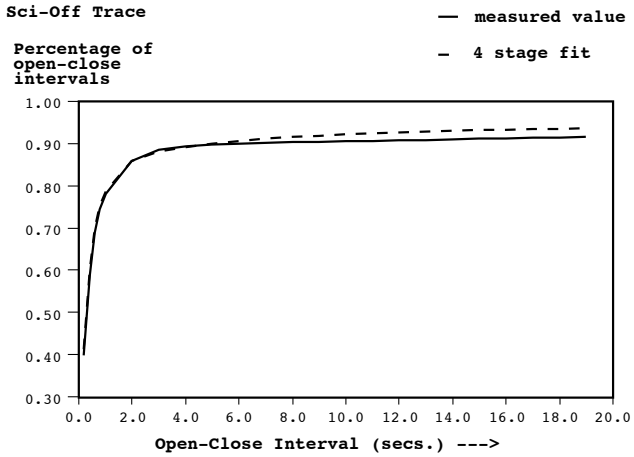


Figure 4.4: A two-stage hyper-exponential curve-fit: for Open-Close time distribution.

Different values of *p* and the mean values of the two stages give us a family of curves shown in

Figure 4.4, that fit the values for the mean and variance of the trace data. We use the least-square estimate to find the best fit for the different values of *p* used, and use the Kolmogorov-Smirnov test to ensure that the confidence level is over 95% for the curve fit. More details for such curve fits may be found in [8].



Stage	Mean (sec.)	Probability
1	0.232	0.593
2	0.988	0.306
3	77.5	0.092
4	3975.7	0.009

Figure 4.5: Open-Close time distribution with 4-stage hyper-exponential fit

We observed that the 2-stage hyper-exponential distribution is not sufficiently close to the trace data for any value of *p* chosen. We then used a 4-stage hyper-exponential distribution, which provides a much better fit. Figure 4.5 shows that we obtain an accurate fit for the most dominant portion of the data. Furthermore, the error in the values for the outliers between the measured and fitted distributions never exceeds 2%. The mean of each of the different stages and the probability of going to each stage is shown in Figure 4.5.

Open Characteristics: We observed in Section 4.1 that the organization and file open-mode has a significant influence on the sharing and activity pattern of active files. Therefore, when a user opens a file we use the following three important characteristics to determine the amount and

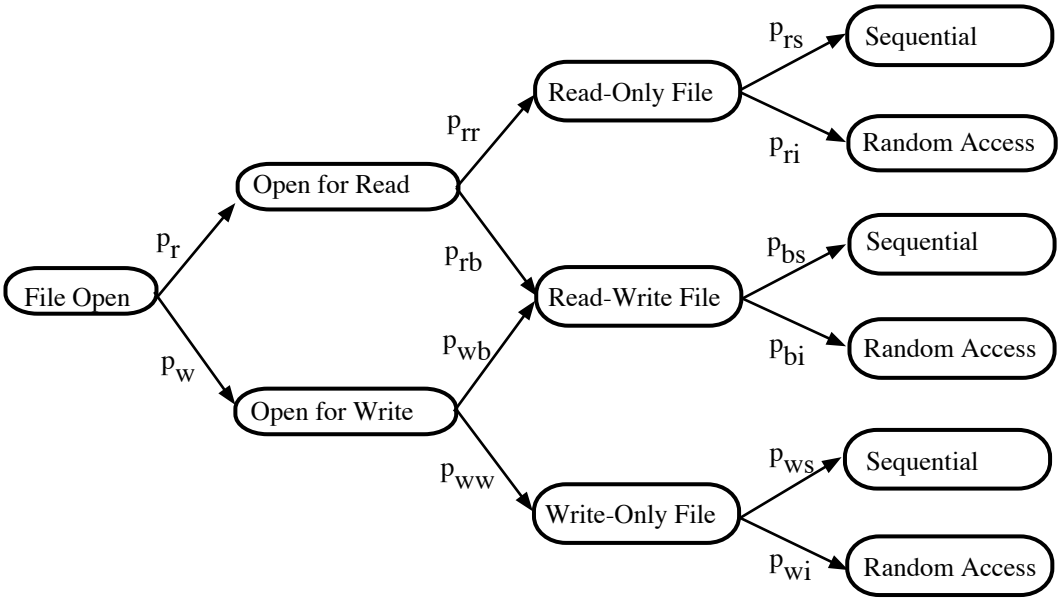
pattern of access to this file. These factors are:

			%Active Files	%Open Request	File Size (Kbytes)
Sci-Off	Read-Only	Sequential	31.76	50.58	17.5
		Random Access	0.55	0.87	86.9
	Write-Only	Sequential	32.49	4.43	66.8
		Random Access	2.27	13.59	60.5
	Read-Write	Sequential	27.61	22.26	6.9
		Random Access	5.32	8.27	131.1
Off-Day	Read-Only	Sequential	28.28	48.78	102.6
		Random Access	2.47	2.89	35.9
	Write-Only	Sequential	23.62	9.23	91.2
		Random Access	10.04	4.07	80.9
	Read-Write	Sequential	35.26	23.19	18.3
		Random Access	0.34	11.84	2991.5

Table 4.2: Percentage of Files, Opens and Average File Size for the six file categories

1. The file open mode -- i.e., whether the file is open for reading or writing: This is given by the probabilities p_r and p_w shown in Figure 4.6 We observed that about 75% of all opens were done with the intent to read a file.
2. The ‘‘open-mode’’ category of the file being opened - whether the file is a R-ONLY, W_ONLY, or RW file (as described in section 4.1). Clearly an open for read cannot be directed to a file that belongs to the W-ONLY category and similarly a R-ONLY file cannot be opened for write. We therefore evaluate the conditional probabilities p_{rr}, p_{rb}, p_{wb}, and p_{ww} as shown in Figure 4.6 for use in our distributed environment simulation. For example, about 68-70% of open for reads are done on R-ONLY files whereas about 55-67% of the opens for write are directed to W-ONLY files.
3. The file organization -- i.e., whether the file opened is a sequential or random access file. It was noted in Table 4.1 that about 87-92% of the active files are sequential files, but we observe that this ratio doesn’t hold across the open-mode file categories. For both the traces we use in this study, we find that the R-ONLY files predominantly tend to be sequential files while the RW files in the Sci-Off trace and the W-ONLY files in the Off-Day trace had a

relatively larger proportion of random access files (Table 4.2).



	p_r	p_{rr}	p_{ww}	p_{rs}	p_{bs}	p_{ws}
Sci-Off Trace	0.731	0.704	0.668	0.983	0.246	0.729
Off-Day Trace	0.758	0.682	0.549	0.944	0.694	0.662

Figure 4.6: File Open Characteristics

NOTE: Probabilities that are not shown in the above table can be calculated. For example, $p_w = 1 - p_r$, $p_{rb} = 1 - p_{rr}$, etc.

Read-Write Characteristics: The way a file is read or written depends on whether the file is a sequential access or random access file. The total number of blocks accessed also depends on the file organization. The only exception we observed is that small files (< 48 Kbytes in our case), irrespective of their file organization, are usually accessed in their entirety every time they are opened.

When a sequential file is opened, we assume that 80% of the time the entire file is accessed [2]. The rest of the time, we assume that on an average 50% of the file is accessed. The fraction of the file accessed when the whole file is not accessed is drawn from a uniform distribution between 0 and 1. The starting point of access within the file is assumed to be randomly distributed, between the start of the file and a point such that the fraction accessed is within the file.

When a random access file is opened for reading we assume that the number of file blocks accessed is 40% of the file size. Random access files are usually traversed using some form of an index structure and the file blocks containing the indices are more frequently read. Therefore we assume that 20% of the file is ‘‘hotter’’ than the rest of the file and takes 80% of the file read operations. From our analysis of the trace data, write operations to random access files were found to be significantly less frequent and also to not display the skew in the access pattern that

was observed for reads. This is because the index blocks, which are read frequently, do not have to be updated as often. We match the amount of writes to a random access file within an open-close session to that observed in the trace data. Based on this, we use a mean value of 5% for the fraction of the random access file written in an average file activity session. The number of blocks written in a particular session is derived using a truncated exponential distribution and the blocks accessed are uniformly distributed across the file.

We model the reads and writes to files as requests from the client for 4K byte blocks. The number of read or write operations is determined by how much of the file is accessed.

Number of read/write operations = (size of file * fraction of file accessed)/ 4K bytes.

File Size Characteristics: As described above, the size of the file is an important workload parameter. We attempt to find a synthetic model for the distribution of the file sizes from our trace data. There are significant differences between the file size distributions for the three different characteristics of files (R_ONLY, W_ONLY and RW files).

		Mean File Size(KB)	Std. Dev. (KB)
Sci-Off	Read-Only	18.6	293.8
	Write-Only	66.4	4610.2
	Read-Write	27.0	261.8
Off-Day	Read-Only	97.6	1870.3
	Write-Only	88.2	1998.4
	Read-Write	46.3	2574.7

Table 4.3: Mean and Standard Deviation for overall File Size

Just observing the mean and standard deviation for the file sizes for the Sci-Off and Off-Day traces of the 3 categories in Table 4.3 indicates that it is appropriate to consider the files of the different categories separately. Just as with the open-close times, we observed that there is a long tail in the distribution for the file sizes. This is also reflected by the fact that the standard deviation is significantly larger than the mean in Table 4.3. To perform a curve-fit (e.g., Sci-Off trace), we divide the file size data into two groups. The first contains the first 95 percentile of files by size (smallest 95%) and the second the top 5 percentile of files by size (largest 5%). We first attempted to fit a 2-stage hyper-exponential distribution for the file sizes to each of these categories. Figures 4.7(a) and 4.7(b) show the 2-stage hyper-exponential fit for the W_ONLY files for the smaller 95% of the files (less than 32 KBytes) and the top 5 percentile of files, which are larger than 32 Kbytes, respectively. We see that the 2-stage fit is good for the smaller 95% of the files, but not as good for the larger files. We then use these to obtain an approximate 4-stage hyper-exponential fit for the entire set of files, by a direct combination of the two 2-stage fits. The 4-stage fit is shown in Figure 4.8, and appears to be quite reasonable. The primary reason for

this may be that the fit for the dominant 95% of the files is quite good and that this dominates the overall distribution. We also show in Figures 4.9 the overall 4-stage fits for the R_ONLY and RW files for the Sci-Off traces respectively.

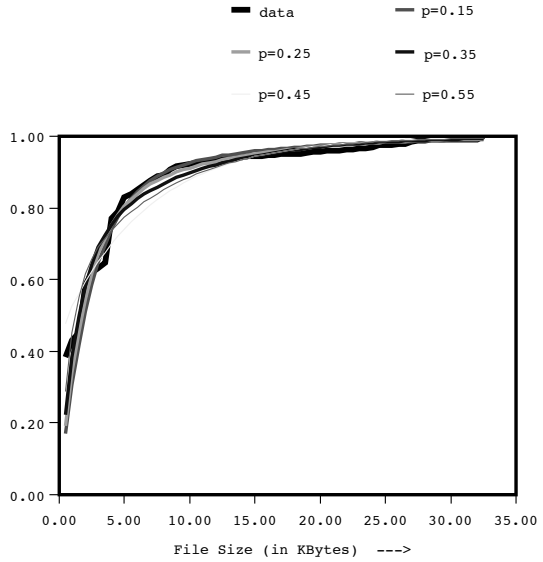


Figure 4.7(a): Sci-Off Trace, W_ONLY files, 95% of the files (files < 32 KB)

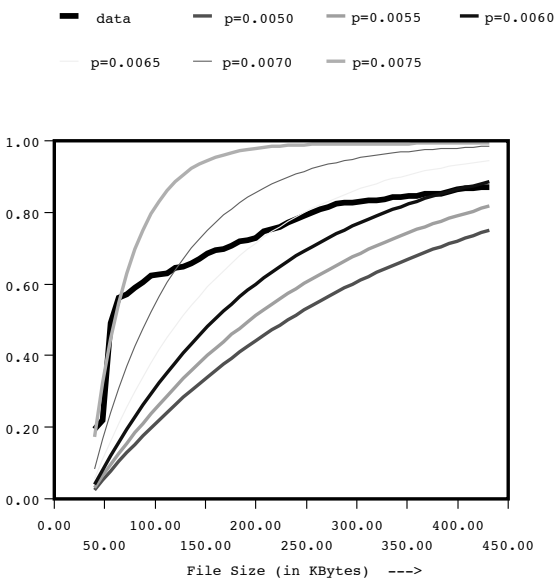


Figure 4.7(b): Sci-Off Trace, W_ONLY files, top 5% of the files(files > 32 KB)

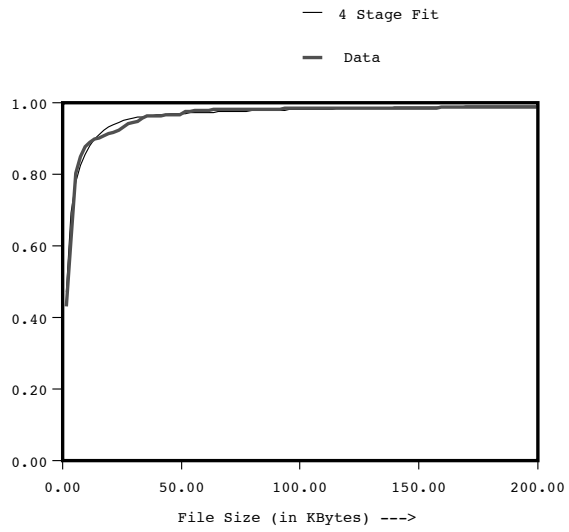


Figure 4.8: Sci-Off Trace, W_ONLY files, combined 4 stage fit for all data.

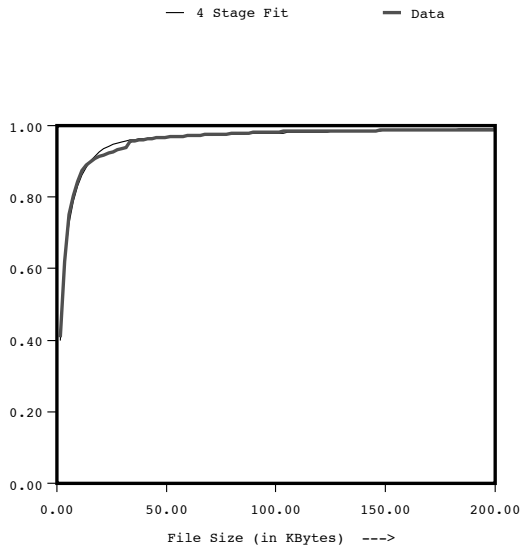


Figure 4.9: Sci-Off Trace, R_ONLY files, combined 4 stage fit for all data.

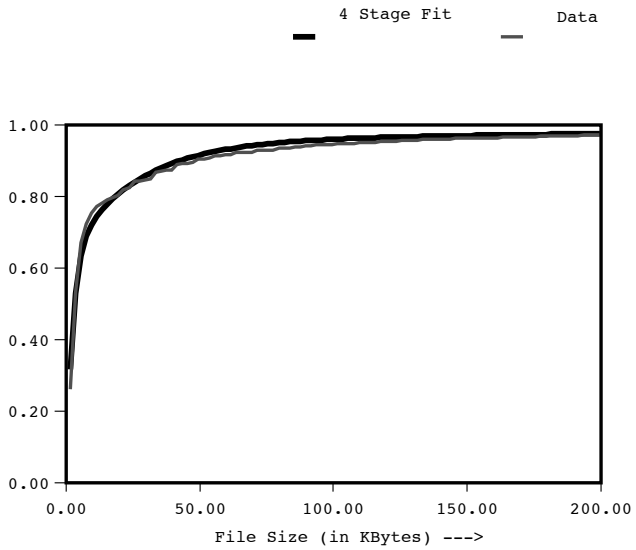


Figure 4.10: Sci-Off Trace, RW files, combined 4 stage fit for all data.

5 SIMULATION METHODOLOGY

We use the CSIM simulation package [16] to model a distributed file system consisting of a number of active client workstations or personal computers accessing files from a single file server. The location of a file in the file system is typically determined using a global name service and we assume that the name server is co-located with the file server. The client systems are 24 SPECmark [13] rated systems with a volatile in-memory file block cache. In addition, the clients may also have a non-volatile (NV) write cache. The server machine is also a 24 SPECmark system with 5 disk drives. The active files are assumed to be uniformly distributed over the five disks. A 5-disk configuration is chosen to produce a well balanced system. The server also has a volatile file block cache and an optional NV write cache. The clients and servers are connected via an Ethernet based local area network.

5.1 Distributed Environment Model

The file server is modeled in detail with the processing unit (CPU) and the disks each represented by a separate queue and server. In addition, the file system caches at the server are also modeled. At the client, only the CPU and file system caches are modeled. The queueing discipline at the CPU at both the server and the clients is first-come-first-serve (FCFS). The CPU service time includes the cost of file system operations like, open, close, read and write; processing cost for handling disk I/O and for performing network operations. These service times are derived from measurements on a RISC Ultrix workstation (Table 5.1). The CPU service time is assumed to be the same at both the client and the server. The service time for the disk is computed from the specifications of the device (Table 5.1). The service times for the file system cache replacement and consistency operations are also accounted for.

For this study the network is modeled as a single server queue with FCFS queuing discipline. We assume just two message size categories - small 64 byte messages for control operations and 4 Kbyte messages for data movement operations between the clients and the server. All the clients and the file server are assumed to be on a single 10 Mbit/second Ethernet segment. Studies have shown that this simple model may be acceptable if the network is not the bottleneck. The client and server CPUs are involved in processing the messages and we account for the CPU service times for protocol processing and network I/O. The service time for the server representing the network is calculated based on the message size (Table 5.1).

Operation	CPU Service Time (msec)
File Open	3.0
File Close	1.0
File Read/Write	2.0
Disk Read I/O	1.0
Disk Write I/O	1.5
64 byte network I/O	0.4
4 Kbyte network I/O	1.5

Operation	Disk Service Time (msec)
Seek Time	16.0
Rotational Latency	8.0
Xfer time for 4 KByte	2.5

Message Size	Network Delay (msec)
64 bytes	0.05
4 KBytes	3.28

Table 5.1: Service times used in the simulation

5.2 File System Setup

Prior to the simulation, the file system is initialized by the following procedure:

Step 1: The number of active files accessed is calculated using the following relationship:

#Active Files = (#Active Files in Trace data / # Users in trace)*NCLIENTS

NCLIENTS, the number of users, is an input parameter to the simulation.

Step 2: We use the ratio of the number of sequential to random access files observed in the trace data (Table 4.1) to assign a file organization attribute to each active file.

Step 3: Files have specific access properties. Some are only accessed for reading (R_ONLY), others for writing only (W_ONLY), while the rest are both read and written (RW). We observed that the distribution of files in these three categories depends on the file organization (sequential or random access). The proportion of files in these three access categories is known for the two file organizations from the traces (Table 4.2) and is used to predetermine the access attribute for each file.

Step 4: Each file is then initialized to a certain size depending on its file open mode and organization. Table 4.2 shows the average file size for the different file categories observed in the file system traces. We use exponential distributions with the given means to derive the file sizes. The

rigorous analysis of the trace data shows that the data is better described by a multistage hyper-exponential distribution. However, this distribution exhibits a long tail because there are a small number of very large files. In the multistage distribution, there is a corresponding stage with a very large mean (and a very small probability of going to that stage). As the file systems we simulate contain only a few thousand (up to a maximum of 25000) active files, the stage that contributed to the outliers has very few files generated by it. For generating reproducible results in our simulation, we needed a reasonable number of files in each stage, thus requiring the environment to have a very large number of files. To keep the processing time and memory requirements for the simulation within reasonable bounds, we used the simpler model of an exponential distribution for the file sizes.

Step 5: We observed in section 4.1 that the sharing characteristics varies significantly between files with different access properties (R_ONLY, W_ONLY, or RW). Hence, the number of clients that access a file is determined by the appropriate file sharing distribution (Figure 4.2 and 4.3). For a majority of the files the sharing characteristics (number of clients accessing file) is independent of the total number of clients. However, for a small set of files, the tail of the distribution (the number of clients accessing the shared file) increases with increasing client population. This effect was captured in the empirical sharing distributions used. The specific clients that access a file are chosen by randomly selecting the number of clients determined by the sharing distribution. Once the identity of clients that access a file is determined, that file is put in the file list at that client. Ideally there are six file lists at each client based on the three open modes and two file organization types. In our simulation we attempted to merge some of these lists. For example, we observed in the Sci-Off environment that there were very few files in the random access R-ONLY file category. Therefore we merged that list with the sequential R-ONLY list to have just one R-ONLY list at each client.

This completes the initialization of the file system. Each file has a specific organization, a size and an open-mode category.

5.3 Client-Server Operations

A typical client interaction with the server involves a file open, several reads or writes and a close. We shall refer to all of the activity between the execution of an open and a close on a file as an ‘‘activity session’’. The user at the client system is assumed to ‘‘think’’ for a certain amount of time between each file open request. It is also quite likely that a client may have multiple files open simultaneously. Therefore, we model each file open-close activity session at a client as an independent thread. A pictorial view of a client’s operations on files is shown in Figure 5.1. There is usually a think time between each read or write operation which may be attributed to either processing on the client or user ‘‘think time’’. We choose to represent the workload as a sequence of read or write operations with no processing on the client and have an aggregate think time at the completion of the reads and writes before closing the file. In between successive open requests, the client is assumed to spend another think time. The client then initiates an open-close activity session which is modeled as another independent thread. The last

think time, the inter-open time between the start of each file open-close activity sessions, is fairly large with a mean of 17.5 seconds for the Sci-Off trace. In addition, the time between the last read/write activity and the closing of the file is also quite large, with a mean of 42.5 seconds. In comparison, the total amount of time taken for the file open, all of the reads and writes and the close operation to complete, in an open-close activity session, averaged only about 1.8 seconds (which includes all the queueing time, network delay, etc.) for 700 clients in the baseline case. Therefore, an individual client presents a relatively light load on the server and we observe that a single server is able to support a large number of clients before a bottleneck resource in the server is saturated.

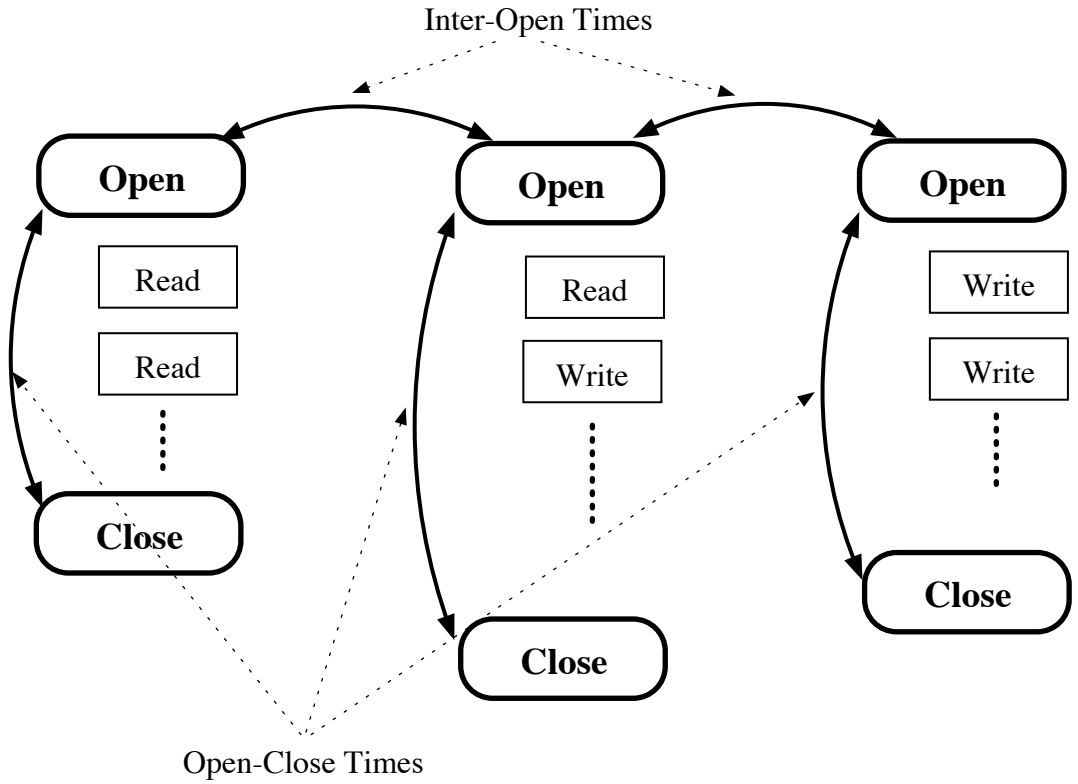


Figure 5.1: Model of Open-Close Transactions at a Client

The client initiates activity by first opening a file. First, we determine whether the open request is for reading or writing the file based on the probabilities shown in Figure 4.5. A very simple file locality model is chosen. We have different probabilities of accessing the file categories. If a file is to be read, we select a file from either the R-ONLY or RW pool for that client with probabilities ($\mathbf{p_{rr}}$ and $\mathbf{p_{rb}}$ respectively) specified in Figure 4.5. Similarly, a file to be opened for writing is chosen from the W-ONLY or RW pools with the specified probabilities ($\mathbf{p_{ww}}$ or $\mathbf{p_{wb}}$). Files are chosen randomly from within each pool. The client sends the open request to the file server that owns the file. We account for the service times on the client CPU for handling the open and for generating the network message.

At the server, first the file name is resolved to a file identifier. In this study, we assume the name service and file service to be collocated. The CPU service time for the name lookup is accounted for. We also assume 20% of the lookups result in visits to the disk with the name table information.

First we describe the operations at the server for a file that is not shared by any other clients. The server updates the file meta-data information of the file, records the client id and the mode (read or write) in which the file is opened. Again we assume that 20% of the opens result in a disk access to the meta-data information. The CPU cost for file open are presented in Table 5.1.

Let us next consider the case where the file may be shared among multiple clients. If the clients use some form of cache writeback mechanism then the possibility of data inconsistency at the client caches arises. There are two different sharing cases to consider:

(a) *Sequential Shared Files*: Consider the case where another client had previously opened, updated, and closed the file. The server always tracks the last writer to have closed the file. On encountering an open from a client, the server sends a call back request to the last writer to flush all dirty blocks of this file. The open operation blocks till the last writer has flushed all dirty blocks and the operations have been handled at the server. The server then responds to the client with the current version number of the file. In the case where the version number received from the server is higher than the version number at the client, the client removes all of the blocks belonging to this file from its cache before signaling completion of the open request to the application. We account for the CPU, network, and possible disk overhead at the server, the last writer, and the requesting client for these cache consistency operations.

(b) *Simultaneously Shared Files*: If one or more clients are found in the list of clients that already have the file open, then the server checks if the current open causes the file to become ''write shared'' i.e. a sharing mode in which one or more clients are updating the file. There are two ways in which an open causes the file to become ''write shared'':

1. the file is already opened by another client for write.
2. the file is being read by other clients while the current open is for write.

If the file becomes ''write shared'' as a result of an open, then cache consistency actions have to be taken. Clients which already have the file open are asked to flush their read and write caches and caching is disabled for the file. The open from the requesting client is complete at this point and all subsequent reads and writes requests are sent to the server. We assume UNIX like semantics for write operations from these multiple clients.

After receiving the successful completion of the open from the server, the client reads or writes the file in the fashion described in the workload section. For a read request, first the read cache at the client is searched for the requested file block. If the read is not resolved there, then the NV write cache is searched, if one is present. If the block is found in the write cache and it is found to be ''clean'' (i.e. it has been committed to the server), then it is removed from the write cache and inserted into the read cache. Blocks are maintained in Least Recently Used (LRU) order in both the caches. If a requested block is not found in either of the client caches, then the read

request is forwarded to the file server.

A write request is handled in a slightly different fashion. When a file block is modified, it is inserted into the NV write cache at the client. An older version of the block, if present, is removed. If the written block was previously read into the read cache, then it is removed from there at this time. This is done because the block is available for reads in the write cache. We do not assume any differences in the access speed of the volatile and non-volatile caches. If the write cache is full, then the LRU block is replaced by the new block written. If the LRU block is ''dirty'', then it has to be written back to the server. The caches at the server operate exactly in the same way as the client caches. The only difference is that when a dirty block is removed from the server write cache, it is sent to the appropriate disk.

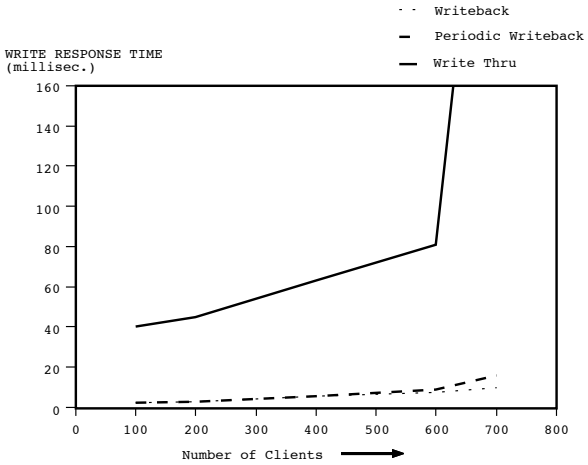
In our simulation, we ensure that all shared data structures, such as the file meta-data at the server and all the caches at the server, are locked before they are updated. This is also true for accesses by the client to its local cache when a file is being flushed to the server. Therefore, we model the delays encountered by client operations as a result of lock conflicts. It must also be noted that we have not modeled file create and delete operations and have assumed that the clients have local paging devices.

6 RESULTS

We examine alternative policies for managing the NV cache at the client and server and quantify the actual performance improvement achieved by including NV caches. Further, we examine the sensitivity to variation in size of the NV cache. We base our results on a simulation of the environment described in Section 5, for different numbers of clients in the range 100 to 700 clients. We do not go beyond 700 clients because the server resources are already heavily utilized. All of our studies are based on a single file server supporting all the clients.

Our baseline configuration is a system in where each client has 8 MBytes and the server has 32 MBytes of volatile cache. The best case, without regard to vulnerability of the data in the cache, is to provide a writeback policy at all the caches. The writeback policy in the baseline case uses an LRU list for all the blocks in the cache. When a dirty block is replaced at the client, the block is written back to the server. When a dirty block is replaced at the server, that block is written to the disk. We shall refer to this as the ''Writeback'' policy. We compare the baseline ''Writeback'' policy with a ''Periodic Writeback'' policy where the caches are checked every 5 seconds, and any cache blocks that are dirty for more than 30 seconds are written back. This form of ''Periodic Writeback'' policy has been used in the Sprite distributed file system. The write response time is shown in Figure 6.1 for a client population varying from 100 to 700 clients. We also compare, these policies with the ''Write-thru'' scheme where all writes are committed to the server's disk before an application's write is completed. Figure 6.1 shows that the Write-thru performance is not only poor even for low loads (since all writes suffer the overhead of going to the server's disk), but also degrades dramatically at high loads due to saturation of the server's disk subsystem. The two writeback policies significantly reduce write response time and the server's I/O subsystem is not saturated even at 700 clients (Figure 6.1). While the performance

of the writeback mechanisms are good, the problem of the vulnerability of the data in the volatile caches still remains. Therefore, when either the client, the network, or the server fails, updates to the data may be lost, resulting potentially in inconsistencies in the file system. We examine the benefits of introducing non-volatile (NV) write caches. Because non-volatile memory is still quite expensive [1,5] we investigate the performance of different algorithms to efficiently manage the write caches. In addition, we estimate the minimum size of the write cache required for the scientific and office environments that the synthetic workload represents, to achieve good overall performance.



	Avg. Server System Disk Utilization(%)		
	100 clients	400 clients	700 clients
Writeback	7.9	20.2	56.4
Periodic Writeback	12.2	41.5	68.8
Writethru	12.3	43.8	79.2

Figure 6.1: Comparison of average write response times and server system disk utilization for Writeback, Periodic Writeback and Writethru cache policies.
Client Caches = 8 Mbytes, Server Cache = 32 Mbytes.

6.1 Impact of the NV Write Cache at the Server

We begin with the introduction of an additional NV cache at the server alone, while maintaining the same amount of volatile cache at the client and server. We compare three different server NV write cache writeback policies:

1. ‘‘Periodic’’ writeback,
2. ‘‘Threshold’’ writeback, and
3. ‘‘LRU’’ writeback.

The volatile cache on the client uses a ‘‘Write-thru’’ policy, and the volatile cache on the server is used only for reads, with a simple LRU replacement policy. The server NV cache is assumed to have a capacity of 8 Mbytes. Under the ‘‘Periodic’’ writeback, we use an aging time of 30 seconds for replacement. With ‘‘Threshold’’ writeback, we use a Hi-Lim value of 90% and a Lo-Lim value of 50%. We will motivate these choices of thresholds in the next subsection. The interaction between the volatile read-cache and the NV write cache at the server was described in Section 2.2.

The variation in the average write response time with the number of clients for these three different policies is shown in Figure 6.2. The simple ‘‘LRU’’ writeback policy for the server NV cache performs relatively poorly at all loads compared to the other two policies because of a sub-

stantial number of ‘‘immediate’’ writes to the disk. These immediate writes occur as a result of the NV cache being full of dirty blocks. We observe two points of inflection for the simple ‘‘LRU’’ writeback policy, one at a load of 200 clients, the other at a load of 600 clients. In the 100 to 200 client range, the increase in the average write response time is primarily due to an increase in the number of immediate writes from a full write cache. Between 200 and 600 clients, the average write response time remains relatively flat. This is because there is substantial sharing which results in a higher number of over-writes (‘‘write hits’’ on a dirty block in the NV cache). Beyond 600 clients, the server’s CPU and disk becomes heavily utilized and this increases the average write response time (Table 6.2).

When we consider the ‘‘Periodic’’ writeback policy, the average write response time is substantially smaller at light loads compared to the ‘‘LRU’’ writeback policy. However, at 700 clients, the average write response time with the ‘‘Periodic’’ writeback policy is only marginally better than with the ‘‘LRU’’ writeback policy. This is because the server becomes heavily utilized due to the large number of disk writes generated by the ‘‘Periodic’’ writeback policy. In fact, across the entire range of 100 to 700 clients, the ‘‘Periodic’’ writeback policy produces a larger number of writes to the server disk than the ‘‘LRU’’ writeback policy. However, because a large number of these writes to the server disk are not synchronized with the user application’s write operation, the write response time seen by the application is lower.

Finally, when we look at the ‘‘Threshold’’ based writeback policy for the server NV cache, the average write response time is even lower. The ‘‘Threshold’’ policy generates fewer writes to disk than the ‘‘Periodic’’ policy and the number of writes to the server disk with the threshold policy is only marginally larger than the ‘‘LRU’’ Writeback policy. But, since most writes to the server disk are not synchronous with the user application’s writes, we see a substantial reduction in the average write response time. In the case of 700 clients, the server CPU and disk are quite heavily used. This in combination with the limited size of the NV write cache, produces a relatively large number of immediate writes from a fully dirty NV write cache. As a result, the average write response time increases substantially between 600 and 700 clients. However, the percentage of ‘‘immediate’’ writes is always lower for the ‘‘Threshold’’ policy than for the

‘‘Periodic’’ case.

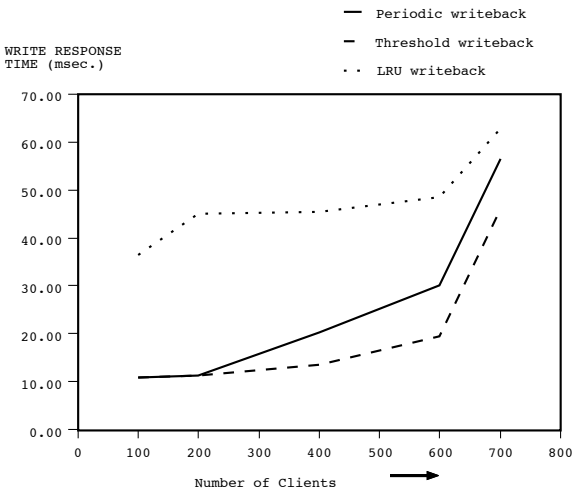


Figure 6.2: Average Write Response Time for three alternative NV writeback policies

	Avg. Read response (ms)		
	100 clients	400 clients	700 clients
Periodic	6.2	11.5	25.8
Threshold	6.2	11.2	25.5
LRU	6.2	11.5	23.9

Table 6.1: Average Read Response Time for three alternative NV writeback policies

An important consideration is that the application’s read response time should not be affected by using a sophisticated policy for writeback from the server NV cache. This is because these policies may send several writes to the disk to clean the write cache (e.g., when the threshold is crossed). Hence, the write traffic to the disk becomes bursty, and read operations that miss at the client may be queued at the server CPU or disk, behind such a burst of writes to the server disk. However, with the Threshold and Periodic policies, we see that at small loads (Table 6.1), the average read response time is not degraded at all (in comparison to an LRU policy). The impact is only marginal even at very high loads when the server CPU and disk are heavily utilized (Table 6.2).

	Avg. Server CPU Utilization (%)			Avg. Server System Disk Utilization (%)		
	100 clients	400 clients	700 clients	100 clients	400 clients	700 clients
Periodic	11.6	46.7	75.9	12.1	40.8	69.4
Threshold	11.2	46.1	75.2	11.5	40.2	67.5
LRU	9.5	41.3	72.2	11.3	40.1	67.0

Table 6.2: File Server CPU and System Disk Utilizations with server NV cache

6.2 Effect of Cache Writeback Policy Parameters

The policies for writeback from the server NV write cache include a number of parameters whose values affect performance. We study the effect of these parameters on performance when there are 100 clients and a 4 Mbyte server NV write cache. In the case of the periodic writeback policy, we have a daemon which checks the age of the blocks of the server NV cache every 5

seconds. If the age of a dirty block exceeds a threshold (time-out age), it is cleaned out of the cache. Table 6.3 shows the effect that the time-out age has on the average write response time. We see that the average write response time reduces as we increase the time-out age from 0 (implying that writing out all dirty blocks prevents us from benefiting from overwrites of dirty blocks in the write cache) until we are in the 10-30 second range. When the time-out value is chosen to be 100 seconds or more, the average write response time increases substantially. This is caused by a large number of ''immediate'' writes from a fully dirty write cache. We conclude from this that a 30 second time-out age value is reasonable and use it in the rest of this study.

	Time-out Age (sec)				
	0	**10**	**30**	**100**	**300**
Avg. Write Response Time (ms)	20.97	10.80	10.79	19.97	39.64

Table 6.3: Average Write Response Time for different Time-out Age values

The Threshold-based Write back policy uses a hysterisis policy with two thresholds for determining when to begin (Hi-Lim) writeback of dirty blocks and to stop the writeback (Lo-Lim). We had studied this previously in the case of a disk controller cache and observed that such a hysterisis policy reduced the number of writes to the disk substantially [3]. We study a similar variation of the thresholds here, by first holding the Lo-Lim (dirty blocks as a percentage of the total NV cache) at 30% of the NV write cache and varying Hi-Lim. Table 6.4 shows that the average write response time is insensitive to the choice of Hi-Lim. Looking at the secondary statistics such as the percentage of writes that go to the disk due to ''Immediate'' writes because of a fully dirty cache, we observe that a choice of Hi-Lim close to 100% is somewhat poor. On the other hand when Hi-Lim has a low value, we observe that the percentage of writes that go to the disk increase, indicating that the cache is being cleaned out prematurely, leaving a larger number of clean blocks in the NV cache than necessary. Therefore, a choice of Hi-Lim of 90% appears to be reasonable.

	Hi-Limit value		
	99%	**90%**	**70%**
Avg. Write Response Time (ms)	11.6	10.8	10.8
% Writes to Disk	90.2%	90.8%	92.2%
% Immediate Writes to Disk	0.99%	0.05%	0.01%

Table 6.4: Important Write Characteristics for different Hi-Lim values, Lo-Lim=30%

We next fixed the value of Hi-Lim at 90% and varied the value of Lo-Lim. For very low values of Lo-Lim, Table 6.5 shows that the number of writes to disk increases. A high value of Lo-Lim causes a slightly higher number of immediate writes to the disk. The tradeoff is similar to the one made above, and we believe that 50% is a reasonable choice for Lo-Lim. The important point to

make is that the average response time and even the secondary statistics are relatively insensitive to the threshold values. As shown in the earlier study [3], performance is insensitive to threshold values as long as Hi-Lim is not close to 100% and Lo-Lim is not too close to Hi-Lim. Thus, for the rest of this study, a value of Hi-Lim of 90% and a value of Lo-Lim of 50% are chosen.

	Lo-Limit value		
	70%	**50%**	**30%**
Avg. Write Response Time (ms)	10.9	10.8	10.8
% Writes to Disk	88.4%	89.7%	90.8%
% Immediate Writes to Disk	0.18%	0.01%	0.05%

Table 6.5: Important Write Characteristics for different Lo-Lim values, Hi-Lim=90%

6.3 Server NV Write Cache Size Variation

In this subsection, we investigate the effect of varying the server NV write cache size for both the threshold based and periodic writeback policies. In Figures 6.3-6.4 we plot the average write response time for different server NV cache sizes at three different client load levels. We observe that at 100 clients, 4 Mbytes of server NV cache is more than adequate and the average write response time does not improve with larger caches. At 400 clients, there is a small improvement in the average write response as the cache size is increased to 8 Mbytes but thereafter there is no improvement in the average write response time. At 700 clients, the average write response time decreases dramatically (almost 33%) when the server NV write cache is increased from 4 Mbytes to 8 Mbytes but thereafter, the average write response time decreases only slightly. We conclude that for this workload, an 8 Mbyte server NV write cache is adequate in the operational environment where the system resources are not heavily utilized. The behavior for the threshold and periodic policies are similar, except that the periodic policy exhibits a slightly higher average write response time.

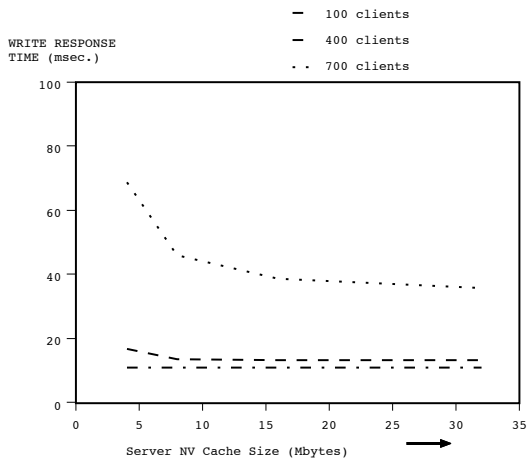


Figure 6.3: Server NV Cache Size Variation Threshold Policy

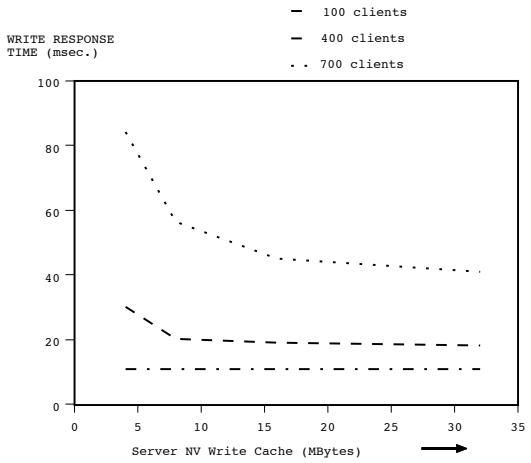


Figure 6.4: Server NV Cache Size Variation Periodic Policy

6.4 Impact of NV Write Cache at the Clients

We now introduce a small NV write cache at the client in addition to the 8 Mbyte server NV write cache. The size of the NV write cache at the client is kept at only 0.5 Mbytes. We compare the average write response time for this configuration with both the periodic and threshold-based writeback polices against that with no client NV write cache (Figure 6.5).We observe that the introduction of a client NV cache substantially reduces the average write response time. The degradation of average write response time with load is also more gradual when the client NV cache is used. The periodic writeback policy is better for small numbers of clients when the file server is lightly loaded (Table 6.7). But for large numbers of clients (600 and beyond) the threshold based policy is better. When a threshold based policy is used, dirty blocks are allowed to stay longer in the client caches and a large number of blocks are either overwritten or written back to the server by the cache consistency algorithms. This reduces the total number of writes to the server and consequently gets better performance.

When we use any form of writeback policy at the client it is necessary to introduce algorithms (as described in Section 2.2) to maintain data consistency at the clients and the server. Most of these cache consistency policies are invoked when a file is opened and therefore we need to ensure that the average file open time is not seriously impacted by the introduction of the client NV write cache. We see from Table 6.6 that the average file open time is larger for the periodic writeback policy than when no client write caching is used. For the threshold based writeback policy the average file open time is higher only at very low loads (100 clients). But, at higher loads (400 and 700 clients), the threshold based policy in fact reduces the mean file open time below that for the no-client cache case. This improvement is due to the reduced load placed on the file server by the threshold policy compared to the other two policies (Table 6.7).

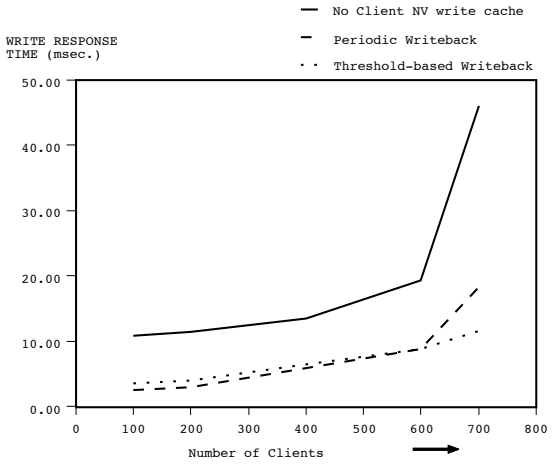


Figure 6.5: Impact of Client NV caching on the average Write Response Times

	Average Open response Time (ms)		
	100 clients	400 clients	700 clients
No client cache	51	67	119
Periodic	53	70	136
Threshold	54	67	97

Table 6.6: Impact of Client NV caching on the average File Open Response Times

	Avg. Server CPU Utilization (%)				Avg. Server System Disk Utilization (%)		
	100 clients	400 clients	700 clients		100 clients	400 clients	700 clients
No client cache	11.2	46.1	75.2		11.5	40.2	67.5
Periodic	11.9	48.0	77.3		12.2	41.6	68.0
Threshold	8.9	40.1	71.6		10.1	36.0	61.4

Table 6.7: File Server CPU and System Disk Utilizations with server and client NV caches

In Table 6.8 we look closely at the write traffic going from the clients to the server under the Threshold based and Periodic writeback policies. We consider the percentage of all the application writes that go to the server and further break down those writes going to the server into three categories - writes to maintain cache consistency, writes due to the invocation of the writeback policy (threshold or periodic), and immediate writes that are required when the client NV cache become completely full with dirty blocks and cannot accommodate an incoming dirty block. The threshold-based policy performs fewer overall writes to the server in comparison to the periodic writeback policy. The fraction of those writes due to consistency traffic is much higher for the threshold-based policy compared to the periodic policy. This indicates that the threshold policy adapts better to the dynamics of the workload and writes to the server only when it is required by the consistency mechanisms. However, the number of writes to the server because the client cache is completely full with dirty blocks (immediate writes) is larger with the threshold-based writeback policy than with the periodic writeback policy. The periodic writeback policy is continually writing dirty blocks out, potentially keeping more of the client cache clean and, therefore, can handle bursts of writes at the client. This impacts the average write response time for the threshold based writeback policy at low loads. But the significant reduction in the number of overall writes to the server with the threshold based writeback provides the advantage at higher loads (Figure 6.5) so that the average write response time is lower than under the periodic write-

back at 700 clients.

	Threshold Policy			Periodic Policy		
	100 clients	400 clients	700 clients	100 clients	400 clients	700 clients
% Writes to Server	59.77	73.26	79.56	99.04	99.47	99.67
% Writes to Server - Consistency	41.14	69.24	82.44	9.43	40.31	55.85
% Writes to Server - Threshold/Periodic	42.30	21.20	11.37	90.53	59.60	44.13
% Writes to Server - Immediate	16.56	9.56	6.19	0.04	0.09	0.02

Table 6.8: Percentage of writes from clients to the server and components of the write traffic

6.5 Client NV Write Cache Size Variation

Next we study the effect of varying the client NV write caches on performance while keeping the server NV cache constant at 8 Mbytes. In Figure 6.6, we plot the variation of the average write

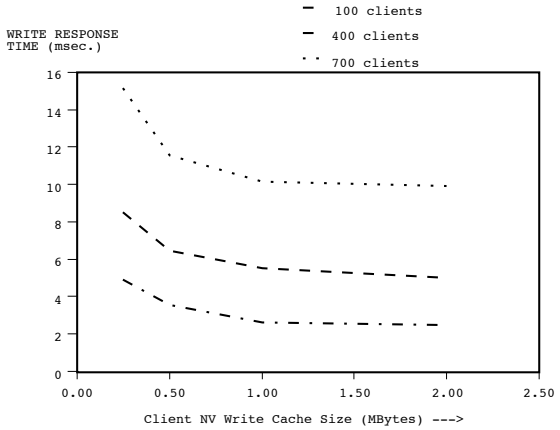


Figure 6.6: Impact of Client NV cache size on the average Write Response Times at three different client load levels

response time as seen by the client application as the client NV write cache is varied between 0.25 MBytes to 2 MBytes. We consider a system that uses the threshold based writeback policy at both the client and the server. We notice that at all load levels, the average write response time decreases until the client NV write cache is increased to about 1.0 MByte. The average write response time is unaffected by an increase in the client NV write cache boyond 1.0 MByte.

6.6 Comparison: Full Cache Writeback vs. Small Fraction NV Write Cache

In the previous subsections, we examined the performance improvements resulting from including NV write caches at the server and at the clients. We introduced these as additive NV write caches over and above the volatile read caches at the server (32 Mbytes) and the clients (8 Mbytes each). We now show that the introduction of a small amount of NV write caches yields

almost all of the performance benefit obtained by replacing the volatile cache completely by an NV cache. We consider, as our base case, a single writeback cache at the client of 8 Mbytes and at the server of 32 Mbytes. We then compare the average write and read response times of the base case with the average read and write response times for the following configuration:

- server - 8 Mbytes of NV write cache and 24 Mbytes of a volatile read cache,
- client - 2 Mbytes of NV write cache and 6 Mbytes of volatile read cache.

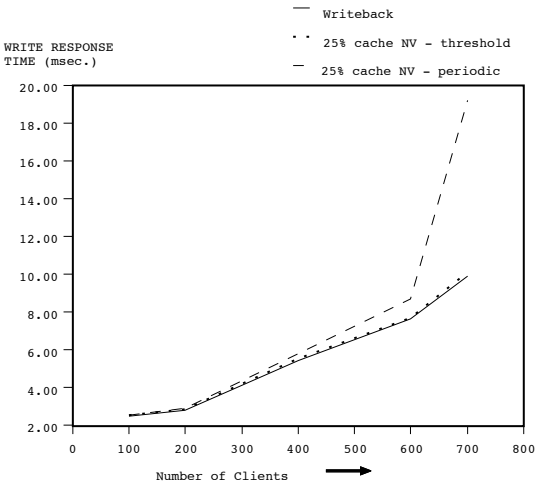


Figure 6.7: Average Write Response Times for full cache writeback vs. 25% cache NV with two alternative cache write policies

	Read response (ms)		
	100 clients	400 clients	700 clients
Periodic	6.3	11.5	34.6
Threshold	5.6	9.5	16.2
Writeback	5.6	10.3	17.1

Table 6.9: Average Read Response Time comparison at 3 client load levels

We study both the periodic and threshold based writeback policies in this configuration. At light loads, (Figure 6.7) the average write response time with the NV write cache is almost as low as that obtained on a system with a full writeback cache of a size equal to the sum of the large read cache and the small NV write cache. At heavy loads, the NV write cache with a threshold write-back policy still performs as well as the large single cache with full writeback. This is true not only for the average write response time, but also for the average read response time as seen in Table 6.9. Thus, it is possible to have a system that is not vulnerable to failures while increasing the cost of the system only slightly for the small NV write cache. However, for periodic write-back the average read and write response times are substantially higher at heavy load.

6.7 NV Cache Location Tradeoff

If a fixed amount of NV storage is available, how should it be distributed between the clients and the server to maximize the performance improvement? Let us first look at the case, where we have 50 MBytes of total NV storage to be distributed among 100 clients and one file server. Figure 6.8 shows the average response time of write operations as the server NV cache size is varied from 0 to 50 Mbytes. Note that increasing the server cache size implies decreasing in the client NV cache size. The client NV cache size for a given server NV cache size is calculated as fol-

lows:

$$Client\ NV\ Cache\ Size\ per\ client\ =\ \frac{Total\ NV\ Storage\ Available\ -\ Server\ NV\ Cache\ Size}{Number\ of\ Clients}$$

When the whole NV storage is used at the server, the clients use a write-though policy to the server. Similarly, when all 50 MBytes of the NV storage is used at the clients, the server uses a write-through policy to its disk. We focus only on the average write response time because that is the metric that is primarily impacted by the size of the NV cache.

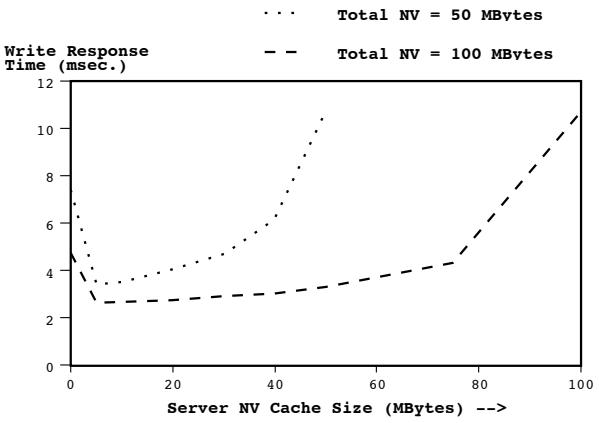


Figure 6.8 : Write Response Times for Varying Server NV Cache Size. Total NV Storage Fixed at 50 and 100 MBytes

Figure 6.8 shows that if a write-through policy is used at either the clients or the server (represented by server NV cache sizes of 50 and 0 MBytes), it results in poor performance. It appears that a small server NV cache is adequate and the rest of the NV storage should be used at the clients. Placing all 50 MBytes at the server yields an average write response time of 10.74 milliseconds while dividing the 50 MBytes among the clients (0.5 MBytes per client) yields a average write response time of 7.40 milliseconds. But if we place 10 MBytes at the server and 0.4 MBytes at each client, the average write response time reduces to 3.52 milliseconds.

Figure 6.8 also shows the impact of varying the server and client server caches when we have 100 MBytes of NV storage for the same environment of 100 clients and one server. Again, we observe similar characteristics with a small 5 MByte server NV cache and 0.95 MByte client NV caches yielding the lowest average write response time.

6.8 Effect of File Size Variation

Results from two studies of the same environment, one reported in 1985 and the other published in 1991, shows that file sizes are increasing over time. The data used in this paper was collected during 1988-89 and therefore it is important to investigate the impact of increasing file size on

the performance of the distributed file system.

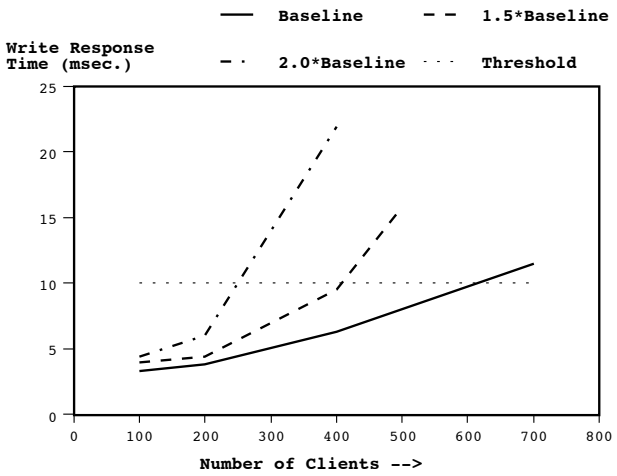


Figure 6.9: Write Response Times for Varying File Size

Figure 6.9 compares the average response time for write operations for three different file sizes. Here *Baseline* represents the case where the file sizes observed in the trace data are used. The other two cases, *1.5*Baseline* and *2.0*Baseline*, are two projected workloads where the average file size is multiplied by the corresponding factor of 1.5 or 2.

Figure 6.9 shows that file sizes have significant impact on the average response time of write operations. Table 6.10 also shows that increasing file size increases server resource utilization and therefore affects the response times of all file system operations. For example, if we impose an ‘‘arbitrary’’ limit of 10 millisecond as acceptable average write response time, this limit is crossed at a load of about 600 clients for the *Baseline* whereas the same threshold is crossed at about 400 and 250 clients when the average file size is increased by a factor of 1.5 and 2.0 respectively.

7 CONCLUSIONS

The I/O subsystem in computer systems has become the bottleneck because of recent dramatic improvements in processor speeds. In particular, disk characteristics have not changed as dramatically as the rest of the computer system. As a result, distributed file systems require caches at various locations in order to alleviate the limitations due to this bottleneck. Caches both at the file server as well as at the clients are commonly used. These traditionally have improved the performance of read operations as seen by the user application. However, to provide a consistent view of the data seen by all the clients of the distributed file system, and for reliability in the face of failures, write operations are often allowed to complete only after the data has been committed to stable storage, the disk at the file server. Therefore, the dominant load on the file server consists of writes. Allowing for writebacks from the clients can reduce the load on the server but exposes the data to loss due to failures. Fortunately, non-volatile random access memory (NV-RAM) is becoming cheaper. In this paper, we considered the performance issues introduced by

combining non-volatile write caches with separate read caches in the file system.

Table 6.10: Server Utilizations and Response Time Comparison.

(For Varying File Sizes at Three Load Levels)

	Server CPU Utilization (%)			Server System Disk Utilization (%)		
	100 clients	200 clients	400 clients	100 clients	200 clients	400 clients
Baseline	9.2	19.0	40.7	10.2	20.4	36.4
1.5*Baseline	13.9	28.4	58.1	13.1	26.6	45.2
2.0*Baseline	18.6	37.3	72.5	16.3	33.2	55.3

	Read Response Time (msec.)			Open Response Time (msec.)		
	100 clients	200 clients	400 clients	100 clients	200 clients	400 clients
Baseline	6.6	8.2	10.9	54	58	67
1.5*Baseline	8.3	10.4	14.7	57	63	85
2.0*Baseline	9.6	12.4	23.5	59	71	140

The synthetic workload we use for this simulation study is based on parameters derived from analysis of I/O traces collected from two operational commercial sites representing interactive scientific and office environments. The important file access characteristics modeled by the synthetic workload includes the file inter-open time, the file open-close time, the file sizes and most importantly, the characteristics of sharing. We further subdivide the classes of active files based on their file-open modes (e.g., Read-Only, Write-Only, Read-Write) and the organization of the files (sequential, random access). We observe that the sharing characteristics differ among the different classes of files and the access characteristics differ according to the file organization. Therefore, we divide the active files into six categories (sequential-read-only, random access-read-only, etc.) and look at the file size, file access pattern and sharing separately for each of these classes.

The study of the effectiveness of the NV write cache is based on a simulation model of a distributed environment of a large number of clients and a single file server. We parameterize the service times for the components at the client (CPU), the server (CPU, disk) and the network for the different file system operations (open, read, write, close) based on careful measurements of resource consumption on a representative workstation. We include the overhead for the cache con-

sistency mechanisms and the system overheads for network I/O.

The primary observation we make from the study is that a small NV write cache at the server and at the clients is quite effective. This reduces the average write response time (which we consider the primary metric to understand the effectiveness of the NV cache) and the load on the file server dramatically, thus improving the scalability of the system. In fact, for this workload, we found that a NV write cache of 1 Mbyte at the client (with a volatile read cache of about 8 Mbytes) and a NV write cache of 8 Mbytes at the server (with a volatile read cache of about 32 Mbytes) reduces the average write response time by over 90% in comparison to having a simple write-through policy at the clients and the server even under light loads. We also ensured that there is no significant impact on the average read response times, the average response time of file opens and the resource utilization due to the cache writeback policies.

We studied the comparative benefits of two alternate writeback policies for the NV write caches. The first is a ‘‘periodic’’ writeback policy (checking every 5 sec. for dirty blocks older than 30 sec.) and the other is our proposed policy, called the ‘‘threshold’’ based writeback policy. The threshold based writeback scheme uses a hysterisis policy for determining when to writeback dirty blocks from the NV write cache. At low loads, the policy for replacing blocks from the NV write cache is not critical as the system utilization and the average write response time is quite low. But at high loads, the ‘‘threshold’’ policy, which reduces the load on the server significantly compared to the ‘‘periodic’’ scheme, performs the best. In the ‘‘threshold’’ policy, dirty blocks tend to held longer in the write cache thereby increasing the probability of being overwritten or being called back to the server only when it is required by another client.

With only a server NV write cache, the write response time cannot be improved much beyond a certain point. Introducing a small client NV cache, together with the server NV cache, provides further improvements. For example, the average write response time at a load of 100 clients with a server NV write cache of 50 Mbytes is 10.74 msecs. But when the same NV cache is distributed into small, 400 Kbyte, NV write caches at each of the 100 clients and an 10 Mbyte server NV write cache, the average write response time drops down to only 3.52 msecs. An advantage of the ‘‘threshold’’ policy is that the performance of the policy is not sensitive to the exact choice of the threshold values.

We have shown that the use of NV write caches at the client and the file server with simple policies for managing these caches can result in substantial improvements in write response time and the ability to support a large number of clients.

8 REFERENCES

1. Baker, M.G., et al, "*No-Volatile Memory for Fast, Reliable File Systems,*" Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V), October, 1992.
2. Baker, M.G., et al, "*Measurements of a Distributed File System,*" Proceedings of the 13th Symposium on Operating Systems Principles (SOSP), October, 1991.

3. Biswas, P., et al, "*Trace Driven Analysis of Write Caching Policies for Disks,*" To appear in the Proceedings of the 1993 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, May 1993.

4. Biswas, P., Ramakrishnan, K.K., "*File Access Characterization of VAX/VMS Environments*," Proceedings of the 10th International Conference on Distributed Computing Systems, May 1990.

5. Copeland, G., et al, "*The Case For Safe RAM,*" Proceedings of the 15th International Conference on Very Large Data Bases, August 1989.

6. Howard, J.H., et al, "*Scale and Performance in a Distributed File System*," ACM Transactions on Computer Systems 6(1), February 1988.

7. Kure, O., "*Optimization of File Migration in Distributed Systems*," Ph.D. thesis, Computer Science Division, University of California, Berkeley, CA, UCB-CSD-88-413, 1988.

8. Lazowska, E., "*The Use of Percentiles in Modeling CPU Service Time Distributions*", Computer Performance, K. M. Chandy and M. Reiser (Eds.), North Holland Publishing Company, 1977.

9. Lazowska, E., Zahorjan, J., Cheriton, D., Zwaenepoel, W., "*File Access Performance of Diskless Workstations,*" ACM Transactions on Computer Systems 4(2), August 1986.

10. Menon, J., Hartung, M., "*The IBM 3990 Disk Cache,*" Proceedings of the IEEE Computer Society International COMPCON Conference, 1988.

11. Moran, J., et al, "*Breaking Through the NFS Performance Barrier,*" Proceedings of the EUUG Spring 1990, Munich, Germany, April 1990.

12. Nelson, M.N., et al, "*Caching in the Sprite Network File System,*" ACM Transactions on Computer Systems, February, 1988.

13. Nielsen, M.J.K., "*DECstation 5000 Model 200,*" Proceedings of the 36th IEEE Computer Society International Conference, COMPCON 1991, February 1991.

14. Ramakrishnan, K.K., Emer, J.S., "*Performance Analysis of Mass Storage Service Alternatives for Distributed Systems*," IEEE Transactions on Software Engineering 15(2), February 1989.

15. Ramakrishnan, K.K., et al, "*Analysis of File I/O Traces in Commercial Computing Environments,*" Proceedings of the 1992 ACM SIGMETRICS and PERFORMANCE '92 International Conference on Measurement and Modeling of Computer Systems, June 1992.

16. Schwetman, H.D., "*CSIM Reference Manual,*" MCC Technical Report, ACA-ST-257-87 Rev 14, March 1990.

17. Thompson, J.G., "*Efficient Analysis of Caching Systems,*" Ph.D. thesis, Computer Science Division, University of California, Berkeley, CA, UCB-CSD-87-374, 1987.