Everything You Always Wanted to Know about NFS Trace Analysis, but Were Afraid to Ask

Dan Ellard
Jonathan Ledlie
Pia Malkani
and
Margo Seltzer

TR-06-02

Computer Science Group
Harvard University
Cambridge, Massachusetts

SOS Technical Report: Everything You Always Wanted to Know about NFS Trace Analysis, but Were Afraid to Ask

Dan Ellard, Jonathan Ledlie, Pia Malkani, Margo Seltzer
{*ellard,jonathan,malkani,margo*}*@eecs.harvard.edu*
Harvard University

June 25, 2002

Abstract

The past two decades in file system design have been driven by the sequence of trace-based file system studies that have informed the research community of the general access patterns applied to secondary storage systems. Those same two decades have witnessed a radical shift in the types of people who use computer systems and a resulting change in the workloads to which today's systems are subjected. In this paper, we continue the tradition of trace-based studies, but with some important differences. First, we use passive monitoring of NFS traffic to collect our data; we believe that the non-invasive nature of this technique will improve the community's ability to collect a wider range of workloads. Second, we focus on two different types of workloads, the traditional CS departmental load typically used in such studies and the load from a campus-wide, ISP-like environment. As might be expected, these workloads differ dramatically. The departmental workload does not follow previous predictions as directly as might have been expected and the ISP-like workload turns out to provide a detailed analysis of file-system-based mail handling. Our findings demonstrate that the set of static parameters and heuristics that most file systems provide is inadequate to adapt to complex, specialized workloads.

1 Introduction

There is a rich history in trace-based file system analysis, and the results of such studies have driven file system design for the past two decades. The original analysis of the 4.2BSD file system [3] in conjunction with the introduction of RAID [4] motivated many of the design decisions of the log-structured file system (LFS) [6]. The revisitation of the original BSD study [1] confirmed the community's earlier results and further drove file system design and evaluation towards the support of the computer science engineering and research workloads. In the late 1990's, our repertoire of trace-based studies was expanded to include the increasingly dominant desktop systems of Microsoft [5, 9]. This natural progression leads to this paper, which expands the domain of analysis to include large server systems supporting thousands of non-technical users. By non-technical users, we mean that the systems under study are used neither for programming and conducting computer science research nor to support a common time-shared environment. Instead, these large server systems serve a population of users who (as it turns out) spend the majority of their file system accesses reading email.

While at the time of the earliest trace-based studies, most computer users were technical users, on today's systems, the majority of users fall into this "non-technical" category. As such, the file systems with the greatest user impact are those that address the needs of this community. In order to drive file system evolution in the direction of greatest utility, we must understand these new workloads.

In order to perform an in-depth examination of new workloads, we needed to gather traces from these systems. Unfortunately, tracing a production system presents serious challenges. The principle practical obstacle is that system administrators are unwilling to modify their systems. This is largely due to the risk associated with loading new, non-production, and possibly buggy (or malicious) kernel patches into their critical systems as well as the concern about additional load it places on their systems.

To overcome this obstacle, we perform data collection using passive NFS monitoring. Capturing and recording NFS packets on the network requires no changes to the NFS server host and does not introduce any new load on the system. In our data collection, we used mirror ports on the network to copy all packets to and from the NFS server to our monitor.

Although analyzing NFS traces properly introduces some new technical challenges, the social and legal challenges involved in gathering traces from production systems are even more difficult to overcome — detailed file system traces can reveal an enormous amount of sensitive information about the activities of individual users, and the administrators of these systems have justifiable concerns about protecting the privacy of their users. At the beginning of our project, we contacted several ISPs, hoping to gather traces from a variety of sites. Although several of these ISPs expressed interest, none of them were able to let us gather traces. The only information we were able to get were aggregate statistics from one ISP. This analysis looks at one of many new non-technical workloads; hopefully, it will promote more ISPs to submit their traces to more rigorous and open analysis, which, in turn, will invigorate contemporary file system design.

The contributions of this work are: an in-depth analysis of a non-technical workload, a collection of new tools and techniques for analyzing NFS traces, another data point in the timeline of research-environment-based file system analysis, a new method for quantifying workload sequentiality and understanding its impact on file servers, and some modern guidelines to drive file system design.

The rest of this paper is organized as follows. In Section 2, we discuss our tracing methodology and the different environments on which we collected traces. Sections 3, 4, and 5 present the bulk of the trace analysis, discussing reference locality, block lifetimes, and the patterns of accesses within files. These are some of the critical elements of a workload description that can be used by file and storage systems to optimize their performance. In section 6 we summarize our study, making suggestions for the future file system designs.

2 Gathering New Traces

CAMPUS	EECS
Storage for the campus SMTP, POP and login servers.	Storage for the EE/CS department home directories.
Most of the NFS requests are for data.	Most of the NFS requests are for metadata.
Reads significantly outnumber writes (2.5:1).	Writes significantly outnumber reads.
Peak load periods highly correlated with day of week and time of day.	Unpredictable interactive load, but predictable background activity.
20% of the files accessed, and 95+% of the data read and written comes from mailboxes.	No mailboxes, some mail lock and temporary files.
50% of files accessed are mailbox locks.	Large number of mail application locks, but also many other kinds of locks.
Most blocks live for at least ten minutes.	Most blocks die in less than one second.
Almost all blocks die due to overwriting.	Blocks die due to a mix of overwriting and file deletion. Many deleted blocks are from files in browser caches.

Table 1: Characteristics of CAMPUS and EECS.

Most analysis of file system traces has been from a thin slice of the numerous workload types that exist. Almost all system-call level file system work has been on data gathered from research labs, computer science departments, or web servers. Our goals were to contrast this previous work to contemporary workloads, especially ISP-like workloads, and to show how to perform detailed workload analysis without invasive changes on the traced host.

In order to both examine new workloads and place our work in the context of previous work, we gathered two sets of traces. The EECS trace is reminiscent of the frequently studied computer science departmental workload. The CAMPUS workload is from our University's central computing center. The two workloads are summarized in Table 1. Both of these systems use NFS extensively to provide file access; in section 2.3 we discuss how we took advantage of NFS to gather our data.

2.1 The EECS System

The main EECS NFS server is a Network Appliance Filer that serves as the primary home directory server for our computer science department. The workload is a mix of research, software development, and course work. EECS users can directly mount their home directories onto their PCs via NFS, Samba, or other protocols. In the EECS system, all non-NFS file access protocols are routed through an intermediate host, preventing us from directly identifying the source of these requests.

Although there is no standardized EECS client, a typical client is a UNIX or NT machine with more than 128M RAM and local copies of system software and most utilities. The EECS server is used primarily for home directories

date	reads	%	writes	%	lookup	%	getattr	%	total	read M	write M	r/w ratio
10/07/2001	2915	17.92	3115	19.15	5425	33.34	2773	17.04	16271	16479	14950	0.936
10/14/2001	2818	10.94	4904	19.04	11233	43.62	3289	12.77	25755	15403	23379	0.575
10/21/2001	3305	10.63	4809	15.47	13478	43.34	6029	19.39	31098	35672	63594	0.687
11/11/2001	1876	14.83	3484	27.54	4069	32.17	867	6.85	12649	13387	27853	0.538
11/18/2001	1720	18.86	1490	16.34	2208	24.20	2386	26.16	9122	12039	10059	1.154

Table 2: EECS aggregate statistics for several week-long periods. (Some week periods are omitted because they are missing more than one days worth of data.)

and shared project and data files. Unlike what we will see in the CAMPUS workload, the EECS traces do not contain any email traffic. In the EECS system, email and WWW service is provided by other servers, and user inboxes are accessed via a dedicated partition on a separate server, instead of being located in each user's home directory.

There are no user quotas on the EECS system. The aggregate disk capacity of the EECS client machines is significantly larger than the capacity of the EECS server. Much of the research data used in the EECS system is distributed over other servers; the only information stored on EECS is data that needs to be shared, or data that users want to have backed up. (The traces do not include any backup activity.)

Somewhat perversely, much of the EECS workload is caching web pages viewed by users running on client workstations. By default, these browser caches are created in a subdirectory of the user's home directory, so they are "cached" on the central file server instead of locally to each machine. Another somewhat surprising behavior is that the most frequently created and deleted files on EECS are files created by the window managers and desktop applications of some users (for example, `Applet` files created by GNOME).

Tables **??** and **??** show the aggregate and hourly average statistics for the CAMPUS server.

2.2 The CAMPUS System

CAMPUS is a collection of machines serving the computing needs for the bulk of the administration, college, and graduate school at the University. It handles email and web service for the majority of the students, faculty, and administrators, and has approximately 10,000 active user accounts.

CAMPUS storage is distributed over fourteen 53GB disk arrays. These fourteen disk arrays are hosted by three NFS servers. Because each array is given its own virtual host, all NFS traffic to a particular disk array uses an IP address unique to that disk array.

Each of the fourteen disk arrays contains the home directories for a subset of the CAMPUS users. Users are given a default quota of 50 MB for their home directory. Users are distributed among the disk arrays according to the first letters of their login names. We gathered long-term traces for two arrays, and short-term traces for several of the other arrays. The behavior of all of the CAMPUS traces was similar; we chose one array, "home02", for our analysis.

The central email, WWW, general login, and CS course servers mount the disk arrays via different networks. Because our traces capture the traffic between the email and general login servers and the disk arrays, we do not capture

All Hours													
week	total		readM		reads		writeM		write		r/w ratio		
10/07/01	96850	(75847)	98	(133)	17352	(22680)	89	(138)	18542	(29356)	2.42	(2.57)	
10/14/01	153304	(175656)	92	(121)	16772	(28968)	139	(233)	29188	(54714)	2.70	(3.96)	
10/21/01	185106	(161463)	212	(351)	19673	(21601)	378	(932)	28627	(57463)	3.16	(7.66)	
11/11/01	75293	(164772)	80	(189)	11165	(32793)	166	(506)	20736	(57413)	1.32	(2.23)	
11/18/01	54296	(35923)	72	(67)	10238	(9539)	60	(64)	8872	(8081)	1.47	(1.27)	
Peak Hours Only (Weekdays 9:00am - 6:00pm)													
week	total		readM		reads		writeM		write		r/w ratio		
10/07/01	170276	(91231)	189	(150)	34076	(23064)	193	(190)	40174	(44561)	1.62	(1.68)	
10/14/01	243722	(111885)	161	(92)	29012	(15757)	292	(313)	62352	(79794)	1.07	(0.92)	
10/21/01	267227	(183498)	268	(391)	29240	(22531)	439	(1002)	34073	(54014)	1.13	(1.20)	
11/11/01	106724	(154813)	73	(75)	9625	(12361)	290	(579)	35133	(64149)	0.81	(0.63)	
11/18/01	78697	(41729)	107	(79)	14962	(10132)	91	(77)	12386	(7669)	1.50	(1.58)	

Table 3: Hourly average statistics for EECS (lair62), with standard deviations in parentheses, for all hours in each week and for just the peak hours of each week.

date	reads	%	writes	%	lookup	%	getattr	%	total	read (M)	write (M)	r/w ratio
10/07/2001	101959	64.09	32648	20.52	10240	6.44	3872	2.43	159083	727546	253339	3.123
10/14/2001	114283	64.16	37807	21.22	10739	6.03	4219	2.37	178132	802308	293608	3.023
10/21/2001	120780	64.61	40126	21.46	10739	5.74	4211	2.25	186941	837798	311825	3.010
11/11/2001	125549	66.37	39400	20.83	98077	5.18	4048	2.14	189170	874994	306223	3.186
11/18/2001	77169	66.71	22449	19.41	6619	5.72	2671	2.31	115675	553696	174316	3.437

Table 4: CAMPUS aggregate statistics for several week-long periods. (Some week periods are omitted because they are missing more than one days worth of data.)

traffic generated by serving personal home pages or by students working on CS assignments. Statistics provided by CAMPUS show that the email and general-purpose login subnet carries the vast majority of the total traffic. The CAMPUS traces do not include backup activity.

Mail to each CAMPUS user is delivered to an inbox located inside his or her home directory. This is unlike many other systems, where the mail spools are kept in dedicated partitions. The main activity of the CAMPUS system is email. Most users of the CAMPUS system access it remotely via a POP or SMTP server from their PC or Macintosh.

Tables 4 and 5 show the aggregate and hourly average statistics for the CAMPUS server.

2.3 Gathering Traces

Although we gathered our traces in a manner different from previous studies, we wished to provide some of the same analyses found in those earlier papers. We found that this was complicated by the difficulty in inferring the client-level

All Hours						
week	total	readM	reads	writeM	write	r/w ratio
10/07/01	946922 (496216)	4330 (2105)	606898 (313308)	1507 (907)	194335 (116896)	3.53 (2.19)
10/14/01	1060311 (531658)	4775 (2167)	680254 (331211)	1747 (1056)	225040 (135898)	3.32 (1.45)
10/21/01	1112741 (551094)	4986 (2235)	718927 (347657)	1856 (1076)	238845 (138417)	3.27 (1.58)
11/11/01	1119973 (610743)	5180 (2 526)	743140 (391183)	1813 (1157)	233316 (148738)	3.51 (1.33)
11/18/01	688544 (585054)	3295 (2570)	459338 (382925)	1037 (1047)	133626 (134664)	4.58 (2.57)
Peak Hours Only (Weekdays 9:00am - 6:00pm)						
week	total	readM	reads	writeM	write	r/w ratio
10/07/01	1462787 (198471)	6376 (720)	931065 (123707)	2439 (416)	314315 (53410)	2.66 (0.32)
10/14/01	1656301 (182642)	7009 (585)	1044012 (98762)	2881 (595)	370960 (76315)	2.49 (0.32)
10/21/01	1698972 (129744)	7153 (438)	1088060 (77248)	2934 (353)	377459 (45187)	2.46 (0.25)
11/11/01	1790723 (256633)	7764 (1201)	1168067 (170450)	3063 (446)	393909 (57315)	2.54 (0.22)
11/18/01	1061569 (682277)	4834 (2840)	697072 (443012)	1698 (1286)	218547 (165120)	3.86 (1.79)

Table 5: Hourly average statistics for CAMPUS (home02), with standard deviations in parentheses, for all hours in each week and for just the peak hours of each week.

workload reflected in the NFS trace. For example, some analyses hinge upon explicit knowledge of `open` and `close` system calls, but NFS does not provide this information.

Although the lack of traces at the same interface as other studies (the system call level, in most cases) hinders our ability to reproduce some analyses in their entirety, we believe that our traces do have a great deal of value, because they represent the world as it is seen by an NFS server. In theory, an NFS server might benefit from seeing more of the system call information (for example, advisory "open" and "close" messages could be sent to the NFS server) but for now, we are limited to the NFS protocol.

2.4 Analyzing NFS Traces

In addition to the problem of collecting traces at a different interface, there were three problems that we encountered due to our collection methodology: missing packets, out-of-order requests, and difficulty in identifying or counting users.

Missing packets occurred on the CAMPUS system because our monitor consisted of a single gigabit Ethernet port on a fully-switched gigabit network. During bursts of heavy activity, the monitor port simply did not have the bandwidth to forward all of the network traffic. This problem is compounded by the fact that it is impossible to decode an NFS response without seeing the call, so losing a request packet effectively results in missing both. We estimate that during periods of heavy use, as many as 10% of the packets may have been lost, and for short bursts the percentage could be even higher. On EECS, the mirror port was as fast as the port to the server, so we did not observe a problem with missing packets.

Out-of-order requests occurred when the NFS requests are sent to the server (and observed by our monitor) in

	10-minute Intervals					
	CAMPUS (home02)	CAMPUS-3 (home03)	EECS (lair62)	Windows NT	Sprite	BSD
Max # of active users	482	328	249	45	27	31
Ave # of active users	253.1 (106.1)	192.3 (72.5)	44.3 (39.3)	28.9 (21.6)	9.1 (5.1)	12.6
Ave throughput per user (K/s)	8.1 (2.5)	7.0 (2.3)	7.2 (18.5)	24.4 (57.9)	8.0 (36)	0.40 (0.4)
Peak throughput per user (K/s)	20.1	16.5	97.2	814	458	NA
Peak total throughput (K/s)	6431.9	2696.6	3222.5	814	681	NA
	10-second Intervals					
	CAMPUS (home02)	CAMPUS-3 (home03)	EECS (lair62)	Windows NT	Sprite	BSD
Max # of active users	233	125	247	45	12	NA
Ave # of active users	13.7 (8.2)	10.4 (5.3)	6.9 (6.6)	6.3 (15.3)	1.6 (1.5)	2.5 (1.5)
Ave throughput per user (K/s)	147.2 (209.3)	126.6 (187.2)	37.7 (138.2)	42.5 (191)	47.0 (268)	1.5 (808)
Peak throughput per user (K/s)	10700.4	2733.7	1456.2	8910	9871	NA
Peak total throughput (K/s)	203308.1	13057.0	6720.8	8910	9977	NA

Table 6: User activity over 10-second and 10-minute intervals. The EECS and CAMPUS numbers are for the single day 10/22/2001. The quantities in parenthesis are standard deviations. The numbers for the Windows NT, Sprite, and BSD columns are taken from earlier trace studies[3, 1, 9].

an order different than they were issued from the application. This reordering is an artifact of conventional NFS architectures in which separate processes called `nfsiod`'s issue the network requests. Although a client's requests are dispatched to `nfsiod`'s in order, the process scheduler is the authority that determines the actual order in which these requests will be issued. Unfortunately, out-of-order delivery wreaks havoc with standard notions of sequentiality. More importantly, this reordering might have an important and hidden effect on the apparent performance of NFS servers: the client OS's file system might be making an earnest attempt to order its requests sequentially (or at least sensibly) in order to make the job of the server easier, but this attempt is being thwarted by the `nfsiod`s. Section 5 includes our solution to this problem.

Some metrics used in earlier work measure load per user. Because our traces do not include the RPC user authentication information, it is not possible to reliably identify the user on whose behalf each NFS call was made. In order to generate per-user statistics to compare with earlier research, we use a heuristic that most files are accessed only by their owner, and therefore if we know the owner of a file being accessed, we can guess who is doing the access. This is not completely foolproof, however, particularly on EECS, where there are many shared project directories, and also there are several administrative jobs that sweep through the entire disk every day. In these cases, our heuristic will overestimate the number of users. This means that that average number of users reported in Table 6 is slightly high (especially the maximum number of users per time period), and so all the per-user I/O rates are slightly low. Nonetheless, we believe that they are close enough to provide a meaningful basis for comparison with earlier studies.

	CAMPUS (home02)	CAMPUS-3 (home03)	EECS (lair62)	INS	RES	NT	Sprite
Year of Trace	2001	2001	2001	2000	2000	2000	1991
Days	7	7	7	31	31	31	8
Total ops (millions)	187	128	31.1	318	122	145	4.6
Data read (GB)	837	587	35.7	94.6	52.7	125.3	42.9
Read ops (millions)	121	83.9	3.30	71.9	9.4	39.3	1.66
Data written (GB)	312	195	63.6	16.8	14.1	19.8	9.3
Write ops (millions)	40.1	25.2	4.80	4.65	2.2	7.16	0.46
Read/Write ratio	3.01	3.33	0.69	5.6	3.7	6.3	4.6

Table 7: A Summary of Activity During Trace Periods. The INS, RES, NT, and Sprite numbers are from earlier trace studies[1, 5]. The year row shows when those traces were performed. The workload on INS was instructional, and RES was research.

2.5 Data Used

We gathered several months of traces, and computed summary statistics for the entire trace period. Unfortunately, the sheer volume of CAMPUS data necessitated that we choose a subset of the larger trace period to analyze in depth.

For our analyses, we selected the one-week period of midnight, October 21 through midnight October 27 of 2001, for a single disk array from the CAMPUS system. For CAMPUS, this period of time is slightly busier than average, due to the general trend that systems become busier later in the semester. The EECS data shows so much variation from week to week that defining a "typical" week is difficult, but this week is reasonably representative.

Some of the analyses focus on the weekdays (October 22 through October 26) or the peak load periods (9:00am through 6:00pm on weekdays) or a single day. Each table or graph is labeled with the subset of the data used.

A summary of the total volume of traffic for the traced systems is shown in Table 7, along with comparisons to the same statistics from the Baker and Roselli traces [1, 5]. It is clear that the CAMPUS system is an order of magnitude busier than any of the other systems, particularly in terms of the amount of data read and written. However, as shown in Table 6, the per-user statistics have not changed significantly.

One thing to note is the apparent contradiction that the average hourly read/write ratio is not the same as the read/write ratio for the entire trace period, particularly for the EECS data. It is the nature of the calculation that the average of the ratios is not necessarily the same as the ratio of the averages, but were the variance less extreme, one would observe a closer similarity.

2.6 Load Characterization Over Time

As reported in other studies, most notably Vogels [9], the variance of the load characteristics over time is extremely large. The behavior of CAMPUS and EECS do not converge over time; the statistics for any randomly chosen hour are not likely to be meaningfully close to the daily or weekly averages.

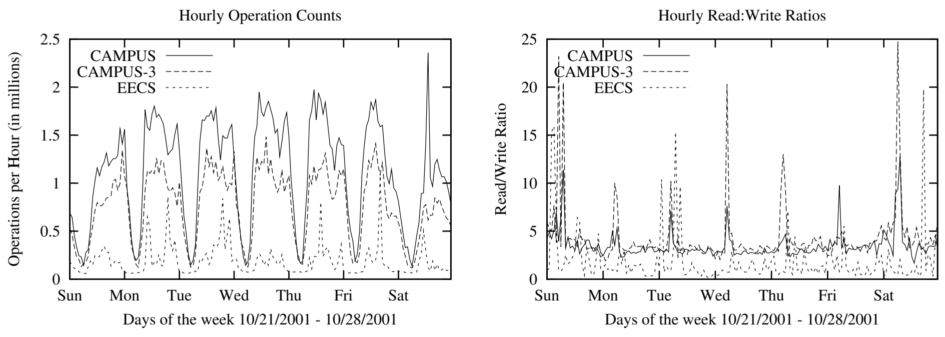

Figure 1: Hourly total operation count and read/write ratios for the week of 10/21/2001.

	All Hours						**Peak Hours Only**					
Hourly Averages	CAMPUS (home02)		CAMPUS-3 (home03)		EECS (lair62)		CAMPUS (home02)		CAMPUS-3 (home03)		EECS (lair62)	
Total Ops (1000s)	1119	(548)	763	(370)	176	(162)	1699	(130)	1151	(125)	267	(183)
Data Read (MB)	5030	(2233)	3497	(1688)	193	(351)	7153	(437)	5129	(1268)	268	(391)
Read Ops (1000s)	725	(347)	499	(241)	17	(21)	1088	(77)	758	(81)	29	(22)
Data Written (MB)	1869	(1067)	1164	(681)	366	(930)	2934	(353)	1802	(264)	439	(1001)
Write Ops (1000s)	240	(137)	150	(87)	27	(57)	377	(45)	232	(34)	341	(540)
Read/Write Ratio	3.23	(1.48)	4.07	(3.36)	1.94	(3.47)	2.46	(0.25)	2.86	(0.60)	1.13	(1.20)

Table 8: Average Hourly Activity. The *All Hours* columns are for the entire week of 10/21-10/27/2001. The peak hours are the hours 9:00am - 6:00pm, Monday 10/22/2001 through Friday 10/26/2001. The numbers in parentheses are the standard deviations of the hourly averages.

After examining the variance of the data over time, we found that much of the variance on CAMPUS can be accounted for by periodic and predictable rhythms in user activity. As illustrated in Figure 1, the load on the CAMPUS system changes over time in a regular pattern, and the most striking difference is between the peak load hours (9:00am to 6:00pm on weekdays), and the lightly-loaded hours during nights and weekends. During the peak hours, both the load and the read/write ratio are fairly consistent, but in the off-peak hours, the load drops substantially, while the variance of the read/write ratio increases, with spikes of very high read/write ratios.

For the EECS data, the peak hours are much harder to define. However, in many cases, load spikes can be traced back to specific causes, such as software builds, large experiments, or data processing tasks (often run via `cron` during off-hours). Much of the total variation in load can be attributed to these causes.

Table 8 shows the hourly average and standard deviations for the statistics for both the entire week and the peak hours. The great reduction in variance for the statistics during the peak load hours shows that time is a strong predictor of these statistics for CAMPUS. For EECS, there is no correlation.

Operations	CAMPUS (home02)	CAMPUS-3 (home03)	EECS (lair62)	ISP-HTTP ISP-HTTP	ISP-FS ISP-FS	ISP-POP ISP-POP	ISP-SMTP ISP-SMTP
Total	186940575	128334193	31097728	9821011	394251241	325700225	51194858
Read	64.6 %	65.4 %	10.6 %	8.8 %	1.2 %	1.0 %	1.4 %
Write	21.5 %	19.6 %	15.5 %	1.0 %	1.0 %	0.3 %	5.2 %
Lookup	5.74 %	6.44 %	43.34 %	70.3 %	33.8 %	30.3 %	53.0 %
Getattr	2.25 %	2.36 %	19.4 %	11.0 %	30.4 %	33.6 %	18.0 %
Access	2.85	2.92	5.91	8.6 %	29.6 %	32.9 %	16.8 %
Read:Write Ratio	3.01	3.33	0.69	9.196	1.250	3.099	0.261

Table 9: Summary of NFS activity for CAMPUS (home02), CAMPUS-3 (home03), EECS, and ISP during the week 10/21 - 10/27/2001. The ISP totals are missing some data from the afternoon hours of Monday 10/22/2001, due to a network error. ISP-HTTP is a WWW server, ISP-FS is a general login server ISP-POP is a POP server, and ISP-SMTP is an SMTP server.

2.7 Discussion

2.7.1 Other Workloads

We were able to gather aggregate statistics from one ISP about traffic between four NFS servers serving four different kinds of clients. The aggregate data illustrates the widely divergent workloads faced by ISP administrators. Table 9 shows a summary of the traffic for the week of 10/21/2001 through 10/27/2001 for the four ISP workloads, in addition to the CAMPUS and EECS workloads.

2.7.2 EECS Workload

Not surprisingly, the EECS traffic patterns fit the model predicted by Ousterhout *et al.* fifteen years ago, also drawn from a computer science environment [3].

The distribution of NFS calls on EECS is predominantly file attribute calls (`lookup`, `getattr`, and `access`). Most of these calls occur because clients are simply checking to see whether a file has been updated or whether they can use a locally cached copy. The overall distribution between reading and writing is approximately equal, but varies widely over time.

2.7.3 CAMPUS Workload

The user-level workload of the CAMPUS environment is different from EECS and from the workload in the earlier traces [1, 5, 3, 9].

On CAMPUS, email is the dominant cause of traffic. Table 10 illustrates how significant this role is during the peak load hours. During these hours, about 20% of the unique files referenced are user inboxes, and another 50% are lock files used to control concurrent access to these inboxes. Many CAMPUS users use email applications that access additional configuration files, put incoming email into other mailboxes, and create temporary files to hold messages

	% Accesses		% Data	
Date	**Inbox**	**Locks**	**Read**	**Write**
10/22 (Mon)	19.4	47.3	96.51	98.24
10/23 (Tue)	19.3	46.8	96.50	96.89
10/24 (Wed)	19.9	48.8	97.34	98.73
10/25 (Thu)	18.2	44.7	96.86	98.25
10/26 (Fri)	18.5	45.7	98.34	98.34

Table 10: CAMPUS (home02) Load Due to Email. All numbers are percentages of the total traffic during the peak load period. The Access columns show the percentage of files opened during the trace period that were either a user inbox or a lock file protecting an inbox. The Data columns show the number of bytes read and written to a user's primary inbox as percentage of the total.

	% Accesses		% Data	
Date	**Inbox**	**Locks**	**Read**	**Write**
10/22 (Mon)	21.1	52.5	96.89	98.65
10/23 (Tue)	21.2	52.3	96.79	98.88
10/24 (Wed)	20.1	52.2	96.48	98.10
10/25 (Thu)	21.8	54.5	97.17	98.86
10/26 (Fri)	21.0	53.6	97.62	98.44

Table 11: CAMPUS-3 (home03) Load Due to Email. All numbers are percentages of the total traffic during the peak load period. The Access columns show the percentage of files opened during the trace period that were either a user inbox or a lock file protecting an inbox. The Data columns show the number of bytes read and written to a user's primary inbox as percentage of the total.

during composition. We do not attempt to identify each of these kinds of files, but we have observed that a large fraction of the remaining accessed files are also related to email use.

These numbers and our analysis of the traces support the hypothesis that most CAMPUS users do little else on the system besides use email. A typical user session involves logging in (accessing `.cshrc` and possibly `.login`), starting an email client (accessing `.inbox` and any configuration such as `.pinerc`, and possibly creating and deleting a lock file for the inbox). During a mail session, mail applications may rescan the mailbox several times. Composing email messages may, depending on the mail program, create temporary files in home directories. Viewing or extracting attachments may also create files. Typical users do little else, although a small population uses the system in other ways.

Even more dominant is the contribution of email to the total quantity of data movement. As shown in Tables 10 and 11, more than 95% of the data read and written involve a user's primary inbox. This percentage is true for both the total number of read and write operations and the total volume of data transferred.

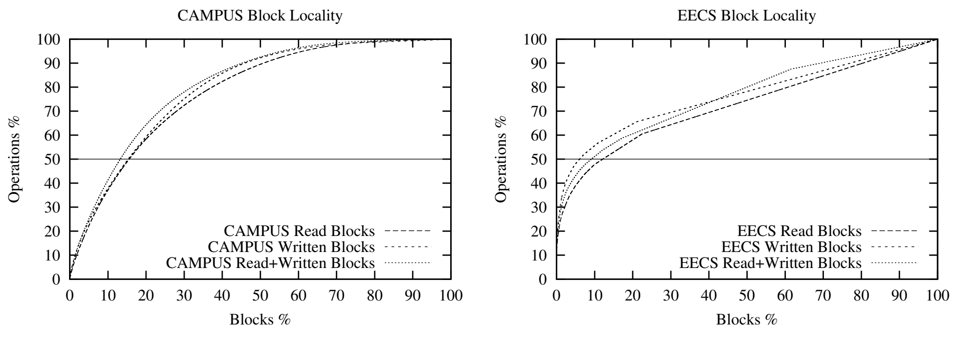

Figure 2: Locality of block reference. Cumulative distribution of operations per logical 8K data block, sorted by frequency of occurrence, for the peak load hours of weekdays 10/22/2001 through 10/26/2001.

3 Reference Locality

The end goal of analyzing file system workloads is to enable the construction of better file and storage systems. While it might be desirable to globally optimize a file system's behavior, this problem is computationally intractable. However, with practical, limited resources, it is important to understand how much benefit can be gained by focusing on a subset of the total system. To this end, we begin our trace analysis by examining the reference locality present in our workloads. We investigate both block-level locality and file locality to provide guidance for deciding at what level in the storage system to optimize. We begin with block reference locality.

3.1 Locality of Block Reference

As shown in Figure 2, the locality of reference for 8k file data blocks is heavily skewed on both systems, but in different ways. On EECS, only 10% of the blocks account for half of the accesses, while on CAMPUS, 20% of the blocks are required to account for half the accesses. However, if one wishes to account for 90% of the accesses, it takes only 50% of the blocks on CAMPUS, but 90% of the blocks on EECS.

Considering the data another way, one might ask what percentage of the blocks ought to be considered for optimization. If we use the criteria that we should continue including blocks in our optimization strategy until the marginal utility of including more blocks is less than the benefit derived from including them, we would conclude that we should optimize whatever fraction of the blocks occur before the knee in the cumulative distribution. For EECS, we would focus on approximately 20% of the blocks while on CAMPUS, we might want to consider as many as 50% of the blocks. Alternately, we might impose a constraint on the percentage of blocks that could be managed, and use that to cap the benefit we might achieve. Let's say that it is practical to optimize 10% of the blocks, then we would be able to improve approximately 50% of the EECS accesses, but only about 30% of the CAMPUS accesses. Earlier trace-based studies have not reported on block reference locality, so we cannot say anything about trends in this area. However, the fact that the CAMPUS workload requires a much larger fraction of the working set to capture a significant percentage of requests does not bode well for our ability to optimize the majority of file system accesses.

3.2 Locality of File Reference

Given that email is the dominant application on CAMPUS, it is not surprising that the locality of reference is even higher on a file basis. Nearly 99% of the data read and written on CAMPUS come from the most heavily accessed 5% of the files. Although during the course of the day there are many tens of thousands of files accessed on CAMPUS, there are only a few hundred user inboxes accessed on any given day, and these inboxes alone account for more than 96% of the data read and written.

Despite the very different workload, the locality of file reference on EECS is also high. More than 90% of the data read and written on EECS are accounted for by fewer than 10% of the files.

This is better news for an optimizing file system. It says that if we focus optimization policies on specific files, then we can achieve significant improvement.

3.3 Locality of File Position Reference

We hypothesized that on CAMPUS the position of a block in a file would correlate with its access frequency for the following reasons: (1) most write accesses are sequential, and therefore when a file is growing, most writes should come at the end of the file, (2) email is delivered by appending it to the end of the inbox, and (3) since the most recent email is near the end of the inbox, we theorized that users would be most likely to visit the end of their mailbox to read their new email.

The good news is that the data support our hypothesis. The first and last blocks of a file are several times more likely to be accessed than any of the other blocks in the same file. Across the rest of the blocks in the file, there is remarkable uniformity in the frequency of reference on both CAMPUS and EECS.

The bad news is that since the bulk of block accesses on both systems (and particularly CAMPUS) are to files containing dozens or hundreds of blocks, and the most common access pattern is to read the entire file, the first and last blocks account for only a small percentage of the total accesses to an average file. Therefore, there does not appear to be any great advantage to optimizing access to these blocks, at least not on these systems.

4 Block Lifetimes

Another workload characteristic that can be exploited by optimizing file systems is block lifetimes. If blocks do not live long, it may be possible to keep them in memory and avoid ever writing them to disk. Additionally, the block lifetimes are bimodal, then once a block lives sufficiently long, it might be beneficial to optimize its on-disk layout under the assumption that the block will live for a long time.

We used the "create-based" method of Roselli [5] to calculate block lifetime statistics. We partitioned sections of the data into pairs of time periods, and processed each in a separate phase. In the first phase, we record both block births and deaths. In the second phase, we record only block deaths. The output of the second phase is filtered to remove any death records for blocks with lifespans longer than the length of the second phase (called the *end margin* by Roselli *et al.*), to remove sampling bias for blocks born early in the first phase. Any blocks that were born in the first phase but did not die within the end-margin are considered *surplus*.

Calculating block lifespan from only NFS traces is complicated by the fact that we do not have total knowledge of the file system hierarchy. Some operations that create or destroy blocks (such as `write`) refer to files via a file handle, while others (such as `remove`) refer to files via a name and the file handle of the parent directory. Without some knowledge of the directory structure, it is impossible to know whether the same blocks are affected by these two kinds of call. Fortunately, we found that we could infer almost all the necessary information by tracing client requests for information about the file system (principally `lookup`) and tracking changes to the file system as they occurred.

4.1 Analysis

We ran a set of five 24-hour block life analyses on CAMPUS and EECS for the weekdays in our trace period. The first phase of each test began at 9:00am and ran for 24-hours with a 24-hour end margin. We chose a 24-hour end margin because shorter ones did not capture the relatively high percentage of deaths that occur between 18 hours and 24 hours. Other, longer tests agree with earlier observations that a block that lives for a day is likely to live for a relatively long time (perhaps until the students owning those blocks graduate and their accounts are reclaimed!). The end surplus for CAMPUS ranged between 2.1% and 5.9% and for EECS between 3.5% and 9.5%.

4.1.1 Summary of Block Births and Deaths

We first calculated summary statistics for block deaths and births as shown in Table 12.

	CAMPUS	**EECS**
Total Births (millions)	28.4	9.8
Due to Writes	99.9 %	75.5 %
Due to Extension	$\ll$ 0.1 %	24.5 %
Total deaths (millions)	27.5	9.2
Due to Overwrites	99.1 %	42.4 %
Due to Truncates	0.6 %	5.8 %
Due to Removes	0.3 %	51.8 %

Table 12: Block life statistics for 10/22-10/26/2001.

Both CAMPUS and EECS exhibit similar trends with respect to block births. In both systems, most births are due to actual data writes, as opposed to metadata operations (*e.g.,* file extensions). In fact, the number of file extensions is mildly exaggerated, because writes that follow an `lseek` past the end-of-file are interpreted as extension writes to all the newly created blocks; that is, not only the block explicitly written, but all unwritten blocks between the previous end-of-file and the new block are counted as extensions. In the future it might be useful to distinguish between blocks born due to overwriting existing blocks and blocks born due to writing past the end of a file, but we do not distinguish these cases in our analysis because we can not always distinguish them from the trace data. Even with this overcounting, relatively few blocks are created via file extensions, particularly on CAMPUS.

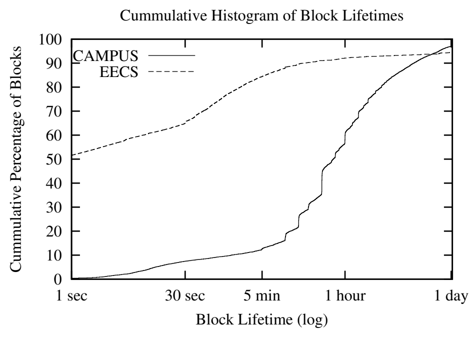

Figure 3: Block Lifetimes. The cumulative distribution of block lifetimes for each day 10/22-10/26/2001.

In the case of block deaths, our results agree with earlier results that showed that most blocks die due to overwriting. For example, in the Roselli trace results [5], the percentage of blocks that die due to overwriting is always greater than 50% and often between 85%-99%. This is especially true for CAMPUS, where 99% of the blocks die due to overwrites. On CAMPUS almost all the data written is to mailboxes, which are never deleted but are overwritten frequently, so this is not surprising.

The distribution is more varied on EECS. There are considerably fewer overwrites on EECS than CAMPUS, and many more removes. This is due to the research-oriented workload on EECS: tasks such as compiling programs, using source control tools, and manipulating data can create and delete many temporary files.

Like CAMPUS, the most commonly deleted files on EECS are zero-length lock files. However, there are also approximately 10,000 deletes per day of small files with names of the form "`Applet_*_Extern`", which are files created by Unix window managers. Web browser caches and temporary files used to compose email messages make up most of the rest of the deleted files.

4.1.2 The Lifespan of Blocks

We calculated block lifetimes using the same two-phase method. The lifespan of a block is defined as its time-of-death minus its time-of-birth. Figure 3 illustrates that the two systems are quite different. On EECS, over half the blocks die in less than a second and relatively few blocks live for an entire day; this is similar to the results of the NT study by Vogels [9]. However, on CAMPUS, blocks live longer; about half live longer than 10-15 minutes. These results are more reminiscent of the results of Baker [1] than they are of the more recent studies by Vogels [9] and Roselli [5], although the shape of our curves is similar.

In the CAMPUS computing environment, blocks live longer because blocks are born due to mail delivery (at the end of the mailbox) or saving the mailbox before exiting their email client, and blocks are killed when mail messages are removed from the mailbox, a process which most email clients do in a batch operation. Thus we expect many blocks to live for approximately the same length of time as an average email session. Although we do not know this length, our data agrees with estimated mail-reading session times between fifteen minutes and an hour.

In comparison with Roselli's block lifetime analysis results [5], we see that our results are for the most part

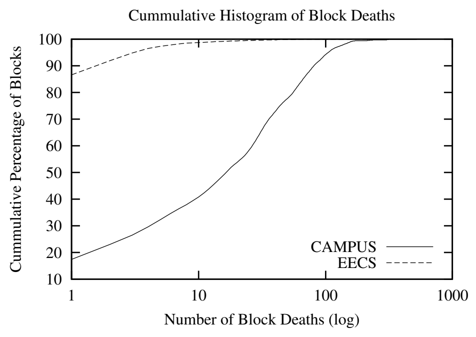

Figure 4: Cumulative distribution of deaths per block, sorted by frequency of death, for each day 10/22-10/26/2001.

different. The only similarity we observed was the fact that both our CAMPUS trace and Roselli's RES trace have a "knee" at approximately the 10-minute mark. However after this 10-minute mark, CAMPUS has a more gradual increase of block lifetimes, whereas the change in RES is abrupt. For the CAMPUS workload, hardly any blocks live for less than a second. Approximately 50% and 20% die within a second for EECS and Roselli's traces, respectively. The surplus, those blocks that live longer than a day, is about 5% - 7% for CAMPUS and EECS. For Roselli, this percentage varies between 10%-30% for non-NT traces and is about 70% for the NT traces.

4.1.3 Number of Deaths per Block

We examine the distribution of the number of deaths that each block experiences over time to see if some blocks are likely to be overwritten much more frequently than others. Figure 4 illustrates this distribution.

The results here show that on EECS the great majority of blocks die only one death. This makes sense since most blocks die due to removal. CAMPUS sees many more recurring deaths. This too is expected given how much data is written each day (almost enough to fill the entire CAMPUS disk every weekday) and the fact that few files are ever removed. The new data must go somewhere, which translates to overwriting existing data.

4.2 Discussion

Our results indicate that, as shown in previous studies, there is a significant benefit from letting young blocks die in a cache. In addition, we observe that applications regularly use the file system for temporary data storage (the GNOME and KDE files) and for locking. These operations can impose a significant load on file systems. Either file system designers should be sure that systems can identify and handle these cases efficiently, or application designers should be encouraged to use more suitable metaphors for locking and temporary storage.

5 Run Patterns

File systems use heuristics to most efficiently access a stream of requests to a file. For example, FFS prefetches blocks when an access appears sequential and does not when it appears random. Determining a client's underlying access pattern is an important part of optimizing file system access. We find that simple heuristics that rely only on a client's unprocessed request stream and on the conventional random/sequential divide are ill-equiped for a NAS environment.

The earlier studies [3, 1, 9, 5] describe workloads in terms of sequential runs. In order to make comparisons, we needed a method to divide NFS records into runs on a file. When we applied the obvious analysis (considering all consecutive accesses to a single file as a sequential run), we obtained a much higher percentage of random accesses than we had expected. We determined that the apparent randomness was due to two factors: the reordering of requests introduced by multiple client-side `nfsiod` processes and the omission of a small percentage of NFS records from our logs. We discuss the results after applying mechanisms to correct for these problems and show that the read-write percentages are significantly different from previous work, but that the amount of random accesses is comparable. At the end of this section, we introduce a method for quantifying the randomness in a random access pattern which grew from our mechanism for dealing with the NFS reordering problem.

Previous work defines a *run* as the series of accesses to a file between each `open` and subsequent `close`, and an *access* as either a single read or write. Because `open` and `close` do not exist in NFS, we use the following methodology to turn lists of write and read operations into runs:

1. We associate a list of accesses with each file, adding a record to this list whenever we see a `read` or `write` to the file in the trace.
2. Periodically, we purge these per-file access lists. If the last item in a list is recent (*e.g.,* younger than 30 seconds), we continue adding records to it as in (1). If the last item is not recent, we begin a new run.
3. Additionally, if during a run we see an end of file, we begin a new run.

Using this mechanism, a series of accesses can be split into one or more runs. We evaluated other methods to break lists of accesses into runs. For example, a break might occur when a gap of several seconds occurs between two accesses or when a large backwards seek occurs. We found that these other methods produced inconsistent results.

After splitting accesses into runs, we separately determine each run's access pattern. Informally, a run is sequential if each request begins where the next left off, for all requests. In NFS terms, a run is sequential if, for all accesses in the run, $\text{offset}_i = \text{offset}_{i-1} + \text{count}_{i-1}$, where i is the index of the current access in a run and count_{i-1} is the number of bytes accessed in the previous access. Offsets and counts were rounded to blocksizes of 8 kilobytes. For example if a series of offset(counts) were 0k(8k), 8k(8k), 16k(7k), 24k(8k), we would consider this sequential. If a run has the qualities of being sequential and also accesses the file from offset 0 through to *eof*, it is called *entire*. If it is neither sequential nor entire, the run is called *random*. A run is called "read" if it contains only read requests, "write" if is contains only writes, and "read-write" if it contains one or more of each.

The first two columns of Table 13 show the result of categorizing all EECS and CAMPUS runs using this methodology. It includes three other traces for comparison. Both EECS and CAMPUS have significantly different run patterns from the previous results. Both workloads contain a much higher percentage of write runs, especially EECS. We initially believed that much of the EECS write activity was due to late night `cron` activity, but screening for only peak hours revealed essentially the same percentages. Screening only the peak hours for CAMPUS also gives the same percentages seen in Table 13. In EECS, the dominance of write traffic is because most client machines are used by one

Access Pattern	CAMPUS	EECS	NT	Sprite	BSD	CAMPUS	EECS
	Raw					Processed	
Reads (% total)	53.1	16.6	73.8	83.5	64.5	53.1	16.5
Entire (% read)	47.7	53.9	64.6	72.5	67.1	57.6	57.2
Sequential (% read)	29.3	36.8	7.1	25.4	24.0	33.9	39.0
Random (% read)	23.0	9.3	28.3	2.1	8.9	8.6	3.8
Writes (% total)	43.8	82.3	23.5	15.4	27.5	43.9	82.3
Entire (% write)	37.2	19.6	41.6	67.0	82.5	37.8	19.6
Sequential (% write)	52.3	76.2	57.1	28.9	17.2	53.2	78.3
Random (% write)	10.5	4.1	1.3	4.0	0.3	9.0	2.1
Read-Write (% total)	3.1	1.1	2.7	1.1	7.9	3.0	1.1
Entire (% r-w)	1.4	4.4	15.9	0.1	NA	3.5	5.8
Sequential (% r-w)	0.9	1.8	0.3	0.0	NA	2.1	7.3
Random (% r-w)	97.8	93.9	83.8	99.9	75.1	94.3	86.8

Table 13: File Access Patterns using *entire/sequential/random* categorization. The first and second columns show CAMPUS and EECS traces divided into runs according to the mechanism enumerated in this section, but otherwise unaltered. The third, fourth, and fifth columns contains Roselli's NT results, the Sprite, and BSD results, respectively, for reference. The last two columns show the CAMPUS and EECS traces divided into runs using the methodology we present in section 5.1. This methodology accounts for request reordering produced by `nfsiod`s and allows for very small seeks of less than 10 blocks per access to not turn a run from sequential to random. In both this presentation and that in section 5.1, singleton runs are either sequential (if they access only part of the file) or entire (if they access the entire file).

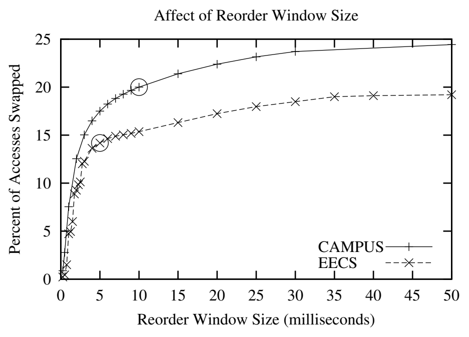

Figure 5: Effect of different window sizes on the number of swapped accesses.

user and that user's files experience little cache invalidation. In CAMPUS, the read-write imbalance has a different cause. There, most writes are short appendages to mailbox files, and most reads are long reads of mailbox and other mail related files.

Neither of the causes for write-domination helped explain the high percentage of random accesses for both reads and writes. After experiments on an isolated system, we determined the cause to be client-based `nfsiod`s and the loss of a small percentage of requests. We developed separate mechanisms to deal with each of these two problems.

In order to handle `nfsiod` reordering, we partially sorted requests in ascending order within a small temporal window, which we call a *reorder window*. For each request, we look ahead t milliseconds to see if a nearby request should be swapped with the current request, and swap them if they are out of order. In an experiment on an isolated system where the client was significantly less powerful than the server, we found that `nfsiod`s reordered requests by as much as one second. In EECS and CAMPUS, both client and servers occasionally became overloaded. We show the results of using differing *reorder window* sizes on a subset of the data in Figure 5.

Both workloads exhibit a knee, after which larger sorting windows do not make the data any more sequential; we tried windows as large as 2 seconds. The results from this analysis suggested that we should use a window size of $t = 5$ milliseconds for EECS and $t = 10$ milliseconds for CAMPUS. We inserted the sorting process before step (b) in our methodology for separating accesses into runs.

This mechanism for approximating the client-side access pattern gets at the larger question of how a NAS file system needs to react to request streams. NAS servers must be intelligent in categorizing a client's access: a single out-of-order access should not relegate it to the "random" dustbin. The vast majority of non-adjacent seeks were to blocks two or three away from the current offset. Runs containing only adjacent and nearby requests would be best served by being characterized as sequential, not as random: small forward jumps in an access should not automatically push a run into the "random" category. In our categorization, we decided that jumps of fewer than 10 blocks should be permitted for a run to remain sequential, because a logical seek of less than 10 blocks is highly unlikely to induce a physical seek and because it was certain to include the outliers that had jumps of five or six blocks.

The right side of Table 13 gives the results of the run patterns after they have been processed with both of these mechanisms. Clearly, the partially sorted runs are much less random than the unprocessed ones, giving a much better approximation of client's underlying run patterns. What this conveys is that client seek patterns have stayed fairly

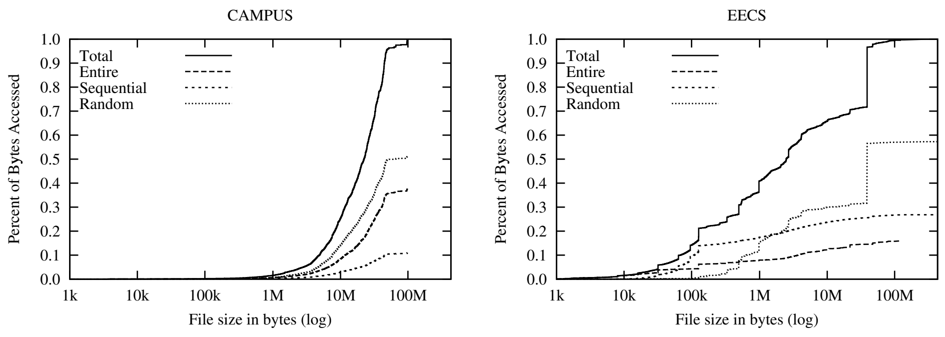

Figure 6: Percentage of bytes accessed randomly, sequentially, or in their entirety verses the total bytes accessed. Each run is categorized according to its access pattern and then all of the bytes accessed in this run are added to the subtotal for this category.

constant and that traditional read-ahead mechanisms will continue to work, as long as the file system can be intelligent enough to accurately determine the actual access pattern.

In Figure 6, we present the file-size based access patterns of EECS and CAMPUS. EECS is similar to previously analyzed file systems where a large percentage of accesses come from files smaller than 1M. Almost 60% of bytes are accessed randomly, which is about the same as Roselli's NT workload and much larger than most other workloads. Again like the NT workload, long files read in their entirety constitute 30% of the total in EECS. CAMPUS is similarly dominated by random and entire runs, although as discussed in Section 5.1 most of these runs that are categorized as "random" here are, in fact, highly sequential. The vast majority of CAMPUS bytes transferred come from files larger than 1M. This is unlike almost all of the previous work on run analysis, except for the two outlying traces of the Baker study [1], which presented an almost identical distribution.

5.1 Sequentiality Metric

Previous analysis, Roselli in particular, has found that the majority of bytes from files over 100k to be accessed randomly. Our evidence shows "random" to be too strong a term and too coarse a measurement. Most runs that would be categorized as random in the entire/sequential/random split are actually composed mainly of sequential sub-runs with jumps in between. Most of these jumps are to blocks less than ten logical blocks away from the current offset. In most file system layouts, a logical jump of this length will not require changing the current physical track or head position, translating into only a tiny performance penalty. In fact, the vast majority of the jumps within the ten block range are to five or fewer blocks away.

We introduce a *sequentiality metric*, based on Keith Smith's *Layout Score* [8], for measuring sequentiality of file access. The *sequentiality metric* is the percentage of blocks that have been accessed sequentially. A block is accessed sequentially if it is consecutive to the previous access. A benefit of computing sequentiality as $\frac{sequentialCount}{accessCount}$ is that it quantifies the degree of randomness in a run. A run with a sequentiality metric close to 1 is almost sequential and should be processed by the file system and disk as if it were. A run with a metric less than about one-half should

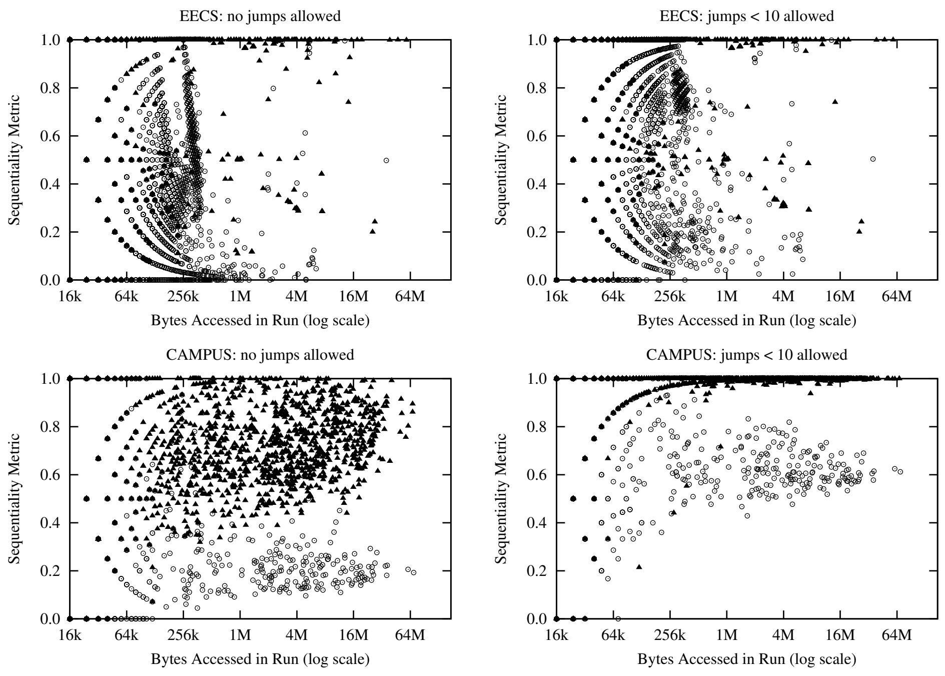

Figure 7: Run Length vs Sequentiality Metric. One point per run for a subset of the week. A triangle signifies a read run; a circle signifies a write run; read-write runs are not shown. For clarity, only a subset of the data is shown. For CAMPUS, it is 10/23 4-5pm and for EECS, it is 10/23 10am-7pm.

Seq. Metric (Range)	CAMPUS no jumps	CAMPUS jumps < 10	EECS no jumps	EECS jumps < 10
Singletons (%total)	837938 (76.3%)	837938 (76.3%)	2860510 (92.2%)	2860510 (92.2%)
> 1 Access (% total)	260638 (23.7%)	260638 (23.7%)	242193 (7.8%)	242193 (7.8%)
(0.0-0.1]	37408 (14.4%)	26484 (10.2%)	80900 (33.4%)	38139 (15.7%)
(0.1-0.2]	12804 (4.9%)	338 (0.1%)	10430 (4.3%)	9772 (4.0%)
(0.2-0.3]	13426 (5.2%)	1327 (0.5%)	9357 (3.9%)	6214 (2.6%)
(0.3-0.4]	7881 (3.0%)	2338 (0.9%)	9056 (3.7%)	5247 (2.2%)
(0.4-0.5]	7852 (3.0%)	1701 (0.7%)	3702 (1.5%)	3123 (1.3%)
(0.5-0.6]	17769 (6.8%)	21100 (8.1%)	17353 (7.2%)	11952 (4.9%)
(0.6-0.7]	19281 (7.4%)	12324 (4.7%)	10209 (4.2%)	7644 (3.2%)
(0.7-0.8]	22299 (8.6%)	12000 (4.6%)	4720 (1.9%)	9375 (3.9%)
(0.8-0.9]	25792 (9.9%)	10861 (4.2%)	2900 (1.2%)	8089 (3.3%)
(0.9-1.0)	96126 (36.9%)	172165 (66.1%)	93566 (38.6%)	142638 (58.9%)

Table 14: Histogram of how many runs fall into a range of the sequentiality metric. All percentages of sequentiality metrics are of the number of accesses > 1 access. This table includes all of the days under study. The data has been partially sorted to account for the `nfsiod` reordering problem.

be categorized as highly random and treated as a special case. We ignore single-access runs for the purposes of the sequentiality metric.

To account for the minimal penalty introduced by small jumps, we introduce δ-consecutive to mean that a block is within δ blocks of its predecessor. In our evaluation $\delta = 10$ blocks, but this penalty could be adjusted depending on the disk's characteristics. In the future, we will examine how the penalty induced by a seek can be modeled as a continuous function based on the geometry of the storage system, rather than as a simple "nearby" or "not nearby".

Figure 7 portrays visually how many of the runs that would qualify as random are actually highly sequential. In particular, with a strict sequentiality metric, CAMPUS read runs >256KB initially cover a broad range of $0.3 - 1.0$. With the more relaxed metric, they are nearly all sequential, showing that most accesses that were not adjacent were nearly so. In contrast, longer write runs were often composed of two or three adjacent or nearby accesses followed by a long seek, resulting in metrics around 0.6 when jumps were allowed. In many cases, this pattern of a few close accesses followed by a long jump was repeated in the same run hundreds of times within the same second. Although less consistently than CAMPUS, EECS exhibits the same trend where long writes are more random than long reads. Shorter read and write runs from both workloads exhibit all or most of the possible sequentiality metrics in this range, resulting in the semicircular pattern.

Table 14 provides degrees of randomness for all files over the week. It shows that most runs contain one access. It also quantifies the shift of runs (mainly write runs) from around 0.3 to 0.7 when small jumps are allowed.

5.2 How Run Patterns Affect Storage

The central difficulty in the results produced through the mechanisms of partial reordering and allowing small jumps is that they enable only approximations of the clients' behaviors and that they may be obscuring true client activity. If a request was genuinely out-of-order or made small jumps, these two mechanisms would obscure these facts. However, an NFS server must take into account client activity when deciding to prefetch. That requests are being reordered despite the best efforts at the client suggests that a more intelligent algorithm needs to be used at the server to optimize accesses.

6 Conclusion

Using techniques described in this paper, one can monitor and analyze the traffic of an NAS server and learn useful information about its workloads. From measurements of the workload on the behavior of a file system, one can predict the future behavior based on the time of day and day of week. The most important contributors to file system behavior, however, are the sets of applications actually in use on the system. We draw three conclusions from our analysis of NAS workloads:

1. **Application-specific design is important.** We observe that file servers have become, in many cases, dedicated to handling the traffic generated by a diminishing number of applications. In the CAMPUS environment, file servers handle traffic created almost exclusively by POP and SMTP requests, and in the ISP environment, this specialization is taken even further, with one file system dedicated to POP and another to SMTP. We expect this trend to continue and, as a result, we expect general-purpose file servers to become the exception rather than the rule. Designers of file systems and network storage devices should embrace this philosophy of application-specific design, as has already happened with database and WWW servers, rather than adhering to the specious ideal that monolithic file systems can perform well under widely divergent workloads [7].

2. **Antiquated and inefficient file system usage can create an enormous load.** Many applications use the file system in antiquated and arguably inappropriate ways. The use of lock files to provide resource locking in a distributed environment is a particularly widespread example. It is seductively convenient and simple to do locking via the file system, but it is also inefficient. We believe that there could be a benefit to a new locking protocol that assumes that the underlying system is distributed, rather than assuming that it is monolithic.

   Many applications make inefficient use of the file system. In particular, keeping mail spools in a single flat file adds a huge overhead to many operations. For example, reading a single piece of email from the end of a flat-file mailbox requires, for many mail agents, reading the entire mailbox. To make matters worse, some mail agents intentionally periodically rescan the mailbox file to look for new messages. The amount of traffic generated by reading email this way is enormous, and very little of it is actually necessary. In desperation, many ISPs are turning to database systems to manage their email, but this transition is not yet widespread.

3. **New NAS measurement techniques are needed.** Utilizing passive traces will allow researchers to sample a wider set of environments, but this approach introduces analysis issues that did not exist with kernel-assisted

logging. We illustrate mechanisms for coping with request reordering, packet loss, and anonymity. Heeding demonstrated requests from the OS community, these techniques lead us to suggest a new metric for measuring sequentiality that applies to both NAS and non-NAS environments [2]. File system designers must remain vigilant to the different views of the file system seen by client and server even in a tightly integrated LAN environment.

Our original motivation for gathering traces was to examine file system behavior and find how to design better block allocation and layout strategies. While we still believe that this approach has merit, the trends that we observed in our data lead us to believe that there is a much larger potential for improvement waiting at the application layer. OS and file system designers should work to advocate better use of resources at the system and user-level levels.

References

[1] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. Measurements of a distributed file system. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 198–212. Association for Computing Machinery SIGOPS, 1991.

[2] J. Mogul. Brittle metrics in operating systems research. In *The Seventh Workshop on Hot Topics in Operating Systems: [HotOS-VII]: 29–30 March 1999, Rio Rico, Arizona*, pages 90–95, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1999. IEEE Computer Society Press.

[3] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A trace-driven analysis of the UNIX 4.2 BSD file system. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles, (10th SOSP'85), Operating Systems Review*, pages 15–24, Orcas Island, Washington, December 1985.

[4] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *In Proceedings of the ACM Conference on Management of Data (SIGMOD)*, June 1988.

[5] Drew Roselli, Jacob Lorch, and Thomas Anderson. A comparison of file system workloads. In *Proceedings of the USENIX 2000 Technical Conference*, Berkeley, CA, 2000. USENIX Association.

[6] Mendel Rosenblum and John K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, 1992.

[7] Margo I. Seltzer, David Krinsky, Keith A. Smith, and Xiaolan Zhang. The case for application-specific benchmarking. In *Workshop on Hot Topics in Operating Systems*, pages 102–107, 1999.

[8] Keith A. Smith and Margo I. Seltzer. File system aging - increasing the relevance of file system benchmarks. In *Proceedings of SIGMETRICS 1997: Measurement and Modeling of Computer Systems*, pages 203–213, 1997.

[9] Werner Vogels. File system usage in Windows NT. In *Proceedings of the Seventeenth ACM Symposium on*

Operating Systems Principles, (17th SOSP'99), Operating Systems Review, pages 93–109, Kiawah Island, SC, December 1999.