RC22629 (W0211-052) November 6, 2002
Computer Science

IBM Research Report

A Study of the Performance of Diskless Web Servers

T.W. Keller, Michael D. Kistler, Ramakrishnan Rajamony, Freeman L. Rawson
Austin Research Laboratory
International Business Machines
Austin, TX 78758

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

A Study of the Performance of Diskless Web Servers

Tom Keller, Mike Kistler, Ram Rajamony, and Freeman Rawson
IBM Austin Research Laboratory

November 1, 2002

Abstract

One commonly held assumption is that one or more high performance local disks are required for good server performance. This report examines this assumption for one important category of servers, web servers, by comparing their performance in three different configurations – with all files on a local disk, with the content served accessed from a remote machine using NFS and with all files, content and system files, accessed using NFS. Based on an examination of the performance of three static web-serving workloads constructed from traces on typical, commodity hardware, we conclude that the location of the content makes little difference in the responsiveness. Additionally, using fully diskless operation with all files, both system and content, accessed over NFS makes very little difference in responsiveness. To stress the remote file access subsystem, we increased the intensity of the workloads to the point at which response time became unacceptable. We found that by the time that it became unacceptable for remote file access, it was also unacceptable for local file access. The differences in the points of degradation that are observed may be attributable to memory overload in the remote access case, particularly in the case which uses no local disks at all. However, verfication of this hypothesis is beyond the scope of this report. Based on this evaluation, we conclude that any performance loss experienced by diskless web servers is outweighted by their other benefits, such as their simpler administration, lower power consumption and lower heat production.

1 Introduction

In this report we evaluate the performance of web servers that make limited use of local disks or do not have them at all. Instead of using local disks for file storage, these systems access some or all of their files from network-attached storage, a storage-area network or a standard remote server system acting as a Network File System (NFS) server. We consider both the case in which the system files are local but the content served by the web server is remote and the one in which all files are remote. There are a variety of reasons for locating both web server content and system files remotely rather than on a local disk.

1. When the web server runs on a cluster, a single copy of the data can be maintained and served to all the servers.
2. If the system files are also stored remotely, only a single copy of most of them is required, making it much easier to manage and maintain the environment.
3. By using remote storage and single copies, the total number of disk devices required is decreased significantly, reducing the overall equipment cost for the installation.
4. The processor and memory required to operate the web server have significantly different manageability, maintenance, and environmental requirements from storage devices. In particular, it

is much easier to replace failing disks when storage is centralized than it is to locate and replace a failed disk in a web server within a rack-mounted web server cluster.

5. Most storage networking solutions provide special features such as backup, high availability and space management that are much more sophisticated than can be implemented when data is spread across multiple machines.
6. Networked-storage devices and servers often employ RAID hardware to provide enhanced performance and reliability as compared to data stored on individual drives in separate machines.

The approach of this study was to implement, measure and compare three distinct cases:

1. a standard, disk-based web server, illustrated in Figure 1,
2. a web server with a local disk for system files and accessing the content that it serves remotely, illustrated in Figure 2,
3. a web server with no local disk and accessing all files, system and content, remotely, illustrated in Figure 3.

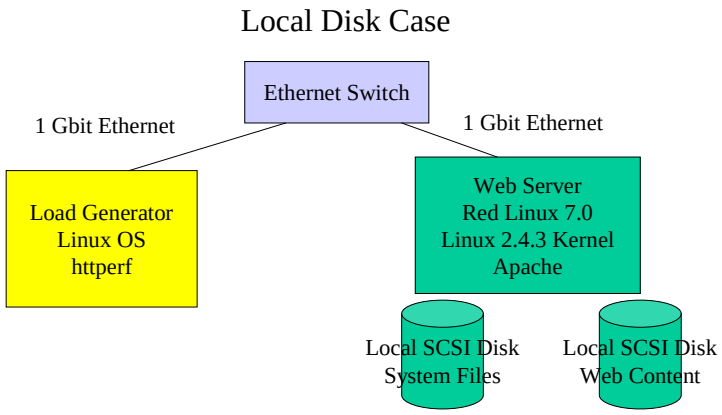

Figure 1: Test Environment for Web Server with Local System Files and Web Content

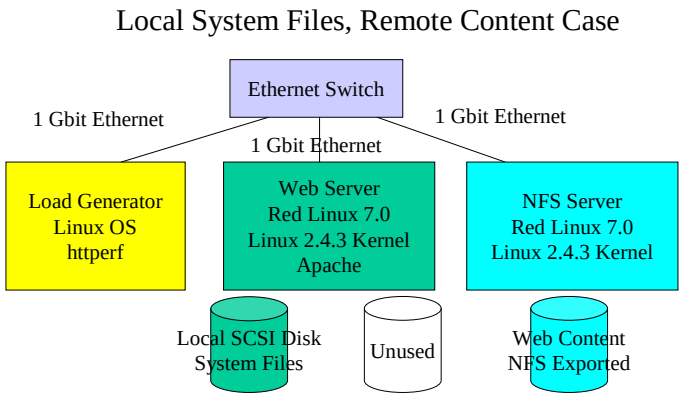

Figure 2: Test Environment for Web Server with Local System Files and Remote Web Content

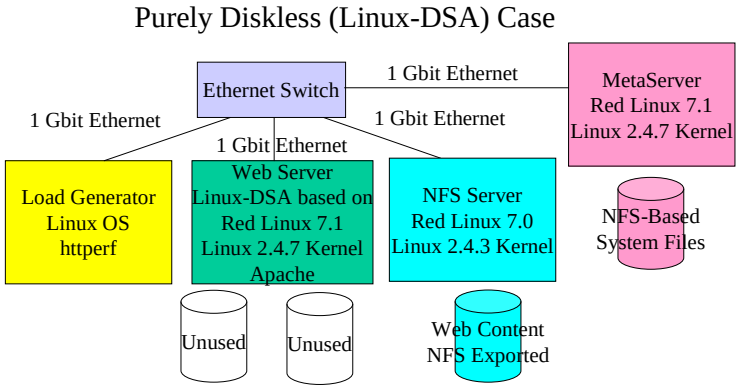

Figure 3: Test Environment for Web Server with Remote System Files and Web Content

Other than file location, the configurations are identical in every respect. We used post-processed traces collected from three real websites. While conceptually simple, this approach was difficult to implement and very time consuming because the workloads themselves require "real" web servers. Our analysis is limited to doing comparisons of the performance of the workloads on our three configurations and determining that our comparisons are fair. Since the focus of this study is the fair quantification of the relative performance of web servers with and without local disks, we did not attempt, during our experiments, to improve the performance of our inexpensive, commodity hardware and software by, for example, replacing our standard Linux-based NFS server with a true network-storage device such as a NAS box.

The remainder of this report is organized as follows. In Section 2 we describe the motivation for this study. We give the methodology used in our experiments in Section 3, including a detailed description of the hardware and software configuration and the workloads used. We present the results of our experiments along with our analysis in Section 4. To complement our experiments, we give in Section 5 a theoretical analysis of the performance of diskless web servers on the SPECWeb99 benchmark. We discuss areas of future work in Section 6 and related prior work in Section 7. Finally, we summarize the conclusions of our work in Section 8.

2 Motivation

The use of diskless server systems and the aggregation of storage on specialized storage nodes have a number of advantages in an era of large main memories and very high-speed network interconnects. By making the server systems diskless, they become essentially stateless, which allows very rapid changes in configuration and even operating system selection. For example, an alternative OS image can be prepared on a disk server while the target server is still running its current OS and workload. When the image is ready, all that has to be done is to reboot the target system with the new image: no disk clean and install cycle is necessary. In addition, the centralization of disk storage economizes on the number of disk drives used since there are far fewer redundant copies of both system and data files. Rather than each server having its own copy, the files are shared by a whole pool of servers.

These same advantages motivated a number of technically and commercially successful implementations of diskless workstation environments during the 1980s. Examples include the SPUR workstation with its accompanying Sprite operating system developed at the University of California at Berkeley and Sun's line of diskless workstations whose development motivated the initial introduction of NFS [4, 13]. More recently, the diskless paradigm has been used in the Plan 9 operating environment developed by Bell Laboratories [14].

However, for server systems, the conventional wisdom is that these advantages are outweighed by the performance penalty imposed by distributed file system protocols and network latency. This view is based on two assumptions – that server workloads are inherently disk-bound and that the performance of disk accesses over a network is poor. Further reinforcing this view is the reputation of NFS, both the protocol and its implementations, for poor performance.

Our motivation in this study is to re-consider the conventional wisdom and to investigate the performance of an interesting server application, serving web pages using a standard HTTP server, in a diskless server environment. If the performance penalty for running diskless is modest, the other

advantages of diskless servers make them attractive. Diskless operation is especially attractive for density, power and heat reasons in environments with very fast local interconnects and large numbers of densely packed servers, such as are increasingly found in data centers.

3 Methodology

3.1 Philosophy

Our goal in this study is to determine the performance impact of locating all of the files used by a web server on network-attached storage. We approximate this by comparing the performance of a standard web server that accesses all of its files from a locally attached disk, a web server that accesses the content that it serves from a general-purpose server system using NFS and a diskless web server that does all of its file accesses from a general-purpose system with NFS. These configurations differ only in the location of the files that they access and are otherwise identical. In addition, in the second and third cases, the file server configurations differ only in the set of files that they serve and are otherwise identical. The workloads in this study are all static data, which actually represents a worst case since data access time is a major determinant of their performance.

3.2 Environment

The systems are commodity 2U-high machines with commodity, inexpensive web serving software:

- 2 Pentium III 866MHz processors with 512KB of off-chip L2 cache
- ASUS CURDLS motherboard with a 133MHz PCI bus and an on-board 100Mbit ethernet
- 2 IBM 36GB Ultrastar Ultra160-SCSI disks (10K RPM)
- 1024MB IBM PC133 SDRAM (2 DIMMs)
- Alteon ACEnic 1Gbit Ethernet card
- Red Hat Linux Release 7.0 and a 2.4.3 uniprocessor kernel
- Apache 1.3.14 web server as packaged in the Red Hat RPM.

The slightly specialized requirements of fully diskless operation have forced us to use a modified Linux kernel on the web server in that case: it is based on Linux 2.4.7. In all three cases, Linux is configured to run in uniprocessor mode, so the second processor is not used. Furthermore, the 100Mbit Ethernet interface on the motherboard is not enabled after initial boot, so all network communication is performed using the gigabit ACEnic interface. Thus, all of the network traffic, both HTTP and NFS, flows over a single network interface. The ACEnic gigabit Ethernet adapters can support Maximum Transmission Unit (MTU) sizes of up to 9000 bytes, which provides higher bandwidth communication. However, we configured the ACEnic with the standard MTU of 1500 bytes for all our experiments. The single switch directly connecting the client, the HTTP server and the NFS server, if used, is a powerful, under-utilized, Extreme Networks' 48-port BlackDiamond 6808 [3]. All computers are connected over a single, 1 Gbit Ethernet subnetwork.

Since our goal was to test the limits of server performance and since the default Apache configuration used by the Red Hat RPM assumes a modestly sized system, we had to make two configuration

changes. First, we increased `MaxRequestsPerChild` from 100 to 10,000. Second, we changed the `MaxClients` setting from 100 to 1,500. Since the Apache binary included in the Red Hat RPM only allows a `MaxClients` value of up to 256, we were forced to change the limit in the Red Hat sources and recompile Apache. Other than applying the maintenance fixes provided by Red Hat, which included a new version of the nfs-utils package, no changes were made to the standard Red Hat 7.0 NFS configuration. All NFS communications use the NFS version 2 protocol and UDP communication. The NFS server uses the same hardware and software configuration as the web server. In the case of a completely diskless web server, there are two NFS servers, one serving the content and one serving all of the other files.

Requests are sent to the web server from a single client machine with the same configuration as our disk-based web-server system. We chose to use a powerful system for the client to ensure that performance is not artificially limited by client resources.

3.3 Workloads

Since our goal is to study the performance of diskless web servers under real workloads, we constructed workloads using web server logs obtained from several production Internet servers. Each access log represents 24 hours of activity. The first workload is derived from the web server access logs of the 1998 Nagano Winter Olympics web site. These logs contain all of the requests processed over the course of one day at one of the four geographically distributed sites used for the Olympics. This workload is analyzed in great detail in [7]. The second workload is derived from the access log of one day's activity on the New York Stock Exchange shareholder information service web server, www.nyse.com. The third workload is derived from a log of a proxy server operated by the Information Resource Caching (IRCache) Project [6], which is affiliated with the National Laboratory for Applied Network Research (NLANR). The IRCache Project operates a number of caching proxy servers on the Internet and publishes the logs from these servers as a way of promoting research on web caching. All IRCache servers run Squid, which is also supported by the IRCache project. Strictly speaking, a proxy server is not a web server in that it does not have its own content, but rather stores frequently referenced content from other servers to allow for more rapid access by clients. Our motivation for creating a workload based on a proxy server was to study a workload with very low locality and occasionally high throughput, thereby placing a "worst case" load on the file access mechanism.

For each workload, we construct a stream of HTTP requests for static files based on the server access log. We exclude all requests that provide parameters as part of the URL, since these are obviously for dynamic content. Such dynamic requests account for 2.5% or less of the Olympics98 and NYSE logs, and none for the Proxy server log. We treat all remaining requests as ones for static files. However, to account for any remaining URLs that may generate dynamic content, all file names are augmented with the size of the file returned, so that each dynamic response of a unique size is treated as a separate file. We have used this approach to deal with dynamic content because we do not have access to the dynamic scripts used at any of these sites and because serving static content is more likely to be disk-bound. Eliminating this small dynamic load results in a small disadvantage to the two NFS-based configurations.

Persistent connections, a feature of the HTTP/1.1 protocol [19], allow a client to make multiple requests to the web server over a single TCP connection, thus amortizing the cost of connection establishment

and termination. While the access logs indicate whether the client used the HTTP/1.0 or HTTP/1.1 protocol, there is no explicit indication of which requests were grouped onto a single connection. To approximate the effect of persistent connections, we use the following rule to group requests into connections: any request received from the same client within 15 seconds of a prior request is grouped into a connection with that prior request. Since the timestamp in the access log has only a one-second resolution, we randomly distribute the arrival time of all HTTP/1.0 requests and new HTTP/1.1 connections within that second. If only one follow-up request in a persistent connection is logged in a second, we assume that it arrived at a random time within that second. We assume that multiple follow-up requests within the same second arrived at the server as a burst at a random time within that second. This models an aggressive client requesting some number of objects linked by a previously requested page, for example. However, the logs used in our work do not provide information which would let us model the structure or approximate the complexity of the pages being accessed.

The characteristics of our three workloads are summarized in Table 1. The 97%/98%/99% data size figures are particularly important for our study since they indicate the amount of memory needed to hold the unique files for 97%/98%/99% of all requests. For example, in the Olympics98 workload, 99% of requests could be served from memory using a memory cache size of only 141 MB. Since our web server has 1024 MB of memory, caching files in memory may significantly reduce the disk activity required to serve the workload. However, we should note that it is difficult to make a direct correlation between the 97%/98%/99% data size figures and the cache space required by the web server, since these figures were determined with complete knowledge of the request stream. On the other hand, the figures are for a "static" cache, and could be reduced even further if cache contents were dynamically managed. On balance, these figures should simply be viewed as indicators of what could be achieved with a reasonably effective caching strategy.

Since caching should significantly reduce disk activity for both the Olympics98 and NYSE workloads, we wanted to balance them with a workload that requires larger amounts of disk I/O. The Proxy workload has this feature. The total response size of the Proxy workload, at 10.2 GB, is less than twice as large as the total file size, indicating relatively little reuse of previously served data. Thus, it will have significant disk activity despite the use of almost all of the 1 GB memory on the machine as a file cache. Furthermore, even with a perfect caching strategy, only about 75% of the Proxy workload will fit into our server's RAM cache. As a result, it is unlikely that caching will significantly decrease disk I/O, and for many requests the server will be required to read the file from disk in order to generate the response. A more precise formulation of this point would require measuring or deriving the cache hit rate or, alternatively, the replacement rate, which is beyond the scope of this study.

Web serving workloads are notorious for having high variability [2], and these three workloads are true to form. As indicated in Table 1, the peak request rate measured in one-minute intervals is anywhere from 1.5 to 3 times larger than the average request rate. Moreover, each workload shows its own distinct pattern of request arrivals, which are illustrated in Figures 4, 5, and 6. These variability characteristics made it difficult to configure the hardware such that the HTTP servers and NFS servers were stressed, but still usefully operational. For example, if one configures the system to satisfy the average load, then it is hopelessly under-powered for peak periods and typically fails. On the other hand, configuring for the seldom-seen peak periods results in utilizations in the single percentage digits the rest of the day, making any comparison between "local data" and "remote data" systems difficult because the I/O load is so very small for each. However, we realize that this is "reality."

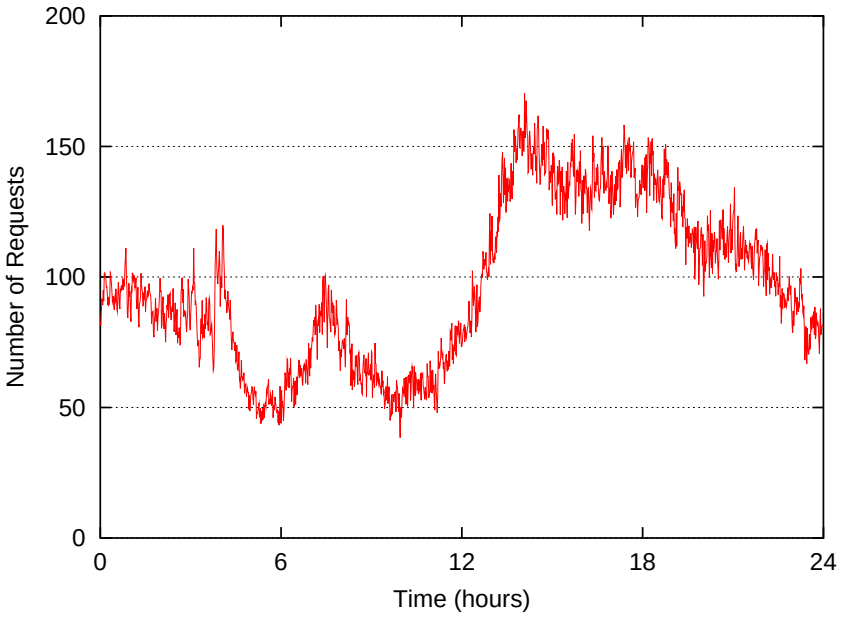

Figure 4: Request rate seen at one of the Olympics web sites on February 19, 1998

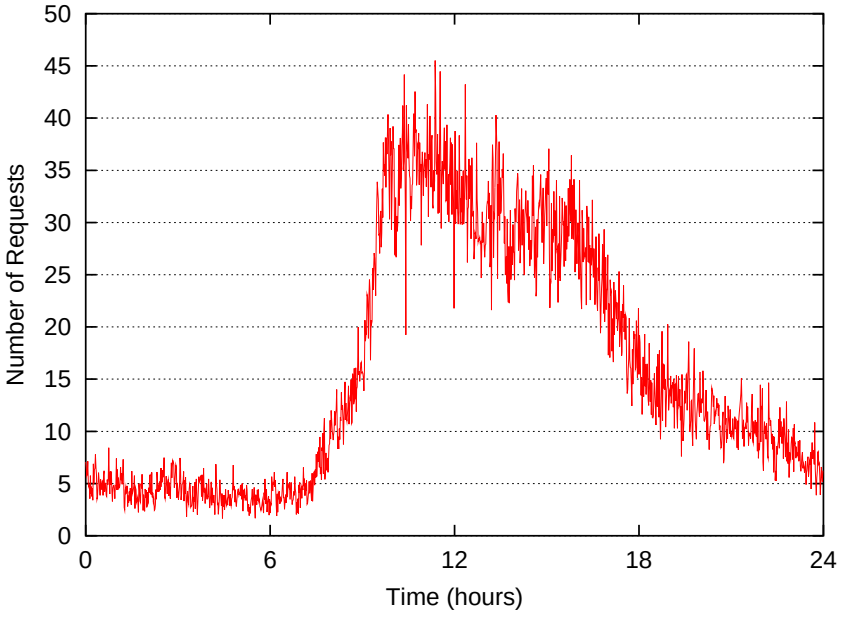

Figure 5: Request rate of a New York Stock Exchange server www.nyse.com on Oct. 19, 1999

Workload	Olympics98	NYSE	Proxy
Avg requests / sec	97	16	15
Peak requests / sec	171	46	30
Avg requests / conn	12	8.5	31
Files	61,807	16,872	698,232
Total file size	705 MB	171 MB	6,205 MB
Requests	8,370,093	1,360,886	1,290,196
Total response size	49,871 MB	2,811 MB	10,172 MB
97%/98%/99% (MB)	24.8 / 50.9 / 141	3.74 / 6.46 / 13.9	2,498 / 2,860 / 3,382

Table 1: Characteristics of three web server workloads. Average requests per second is the average request rate over the entire 24 hour period, and peak requests per second is the highest observed rate for a one minute period. The average requests per connection is based on our technique of grouping requests into connections. Files is the number of unique files requested, and total file size is the total size of these unique files. Requests is the number of distinct HTTP requests, and total response size is the total size of response data sent for these requests excluding HTTP headers. 97%/98%/99% data is the amount of memory needed to hold the unique files for 97%/98%/99% of all requests.

Real web serving workloads are not conveniently measured, nor are real web servers conveniently configured for highly variable loads. These workloads are valuable in this study because the challenges they pose are exactly the same as those that will be faced by web-serving sites considering diskless systems.

3.4 Replay Program

We use a modified version of the `httperf` workload generator [11] to generate an HTTP load on the web server. It uses an event-driven model to initiate, maintain, and close multiple connections to a client. The process described in the previous section converts an access log into a trace file that contains a sequence of connections, each consisting of a sequence of HTTP requests. The trace file specifies the time interval between the initiation of two successive connections and the time interval between the initiation of successive HTTP requests within a connection. We modified `httperf` to replay server access log by initiating connections at the appropriate times, generating requests at specified time intervals, and then closing the connections. In addition, we added the capability to scale the inter-arrival time of connections, but not requests within a connection, by a user-specified amount. A scale factor of "2×" corresponds to reducing the inter-arrival time of connections by 50%. Since each connection roughly corresponds to a client, reducing the connection inter-arrival time effectively generates a heavier client load, but with the same basic pattern of connection arrivals. The inter-arrival time of requests within a connection is not scaled since these times represent user think time or the network and client overhead involved in retrieving multiple components of a web page. We use this scaling mechanism to evaluate server performance for a range of client load intensities. Thus, we are able to scale the intensity of any workload to "1×", "2×", "2.5×", etc.

We discovered in some preliminary experiments that our replay program could encounter errors due to the Linux-imposed limit on the number of open file handles, We solved this problem by increasing

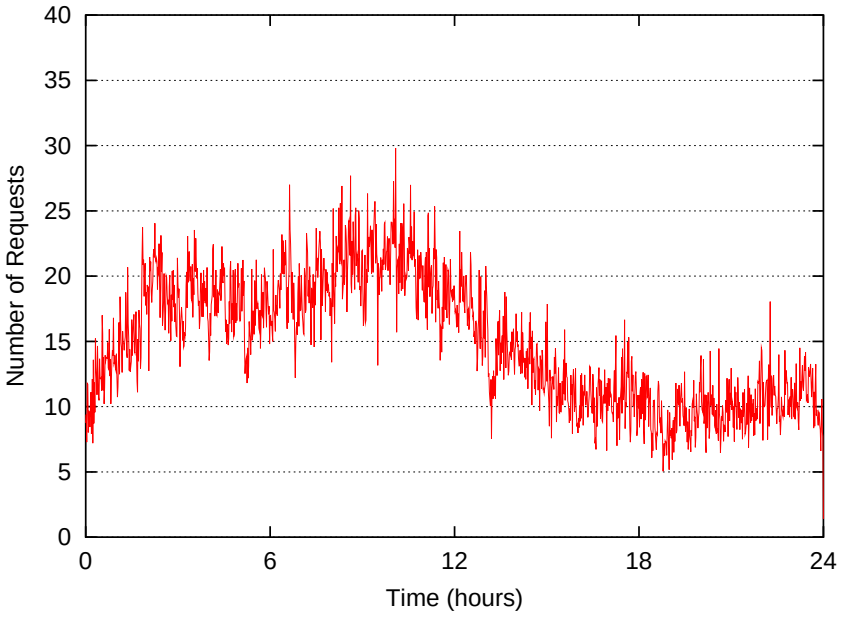

Figure 6: Request rate seen at IRCache Project Silicon Valley proxy server on May 2, 2001

the Linux limit on the maximum number of open file handles in `/proc/sys/fs/file-max` from 16K to 64K and setting the per-process limit to unlimited.

3.5 Metrics

"Responsiveness" can be measured and compared in a number of ways. There are a number of different intervals that we can define as "user response." Each web serving request consists of a connection (perhaps persistent), followed by a number (perhaps 1) of file requests. Each file request consists of a number of packets. The responsiveness of connection establishment is independent of whether the requested file resides on a local disk or is served over NFS: it depends on the load on the computational resources. The time elapsed to receive a file is also partially dependent on the computational load, but it is mostly determined by the size of the file. We observed that the time required to establish a connection is uniformly small, on the order of a tenth of a millisecond or less. Since the time to establish a connection does not contribute to a direct comparison between "local data" and "remote data" configurations, and because it must be allocated among one or more files, we chose not to include it in our responsiveness metric. File sizes range from a few tens of bytes to more than thirty megabytes in the case of the Proxy workload. In comparing the three workloads we chose not to use the entire file retrieval time for our metric, but rather the time required to retrieve the first packet of the requested file. Otherwise, very large files would contribute to the percentile metrics in a weighting proportional to their file sizes, rather than in proportion to the slower response of NFS. In other words, by choosing this responsiveness metric, we believe that we emphasize the impact of slower NFS file retrieval. In summary, the response interval we chose can be phrased as "the time elapsed since a file is requested until the first packet of that file is received by the client, given that a connection is already established."

The metrics we derive from each workload run for the above interval include the mean, the median, the

95th percentile and the 99th percentile. We initially chose the mean to compare, until we realized that the mean is much less sensitive to loading and to the "local access" versus "remote access" comparisons. For convenience, and to reduce the complexity of the comparison, we have chosen the 95th percentile as our primary metric of responsiveness to compare the three configurations. This is consistent with many forms of service-level agreements.

A further consideration is whether to use absolute or relative comparisons. Is it more meaningful to say "The 95th percentile of responsiveness for NYSE at 6X the actual load is 20 percent worse for the remote data case compared to the local data case" or "The 95th percentile increases by 0.44 milliseconds for the remote data case when compared to the local data case", when the metric value for local data access is 2.23 ms and is 2.67 ms for remote data access? Which to use depends on whether the differences matter to the usefulness of the system. A difference of 0.44 ms is not perceptible to users making requests of web pages, whereas a difference of 0.5 seconds is. Since many absolute differences are small, we choose to discuss our comparisons in terms of relative differences, while also giving the absolute values. Finally, all comparison plots include a line at the 0.5 second "threshold" of responsiveness. We do this since the range of values plotted is different for each workload, and the 0.5 second value makes it easier to compare the results between the different plots and different workloads.

4 Results

We ran each of our three workloads against a web server with all files on a locally attached disk, against one whose content was stored on a separate server and accessed over NFS and against one where all files were stored on separate servers and accessed over NFS (identified as "DSA" in the tables and graphs). Each workload was run for a number of scale factors in order to discover the knee of the responsiveness curve. Both the web server and NFS caches are flushed prior to the start of each run. The `httperf` replayer measures the elapsed time to create the TCP/IP session, denoted as "connecttime", and the elapsed time between sending a request to the server and receiving the first packet of the corresponding response, denoted as "timetogetfirstbyte", and records this information for each request in an output file. As explained above, our response time metric is the time to receive the first packet of the response, excluding the time to create the TCP/IP session, and thus only the timetogetfirstbyte value is included in our results. As already described, we report the mean, median, 95th percentile, and 99th percentile values.

Figures 7 through 9 present comparisons of the response time delivered for the three configurations on each of our three workloads. In these graphs, the x-axis indicates the load imposed on the web-server in average requests per second over the course of the run, and the y-axis indicates the 95th percentile response time delivered by the webserver. Complete response time measurements for all three workloads and scale factors are shown in Table 2.

The Olympics98 workload was run at request rates from 1 to 12 times the captured rate and shows a dramatic knee around 1000 requests/second. There is no perceptible difference in the 95th percentile responsiveness of the local versus the remote cases across the range of request rates, except near the knee of the curve. Note that the results reported here for runs during which request logging by the web server was disabled.

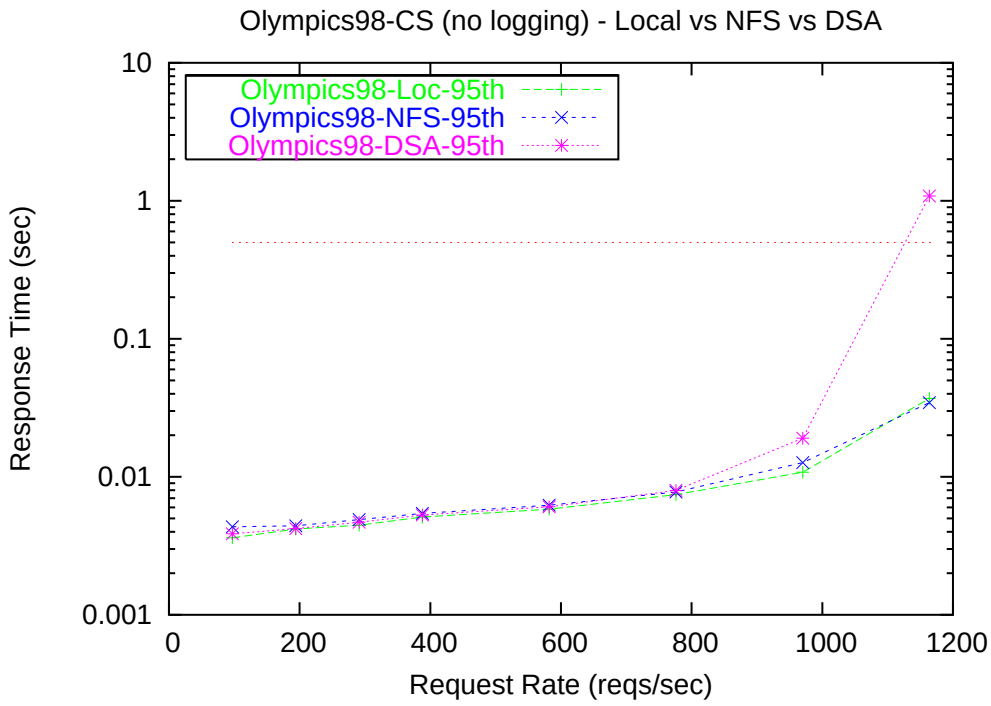

Figure 7: Response time comparison for the Olympics98 workload

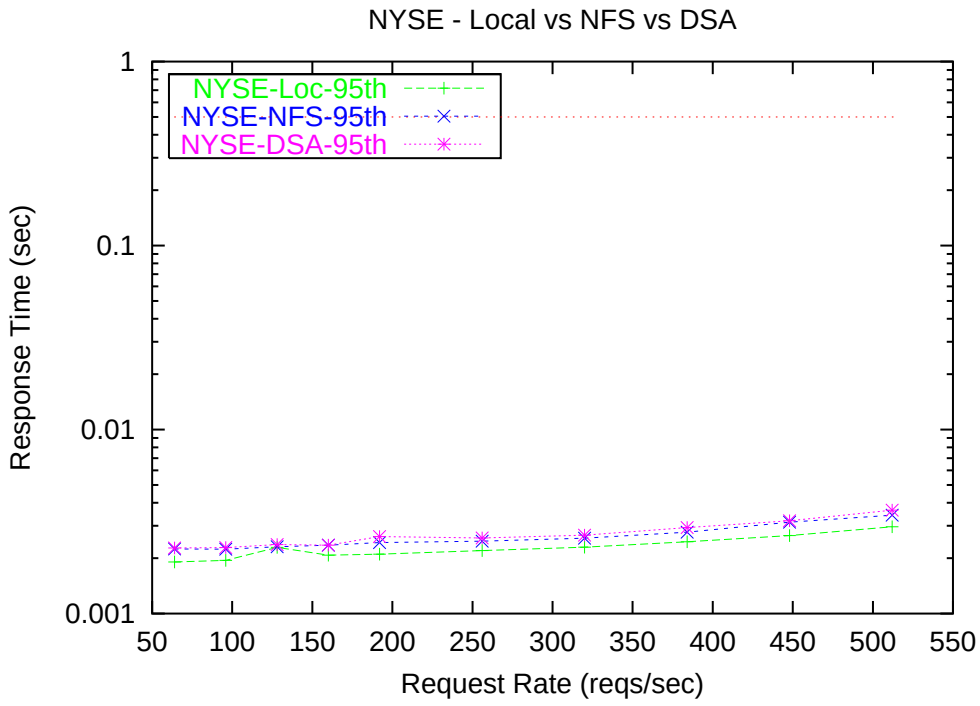

Figure 8: Response time comparison for the NYSE workload

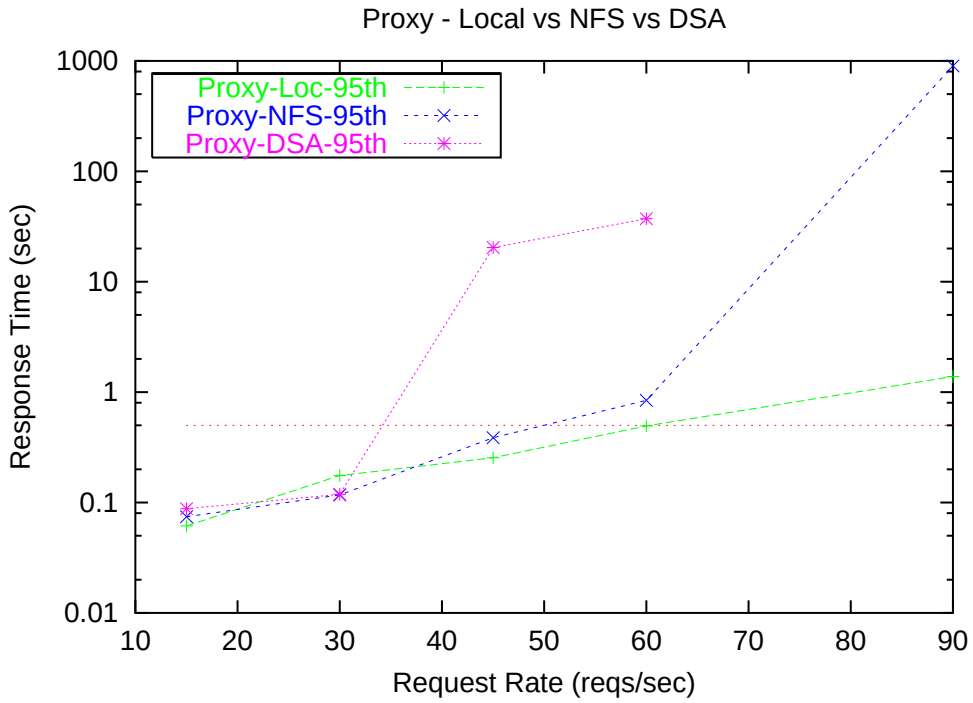

Figure 9: Response time comparison for the Proxy workload

Figure 8 plots the responsiveness of the NYSE workload at request rates from 1 to 32 times the captured rate and displays a slight knee beginning at 250 requests/second. Here the three curves are almost superimposed upon one another with no perceptible difference among them. Again, there is no perceptible difference in the 95th percentile responsiveness of the local versus the remote cases across a dramatic range of request rate intensities.

The Proxy workload results for average request rates from 1 to 6 times the (average) captured rate are displayed in Figure 9 which does not show a knee for local data but shows a dramatic degradation above 60 (average) requests/second for the remote data case while for the fully diskless case there is severe degradation above 30 average requests per second. For Proxy, a rate scale factor of 1.0 is a rate of 15 requests per second. Since we know from other measurements that almost all of the NFS activity during the runs is to fetch the content, it is not clear why the fully diskless configuration performs so poorly relative to the remote content case. In particular, the primary file I/O activity during the run other than content fetch is writing to the web server log files. However, the reason that Proxy fails at a scale factor of 6 for the fully diskless configuration is that the out-of-memory killer begins to run indicating a lack of sufficient memory to support the load being attempted. The fully diskless configuration that we used in these tests does not have any swapping capability, and it is quite possible that even at lower scale factors the absence of swapping capability combined with the pressure on memory is causing the system to discard pages that would otherwise be cached, resulting in the difference from the case with NFS-based content but local system files.

Scale	Request	Local				NFS				DSA			
Factor	Rate	MEAN	50%	95%	99%	MEAN	50%	95%	99%	MEAN	50%	95%	99%
1.0	97	2.315	0.896	3.610	10.70	2.908	1.070	4.341	11.76	2.049	1.030	3.866	9.056
2.0	194	2.352	0.963	4.190	11.95	2.573	1.113	4.422	10.50	2.506	1.056	4.217	10.23
3.0	291	2.741	1.001	4.455	11.98	3.061	1.152	4.892	12.04	2.890	1.100	4.694	11.63
4.0	388	3.119	1.075	5.108	13.30	3.632	1.195	5.434	13.38	3.406	1.148	5.290	13.16
6.0	582	3.855	1.061	5.827	16.03	4.900	1.195	6.224	16.29	4.026	1.148	6.064	16.08
8.0	776	5.014	1.188	7.430	23.84	6.551	1.308	7.766	26.14	4.957	1.283	7.969	27.89
10.0	970	7.485	1.338	10.80	47.23	21.68	1.500	12.65	104.3	72.58	1.501	19.02	1,444
12.0	1164	73.33	1.689	36.97	629.2	228.2	1.814	34.52	2,085	671.2	2.206	1,084	10,742

Olympics98-CS workload

Scale	Request	Local				NFS				DSA			
Factor	Rate	MEAN	50%	95%	99%	MEAN	50%	95%	99%	MEAN	50%	95%	99%
4.0	64	1.368	0.767	1.908	5.615	1.489	0.836	2.245	6.451	1.479	0.855	2.270	6.381
6.0	96	1.603	0.770	1.945	6.163	1.680	0.831	2.237	6.986	1.775	0.848	2.285	7.115
8.0	128	3.606	0.827	2.292	16.79	2.064	0.839	2.305	7.663	2.004	0.858	2.373	7.512
10.0	160	2.167	0.802	2.076	7.465	2.163	0.850	2.356	8.105	2.152	0.848	2.353	8.018
12.0	192	2.063	0.811	2.103	7.413	25.53	0.858	2.432	10.02	5.387	0.858	2.623	112.3
16.0	256	2.572	0.811	2.199	8.317	2.652	0.858	2.472	9.084	2.653	0.890	2.576	9.104
20.0	320	2.537	0.826	2.294	8.425	2.370	0.873	2.568	9.223	2.449	0.906	2.668	9.051
24.0	384	2.843	0.847	2.453	9.179	3.095	0.909	2.767	11.57	3.445	0.940	2.935	12.16
28.0	448	3.224	0.884	2.653	9.799	3.541	0.952	3.136	16.25	3.701	0.971	3.195	12.75
32.0	512	4.022	0.933	2.963	21.85	3.954	0.988	3.424	22.21	4.118	1.035	3.643	19.46

NYSE workload

Scale	Request	Local				NFS				DSA			
Factor	Rate	MEAN	50%	95%	99%	MEAN	50%	95%	99%	MEAN	50%	95%	99%
1.0	15	23.31	6.686	61.45	423.3	24.88	7.743	74.49	340.6	39.42	6.677	87.75	1,053
2.0	30	37.08	8.330	175.5	602.5	26.87	8.747	116.9	384.3	28.37	8.713	118.7	401.8
3.0	45	43.67	8.703	254.1	669.0	67.40	11.60	386.9	837.4	3,251	14.20	20,436	67,815
4.0	60	97.71	17.03	493.2	1,078	193.5	39.59	841.1	1,920	5,665	58.62	37,225	102,363
6.0	90	325.2	43.55	1,382	3,890	233,299	133,254	899,826	1,300,550	N/A	N/A	N/A	N/A

Proxy workload

Table 2: Measured response times (in milliseconds) for our three workloads

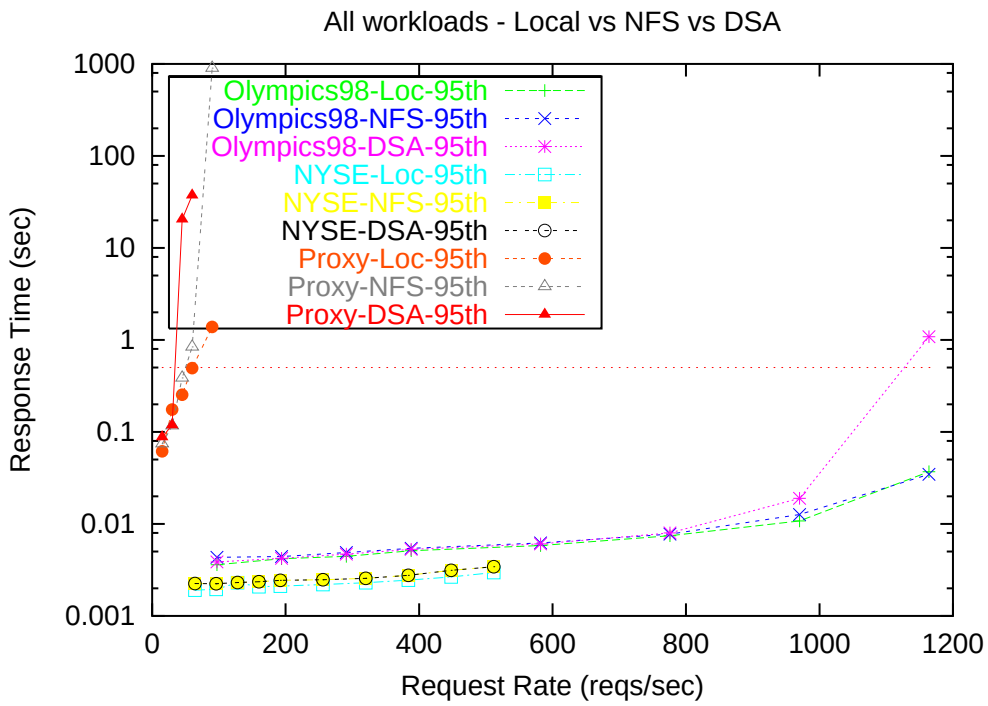

Figure 10: Response time comparison for the three workloads versus request rates

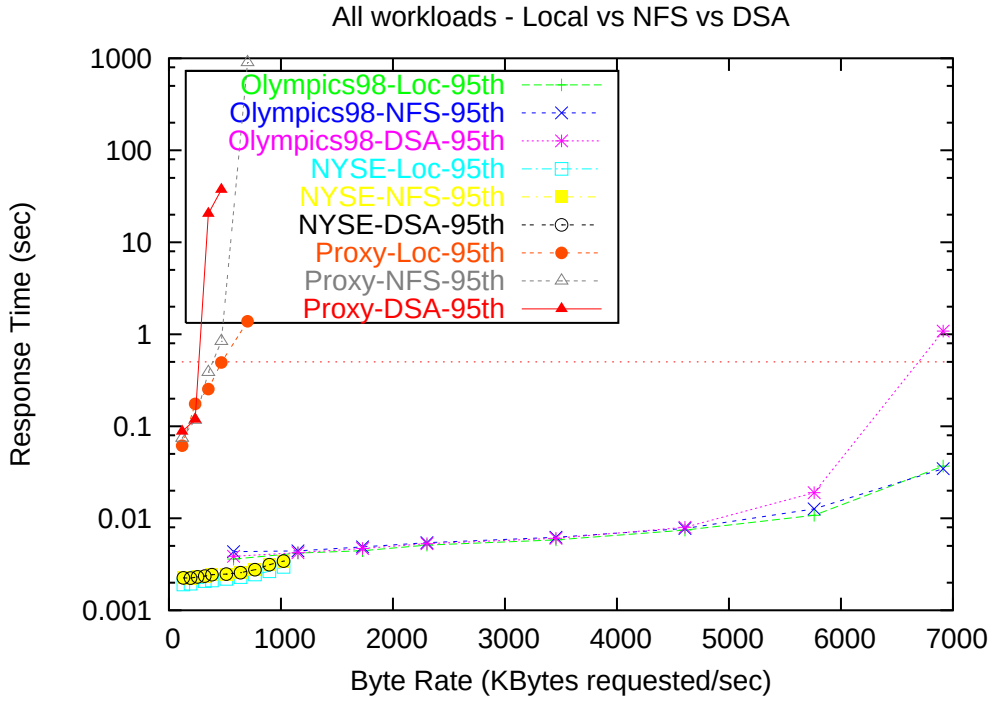

Figure 11: Response time comparison for the three workloads versus bandwidth requirement

Figure 10 illustrates the relationship between responsiveness and the rate of requests handled by

the web server for all of our configurations. In this graph, the x-axis represents the average request rate for the workload and the y-axis is the 95th percentile response time for the workload. Note that the curves for the Olympics98 and NYSE workloads have very similar shapes, with the knee of the Olympics98 workload shifted in its relative location in the curve to a point about 50% higher from its location in the NYSE workload. The reason for this shift is most likely the much larger variability in the request rate of the NYSE workload, where the peak request rate is about 3 times the average request rate, compared to the Olympics98 workload, where the peak request rate is only 1.7 times the average request rate. The curves for the Proxy workload do not show any strong similarity to the NYSE and Olympics98 workloads, since the vast majority of the Proxy requests do not hit in the web server cache, and thus result in actual disk I/O. However, for the purpose of comparing the performance of the three configurations, this graph demonstrates that there is very little difference among them and that those differences that do exist are most pronounced for the Proxy workload, as would be expected.

In order to determine the correspondence between responsiveness and bytes transferred per second, an additional plot is presented. In Figure 11 we see that the knee of the Proxy workload on the DSA configuration occurs at 500 KBytes/second, while for the Olympics workload it occurs at 5500 KBytes/second. There is no knee in the NYSE workload under different loadings, and there is no difference in the Olympics98 remote content and local content performance for different average byte rates.

If requests per second were the single limiting factor of the system, then all three workloads would show a knee in responsiveness at roughly the same value of it. They do not. Likewise, if requested bytes/second were the single limiting factor of the system, then all three workloads would show a knee in responsiveness at the same value of it. They do not. This analysis allows us to conclude that the performance limiting factors of the web server configurations are not simply characterized and that there are a number of interacting features that affect their performance limitations.

5 Analysis for SPECweb99

We have shown in previous sections that web server performance is not significantly impacted by diskless operation for two workloads derived from real internet web sites, and also shown that for a worst-case workload (Proxy) the performance degradation of diskless operation is acceptable. In this section, we perform an analytical study of the performance implications of diskless operation for a known and important web-serving benchmark. Of possible candidates considered, SPECweb99 was chosen because it has the most history, applicability, and widespread acceptance among general-purpose server manufacturers. Our analysis relies heavily on the study of SPECweb99 file placement by Keller, Sankaralingam and Hofstee [8], and extends this work by considering how the placement of of data on local or remote filesystems will affect performance.

5.1 The SPECweb99 Benchmark

SPECweb99 is the latest benchmark for web-serving performance from the SPEC consortium. It is a throughput benchmark. One or more clients drive an HTTP server with requests in a cycle of ”sleep – HTTP request — wait for response”. Requests take the form of either static file serving requests (static GET), dynamic file serving requests (dynamic GET) or logging (POST). In POST requests a cookie is returned by the server, instead of an unchanged file (static GET) or a file generated from a small num-

ber of static files (dynamic GET). Each request results in an operation. Server throughput is measured in operations per second given that a bandwidth requirement is met for each connection. The sleep interval in the client cycle is dynamically varied such that between 40,000 and 50,000 bytes/second per connection are maintained. A connection that meets the bandwidth requirement is counted towards the publishable metric, the number of conforming connections. This bandwidth requirement and the way it is implemented also enforces a de facto response-time requirement of 0.03 seconds per operation.

Static GET requests comprise 70% of the requests, while dynamic GETs comprise 25.2% of the requests with the remaining 4.8% of requests being POSTs. The dynamic GETs return a variety of slightly modified files. Regardless of whether the GET request is static or dynamic, the contents of the request are drawn from a fixed inventory of files. While dynamic GETs are computationally intensive, they add only about 6KB to the size of the returned file. We ignore POSTs in this analysis and concentrate on the static and dynamic GETs which comprise 95.8% of the workload and which fully characterize the file caching demands placed by the workload.

The composition of the files to be served is fully specified by the SPECweb99 benchmark in [17]. We will be brief in our characterization, only touching on those points relevant to this analysis. The set of files is composed of directories. Directories are identical in terms of file counts, sizes and frequencies of access within the directory. Each directory contains 36 files divided into 4 classes of 9 files each. Class 0 contains the smallest files, which range in size from 0.1 KB to 0.9 KB while Class 3 contains the largest files, ranging in size from 100 KB to 900 KB.

A file is uniquely identified by the triple of its Directory Level, Class Level and File Level. The frequency of accesses to the 9 files within each class is fixed and stays the same across all classes. The frequency of access to any particular file can be calculated as the product of (1) the frequency of access to its Directory Level, (2) the frequency of access to its Class Level and (3) the frequency of access to its File Level. The Class Level and File Level frequencies are found in [17]. All frequencies are defined by Zipf distributions with the characteristic alpha set to 1. For a set of files composed of N directories, indexed 1 to N, the probability of accessing directory i is NORM/i, where NORM is the sum of all access probabilities.

Directory Level 1 has the greatest frequency of access while Directory Level N has the smallest frequency of access. For a configuration with 1000 directories, this difference is 1000 to 1. This is a very skewed distribution. The distribution of file sizes and frequencies of access within each directory (while identical for each directory) is also skewed, with the smallest files generally receiving the greatest frequency of accesses. Regardless of the size of the file set, the mean, frequency weighted, size of a file is 14.4 KB, while the median is around 3 KB. The size of each directory is nearly 5.0 MB.

The number of directories comprising the file set is specified by SPECweb99 as a linear function of the number of target connections. The number of achieved connections is the measure reported for the benchmark result and, for all practical purposes, is the number of target connections. Hence, we will only refer to "connections" for the remainder of this paper. This relationship is given by: N = 25 + (40/61)*C, where N is the number of directories and C is the number of connections. We observe that this means that a 4 GB address space can fully cache a file set for 3,800 connections.

5.2 Effect of Diskless Operation

Having introduced the benchmark, we next determine the extent that I/O performance can impact the benchmark ratings. If disk I/O is the bottleneck for a configuration, then the difference in SPECweb99 ratings between local and remote disks will be determined by the relative difference in I/O performance. If disk I/O is not the bottleneck, then it does not matter what the relative difference in I/O performance is, given that the difference is not so great as to shift the bottleneck to disk I/O. In the rest of this section, we will examine the other hardware components of web-serving systems and discover that the relative difference between local and remote disk I/O performance for a SPECweb99 workload is small.

The performance-determining resources within the webserver are network bandwidth, RAM size, disk latencies and bandwidth, and CPU speed. A successfully configured SPECweb99 webserver typically:

- achieves high network bandwidth by multiple network interface cards,
- is configured with as much RAM as possible,
- is configured with a large number of high speed SCSI disk drives, and
- is configured with one or two high speed CPUs.

IBM, Dell, HP and others today (2001-2) offer SPECweb99 connection ratings of 1800 connections for one CPU and 3200 connections for two CPU systems, with a typical configuration being:

CPU speed:	1.2GHz
RAM size:	4 GB
Number of 1Gbit NICs:	4
Number of high-end SCSI disks:	6-8

Next, let us assume a system in which "delivered" disk bandwidth is the system bottleneck so that we can evaluate the difference between systems with local and remote disks, for if delivered disk bandwidth were not the limiting factor, there would be no difference between local and remote disks. We define "delivered disk bandwidth" as the peak, sustainable, measurable bandwidth of a disk(or disks) under a SPECweb99 I/O loading. This bandwidth metric includes disk latency, and is what is meant by "disk bandwidth" henceforth. By definition, if we are disk-bandwidth bottlenecked, then the CPU(s) and NICs are fast enough.

Since SPECweb99 is a throughput benchmark, the network throughput can be estimated directly by using the quantity of 320,000 bits/second per connection, which is defined by the benchmark. In practice, this is rounded to a "rule of thumb" of 400,000 bits/second/connection. So 1800 connections implies 700 Mbits/second of sustained network throughput, while 3200 connections implies 1.3 Gbits/second of sustained throughput. In our laboratory, we have taken measurements that show that the maximum throughput of a 1 Gbit commodity Ethernet NIC is 0.8 Gbit/second for large file transfers. This will diminish somewhat for SPECweb99, which serves different file sizes. Thus one or two 1 Gbit network attachments are sufficient for the one- or two-CPU configurations described earlier.

The RAM size and the efficiency of file caching in RAM are critical to performance. Disks hold files too large to cache in RAM, and the continually updated webserver log is written to disk.

We analyze the impact of "diskless" by estimating the performance contribution of RAM-based files versus disk-based files, then determine the difference that local versus remote disks make to SPECweb99

local disk sustained throughput	34 MB/s
NFS disk sustained throughput	22 MB/s
local disk sustained throughput Class 3 files	17.7 MB/s
NFS disk sustained throughput Class 3 files	14.8 MB/s

Table 3: Measurments of Local and Remote Disk Throughput

performance. We will use today's competitive configurations, cited above, as our fixed point. A 1,000-connection SPECweb99 has a database of 3.139 GB, by definition of the benchmark, which sizes the database linearly with the SPECweb99 rating in connections. Thus our 1,800- or 3,200-connection systems' databases are on the order of 6 GB and 10 GB, respectively. So our fixed point's 4 GB RAM system uses its disk only for the most infrequently accessed files after the RAM cache is warmed up (which it is, under the benchmarking rules). If the entire 4 GB is used for caching, then "only" 1,270 connections can be sustained by the cache, if all files are cached in RAM. We will next see how this can be improved to over 1,700 connections if a local disk system can contribute connections, via an intelligent caching strategy. Because of the skewed sizes and access pattern, about 1% of the accesses go to files comprising 90% of the size of the database. A near optimal file allocation strategy places some of the largest files in cache with the remainder on disk. [8] discusses how an optimal static partitioning of files between RAM and disk results can be made, given RAM size and disk I/O bandwidth. But for this analysis we will use a simpler, more tractable analysis of the partitioning in order to estimate the difference between local and remote I/O systems. We estimate the theoretical number of connections sustainable based on

- the amount of RAM, and
- the deliverable disk bandwidth.

Assume that RAM is packed with all class 0, 1 and 2 files and (some) class 3 files while the remainder of the class 3 files are marked non-cacheable and are served by the disk. We want to make a comparison of local versus remote disk operation. At one extreme, if we have no disk, then all files must be placed in RAM. For example, 1 GB of RAM holds the database for approximately 320 connections. 4 GB of RAM holds the database for 1270 connections. Adding a local disk allows us to migrate some (or all) of the class 3 files to disk and improve our connection rating. The lower the latency and faster the disk drive, the higher the connection rating.

How do we estimate the deliverable disk bandwidth for this "Class 3" workload? We simply measure the sustainable throughput (in MB/second) of:

- a local drive, then
- a remote drive.

The rates in Table 3 are derived from local and remote NFS transfers of a 33MB file for deliverable throughput and a sequence of 1,999 unique accesses to unique files of size and access frequency corresponding to SPECweb99 Class 3 files. These 1,999 files totaled 1,000 MB in size. The sustained throughput case performs `cp VeryLargeFile /dev/null` where VeryLargeFile is 32 MB in size and takes 0.946 seconds and 1.47 seconds for local and remote cases, respectively. The results for the Class 3 files are obtained by executing a script of 1,999 requests of the form `cp Filename /dev/null`, for

1,999 unique Class 3 files. In the "local" case the filesystem is local, while in the "remote" case the filesystem is mounted over NFS. The completion times for the script are 56.6 seconds and 67.6 seconds for the local and remote cases, respectively.

As mentioned previously, each connection requires "roughly" 400,000 bits/second sustained throughput (the 320,000 minimum plus some overhead), or 0.05 MB/sec. Therefore, if a connection consisted of Class 3 files, the throughput of one NFS disk at 14.8 MB/second would sustain 14.8/.05, or 296 connections. More accurately, a throughput of 14.8 MB/second dedicated to Class 3 files enables the remaining classes of files for 296 connections to be served from RAM. Per [8], page 16, the average access size is 14.2668 KB, of which Class 3 contributes 4.8950 KB, so the total number of connections sustainable is 14.2668/4.8950 (or 4.9) times the 296 connections, 1450 connections. Of these 1450 connections, 296 are "sustained" from the single, remote disk while the remaining 1154 are sustained from RAM. Similar calculations for the local, single disk result in 354 connections with a total number of connections of 1734.

How much disk and RAM space is required to serve these connections? From [8] we know that each 1,000 connections requires a database of size 3.2 GB for all files of which 2.8 GB is for Class 3. Since all Class 0, 1 and 2 files are locked in RAM in our scheme, a RAM cache of 0.51 GB is required for 1450 connections and 0.60 GB for 1734 connections. Similarly, the total size of the Class 3 files residing on disk are 4 GB and 4.9 GB for the NFS and local disk cases, respectively. Neither of these sizes are potentially limiting factors. For the above case, the relative difference in SPECweb99 ratings is in the same ratio as the remote to local disk transfer speeds, or 17.7 MB/s to 14.8 MB/s, or a 16% degradation in SPECweb99 ratings.

In an optimal configuration this relative difference is decreased, since the remaining unused RAM can be employed to cache the most frequently accessed Class 3 files (per [8]) thereby increasing the number of connections sustained by RAM while the number of connections sustained by disk remain at 296 and 354 for NFS mounted and locally mounted single disks, respectively. In other words, the fraction of connections sustained by RAM increases relative to disk, as the most frequently accessed Class 3 files are cached in RAM. As this fraction increases, the relative difference in sustained SPECweb99 connections between different I/O rates diminishes. We note that our analysis applies to the NFS configuration but not the DSA configuration. The difference that writing a log over NFS plus doing very infrequent swaps over NFS makes is not known although it is thought to be negligible.

While we could redo the analysis done in [8] to include these larger and more realistic amounts of RAM, and use the measured disk transfer rates and latencies for locally and NFS mounted disk drives, there is no reason to do so simply to belabor the point that the 16% relative difference would be decreased to some smaller value. A relative difference of 16% is acceptably small.

6 Future Work

We plan to perform additional measurements and analysis of diskless servers. In particular, we intend to explore the following issues:

- The poor performance of the fully diskless environment on Proxy at higher scale factors. We plan to investigate the specific reasons why Proxy does poorly at scale factors above 3 and fails entirely at scale factor 6 on the purely diskless system. In particular, we plan to use our implementation

of swapping for the diskless environment to determine if the problem is due to the absence of swap.

- Root causes of the other performance deltas. We plan to use additional performance metrics (e.g. CPU, disk, and network utilization, statistics on Apache and NFS) to identify the reasons for the differences among local and remote content configurations and the fully diskless configuration. Where possible, we hope to discover what limits the performance of each of the configurations and to find explanations for the differences that we observe among them.
- The effect of using true network-attached storage rather than a commodity, general-purpose NFS server since it offers a more realistic environment.
- Alternatively, more efficient data access protocols to determine what effect their use has on our results.
- Benefits/costs of the diskless model for other server applications, such as web application servers, firewalls, workload balancers, and other "appliance-like" applications.

7 Related Work

Although there are a number of reported uses of small-scale web servers in the embedded environment to support such things as configuration, management and diagnostics ([18], [20], [21]) this study considered the classical use of web servers to provide general network access to information, in this case, static information. To our knowledge, this use of a diskless configuration has not been studied in the past. However, there have been a number of studies of over the years of both diskless workstation environments and distributed file system performance.

One of the earliest studies of diskless workstation performance was a set of measurements done on diskless workstations, using three different operating environments, in the middle 1980s by Lazowska, Zahorjan, Cheriton and Zwaenepoel [9]. They concluded that diskless workstations could be made to work acceptably with the right protocol design. A study of a later diskless workstation environment was done by Nelson, Welch and Ousterhout using the Sprite system and appeared in 1988 [12]. More recently, there have been some newer studies of distributed file system performance and diskless workstations such as that done by Baker et al using Sprite that appeared in the early 1990s [1].

NFS and its performance have often been criticized. One of the most important early papers on NFS performance was written by Howard, et al, to compare the performance of NFS with AFS: it appeared in 1988 [5]. Although very interesting and historically very important, it too covers a workstation environment, and the speeds and sizes that it uses are two to three decimal orders of magnitude smaller than those in common use today. More recently, there have a number of attempts to quantify NFS performance for competitive reasons, and there are commercial benchmarks for NFS such as LADDIS [22] and its successors, SFS [16, 15], from SPEC. These benchmarks are used by both system vendors and the manufacturers of network-attached storage systems to back their performance claims. Finally, there is a recent and very interesting piece of work by Culler and Martin [10] that deals with how sensitive NFS performance is to the underlying network performance. They offer both empirical results and an analytic model.

Our work differs from these efforts in several important ways. First, it is an empirical study of the use of NFS in a web server environment where the pages being delivered to the client are fetched over NFS

or where all disk I/O operations are done over NFS. Second, it measures the performance of the web service rather than that of NFS. This is significant because even though NFS and local disk access may have very different performance characteristics, if they have no material effect on the performance of web page delivery, they can be safely ignored.

8 Conclusions

The results from our performance tests of diskless web servers across three real workloads can be summarized in the following.

- For all three workloads, running at their original speed, having the content on a local disk or over NFS makes little difference in responsiveness.
- For all three workloads, running at their original speed, using a fully diskless web server makes little difference in responsiveness.
- For the New York Stock Exchange shareholder information server workload, which has a relatively small footprint and fits easily in the web server's RAM cache, there is little difference in responsiveness among all of the configurations even when the workload executed at 32 times its original rate, at which time the local disk configuration's responsiveness becomes unacceptable.
- For the Nagano 1998 Olympics workload, which also has a data footprint that fits entirely in the web server's RAM cache, there is little difference in responsiveness among all the configurations at rates up to 8 times the original load, above which the DSA case degrades. There is little difference in responsiveness between the local content and NSF content configurations even at a loading that is 12 times the original load.
- For the Proxy workload, which has a multi-gigabyte footprint and does not fit in the RAM cache, there is increasing degradation in responsiveness as the load increases. At the loadings of 1X and 2X both "diskless" cases show 95th percentile responsiveness relative degradation of roughly 25% and 40%, respectively. The remote content configuration survives the 3X and 4X loadings with 52% and 75% relative degradation, respectively, with disastrous responsiveness at a loading of 6X. The DSA configuration degrades catastrophically at 3X loadings and above.
- For SPECweb99, our analysis shows that the relative difference in performance ratings between the local and remote content configurations, for single disk configuations, should be less than 16%.

Since we have bracketed static content workloads by two "best cases" and one "worst case," and found that content served over NFS and full diskless operation using "worst case" remote file servers results in slower, but acceptable performance, we believe that most real workloads suffer little to no performance impact when the web server data is located on network-attached storage. Similarly, we believe that most real workloads suffer little to no performance impact when the web server is diskless and all files are located on network-attached storage or other, specialized network-storage equipment. In light of the benefits of this structure, we believe that it is appropriate for many ISP and ASP installations.

A discussion of the practicality of diskless web servers is incomplete without examining the additional cost of remote file servers. We have compared configurations of one HTTP server versus one HTTP server plus one NFS server for content and one for other files if required. The question arises "How many HTTP servers can a remote file server support?" If the cost of a remote file server is amortized

against a small number of HTTP servers then the practicality of diskless web servers diminishes, while a large amortization number favors the application of diskless web servers. We have not yet determined this number by direct experiment. However, an understanding of the characteristics of the web serving benchmarks reduces this concern. Both the Olympics98 and the NYSE workloads require little or no NFS service once the working set of the data is cached in the HTTP server. It is for this reason that the performance of the remote data case and the performance of the local data case are virtually identical across a range of loadings. This means that an NFS server should support a large number of HTTP servers, which strengthens the case for diskless web servers. The Proxy workload makes a heavy demand on NFS, since the HTTP server cache can never grow warm. This is a worst case test for diskless web servers. The loading placed on the remote data server is high. Examining Figure 9 shows evidence of this. We would expect the configured NFS server to support only one HTTP servers.

SPECweb99 is CPU-bound for today's single-CPU configurations, where 6 to 8 high-speed SCSI drives are used. If the number of disk drives in these configurations is reduced to one, we expect these systems to become disk bottlenecked, offering lower SPECweb99 ratings. In this benchmark the effectiveness of the file caching mechanism is a crucial determiner of the load placed on the disk system. For example, the Apache webserver is notoriously ineffective on SPECweb99 since it uses the operating system's file caching mechanism instead of supplying its own. This means that each time an infrequently accessed and very large Class 3 file is brought into the cache, many small, frequently accessed files are discarded and thrashing results. By contrast, the Zeus [23] webserver, offers a single "size of the largest file to be cached" parameter that can be set to prevent this unfortunate occurence. Thus, the relative difference in performance ratings of local versus remote content configurations may be better or worse than the "worst case" Proxy workload, depending upon the effectiveness of the file caching mechanism used by the webserver. In any case, the number of HTTP servers supported by an NFS server to obtain our 16% relative limit is 1.

We believe that the results reported here support the argument that diskless web servers are practical. We have demonstrated that in two important, "real" workloads, diskless web servers offer the same level of responsiveness as "diskful" web servers, even at loadings as high as 32 times the original, measured load and even when a notoriously inefficient remote file system, NFS, is used. When given a worst case test, a web proxy cache workload, performance of the diskless case is acceptable over a 24 hour period at original, measured loadings. Eliminating disks from web servers allows the volume, power and cooling loads consumed by disk drives to be utilized for higher densities of diskless servers.

References

[1] M. Baker, J. Hartman, M. Kupfer, K. Shirriff, and J. Ousterhout. Measurements of a distributed file system. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 198–212. Association for Computing Machinery, 1991.

[2] M. Crovella and A. Bestavros. Self-similiarity in world wide web traffic: Evidence and possible causes. In *1996 SIGMETRICS Conference on Measurment and Modeling of Computer Systems*, May 1996.

[3] Extreme Networks. Blackdiamond 6808 data sheet. http://www.extremenetworks.com/products/datasheets/bd.asp, 2000.

[4] M. Hill et al. Design decisions in spur. *Computer*, 19(11):8–22, November 1986.

[5] J. Howard, K. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. In *ACM Transactions on Computer Systems*, pages 51–81, February 1988.

[6] The IRCache project. http://www.ircache.net/. This project is supported by the National Science Foundation (grants NCR-9616602 and NCR-9521745), and the National Laboratory for Applied Network Research.

[7] A. Iyengar, M. Squillante, and L. Zhang. Analysis and characterization of large-scale web server access patterns and performance. *World Wide Web*, 2(1-2):85–100, June 1999. also www.research.ibm.com/people/i/iyengar/www99.ps .

[8] T.W. Keller, K. Sankaralingam, and H.P. Hofstee. Towards an optimal file allocation strategy for SPECweb99. In L.K. John and A.M.G. Maynard, editors, *Workload Characterization of Emerging Computer Applications*. Kluwer/Plenum Series in Computer Science, 2001.

[9] E. Lazowska, J. Zahorjan, and D. Cheriton. File access performance of diskless workstations. 4(3):238–268, 1986.

[10] R. Martin and D. Culler. NFS sensitivity to high performance networks. In *SIGMETRICS '99/PERFORMANCE '99 Joint International Conference on Measurement and Modeling of Computer Systems*, May 1999.

[11] D. Mosberger and T. Jin. httperf: A Tool for Measuring Web Server Performance. In *SIGMETRICS First Workshop on Internet Server Performance*, pages 59—67. ACM, June 1998.

[12] M. Nelson, B. Welch, and J. Ousterhout. Caching in the sprite network file system. 6(1):134–154, 1988.

[13] J. Ousterhout et al. The sprite network operating system. *Computer*, pages 23–36, February 1988.

[14] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, P. Winterbottom, and P. From. Plan 9 from bell labs. *USENIX Computing Systems*, 1995.

[15] D. Robinson. The advancement of NFS benchmarking: SFS 2.0. In *LISA '99: 13th Systems Administration Conference*, November 1999.

[16] Standard Performance Evaluation Corporation. SPEC SFS97 (2.0) benchmark. Available at http://www.specbench.org/osg/sfs97, 1997.

[17] Standard Performance Evaluation Corporation. An explanation of the SPECweb99 benchmark. Available at http://www.spec.org, 1999.

[18] Sun Microsystems. http://www.sun.com/software/embeddedserver/.

[19] The World Wide Web Consortium (W3C). RFC 2068: Hypertext transfer protocol – HTTP/1.1, January 1997.

[20] Sid Wentworth. Boa: An embedded web server. *Embedded Linux Journal*, July 2001.

[21] Nick Witchey. Designing an embedded web server. *IEEE Internet Computing Online*, 2(3), May/June 1998.

[22] M. Wittle and B. Keith. LADDIS: The next generation in NFS file server benchmarking. In *USENIX Summer*, pages 111–128, 1993.

[23] http://www.zeus.com/.