Proceedings of the FAST 2002 Conference on File and Storage Technologies

Monterey, California, USA
January 28-30, 2002

Obtaining High Performance for Storage Outsourcing

Wee Teck Ng, Bruce Hillyer, Elizabeth Shriver, Eran Gabber, Banu Özden
Information Sciences Research Center
Bell Laboratories
600 Mountain Avenue, Murray Hill, NJ 07974
{weeteck, bruce, shriver, eran, ozden}@research.bell-labs.com

Abstract

Storage outsourcing is an emerging industry that shields storage users from the complexity of in-house storage management, while providing cost savings and reliability improvements via the aggregation of storage into large, special-purpose facilities. These distributed and replicated facilities are operated by a storage service provider, and are accessed by remote users via high-speed network connections.

The viability of storage outsourcing is critically dependent on the performance of remote storage. In this paper, we measure the performance of I/O benchmarks accessing a remote block-level storage system. We use benchmarks that represent a variety of workloads, running on several operating systems and file systems. Network latencies represent distances ranging from a local neighborhood to halfway across a continent. We vary the network loss characteristics to correspond with the conditions of either dedicated fiber or shared Internet (with loss rates up to 10^{-3}). We examine the effectiveness of basic latency-hiding techniques such as caching, application prefetching, and asynchronous writes. We conclude that remote storage is already viable for a wide variety of active workloads, and we point out areas where new techniques could provide significant additional performance enhancement.

1 Introduction

Storage management is complex and expensive. For example, IDC estimates that for every dollar spent on storage equipment, an additional $6 will be spent on managing the storage [16]. This includes the expert help required to configure the storage systems (e.g., host, RAID, and SAN configuration, cabling, and cooling), to administer it (backup and restore), and to manage it for high availability (capacity planning, disaster recovery). These problems are motivating the emergence of storage service providers, who sell data storage as an outsourced business service. Among the major storage service providers are traditional computing system suppliers (e.g., IBM, HP), telecommunication vendors (e.g., Qwest), and startups such as StorageNetworks Inc. (www.storagenetworks.com).

Remote storage has a long history for applications such as distributed databases and FTP archives, but dramatic improvements in the price and availability of high-speed networking suggest that a much broader scope of applications may thrive in an environment of outsourced storage and commercial storage service providers. The viability of the emerging storage outsourcing industry depends, in part, on whether acceptable performance can be obtained from remotely-accessed storage, but the open literature lacks technical data on the performance of remote storage systems. Can remote storage substitute for host-attached disks, nearby storage area networks, or LAN-based network-attached storage servers?

To explore this question, we measure a variety of benchmarks widely used in the file system and database communities. Our experimental platform consists of PCs, a fiber-based gigabit Ethernet, a router testbed, and our own SCSI over IP implementation, which predates standards, but is largely comparable to iSCSI. We measure benchmarks on an accepted kernel tool (the FreeBSD dummynet package) that introduces “network delay” and loss into the protocol stack. On this platform we measure benchmarks when network propagation delay ranges up to 8 ms, corresponding to 1600 km of fiber. We also investigate the performance of outsourced storage under the network delays and packet losses characteristic of the Internet, using a testbed that consists of two Cisco routers with an OC-3 backbone, and a pair of Smartbits generators that generate background traffic consistent with a traffic profile derived from recent Internet traffic studies [4, 30, 21].

We observe that in lossless network conditions, the remote storage behaves in many respects like a local disk that has a moderately slow access time: the traditional caching, application prefetching, and asynchronous write techniques are typically effective in hiding the access delays. For high performance under loss and delay characteristics similar to the Internet, I/O-intensive benchmarks may require a large cache, network protocol tuning, and network support for packet prioritization.

The structure of this paper is as follows. Section 2 covers related work, Section 3 describes our remote storage testbed and benchmarks, Section 4 presents the performance measurements and discusses techniques to overcome network latency and congestion in a remote-storage environment, and Section 5 gives concluding remarks.

2 Related Work

2.1 Storage over IP Protocols

A key component of storage outsourcing is the transport of stored data over a communication network. Current storage service providers offer raw data block service over local storage-area networks (SAN) and over wide area networks (WAN). They typically use proprietary products such as EMC's SRDF software to replicate data to remote storage [6], and rely on a media-specific protocol (e.g., Fibre Channel protocol, ESCON, ATM) as the transport protocol. An emerging alternative for remote storage access is to encapsulate SCSI disk commands and data in IP packets. This approach is the target of a standardization effort by the Internet Engineering Task Force. iSCSI [24] is a SCSI over TCP/IP protocol proposed by a group including IBM, Cisco, HP, Quantum, SanGate and 3Com. It enables clients to address SCSI devices directly over an IP network. The protocol provides flow control, a method to include phase and tag information in a TCP stream, target buffer management, and resource discovery and management. Several working prototypes have been demonstrated in the InterOperability Lab at the University of New Hampshire [11]. In October 2001, the current iSCSI draft defines the encapsulation mechanism, message format, and session management, but many additional aspects are still under discussion.

Prior research projects have studied the use of IP over LAN or SAN for storage applications. The NASD project at CMU developed a storage architecture that enables direct client access to storage on a LAN [9]. The storage device exports an object interface, and allows clients to read and write to it directly, after securing the proper security credentials from an object manager. Prototypes implemented on Alpha workstations using RPC over UDP/IP demonstrated performance comparable to server-attached disks, but with better scalability. The Netstation project at USC studied the feasibility of using IP as a transport protocol between host and peripherals (e.g., storage devices) [32]. They implemented prototypes on Sun workstations with UDP/IP, and showed that it is possible to achieve 80% of SCSI's maximum throughput without the use of network coprocessors.

2.2 File System Research

Remote storage raises many of the traditional problems seen in file system research. Many file system techniques such as caching and prefetching [15] are effective in hiding the delays associated with access to local disks, so we expect them to be similarly helpful to the performance of remote storage. More recent file system research on minimizing the number of file system synchronous writes [8] and reducing write latency for small writes [33] will also alleviate the impact of network delay on data written to remote storage. See [25] for a summary of recent work in this area.

Most distributed file systems employ latency-hiding techniques to mitigate the effects of network delay on client performance. These techniques include caching data at the client, reducing the number of synchronous writes, and reducing the protocol overheads. For example, NFS v3 [20] uses asynchronous writes with commit. NFS v4 [26] groups related commands into a single compound command to reduce the number of round-trip times. It also delegates data ownership to clients to enable more aggressive client-side caching. Martin et al. [14] analyze the effect of network delay on the SPECsfs benchmark running on NFS v3. They verify their measurements analytically with a queuing model. Their results indicate that NFS is

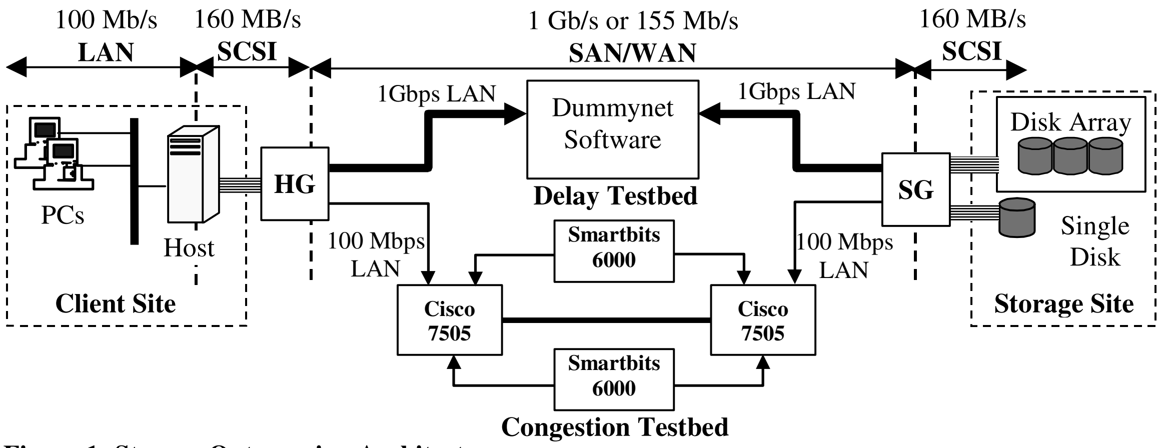


Figure 1: Storage Outsourcing Architecture.

insensitive to network latency up to 150 µs, and that performance decreases linearly with increasing delay.

A large body of research and commercial products show how to implement file systems on top of shared block storage [29]. These works complement our research and can provide the front end to our storage system.

Our paper extends previous work in several ways. We give measurements of SCSI over IP access to remote storage for network latencies corresponding to distances up to 1600 km, for a variety of I/O-intensive low-level and application-level benchmarks. In this context, we compare the performance on several operating systems and file systems, we examine the consequences of packet loss and congestion as seen in the Internet, and we explore the impact of techniques such as server-side and client-side caching and suppression of synchronous writes.

3 Remote-Storage Architecture

In our lab we have several projects that investigate broad issues in remote storage, and that deal with a variety of topics such as storage virtualization, remote replication and failure recovery, the partitioning of storage resources to serve concurrent clients, and implementing all the details of the emerging iSCSI protocol. For the performance experiments described in this paper, we use a simplified remote storage architecture that that consists of a single client site that is networked to a single storage site, as depicted in Figure 1.

3.1 System Description

The center portion of Figure 1 shows a network that connects a pair of machines called the host gateway (HG) and the storage gateway (SG). These machines implement SCSI over IP, and in experiments in Section 4.4 they also perform caching. (In a full storage system, the HG and SG machines would also implement mechanisms such as virtualization, replication, and recovery. In small systems, the HG and SG functionality might be implemented by cards rather than by separate PCs.) On the left side of the figure, the host connects to the HG via a standard SCSI cable, on which the HG appears to be one or more local SCSI disks. On the right side of the figure, the SG behaves like a local host as it accesses standard SCSI disks or RAID arrays on behalf of the HG.

The host, the HG, and the SG are PCs equipped with an Intel 440GX motherboard, dual 700 MHz PIII CPUs, 768 MB SDRAM, Intel 100 Mb/s Fast Ethernet, and Adaptec 29160 Ultra160 SCSI adapter cards. All PCs run FreeBSD OS 3.4 (except for experiments we describe that measure other operating systems). We run most benchmarks directly on the host, but the Surge and TPC-C benchmarks use a 3-level architecture in which a pool of clients access the host machine as a server, which in turn accesses remote back-end storage via the HG. The Surge clients access the host server via http. The TPC-C clients access the host database server via SQLNet. For Surge and TPC-C, the clients are PCs equipped with Intel 440BX motherboard, 400 MHz PII CPU, 64MB SDRAM, and Intel 100 Mb/s Fast Ethernet.

Our version of SCSI over IP for these experiments predates the IETF iSCSI draft [24]. We wrap SCSI messages according to the iSCSI message format, and transport them over TCP/IP. We use the immediate data option to send data along with the **write** command where possible. This version of our SCSI over IP does not implement functionality such as resource discovery and error recovery.

Figure 1 shows that the network connection between HG and SG can take two forms, which we call the *delay testbed*, and the *congestion testbed*. The delay testbed is used for experiments in which the WAN is assumed to be a dedicated high-speed fiber network. In this setting, we take measurements on a short physical network (a Lucent P550 Cajun Gigabit Ethernet Switch between a pair of Alteon ACE Gigabit Ethernet cards) that is augmented with the FreeBSD dummynet tool [22]. Dummynet is a kernel tool that modifies the behavior of the network protocol stack in ways that accurately mimic network behavior with respect to queuing, bandwidth limitations, delays, and packet losses. We use dummynet to vary the one-way propagation delay from 0 ms to 8 ms to reflect the performance impact of network distances up to 1600 kilometers. We calibrate the dummynet settings by measuring the performance of the `ping` command over a pair of actual fibers that form a 60 km bidirectional link (0.3 ms propagation delay each way). Although we use Gigabit Ethernet as the storage interconnect in the delay testbed, we believe that our findings would be similar over alternative storage interconnects such as Fibre Channel.

The congestion testbed simulates the congestion and packet loss of the Internet. The backbone consists of a pair of Cisco 7505 routers connected via an OC-3 link (155 Mb/s). The edge access networks are Fast Ethernets (100 Mb/s). We use a pair of Smartbits hardware traffic generators to impose background traffic on the backbone. Smartbits is more controllable and accurate than is a host-based generator such as *tcplib* [5] — a host-based TCP generator will back off under high load, thus failing to maintain the desired background load, whereas Smartbits can be programmed to maintain a fixed load distribution. Similarly, it would be difficult to simulate network congestion accurately via a simple tool like dummynet, because the various network parameters (e.g., delay and error rate) vary in a non-linear way with increasing load.

Our traffic profile represents current Internet behavior, by contrast with earlier networking studies based on telnet and ftp traffic (e.g., [2]). We obtain detailed statistics for bytes, packets, and flows from [21, 4], and second-order statistics on packet delay from [30]. Our traffic profile is summarized as follow:

Protocol: TCP (95%), UDP (5%), ICMP (<1%).
Application: http (75%), ftp (5%), misc (20%).
Packet length: 50% <44 bytes, 75% <576 bytes, 99% <1500 bytes.

We measure three types of storage devices: an 8 GB IBM DNES disk with 2 MB disk cache, an 18 GB IBM DDYS disk with 4 MB disk cache [10], and a Terrasolution disk array [3] with 128 MB disk cache and eight 18 GB IBM DDYS disks configured as RAID 5 with a 16 KB chunk size. We enable write cache on the disk, assuming that recovery is handled elsewhere (e.g., using NVRAM in the HG.)

Our measurements are conducted on 3 operating systems and 4 file systems. The bulk of our experiments are run on FreeBSD OS v3.4, which has a tunable file cache and a separate VM cache that holds clean data only. On this platform we measure the traditional UNIX FFS that uses synchronous metadata updates to ensure that it can recover the file system data structures to a consistent state after a crash, and we measure the more modern Soft Updates FFS, which uses careful update ordering to reduce the number of synchronous writes while still ensuring integrity [25]. We also repeat selected benchmarks on Microsoft Windows NT4.0, which has a small (and non-tunable) file cache, and on Windows 2000, which has an integrated file and VM cache. The Windows measurements are conducted on both the FAT file system, which uses synchronous writes to maintain metadata integrity, and on the more modern NTFS, which is a journaling file system that avoids synchronous writes by logging metadata updates to two log files on the disk [28].

3.2 Benchmarks

We choose our benchmarks to represent a wide variety of workloads that could access remote storage. The goal of our evaluation is to understand the impact of network delays, file system features, and storage system parameters on application performance when using remote storage. Each data point is obtained from 10 experimental runs. About 2 months are required to run 10 iterations of the full suite of experiments.

We begin with microbenchmarks that measure basic read and write performance, for sequential, random, and identical-block access patterns. (The latter access pattern reveals the time for a hit in the disk's cache.) These microbenchmarks access the raw disk device (i.e., the remote storage that is made to look like a local raw disk by our SCSI over IP software). We use a single-threaded program that reads or writes blocks that range from 1 KB to 64 KB. Our working set is 4 times larger than the remote storage system's cache. The performance metric is bandwidth (**Mbytes/sec**). We are able to model these simple I/Os analytically, and we use these models to give insight into the performance of the application benchmarks described below.

The SSH benchmark [25] represents a software development workload. It unpacks, configures, and builds a software package named SSH. The unpack phase extracts a compressed tar archive containing the source tree (383 files, totaling about 65,000 lines of commented code). It thus reads a large file sequentially and generates many subsequent small metadata operations. The config phase determines what features are available on the host operating system and generates a makefile. To do this, it compiles and executes many small test programs. Because most of the operations are on small files, we expect many metadata operations. The build phase executes the makefile to build the ssh executable. We run the three phases of the benchmark consecutively, so the config and build phases run with the file system cache warmed by the previous phases. We use throughput as a performance metric, by converting the time to run a single SSH benchmark iteration into **iterations/hour**.

The SDET benchmark [7] is designed by SPEC to simulate a typical timesharing workload. It models a software development environment (e.g., editing, compilation, and various UNIX utilities), and makes extensive use of the file system. SDET runs scripts that execute a predetermined mix of commands, and the reported metric is **scripts/hour**, as a function of the number of scripts running concurrently. We report results with 32 concurrent scripts.

The Surge benchmark [1] uses a workload generator to simulate a set of users accessing a web server. It is parameterized by empirical statistics extracted from web server logs. The parameters model server file size distribution, request size distribution, relative file popularity, embedded file references, temporal locality of reference, and idle periods of individual users. We run the benchmark with default settings, except that we increase the data size and load by using 20,000 files, and 4 clients, each running 2 processes and 50 threads per process. These parameters represent a workload with 400 timesharing users accessing a 1 GB data set. The performance metric is **operations/sec**.

The PostMark v1.1 benchmark [12] simulates the workload seen by Internet Service Providers under heavy load: a combination of electronic mail, netnews, and web-based e-commerce transactions. It creates a large set of files with random sizes within a set range, and then executes a large number of file create, delete, read and append operations. We set the benchmark to run with 250,000 files and the default size range (512 bytes to 16 KB), giving a working set of 500 MB. The performance metric is **transactions/sec**.

The TPC-C benchmark [31] simulates an online transaction-processing database workload. It models a wholesale supplier managing orders, and a workload consisting of a specified mix of five transaction types. We store the database in an Oracle 8i database v8.1.7 Enterprise Edition, and run experiments on Windows NT Server 4.0 SP6, and on Windows 2000 Server 5.0. The database is created using default settings recommended by Oracle. The client program is written in the PL/SQL language, adapted from the TPC-C sample program in Appendix A of [31]. We use a scaling factor of 30 to size the database, resulting in a working set of 2 GB (including log files). The database is restored to the same state prior to each run using a disk-level restore command. The complete database (including initialization files) resides in a single file system on the remote storage. For convenience, and as recommended by Oracle, we store the database in a file system rather than in a raw disk partition. Accessing the database through raw I/O instead of through a file system could improve performance by 5–10% [19], but this would not affect our conclusions. The performance metric is **transactions/min**.

4 Performance Results

4.1 Microbenchmark Results

To see the basic effects of network delays, and to discover where the system overheads are, we begin with measurements of blocking 8 KB reads and writes from a single thread that runs on the host PC. We measure I/Os to a local SCSI disk on the host, (reported in the row labeled "local" in Table 1), and I/Os to a local SCSI disk on the HG machine (the row labeled "target"), and I/Os to storage on the SG under various network delays. We conduct measurements on several kinds of storage devices: an IBM DNES disk, an IBM DDYS disk, a RAID 5 array, and /dev/zero (the latter enables us to separate out storage hardware delays from SCSI over IP processing). We also use in-kernel timing measurements to approximate the latency of each component between the host and the remote disk. These latter measurements are given in Figure 2.

The basic performance can be modeled simply, as described in more detail in [18], via this equation.

$$T_{IOLatency} = T_{disk} + 2\times T_{netDelay} + T_{iSCSI} \quad (1)$$

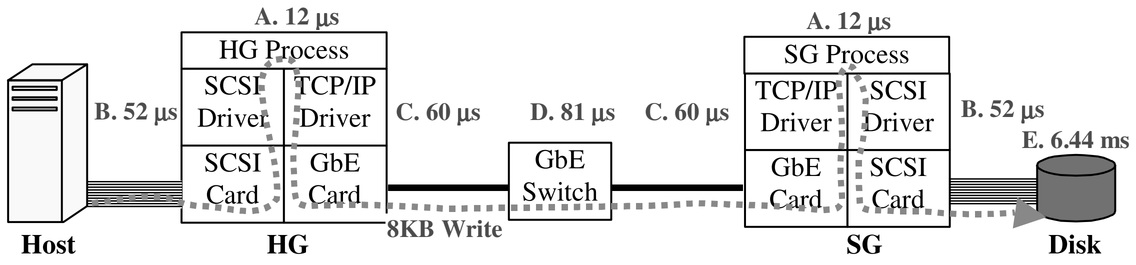
For random I/O, T_{disk} is the disk access time (seek, rotation and command processing) plus the data transfer time (i.e., number of bytes divided by media transfer rate) [10]. For sequential I/O, T_{disk} omits the seek and rotational components. $T_{netDelay}$ is the **one-way** network latency, and T_{iSCSI} is our SCSI over IP protocol processing latency. From Figure 2, we see that, in the case of an 8 KB data transfer to the disk with a 64 B acknowledgement from the SG, T_{iSCSI} is 0.408 ms.

In Table 1 we can see the round-trip overhead of our SCSI over IP implementation in the response time section — it is the difference between the local row and the 0 ms row. The overhead generally ranges from 0.4 to 0.6 ms, which is in

Table 1: Microbenchmark performance (8 KB record size) on FreeBSD. The working set is 4 times the disk cache size for disk-based storage systems, and 1 GB for the null device. Performance is expressed in throughput and response time, and we report the average of ten runs. The standard deviation is less than 2% of the mean; the maximum deviation is 6%. **Local** disk is a server-attached disk, **Target** disk resides on the HG and does not incur SCSI over IP overheads. We vary the **one-way** network delay from 0 to 8 ms.

Throughput (MB/s) for Microbenchmark with 8KB record Size																
Net. Delay (ms)	1 Disk, IBM DNES				1 Disk, IBM DDYS				Disk Array, 8xDDYS				Null Device			
	Random		Sequential		Random		Sequential		Random		Sequential		Random		Sequential	
	Read	Write	Read	Write	Read	Write	Read	Write	Read	Write	Read	Write	Read	Write	Read	Write
Local	0.77	0.70	13.94	20.00	1.07	1.57	34.04	29.40	1.01	3.38	19.52	16.94	1224.18	2501.56	1436.52	3555.56
Target	0.76	0.70	13.26	19.19	1.05	1.40	28.97	29.86	1.00	3.14	12.83	12.94	49.02	51.22	52.51	51.85
0	0.73	0.70	9.86	9.75	0.97	1.43	10.40	9.93	0.95	2.92	8.48	8.43	18.16	18.71	17.96	19.17
1	0.62	0.70	2.84	2.89	0.79	1.40	3.00	2.91	0.77	2.67	2.48	2.77	3.25	3.26	3.28	3.29
2	0.53	0.70	1.65	1.67	0.66	1.41	1.70	1.67	0.62	1.59	1.58	1.63	1.79	1.77	1.80	1.82
4	0.42	0.67	0.90	0.91	0.49	0.90	0.91	0.90	0.47	0.88	0.88	0.88	0.95	0.95	0.94	0.95
8	0.29	0.46	0.47	0.47	0.33	0.47	0.47	0.47	0.32	0.47	0.47	0.47	0.47	0.46	0.46	0.46

Response Time (ms) for a single 8KB I/O																
Net. Delay (ms)	1 Disk, IBM DNES				1 Disk, IBM DDYS				Disk Array, 8xDDYS				Null Device			
	Random		Sequential		Random		Sequential		Random		Sequential		Random		Sequential	
	Read	Write	Read	Write	Read	Write	Read	Write	Read	Write	Read	Write	Read	Write	Read	Write
Local	10.38	11.44	0.57	0.40	7.50	5.10	0.24	0.27	7.89	2.37	0.41	0.47	0.01	0.00	0.01	0.00
Target	10.50	11.45	0.60	0.42	7.60	5.70	0.28	0.27	8.02	2.55	0.62	0.62	0.16	0.16	0.15	0.15
0	10.99	11.38	0.81	0.82	8.24	5.61	0.77	0.81	8.43	2.74	0.94	0.95	0.44	0.43	0.45	0.42
1	12.94	11.48	2.82	2.77	10.09	5.71	2.67	2.75	10.35	3.00	3.22	2.89	2.46	2.46	2.44	2.44
2	15.11	11.44	4.86	4.80	12.21	5.67	4.71	4.80	12.82	5.02	5.08	4.92	4.48	4.52	4.46	4.40
4	19.04	11.89	8.92	8.82	16.29	8.93	8.79	8.86	17.13	9.06	9.08	9.09	8.43	8.42	8.50	8.41
8	27.18	17.47	17.10	17.04	24.46	17.09	16.99	17.02	25.32	17.20	17.17	17.20	17.20	17.35	17.35	17.35



$$T_{iSCSI} = 2\times(52\mu s + 12\mu s + 60\mu s) + 81\mu s + 2\times(1\mu s + 10\mu s + 15\mu s) + 27\mu s = 408\mu s$$

	Module	**Data Size**		
		64 B	**1.5KB**	**8KB**
A	**HG/SG Process**	10 µs	12 µs	12 µs
B	**SCSI (CAM+Adaptec 2960)**	1 µs	11 µs	52 µs
C	**IP (TCP/IP_Alteon NIC)**	15 µs	21 µs	60 µs
D	**GbE Switch (Lucent Cajun)**	27 µs	50 µs	81 µs
E	**Disk Array (Hit in Cache; Random Blocks)**	0.27; 5.65 ms	0.29; 5.75 ms	0.38; 6.44 ms
E	**IBM DDYS Disk (Hit in Cache; Random Blocks)**	0.20; 4.67 ms	0.21; 4.75 ms	0.29; 5.09 ms

Figure 2: Data flow and Latency estimates for a single 8KB I/O in our SCSI over IP prototype.

approximate agreement with the sum of the component overheads shown in Figure 2 (about 0.4 ms). In the throughput table, the null device measurements with 0 ms network latency show that the overhead of this SCSI over IP implementation limits the throughput of a single blocking thread to about 18 MB/s for 8 KB I/Os. The corresponding figure for 64 KB I/Os is about 30 MB/s.

The throughput table shows a huge drop in sequential I/O performance from network delay. The reason is that the sequential workload hits in the disk drive cache, so any network delay gives a huge percentage increase in total service time. For instance, consider sequential writes to the single DDYS disk. The throughput is 34.04 MB/s for the local disk, 10.4 MB/s for a 0 ms propagation delay, and 3.00 MB/s for a 1 ms propagation delay. This performance drop is explained by the response time table. At 34 MB/s, the I/O response time is 0.24 ms (the disk's cache acks the write immediately). Adding an overhead of 0.5 ms (for SCSI over IP) increases the total response time to 0.77 ms: SCSI over IP increases the delay by a factor of 3, which cuts the throughput by a factor of 3. Adding an additional 1 ms of propagation delay each way increases the total latency to 2.75 ms: When the latency grows from 0.24 ms to 2.75 ms, the throughput drops proportionally, from 34.04 MB/s to 3 MB/s. In general, for the sequential I/O microbenchmark the response time doubles as the network latency doubles. The random workloads are less severely impacted by network latency, because the disk random-access latency dilutes the impact of the network latency.

For single disk configurations (shaded columns in Table 1), random write performance is relatively insensitive to network delay. Figure 3 explains this behavior by analyzing the event sequence that occurs when writing to a busy disk. At t_1, a write request reaches the disk and the disk generates a SCSI status message immediately. The disk starts to flush the data to media since its cache is full. At t_2, another write reaches the disk, but it must wait until the previous write completes at t_3, freeing up space in the cache. Thus, if the roundtrip delay ($2\times T_{netDelay}$) is less than the disk response time (T_{disk}), the network delay is masked by the disk latency and the client's response time remains constant (i.e., $T_{disk} + T_{iSCSI}$). If the roundtrip delay is greater than the disk latency, the client's response time is only dependent on the network delay (i.e., $2\times T_{netDelay} + T_{iSCSI}$). This behavior is not seen in the disk array because of its larger cache and much higher aggregate write bandwidth.

For an I/O size smaller than the disk array chunk size (16 KB), the disk array overhead generally impairs the performance of a single-threaded request stream. In the case of a workload that has concurrent I/O requests, we would expect the aggregate performance of the array to be better than that of a single disk.

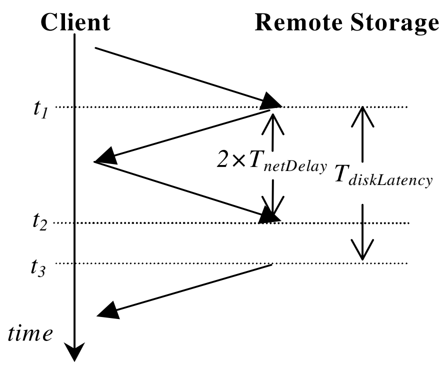


Figure 3: Writing a Single Block to a Random Disk Address.

4.2 Application-Level Benchmarks

The microbenchmark measurements in the previous section indicate that network latency impairs the performance of a single-threaded process that spends 100% of its time trying to do I/O from remote storage. We next consider the performance of a variety of application-level I/O benchmarks, to see the performance of remote-storage on more typical workloads. Except where stated otherwise, the measurements in this section are obtained on a remote storage system that uses a disk array (not a single disk).

4.2.1 Experiments that vary network latency

Table 2 shows the application benchmark performance versus network delays, for the FreeBSD OS with the traditional FFS and the Soft Updates FFS file systems, and for Windows NT and Windows 2000 with the NTFS and the FAT file systems. We measure I/Os to storage on the host, (the row labeled "local"), I/Os to storage on the HG machine (the row labeled "target"), and I/Os to storage on the SG under **one-way** network propagation delays ranging from 0 to 8 ms.

We make the following observations:

1. **Our SCSI over IP implementation overhead is small.**

We observe that our SCSI over IP implementation adds little overhead, both in terms of application performance (Table 2), and host CPU usage. The performance degradation from a host-attached disk to a target disk averages 6% across all the benchmarks (comparing rows labeled **Local** and **Target**). We experienced higher overheads (18%) when accessing a remote disk over the IP protocol stack (comparing rows labeled **Local** and **0 ms**). The CPU usage at the HG is less than 10% for all workloads, implying that we do not need a high performance processor for the SCSI over IP gateway—it would be feasible to implement the gateway on a network interface card.

2. **Network latency has little effect on web, database, and CPU-bound benchmarks.**

The Surge benchmark models a typical web server, where hot documents are accessed very frequently, and thus many accesses hit in the web server's local cache. Consequently, the workload seen by the remote storage consists largely of asynchronous sequential writes to log files. Thus, we observe very little sensitivity to network latency. For the TPC-C benchmark, we observe

Table 2: Application benchmark performance. The **local** row gives the throughput (as defined for each benchmark in Section 3.2) of a host-attached disk array. Other rows give the normalized throughput as a fraction of the local row. The **target** row accesses an array on the HG, without SCSI over IP overheads. Subsequent rows vary the one-way network delay from 0 to 8 ms. **Soft** is the Soft Updates FFS file system. We report the mean over ten runs: the standard deviation is less than 5% of the mean; the maximum deviation is 16%.

Net. Delay (ms)	**FreeBSD 3.4**								**Windows 2000**				**Windows NT Server 4.0**			
	SSH		**SDET**		**PostMark**		**Surge**		**PostMark**		**TPC-C**		**PostMark**		**TPC-C**	
	Soft	**FFS**	**Soft**	**FFS**	**Soft**	**FFS**	**Soft**	**FFS**	**FAT**	**NTFS**	**FAT**	**NTFS**	**FAT**	**NTFS**	**FAT**	**NTFS**
Local	67.3	60.0	4253.7	3376.9	386.2	361.9	3058.0	2765.9	326.3	237.6	730.7	727.1	80.0	61.7	750.1	733.5
Target	0.94	0.96	0.94	0.84	0.91	0.95	0.96	0.96	0.96	0.89	1.01	1.02	0.98	0.91	0.95	0.97
0	0.89	0.98	0.70	0.49	0.56	0.57	0.97	0.84	0.70	0.55	1.00	1.02	0.94	0.92	0.94	0.95
1	0.87	0.90	0.33	0.15	0.20	0.17	0.92	0.84	0.42	0.31	0.99	1.04	0.74	0.79	0.93	0.94
2	0.83	0.74	0.18	0.08	0.11	0.14	0.90	0.92	0.28	0.25	0.98	0.97	0.50	0.54	0.93	0.93
4	0.78	0.61	0.10	0.04	0.06	0.07	0.89	0.92	0.14	0.13	0.93	0.93	0.32	0.35	0.88	0.90
8	0.69	0.49	0.05	0.02	0.03	0.03	0.86	0.88	0.07	0.07	0.88	0.89	0.17	0.19	0.82	0.84

that the performance is immune to small network delays (<2 ms). Performance only degrades modestly with larger network delays, because a modern DBMS is designed to overcome I/O latency. It uses a high multiprogramming level to generate numerous concurrent I/Os, and clever caching and prefetching to reduce the number of I/Os that block progress [27]. The database may prefetch tables to gain the benefit of large sequential reads, and may use vertical partitioning of tables to access and cache only the required columns. Consequently, the database's performance will not suffer, given sufficient concurrency and I/O bandwidth. Although the SSH benchmark has been proposed as an I/O benchmark [25], we find that SSH is mostly CPU bound, especially during the configure and build phases. Thus, SSH's performance decreases by only 33% as the one-way propagation delay increases from 0 ms to 8 ms, versus 10× for PostMark and SDET.

3. Read traffic is less significant than write traffic.

Many of these benchmarks (SSH, SDET, PostMark) have little read traffic that reaches the disk, as they first generate the data needed in subsequent phases of the benchmarks, and much of their working sets fit in main memory. Thus most traffic seen at the disk level is writes. (Many file system workloads have this characteristic because of the effectiveness of read caching.) TPC-C has significant read traffic (35–43% of overall bytes transferred). But most of these reads are either prefetch or not in the critical path. The DBMS prefetches whole tables during the early portion of the benchmark. These prefetches are mostly asynchronous reads from sequential locations, so they are serviced quickly. Because the TPC-C working set is larger than the buffer cache size, pages are subsequently evicted from the database buffer cache during the course of the run. Nevertheless, the number of synchronous reads gradually decreases during the benchmark run. The write traffic in TPC-C appears to be mostly delayed writes and not in the critical path, and thus has little impact on the application performance.

4. Some applications exhibit super-linear degradation (> 2x when delay doubles).

With the FFS file system, PostMark and SDET exhibit a large drop in performance from the local array to remote storage with 0 ms and 1 ms propagation delays (see shaded cells in Table 2). This is the same phenomenon as seen in the large throughput drops for sequential I/O in the microbenchmarks. For PostMark and SDET, the predominant I/O is single-threaded random writes to metadata. These writes take 0.4 ms in the disk array (they are ack'd from the array cache). Since our SCSI over IP overhead is about 0.4 ms for 8KB writes, the degradation is 50% from a local array to a remote array with 0 ms propagation delay. When the propagation delay increases to 1 ms (2 ms round-trip), the response time for a single 8KB I/O increases to 2.8ms, and the performance drops accordingly.

4.2.2 Experiments that vary network congestion

In this section we present the results of the PostMark and TPC-C benchmarks when measured on the congestion testbed with the Smartbits traffic generator applying a background network load. PostMark is I/O bound, and the insights that we obtain from it are

Table 3: **Effects of Congestion on Application Performance.**

Load Factor (%)	Background Traffic Profile				PostMark (Trans/sec)			TPC-C (trans/min)
	BW	σ(BW)	Delay	PktLoss	BSD FFS	BSD SoftUpd	NT4.0 NTFS	NT4.0 NTFS
0	0	0	0.46	0	64	80	52	722
20	31	9.8	0.54	0	63	79	49	719
30	40	11.7	0.54	2×10^{-6}	60	75	46	720
35	43	12.3	0.55	5×10^{-6}	59	74	48	718
40	56	12.7	0.64	2×10^{-5}	56	73	47	719
50	76	12.8	0.70	5×10^{-5}	53	72	47	717
55	86	12.8	0.72	7×10^{-5}	51	63	46	710
60	96	12.8	0.78	1×10^{-4}	44	54	42	700
70	106	12.8	1.07	3×10^{-4}	26	29	26	693
75	110	12.8	1.68	7×10^{-4}	**No Data**			668

applicable to other I/O-bound applications. The TPC-C results demonstrate the effectiveness of application-level latency hiding techniques.

Table 3 shows the results of PostMark on FreeBSD with FFS and Soft Updates, and on Windows NT Server 4.0 with NTFS. This table also shows TPC-C results on NT Server 4.0. We measure the performance of these benchmarks as the background network traffic ranges from 0 to 75% of the backbone link capacity. The background traffic is based on the traffic profile described in Section 3.1, and is characterized by 4 parameters: **BW** is average bandwidth in Mb/s, **σ(BW)** is standard deviation of the measured bandwidth, **Delay** is round trip propagation delay in ms, and **PktLoss** is average packet loss rate. The shaded region in the table indicates network congestion (i.e., the application is starved for network bandwidth).

With no background load, the benchmark performance is limited by the fixed 0.46 ms equipment delay inherent in the router network. As the background load increases, we observe that the application's performance decreases gradually until the onset of congestion. The bandwidth requirement of the application varies from 56 Mb/s for PostMark under FreeBSD OS to 24 Mb/s for TPC-C. When the background traffic grows so large that it begins to take bandwidth that is required by the application (shaded region in Table 3), the application becomes *both* delay and bandwidth bound, so performance deteriorates significantly. Over the Internet, we may need more than caches and synchronous write suppression to obtain good remote I/O performance.

4.3 Identifying bottlenecks in a SCSI over IP system

In this section we analyze various components in our SCSI over IP system to identify areas for improvement. This includes the host, the edge access equipment (HG and SG), the network, and the remote storage system.

4.3.1 Host

The host has several components affecting performance, including the file system, OS, network protocol stack, and SCSI protocol stack.

File system: We observe from Table 2 that the file system design affects performance. The results for Soft Updates versus traditional FFS show the benefit of suppressing synchronous I/Os. For instance, PostMark runs 22–33% faster with Soft Updates under all network delay conditions.

OS: We focus on host buffer cache design as it determines the amount of traffic seen at the disk level. We find that a small (and static-size) file cache is bad. This can be seen by comparing the performance of PostMark on NT Server 4.0 and on Windows 2000. NTFS is supposed to have an integrated VM cache that is dynamically resized according to workload, and its cache size can be controlled by the user (e.g., by setting the *LargeSystemCache* parameter [28] or by using the public domain program *CacheSet* [23]). However, we estimate using the Windows Task Manager tool that the file cache size in NT never goes beyond 10% of the working set under various I/O loads and *CacheSet* settings. The resulting cache capacity misses significantly degrade performance, as seen in PostMark, which runs 4 times slower on NT than on Windows 2000. An application can compensate for a small host cache by caching data in application buffers, as is done in the Oracle database: the TPC-C performance numbers on the NT and Windows 2000 platforms are comparable despite the small host buffer cache in NT. We also find that the number of file cache entries should be dynamically tunable. FreeBSD has a small buffer cache (default 512 buffer cache entries) that is fixed at kernel build time. It also has a large VM cache that holds only clean pages. This limited amount of buffer available to cache asynchronous write traffic causes I/O-bound applications running on Soft Updates to stall during high network delays while waiting for buffer cache cleaning. This is seen in PostMark: the performance of traditional FFS converges with that of FFS with Soft Updates as the delay increases (see Table 2), even though FFS with Soft Updates has very few synchronous I/Os.

TCP/IP: TCP is responsible for 37% of the latency for an 8 KB transfer to remote storage when the network propagation delay is 0. This assumes that we have large TCP windows to minimize unnecessary fragmentation of SCSI data, thereby avoiding extra round-trip delays. Most researchers consider window size to be a critical bottleneck for networks having a high bandwidth-delay product [13]. We find that most application-level benchmarks use record sizes of 4–8 KB (metadata). Larger I/Os (16–256 KB) are mainly asynchronous data transfers and are not in the critical path. In our experiments we verified that a default window size of 64 KB does not harm performance by comparison with a larger window size of 640 KB. However, window size may be important if the application or file system generates a significant volume of synchronous I/O traffic. A related issue is the packet size in the network layer (i.e., the MTU size). To reduce fragmentation overhead, we prefer to use the largest MTU allowable by the network. For example, increasing the MTU from 1.5 KB to 8 KB (i.e., jumbo frame) gives a 15% performance improvement in the 8 KB random-write microbenchmark. We also can verify this from Figure 2, by calculating the 8 KB I/O latency (0.461 ms versus 0.408 ms). In application level benchmarks, using a 1.5 KB MTU instead of 8 KB degrades the performance by 1%–9% when the propagation delay is low (i.e., SAN environment); the degradation is less severe with higher propagation delays (>1 ms). Lastly, we find that data copying in the host protocol stack (<10 µs for 8 KB) is insignificant as it is much smaller than other latencies (e.g., 81 µs for 8 KB in the GbE switch).

SCSI: SCSI over IP is responsible for 25% of the latency for an 8 KB transfer to remote storage when the network propagation delay is 0**.** This assumes that we use the immediate data option that avoids an extra round trip delay that would be incurred if the command were sent separately from the data [24].

Our results indicate that hardware assistance for SCSI and TCP/IP protocol processing may be beneficial in a SAN. In the SAN, the network propagation delay is on the order of 10 µs (2 KM of fiber), whereas our measured protocol processing time for an 8 IB I/O is 53 µs for SCSI, plus 75 µs for TCP/IP. By contrast, in a WAN environment the protocol processing overhead is insignificant: the propagation delay dominates.

4.3.2 Network

Our edge access equipment (HG and SG) is responsible for 327 µs of latency per 8 KB I/O (see Figure 2). Thus, in a SAN environment, replacing the HG and SG by a native implementation of iSCSI in a host adapter and in the remote storage array could significantly improve the performance of remote storage. The network equipment latency (i.e., 81 µs per 8 KB transfer on a current GbE switch) is also significant in a SAN. Application performance degrades with increasing latency to remote storage, but fortunately, end-system optimizations such as caching (see Section 4.4) and synchronous I/O suppression (see Section 4.3.1) often can hide network latency. In a lossy environment such as the Internet, bandwidth impairment is the dominant limitation on remote storage performance, so we may need new network techniques, e.g., to avoid the bandwidth destruction that TCP suffers under conditions of congestion and packet loss.

4.3.3 Remote storage

Slower storage system = better delay tolerance: If the remote storage device is slow, that bottleneck reduces the impact of network latency. This is seen in Figure 4, which compares the throughput of the SDET benchmark on remote storage using either a single IBM disk, or a disk array with 8 disks. The performance on the remote disk array is much more sensitive to network delay than is the single-disk performance.

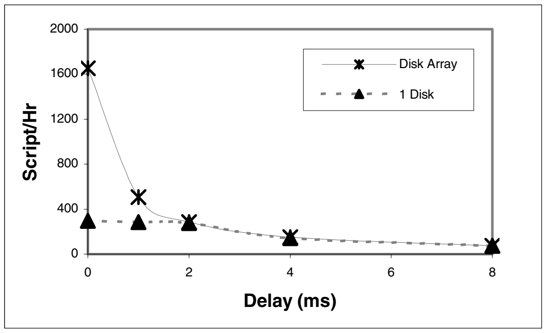


Figure 4: Effects of storage system on SDET performance on FreeBSD and Soft Update file systems.

4.4 Using cache to hide network delay

In this section we study the extent to which caches at the HG or the SG can hide network latency. We assume that we have reliable main memory or NVRAM [17] so that writes can be safely acknowledged from the cache, and we have no cache coherency issues because the remote storage is partitioned over hosts that have exclusive access to virtual volumes.

4.4.1 Cache Location

A cache at the HG tends to hide both disk and network latency. We see this in Figure 5, which uses the 500 MB working set PostMark benchmark on FreeBSD to show the effectiveness of a 1 MB cache that is located at the HG, by comparison with a cache at the SG, or no cache (the latter two curves coincide). We see that the cache at the host side of the network can hide almost all of the network latency.

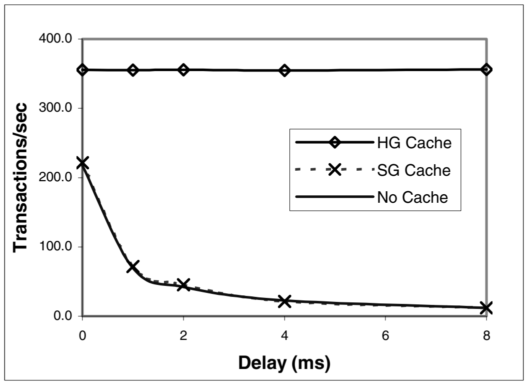


Figure 5: Performance of PostMark with 1 MB cache in HG or SG, versus network delay in ms. (FreeBSD, Soft Updates, delay testbed.)

4.4.2 Write Cache

We now examine the effect of the HG cache on write performance. The size of the write cache that is required to hide the network latency is proportional to the bandwidth-delay product of the network. For instance, an 8 ms network delay will need about 1 MB of write cache on a gigabit/s network (0.008 sec $\times$ 10^9 bits/sec $\times$ 0.125 bytes/bit). For the Internet, with variable delay and congestion, we may need a significantly larger cache.

Figure 6 shows the performance of PostMark running on FreeBSD with the Soft Updates file system on the congestion testbed. We show curves for 4 different HG cache sizes, where the size is given in the legend as a percentage of the 500 MB PostMark working set size. The baseline configuration's performance (no cache in Figure 6) is initially latency bound, due to the 0.46 ms round-trip delay inherent in the router network. It becomes bandwidth bound when the background load exceeds 55% of the backbone link. We can overcome the 0.46 ms router latency with a small amount of cache (i.e., 0.1% or 256 KB). This enables the application performance to approach that of a locally-attached disk array. For the 0.1% cache configuration, PostMark becomes bandwidth bound when the background traffic exceeds 30% of the backbone link. Its performance approaches that of the baseline "no cache" configuration. As we increase the cache size to 10% of the working set, PostMark is able to maintain a high transaction rate up to a 55% load factor. This is because the larger cache filters write traffic effectively, reducing it by 40%.

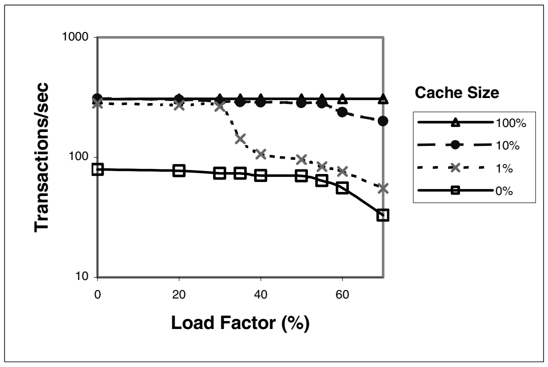
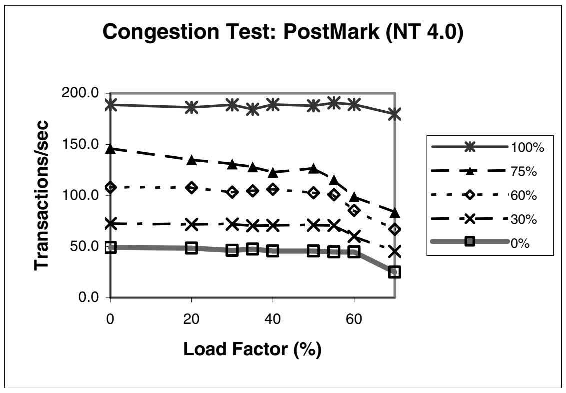


Figure 6: Impact of HG cache size on PostMark (FreeBSD, Soft Updates, congestion testbed; HG cache size specified as a percentage of the 500 MB PostMark working set size.)

4.4.3 Read Cache

Recall from Section 4.3.1 that NT has a very small file system buffer cache, and thus it incurs many capacity read misses. We now use the PostMark measurements on NTFS to examine the effectiveness of the HG read cache.

Figure 7 shows that the HG cache significantly improves performance under all load and delay conditions. We see that a large cache is needed to achieve performance similar to that of a locally attached disk array. For example, in the congestion test we see that with no background traffic and no cache, the throughput is 50 transactions per second. To tolerate a 70% loaded backbone with similar performance, the required cache size is 30% of PostMark's working set. Similarly, the delay test shows that 60% of the working set needs to be cached to mask a one-way propagation delay of 8 ms.



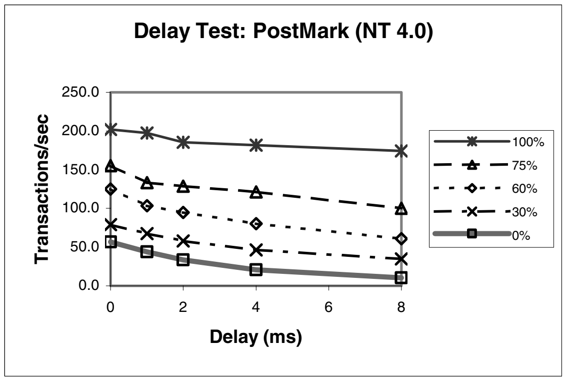


Figure 7: Effect of cache size on PostMark performance in the congestion and delay testbeds. Windows NT Server 4.0, NTFS, with HG cache size specified as a percentage of the 500 MB PostMark working set size. Performance is expressed in terms of Throughput and is based on an average of five runs. The standard deviation is less than 2.5% of the mean, with a maximum deviation of 3.9%.

5 Concluding Remarks

Storage outsourcing is an important emerging industry that forces us to revisit problems at the nexus of storage systems, file systems, and network protocols.

In this paper, we report measurements of the performance of remote storage via extensive benchmark experiments, using benchmarks that are widely applied in file system and database research. Our measurements cover FreeBSD with two versions of FFS, and Windows NT and Windows 2000 with NTFS and the FAT file system. We consider the impact of network delays when a host accesses remote storage over local or long-haul networks, and we also study the effect of congestion and packet loss on the performance of remote storage.

Our measurements give quantitative confirmation, in a SCSI over IP remote storage setting, of the dominant importance of write traffic, the benefits of suppressing synchronous writes, the significance of the file system buffer cache design, and the effectiveness of host-side caches.

Network delay can adversely affect application performance. But it can be masked in several ways, such as by write caching and read prefetching. The impact of network delay on performance is strongly influenced by the remote storage architecture and the network protocol settings. Slow remote storage masks network delay. In the case of fast remote storage, judicious caching and prefetching strategies are important tools to reduce the sensitivity of application performance to network delay.

We have observed quite acceptable performance for application benchmarks that access remote storage, even given network propagation delays that correspond to distances of hundreds of kilometers — but only in the absence of congestion and packet loss. An important problem for future work is to develop network and file system techniques that enable remote storage systems to maintain good performance when faced with the delay and loss characteristics of the Internet. In particular, to maintain high application performance despite high network latency, we need techniques to keep the network connection full of requests and data. Examples of techniques that can help to achieve high I/O concurrency include asynchronous I/O, multithreading, informed prefetching, and set-oriented data access methods. To maintain acceptable application performance in conditions of congestion and loss, we must revisit some traditional problems in networks having a high bandwidth-delay product. TCP bandwidth suffers from an insufficient ability to distinguish between loss and congestion, so SCSI over IP needs assistance, such as from QoS mechanisms or from other techniques that maintain high bandwidth despite loss. Many additional problems also remain for future work. For instance, small blocking I/Os for metadata updates can cause serious performance problems under high network delay. Can the file system maintain safety in a less costly way? Can the OS buffer cache do a better job by combining its write traffic into large scatter-I/O commands?

Acknowledgments

We thank our shepherd, Rod Van Meter, for his careful reading and valuable feedback. We thank the anonymous reviewers for their detailed comments. We thank our colleague Hao Sun for helping to set up the congestion testbed.

References

1 P. Barford, M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", *Proc. ACM SIGMETRICS*, pp. 151-160, Madison, WI, June 1998.

2 L. Brakmo, S. O'Malley, L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance", *Proc. ACM SIGCOMM,* pp. 24-35, London, UK, August 1994.

3 Chaparral Network Storage Inc, *TRC-22 User's Guide*, November 2000.

4 K. Claffy, "The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone", *Proc. Inet '98*, http://www.caida.org/outreach/papers/Inet98.

5 P. Danzig, S. Jamin, "tcplib: A Library of TCP Internetwork Traffic Characteristics", Technical Report CS-SYS-91-495, Computer Science Department, USC, 1991.

6 EMC, *Symmetrix Remote Data Facility Product Description Guide*, 2000.

7 S. Gaede, "Perspectives on the SPEC SDET Benchmarks", http://www.spec.org/osg/sdm91/sdet/index.html.

8 G. Ganger, Y. Patt, "Metadata Update Performance in File Systems", *Proc. 1st USENIX Symp on OSDI.*, pp. 49-60, Monterey, CA, November, 1994.

9 G.A. Gibson, D.F. Nagle, K. Amiri, J. Butler, F.W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, J. Zelenka, "A Cost-Effective, High-Bandwidth Storage Architecture", *Proc. 8th Int. Conf. ASPLOS,* pp. 92-103, San Jose, CA, Oct., 1998.

10 IBM Corp., *Ultrastar 36LZX Hard Disk Specifications, Version 1.0 (Revised)*, December 1999.

11 InterOperability Lab, *iSCSI Implementations,* http://www.iol.unh.edu/consortiums/index.html.

12 J. Katcher, "PostMark: A New File System Benchmark", Technical Report TR3022. Network Appliance Inc, October 1997.

13 T.V. Lakshman, U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss", *IEEE/ACM Trans. Net.* 5(3), pp. 336-350, June 1997.

14 R. Martin, D. Culler, "NFS Sensitivity to High Performance Networks", *Proc. ACM SIGMETRICS,* pp. 71-82, Atlanta, GA, May 1999.

15 M. McKusick, W. Joy, S. Leffler, R. Fabry, "A Fast File System for Unix", *ACM Trans. On Comp. Sys.* 2(3), pp. 181-197, August 1984.

16 F. Neema, D. Waid, "Data Storage Trend", *UNIX Review*, 17(7), June 15, 1999.

17 W. T. Ng, P. M. Chen, "The Design and Verification of the Rio File Cache", *IEEE Trans. Comp.* 50(4), pp. 322-337, April 2001.

18 W. T. Ng, B. Hillyer, "Obtaining High Performance for Storage Outsourcing (Poster Paper)", *Proc. ACM SIGMETRICS,* pp. 322-324, Boston, MA, June 2001.

19 Oracle Corp. *Oracle8i Administrator's Guide, Release 2 (8.1.6)*, pp. 3-31, Jan 2000.

20 B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, D. Hitz, "NFS Version 3 Design and Implementation", *Proc. USENIX Summer 1994 Tech. Conf.*, pp. 137-152, Boston, MA, June 1994.

21 V. Paxson, "End-to-End Internet Packet Dynamics", *IEEE/ACM Trans. Net.,*7(3), pp. 277-292, June 1999.

22 L. Rizzo, "dummynet", http://info.iet.unipi.it/~luigi/ip_dummynet.

23 M. Russinovich, *CacheSet v1.0*, Dec. 1998, http://www.sysinternals.com/ntw2k/source/cacheset.shtml.

24 J. Satran, et. al., "iSCSI", Sep. 2001, http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-08.txt.

25 M. Seltzer, G. Ganger, M. McKusick, K. Smith, C. Soules, C. Stein, "Journaling versus Soft Updates: Asynchronous Metadata Protection in File Systems", *Proc. USENIX Annual Tech. Conf.*, pp. 71-84, San Diego, CA, June 2000.

26 S. Shepler, et. al., "RFC 3010: NFS Version 4 Protocol", Dec. 2000, ftp://ftp.isi.edu/in-notes/rfc3010.txt.

27 A. Silberschatz, H. F. Korth, S. Sudarshan, *Database System Concepts*, McGraw-Hill, 1998.

28 D. Solomon, *Inside Windows NT Second Edition.* Microsoft Press, 1998.

29 C.A. Thekkath, T. Mann, E.K. Lee "Frangipani: A Scalable Distributed File System", *Proc. 16th ACM SOSP,* pp. 224-237, San-Malo, France, Oct., 1997.

30 K. Thompson, G. Miller, R. Wilder, "Wide Area Internet Traffic Patterns and Characteristics", *IEEE Net.*, 11(6), Nov.-Dec. 1997.

31 Transaction Processing Performance Council, *TPC Benchmark C, Standard Specification, Revision 5.0.* February 2001, http://www.tpc.org/tpcc/spec/tpcc_current.pdf.

32 R. Van Meter, G. G. Finn, S. Hotz, "VISA: Netstation's Virtual Internet SCSI Adapter", *Proc. 8th Int. Conf. ASPLOS,* pp. 71-80, San Jose, CA, Oct., 1998.

33 R. Wang, T. Anderson, D. Patterson, "Virtual Log Based File Systems for a Programmable Disk", *Proc. 3rd USENIX Symp on OSDI.*, pp. 29-44, New Orleans, LA, February, 1999.