An Experimental Comparison of Block- and File-Access Protocols for IP-Networked Storage

Abstract

IP-networked storage has become increasingly common in today's LAN environments. There are two fundamentally different abstractions—files and blocks—for accessing remote data in IP-networked storage. In this paper, we conduct an experimental study to determine whether a file-level or a block-level abstraction is better suited for IP-networked storage. We use NFS and iSCSI as representative examples of the file- and block-access protocols, and compare their performance using a variety of micro- and macro-benchmarks. Our results show that block- and file-access protocols are comparable for data-intensive workloads, while the former outperforms the latter by a factor of 2 or more for meta-data-intensive workloads. We identify aggressive meta-data caching and update aggregation in iSCSI to be the primary reasons for this performance difference and propose two enhancements to NFS to overcome these limitations.

1 Introduction

1.1 Motivation

With the advent of high-speed LAN technologies such as Gigabit Ethernet, IP-networked storage has become increasingly common in client-server environments. The availability of 10 Gb/s Ethernet in the near future is likely to further accelerate this trend. IP-networked storage is broadly defined to be any storage technology that permits access to remote data over IP. The traditional method for networking storage over IP is to simply employ a network file system such as NFS [3]. In this approach, the server makes a subset of its local namespace available to clients; clients access meta-data and files on the server using a RPC-based protocol (see Figure 1(a)).

In contrast to this widely used approach, an alternate approach for accessing remote data is to use an IP-based storage area networking (SAN) protocol such as iSCSI [26]. In this approach, a remote disk exports a portion of its storage space to a client. The client handles the remote disk no differently than its local disks—it runs a local file system that reads and writes data blocks to the remote disk. Rather than accessing blocks from a local disk, the I/O operations are carried out over a network using a block access protocol (see Figure1(b)). In case of iSCSI, remote blocks are accessed by encapsulating SCSI commands into TCP/IP packets [26].

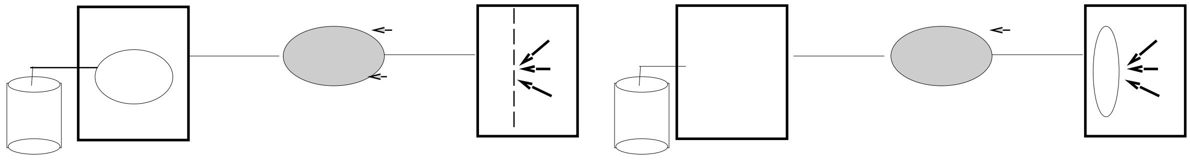
The two techniques for accessing remote data employ fundamentally different abstractions. Whereas a network file system accesses remote data at granularity of files, SAN protocols access remote data at the granularity of disk blocks. We refer to these techniques as *file-access* and *block-access* protocols, respectively. In essence, a file-access protocol makes remote files appear local, whereas a block-access protocol makes remote *disk blocks* appear local. In the former approach, the file system resides at the server, whereas in the latter approach it resides at the client (see Figure 1). Consequently, the network I/O consists of file operations (file and meta-data reads and writes) for file-access protocols and block operations (block reads and writes) for block-access protocols. Furthermore, the implications of caching are different in the two scenarios. Since the file system resides at the client in block access protocols, the file system cache is local, enabling applications to benefit from cached files and meta-data. In a file-access protocol, the corresponding file system cache is located at the server. However, the network file system may employ its own cache at the client to enhance application performance.

File-access protocols have been widely studied over the past two decades in the context of network file systems [3, 10, 15] In contrast, since SAN technologies such as iSCSI are emerging, the design and performance of block access protocols is less well studied. Furthermore, given the above tradeoffs, it is not *a priori* evident whether a file-level or a block-level abstraction is better suited for IP-networked storage. There is no study that systematically compares block- and file-access protocols for IP-networked storage, and this is the focus of the present work.

1.2 Research Contributions of This Paper

In this paper, we experimentally compare file- and block-access protocols using NFS and iSCSI as representative examples of the two approaches. Our experimental study attempts to answer the following research questions:

- Is a file-level or a block-level abstraction better suited for IP-networked storage?
- Under what circumstances does a block-access proto-



(a) File-access Protocol (NFS)

(b) Block-access Protocol (iSCSI)

Figure 1: An overview of file- and block-access protocols.

col outperform a file-access protocol and vice versa? Specifically, what are the network overheads and caching benefits of the two approaches, and what is their impact on application performance?

- What are the limitations of the two approaches, if any, and how can they be overcome?

We answer these questions by conducting an experimental study on a storage system testbed. Our study involves careful micro-benchmarking of three generations of the NFS protocols—NFS versions 2, 3 and 4, and iSCSI. We then measure application performance using a suite of data-intensive and meta-data intensive benchmarks such as PostMark, IOZone, TPC-C and TPC-H on the two systems.

Our results show that iSCSI and NFS are comparable for data-intensive workloads, while the former outperforms the latter by a factor of 2-12 for meta-data intensive workloads. We identify aggressive meta-data caching and update aggregation in iSCSI as the primary reasons for this performance difference. We propose two enhancements to NFS to extract these benefits of meta-data caching and update aggregation.

The rest of this paper is structured as follows. Section 2 provides a brief overview of NFS and iSCSI. Sections 3, 4, and 5 present our experimental comparison of NFS and iSCSI. Implications of our results are discussed in Section 6. Section 7 discusses our observed limitations of NFS and propose two enhancements. Section 8 discusses related work, and we present our conclusions in Section 9.

2 Overview of Block- and File-access Protocols

Since our experimental study uses NFS and iSCSI as representative examples of file- and block-access protocols respectively, we provide a brief overview of NFS and iSCSI.

2.1 NFS Overview

NFS was first proposed as a stateless file access protocol by Sun Microsystems Inc., and later revised by the IETF [24]. This version of NFS was referred to as NFS version 2 (or simply "NFS v2") and consists of four distinct protocols: (i) the base *nfs* protocol, which is used for file creation, access, reading, writing, authentication and statistics; (ii) the *mountd* protocol, which is used to mount server-exported file systems; (iii) the *nsm* protocol, which monitors the status of servers and clients; and (iv) the *nlm* protocol, which provides locking services. In NFS v2, the client and the server communicate via remote procedure calls (RPCs) over UDP. A key design feature of NFS version 2 is its stateless nature—the NFS server does not maintain any state about its clients, and consequently, no state information is lost if the server crashes.

The next version of NFS—NFS version 3—focuses on improving performance and scalability [3, 22]. The key enhancement in NFS version 3 are as follows: (i) support for a variable length file handle of up to 64 bytes, instead of 32 byte files handles; (ii) eliminates the 8 KB limit on the maximum data transfer size; (iii) support for 64 bit offsets for file operations, up from 32 bits; (iv) reduces the number of fetch attribute calls by returning the file attributes on any call that modifies them; (v) supports asynchronous writes to improve performance; and (vi) adds support for TCP as a transport protocol in addition to UDP.

The latest version of NFS—NFS version 4—focuses on enhancing performance over wide area networks such as the Internet, strong security and protocol extensibility [8]. Another goal is to improve the locking and performance for narrow data sharing applications [27]. Some of the key features of NFS version 4 are as follows: (i) it integrates the suite of protocols (nfs, mountd, nlm, nsm) into one single protocol for ease of access across firewalls; (ii) it supports compound operations to coalesce multiple operations into one single message; (iii) it is *stateful* when compared to the previous incarnations of NFS — NFS v4 clients use OPEN and CLOSE calls for stateful interaction with the server; (iv) it introduces the concept of delegation to allow clients to aggressively cache file data; and (v) it mandates strong security using the GSS API.

2.2 iSCSI Overview

In contrast to NFS, iSCSI is a relatively new protocol that was standardized recently by the IETF. iSCSI is a block-level protocol that encapsulates SCSI commands into TCP/IP packets, and thereby leverages the investment in existing IP networks.

SCSI is a popular block transport command protocol that is used for high bandwidth transport of data between hosts and storage systems (e.g., disk, tape). Traditionally, SCSI commands have been transported over dedicated networks such as SCSI buses and Fiber Channel. With the emergence of Gigabit and 10 Gb/s Ethernet LANs, it is now feasible to transport SCSI commands over commodity networks and yet provide high throughput to bandwidth-intensive storage applications. To do so, iSCSI connects a SCSI initiator port on a host to a SCSI target port on a storage subsystem. For the sake of uniformity with NFS, we will refer to the initiator and the target as an iSCSI client and server, respectively.

Some of the salient features of iSCSI are as follows: (i) it uses the notion of a session between the client and the server to identify a communication stream between the two; (ii) it allows multiple connections to be multiplexed into a session; (iii) it supports advanced data integrity, authentication protocols as well as encryption (IPSEC)—these features are negotiated at session-startup time; and (iv) it supports advanced error recovery using explicit retransmission requests, markers and connection allegiance switching [26].

2.3 Caching in NFS and iSCSI

In NFS, the file system is located on the server and so is the file system cache. NFS clients also employ a cache that can hold both data and meta-data. To ensure consistency across clients, NFS v2 and v3 require that client perform consistency checks with the server on cached data and meta-data. The validity of cached data at the client is implementation-dependent—in Linux, cached meta-data is treated as potentially stale after 3 seconds and cached data after 30 seconds. Thus, meta-data and data reads may trigger a message exchange (i.e., a consistency check) with the server even in the event of a cache hit. NFS v4 can avoid this message exchange for data reads if the server supports file delegation. From the perspective of writes, both data and meta-data writes in NFS v2 are synchronous. NFS v3 and v4 supports asynchronous data writes, but meta-data updates continue to be synchronous. Thus, depending on the version, NFS has different degrees of write-through caching.

In iSCSI, the caching policy is governed by the file system. Since the file system cache is located at the client, both data and meta-data reads benefit from any cached content. Data updates are asynchronous in most file systems. In modern file systems, meta-data updates are also asynchronous, since such systems use log-based journaling for faster recovery. In the ext3 file system, for instance, meta-data is written asynchronously at commit points. The asynchrony and frequency of these commit points is a tradeoff between recovery and performance (ext3 uses a commit interval of 5 seconds). Thus, when used in conjunction with ext3, iSCSI supports a fully write-back cache for data and meta-data updates.

3 Experimental Setup and Methodology

In this section, we first describe the storage testbed used for our experiments and then our experimental methodology.

3.1 System Setup

The storage testbed used in our experiments consists of a server and a client connected over an isolated Gigabit Ethernet LAN (see Figure 2). Our server is a dual processor machine with two 933 MHz Pentium-III processors, 256 KB L1 cache, 1 GB of main memory and an Intel 82540EM Gigabit Ethernet card. The server contains an Adaptec ServeRAID adapter card that is connected to a Dell PowerVault disk pack with fourteen SCSI disks; each disk is a 10,000 RPM Ultra-160 SCSI drive with 18 GB storage capacity. For the purpose of our experiments, we configure the storage subsystem as two identical RAID-5 arrays, each in a 4+p configuration (four data disks plus a parity disk). One array is used for our NFS experiments and the other for the iSCSI experiments. The client is a 1 GHz Pentium-III machine with 256KB L1 cache, 512 MB main memory, and an Intel 82540EM Gigabit Ethernet card.

Both machines run RedHat Linux 9. We use version 2.4.20 of the Linux kernel on the client for all our experiments. For the server, we use version 2.4.20 as the default kernel, except for the iSCSI server which requires kernel version 2.4.2 and the NFS version 4 server which requires 2.4.18. We use the default Linux implementation of NFS versions 2 and 3 for our experiments. For NFS version 4, which is yet to be fully supported in vanilla Linux, we use the University of Michigan implementation (release 2 for Linux 2.4).

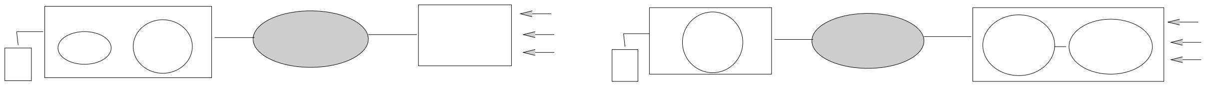
For iSCSI, we employ the open-source SourceForge Linux iSCSI implementation as the client (version 3.3.0.1) and a commercial implementation as the iSCSI server (details withheld for blind reviewing). While we found several high-quality open-source iSCSI client implementations, we were unable to find a stable open-source iSCSI server implementation that was compatible with our hardware setup; consequently, we chose a commercial server implementation.

The default file system used in our experiments is *ext3*. The file system resides at the client for iSCSI and at the server for NFS (see Figure 2). We use TCP as the default transport protocol for both NFS and iSCSI, except for NFS v2 where UDP is the transport protocol.

3.2 Experimental Methodology

We experimentally compare NFS versions 2, 3 and 4 with iSCSI using a combination of micro- and macro-benchmarks. The objective of our micro-benchmarking experiments is to measure the network message overhead of various file and directory operations in the two protocols, while our macro-benchmarks experimentally measure overall application performance.

Our micro-benchmarks measure the network message overhead (number of network messages) for a variety of system calls that perform file and directory operations. We first measure the network message overhead assuming a cold cache at the client and the server and then repeat the experiment for a warm cache. By using a cold and warm cache, our experiments capture the worst and the average case, respectively, for the network message overhead. Since the network message overhead depends on the directory depth (path length), we also measure these overheads for varying directory depths. In case of file reads and writes, the network message overhead is dependent on (i) the I/O size, and (ii) the nature of the workload (i.e., random or sequential). Consequently, we measure the network message overhead for vary-



(a) NFS setup

(b) iSCSI setup

Figure 2: Experimental setup. The figures depict the setup used for our NFS and iSCSI experiments.

ing I/O sizes as well as sequential and random requests. We also study the impact of the network latency between the client and the server on the two systems.

Having understood the overheads imposed by individual file and directory operations (at the granularity of system calls), we then turn our attention to overall application performance. We measure application performance using several popular benchmarks that impose both data- and meta-data-intensive workloads. We use the following benchmarks in our study:

- *PostMark:* This is a file system benchmark that measures performance for ISP (i.e., mail, netnews) and e-commerce workloads. This benchmarks measures system performance for small files that are constantly being created, deleted, read and written, as is common in email, news, and e-commerce environments. The overall workload imposed by PostMark is meta-data intensive.
- *IOZone:* IOZone is a file system benchmark that measures system performance for file system and database environments. It measures I/O performance for a variety of file sizes and a variety of read and write sizes on these files. It also emulates a variety of access patterns (sequential, random, mixed, backward, and strided). IOZone is mostly data-intensive in nature.
- *TPC-C:* TPC-C is a widely used database benchmark that emulates an online transaction processing (OLTP) environment. TPC-C is a data-intensive workload that consists of small random I/O requests. The benchmark consists of a mix of reads and writes (approximately two-thirds reads and one-third writes [2]).
- *TPC-H:* TPC-H is another widely used database benchmark that emulates a decision support system (DSS). TPC-H is a read-intensive workload that measures overall system throughput for long-running ad-hoc queries (large I/O sizes).
- *Other benchmarks:* We also measure the performance of the two systems for the following operations: (i) *untar*, which measures the time to untar the Linux 2.4.20 kernel source tree, (ii) *kernel compilation*, which measures the time to compile the Linux 2.4.20 kernel, (iii) *recursive file listing*, which uses `ls -lR > /dev/null` to recursively lists all files in the Linux kernel source tree, and (iv) *recursive delete*, which deletes the Linux kernel source tree. Observe that untar is write- and meta-data intensive, kernel compilation is cpu-, read-, and write-intensive, while the remaining two operations are meta-data intensive.

We use a variety of tools to understand system behavior for our experiments. We use *Ethereal* to monitor network packets, the *Linux Trace toolkit* and *vmstat* to measure protocol processing times, and *nfsstat* to obtain nfs message statistics. We also instrument the Linux kernel to measure iSCSI network message overheads. Finally, we use logging in the VFS layer to trace the generation of network traffic for NFS. While we use these tools to obtain a detailed understanding of system behavior, reported performance results (for instance, for the various benchmarks) are without the various monitoring tools (to prevent the overhead of these tools from influencing performance results).

4 Micro-benchmarking Experiments

In this section, we compare the network message overheads of various file and directory operations so as to understand the impact of moving file system functionality closer the client. Our micro-benchmark analysis primarily focuses on protocol message counts as well as their sensitivity to file system parameters.

4.1 Overhead of System Calls

Our first experiment determines network message overheads for common file and directory operations at the granularity of system calls. We consider sixteen commonly-used system calls shown in Table 1 and measure their network message overheads using the *Ethereal* packet monitor. Note that this list does not include the *read* and *write* system calls, which are examined separately in Section 4.4.

For each system call, we first measure its network message overhead assuming a cold cache and repeat the experiment for a warm cache. We emulate a cold cache by unmounting and remounting the file system at the client and restarting the NFS server or the iSCSI server; this is done prior to each invocation of a system call. The warm cache is emulated by invoking the system call on a cold cache and then repeating the system call with similar (though not identical) parameters. For instance, to understand warm cache behavior, we create two directories in the same parent directory using `mkdir`, we open two files in the same directory using `open`, or we perform two different `chmod` operation on a file. In each case, the network message overhead of the second invocation is assumed to be the overhead in the presence of a warm cache.[1]

---

[1] Depending on the exact cache contents, the warm cache network message overhead can be different for different caches. We carefully choose the system call parameters so as to emulate a "reasonable" warm cache. Moreover, we deliberately choose slightly different parameters across system call

Table 1: File and directory-related system calls.

Directory operations	File operations
Directory creation (mkdir)	File create (creat)
Directory change (chdir)	File open (open)
Read directory contents (readdir)	Hard link to a file (link)
Directory delete (rmdir)	Truncate a file (truncate)
Symbolic link creation (symlink)	Change access permissions (chmod)
Symbolic link read (readlink)	Change ownership (chown)
Symbolic link delete (unlink)	Query file permissions (access)
	Query file attributes (stat)
	Alter file access time (utime)

The directory structure can impact the network message overhead for a given operation. Consequently, we report overheads for a directory depth of zero and a directory depth of three. Section 4.3 reports additional results obtained by systematically varying the directory depth from 0 to 16.

Table 2: Network message overheads for a cold cache.

	Directory depth 0				Directory depth 3			
Operation	V2	V3	V4	iSCSI	V2	V3	V4	iSCSI
mkdir	2	2	4	7	5	5	10	13
chdir	1	1	3	2	4	4	9	8
readdir	2	2	4	6	5	5	10	12
symlink	3	2	4	6	6	5	10	12
readlink	2	2	3	5	5	5	9	10
unlink	2	2	4	6	5	5	10	11
rmdir	2	2	4	8	5	5	10	14
creat	3	3	10	7	6	6	16	13
open	2	2	7	3	5	5	13	9
link	4	4	7	6	10	9	16	12
rename	4	3	7	6	10	10	16	12
trunc	3	3	8	6	6	6	14	12
chmod	3	3	5	6	6	6	11	12
chown	3	3	5	6	6	6	11	11
access	2	2	5	3	5	5	11	9
stat	3	3	5	3	6	6	11	9
utime	2	2	4	6	5	5	10	12

Table 2 depicts the number of messages exchanged between the client and server for NFS versions 2, 3, 4 and iSCSI assuming a cold cache.

We make three important observations from the table. First, on an average, iSCSI incurs a higher network message overhead than NFS. This is because a single message is sufficient to invoke a file system operation on a path name in case of NFS. In contrast, the path name must be completely resolved in case of iSCSI before the operation can proceed; this results in additional message exchanges. Second, the network message overhead increases as we increase the directory depth. For NFS, this is due to the additional access checks on the pathname. In case of iSCSI, the file system fetches the directory inode and the directory contents at each level in the path name. Since directories and their inodes may be resident on different disk blocks, this triggers additional block reads. Third, NFS version 4 has a higher network message overhead when compared to NFS versions 2 and 3, which have a comparable overhead. The higher overhead in NFS version 4 is due to access checks performed by the client via the *access* RPC call.[2]

We make one additional observation that is not directly reflected in Table 2. The average message size in iSCSI can be higher than that of NFS. Since iSCSI is a block access protocol, the granularity of reads and writes in iSCSI is a disk block, whereas RPCs allow NFS to read or write smaller chunks of data. While reading entire blocks may seem wasteful, a side-effect of this policy is that iSCSI benefits from aggressive caching. For instance, reading an entire disk block of inodes enable applications with meta-data locality to benefit in iSCSI. In the absence of meta-data or data locality, however, reading entire disk blocks may hurt performance.

Table 3: Network message overheads for a warm cache.

	Directory depth 0				Directory depth 3			
Operation	v2	v3	v4	iSCSI	v2	v3	v4	iSCSI
mkdir	2	2	2	2	4	4	3	2
chdir	1	1	0	0	3	3	2	0
readdir	1	1	0	2	3	3	3	2
symlink	3	2	2	2	5	4	4	2
readlink	1	2	0	2	3	3	3	2
unlink	2	2	2	2	5	4	3	2
rmdir	2	2	2	2	4	4	3	2
open	3	2	6	2	5	5	9	2
creat	4	3	2	2	6	4	6	2
open	1	1	4	0	4	4	6	0
rename	4	3	2	2	6	6	6	2
trunc	2	2	4	2	5	5	7	2
chmod	2	2	2	2	4	5	5	2
chown	2	2	2	2	4	5	5	2
access	1	1	1	2	4	4	3	0
stat	2	2	2	2	5	5	5	0
utime	1	1	1	2	4	4	4	2

Table 3 depicts the number of messages exchanged between the client and the server for warm cache operations. Whereas iSCSI incurred a higher network message overhead than NFS in the presence of a cold cache, it incurs a comparable or lower network message overhead than NFS in the presence of a warm cache. Further, the network message overhead is identical for directory depths of zero and three for iSCSI, whereas it increases with directory depth for NFS. Last, both iSCSI and NFS benefit from a warm cache and the overheads for each operation are smaller than those for a cold cache. The better performance of iSCSI can be attributed to aggressive meta-data caching performed by the file system; since the file system is resident at the client, many requests can be serviced directly from the client cache. This is true even for long path names, since all directories in the path may be cached from a prior operation. NFS is unable to extract these benefits despite using a client-side cache, since NFS v2

---

invocations; identical invocations will result in a hot cache (as opposed to a warm cache) and result in zero network message overhead for many operations.

[2] The *access* RPC call was first introduced in NFS V3. Our Ethereal logs did not reveal its use in the Linux NFS v3 implementation, other than for root access checks. However, the NFS v4 client uses it extensively to perform additional access checks on directories and thereby incurs a higher network message overhead.

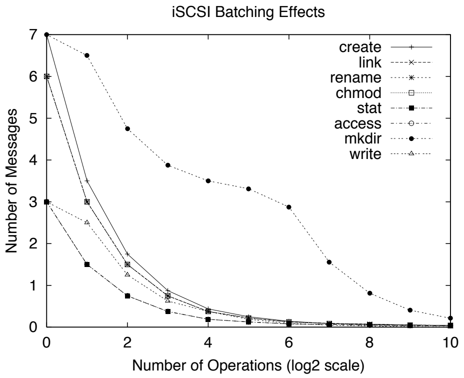


Figure 3: Benefit of meta-data update aggregation and caching in iSCSI. The figure shows the amortized network message overhead per operation for varying batch sizes. The batch size is shown on a logarithmic scale.

and v3 need to perform consistency checks on cached entries, which triggers message exchanges with the server. Further, meta-data update operations are necessarily synchronous in NFS, while they can be asynchronous in iSCSI. This asynchronous nature enables applications to update a dirty cache block multiple times prior to a flush, thereby amortizing multiple meta-data updates into a single network block write.

4.2 Impact of Meta-data Caching and Update Aggregation

Our micro-benchmark experiments revealed two important characteristics of iSCSI—aggressive meta-data caching, which benefits meta-data reads, and update aggregation, which benefits meta-data writes. Recall that, update aggregation enables multiple writes to the same dirty block to be "batched" into a single asynchronous network write. We explore this behavior further by quantifying the benefits of update aggregation and caching in iSCSI.

We choose eight common operations that read and update meta-data, namely `creat`, `link`, `rename`, `chmod`, `stat`, `access`, `write` and `mkdir`. For each operation, we issue a batch of N consecutive calls of that operation and measure the network message overhead of the entire batch. We vary N from 1 to 1024 (e.g., issue 1 mkdir, 2 mkdirs, 4 mkdirs and so on, while starting with a cold cache prior to each batch). Figure 3 plots the amortized network message overhead per operation for varying batch sizes. As shown, the amortized overhead reduces significantly with increasing batch sizes, which demonstrates that update aggregation can indeed significantly reduce the number of network writes. Note that some of the reduction in overhead can be attributed to meta-data caching in iSCSI. Since the cache is warm after the first operation in a batch, subsequent operations do not yield additional caching benefits—any further reduction in overhead is solely due to update aggregation. In general, our experiment demonstrates applications that exhibit meta-data locality can benefit significantly from update aggregation.

4.3 Impact of Directory Depth

Our micro-benchmarking experiments gave a preliminary indication of the sensitivity of the network message overhead to the depth of the directory where the file operation was performed. In this section, we examine this sensitivity in detail by systematically varying the directory depth.

For each operation, we vary the directory depth from 0 to 16 and measure the network message overhead in NFS and iSCSI for the cold and warm cache. A directory depth of i implies that the operation is executed in mnt_point : $/dir1/.../diri$. Figure 4 lists the observed overhead for three different operations. We omit results for the remaining operations due to space constraints.

In case of the cold cache, iSCSI needs two extra messages for each increase in directory depth due to the need to access the directory inode as well as the directory contents. In contrast, NFS v2 and v3 need only one extra message for each increase in directory depth, since only one message is needed to access directory contents—the directory inode lookup is done by the server. As indicated earlier, NFS v4 performs an extra access check on each level of the directory via the *access* call. Due to this extra message, it overhead matches that of iSCSI and increases in tandem.[3] Consequently, as the directory depth is increased, the iSCSI overhead increases faster than NFS for the cold cache.

In contrast, a warm cache results in a constant number of messages independent of directory depth due to meta-data caching at the client for both NFS and iSCSI. The observed messages are solely due to the need to update meta-data at the server.

4.4 Impact of Read and Write Operations

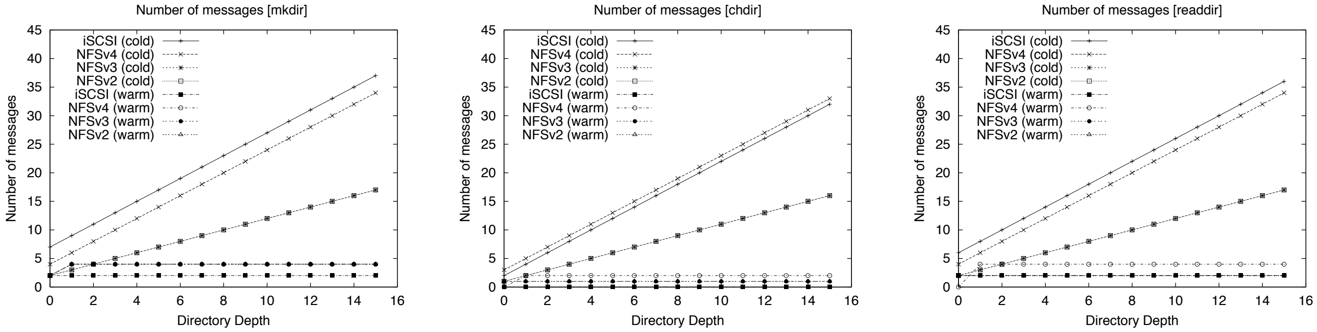
Our experiments thus far have focused on meta-data operations. In this section, we study the efficiency of data operations in NFS and iSCSI. We consider the *read* and *write* system calls and measure their network message overheads in the presence of a cold and a warm cache.

To measure the read overhead, we issue reads of varying sizes—128 bytes to 64 KB—and measure the resulting network message overheads in the two systems. To emulate a cold cache, we ensure that the cache is empty prior to each invocation of read. For the warm cache, we first read the entire file into the cache and then issue sequential reads of increasing sizes. The write overhead is measured similarly for varying write sizes. The cold cache is emulated by emptying the client and server caches prior to the operation. Writes are however not measured in warm cache mode—we use macro-benchmarks to quantify warm cache effects.

Figure 5 plots our results. We make the following observations from our results. For read operations, iSCSI requires

---

[3] The extra overhead of *access* is probably an artifact of the implementation. It is well-known that the Linux NFS implementation does not correctly implement the *access* call due to inadequate caching support at the client [19]. This idiosyncrasy of Linux is the likely cause of the extra overhead in NFS v4.



(a) `mkdir` (b) `chdir` (c) `readdir`

Figure 4: Effect of the directory depth on the network message overhead.

one or two extra messages over NFS so to read or update uncached file meta-data (e.g., inode blocks). While NFS incurs a smaller overhead for small cold reads, the read overhead exceeds that of iSCSI beyond 8KB requests. For NFS v2, this is due to the maximum data transfer limit of 8KB imposed by the protocol specification. Multiple data transfers are needed when the read request size exceeds this limit. Although NFS v3 eliminates this restriction, it appears that the Linux NFS v3 implementation does not take advantage of this flexibility and uses the same transfer limit as NFS v2. Consequently, the cold read overhead of NFS v3 also increases beyond that of iSCSI for large reads. In contrast, the NFS v4 implementation uses larger data transfers and incurs fewer messages. In case of the warm cache, since the file contents are already cached at the client, the incurred overhead in NFS is solely due to the consistency checks performed by the client. The observed overhead for iSCSI is due to the need to update the access time in the inode.

Similar observations are true for write requests (see Figure 5(c)). Initially, the overhead of iSCSI is higher primarily due to the need to access uncached meta-data blocks. For NFS, all meta-data lookups take place at the server and the network messages are dominated by data transfers. The network message overhead for NFS v2 increases once the write request size exceeds the maximum data transfer limit; the overhead remains unchanged for NFS v4.

4.5 Impact of Sequential and Random I/O

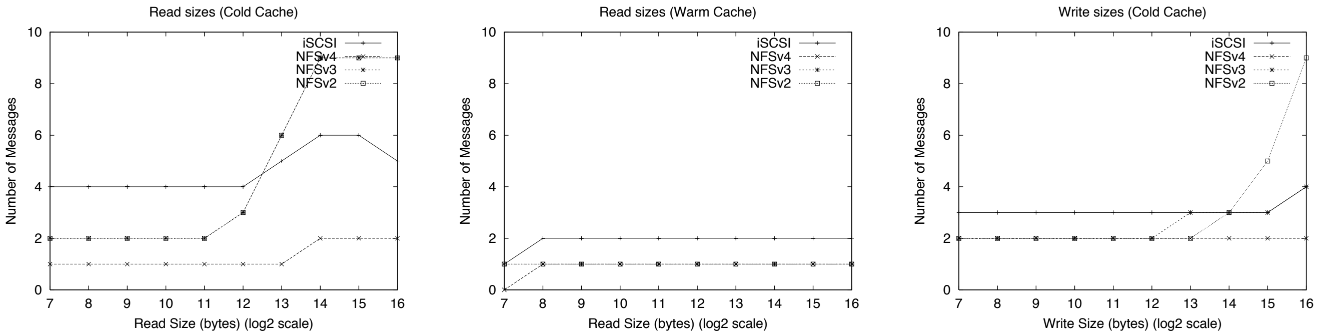
Two key factors impact the network message overheads of data operations—the size of read and write requests and the access characteristics of the requests (sequential or random). The previous section studied the impact of request sizes on the network message overhead. In this section, we study the effect of sequential and random access patterns on network message overheads.

To measure the impact of reads, we create a 128MB file. We then empty the cache and read the file sequentially in 4KB chunks. For random reads, we create a random permutation of the 32K blocks in the file and read the blocks in that order. We perform this experiment first for NFS v3 and then for iSCSI. We do not report NFS v2 and v4 results due to space constraints. Table 4 depicts the completion times, network message overheads and bytes transferred in the two systems. As can be seen, for sequential reads, both NFS and iSCSI yield comparable performance. For random reads, NFS is slightly worse (by about 15%). The network message overheads and the bytes transfered are also comparable for iSCSI and NFS.

Next, we repeat the above experiment for writes. We create an empty file and write 4KB data chunks sequentially to a file until the file size grows to 128MB. For random writes, we generate a random permutation of the 32K blocks in the file and write these blocks to newly created file in that order. Table 4 depicts our results. Unlike reads, where NFS and iSCSI are comparable, we find that iSCSI is significantly faster than NFS for both sequential and random writes. The lower completion time of iSCSI is due to the asynchronous writes in the ext3 file system. Since NFS version 3 also supports asynchronous writes, we expected the NFS performance to be similar to iSCSI. However, it appears that the Linux NFS v3 implementation can not take full advantage of asynchronous writes, since it specifies a limit on the number of pending writes in the cache. Once this limit is exceeded, the write-back caches degenerates to a write-through cache and application writes see a pseudo-synchronous behavior. Consequently, the NFS write performance is significantly worse than iSCSI. Note also, while the byte overhead is comparable in the two systems, the number of messages in iSCSI is significantly smaller than NFS. This is because iSCSI appears to issue very large write requests to the server (mean request size is 128KB as opposed to 4.7KB in NFS).

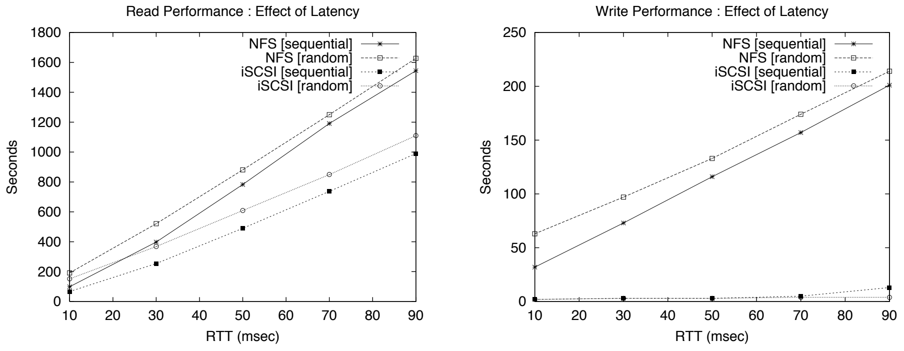
4.6 Impact of Network Latency

Our experiments thus far have assumed a lightly loaded Gigabit Ethernet LAN. The observed round trip times on our LAN is very small (<1ms). In practice, the latency between the client and the server can vary from a few milliseconds to tens of milliseconds depending on the distance between the client and the server. Consequently, in this section, we vary the network latency between the two machines and study its



(a) Cold reads (b) Warm reads (c) Cold Writes

Figure 5: Network message overheads of read and write operations of varying sizes.



(a) Reads (b) Writes

Figure 6: Impact of network latency on read and write performance.

Table 4: Sequential and Random reads and writes: completion times, number of messages and bytes transferred for reading and writing a 128MB file.

	Performance		Messages		Bytes	
	NFS v3	iSCSI	NFS v3	iSCSI	NFS v3	iSCSI
Seq. reads	35s	35s	33,362	32,790	153MB	148MB
Rnd. reads	64s	55s	32,860	32,827	153MB	148MB
Seq. writes	17s	2s	32,990	1135	151MB	143MB
Rnd. writes	21s	5s	33,015	1150	151MB	143MB

impact on performance.

We use the NISTNet package to introduce a latency between the client and the server. NISTNet introduces a preconfigured delay for each outgoing and incoming packet so as to simulate wide-area conditions. We vary the round-trip network latency from 10ms to 90ms and study its impact on the sequential and random reads and writes. The experimental setup is identical to that outlined in the previous section. Figure 6 plots the completion times for reading and writing a 128 MB file for NFS and iSCSI. As shown in Figure 6(a), the completion time increases with the network latency for both systems. However, the increase is greater in NFS than in iSCSI—the two systems are comparable at low latencies ($\leq$ 10ms) and the NFS performance degrades faster than iSCSI for higher latencies. Even though NFS v3 runs over TCP, an *Ethereal* trace reveals an increasing number of RPC retransmissions at higher latencies. The Linux NFS client appears to time-out more frequently at higher latencies and reissues the RPC request, even though the data is in transit, which in turn dregrades performance. An implementation of NFS that exploits the error recovery at the TCP layer will not have this drawback.

In case of writes, the iSCSI completion times are not affected by the network latency due to their asynchronous nature. The NFS performance is impacted by the pseudo-synchronous nature of writes in the Linux NFS implementation (see Section 4.5) and increases with the latency.

5 Macro-benchmarking Experiments

Having examined the behavior of NFS and iSCSI for individual file and directory operations, in this section, we measure overall application performance in the two systems using a number of benchmarks. Our benchmarks have different access profiles with respect to data and meta-data (e.g., meta-data-intensive, data-intensive, read-intensive, mixed, etc.). Consequently, these benchmarks should provide important

Table 5: PostMark Results. Completion times and message counts are reported for 100,000 operations on 1,000, 5,000 and 25,000 files.

Files	Completion time (s)		Messages	
	NFS v3	iSCSI	NFS v3	iSCSI
1,000	146	12	371,963	101
5,000	201	35	451,415	276
25,000	516	208	639,128	66,965

insights into the fundamental behavior of block- and file-access protocols. Due to space constraints, we only compare NFS v3 with iSCSI and omit NFS v2 and v4 results. We also note that NFS v4 support on Linux is not fully mature and less optimized than NFS v3.

5.1 PostMark Results

As explained earlier, PostMark is a benchmark that demonstrates system performance for short-lived small files seen typically in Internet applications such as electronic mail, netnews and web-based commerce. The benchmark creates an initial pool of random text files of varying size. Once the pool has been created, the benchmark performs two types of transactions on the pool: (i) create or delete a file; (ii) read from or append to a file. The incidence of each transaction and its subtype are chosen randomly to eliminate the effect of caching and read-ahead.

In this particular setup, we maintained a equal predisposition to each type of transaction as well as each subtype within a transaction. We performed 100,000 transactions on a pool of files whose size was varied from 1,000 to 25,000 in multiples of 5.

Table 5 depicts our results. As shown in the table, iSCSI generally outperforms NFS v3 due to the meta-data intensive nature of this benchmark. An analysis of the NFS v3 protocol messages exchanged between the server and the client shows that 65% of the messages are meta-data related. Meta-data update aggregation as well as aggressive meta-data caching in iSCSI enables it to have a significantly lower message count than NFS.

As the pool of files is increased, we noted that the benefits of meta-data caching and meta-data update aggregation starts to diminish due to the random nature of the transaction selection. As can be seen in Table 5, the number of messages relative to the file pool size increases faster in iSCSI than that in NFS v3. Consequently, the performance difference between the two decreases. However, as a side effect, the benchmark also reduces the effectiveness of meta-data caching on the NFS server, leading to higher server CPU utilization (see Section 5.5).

5.2 IOZone Results

IOZone is a file system benchmark that analyzes overall system performance. The benchmark tests I/O performance for a variety of file system operations such as read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread, mmap, aio_read and aio_write. Although, we experimented with all permissible operations in Linux, we report only the sequential read and write performance due to space constraints. Our observation also apply to other access patterns and detailed results can be found in an extended technical report.

Figure 7 depicts IOZone performance for sequential reads and writes of different sizes. IOZone also varied the file size from 8KB to 512MB; since the read and write behavior was similar across entire spectrum of file sizes, we report results only for two file sizes—16MB and 256MB.

As shown in Figure 7(a), there is little difference between iSCSI and NFS v3 for sequential reads. This is because an analysis of the NFS v3 protocol traffic shows predominantly read data messages being exchanged between the server and the client. For such a traffic profile, there is little difference in protocol behavior between NFS v3 and iSCSI, and consequently, the IOZone performances of both iSCSI and NFS v3 are identical.

Note that, these results do not reveal any inefficiency in iSCSI for a pure data-intensive application. A read request requires iSCSI to fetch the corresponding file meta-data before the read can be issued. In the case of NFS, a client can immediately issue a read since meta-data lookups take place at the server. Meta-data caching in iSCSI seems to outweigh the inefficiency of this extra meta-data read.

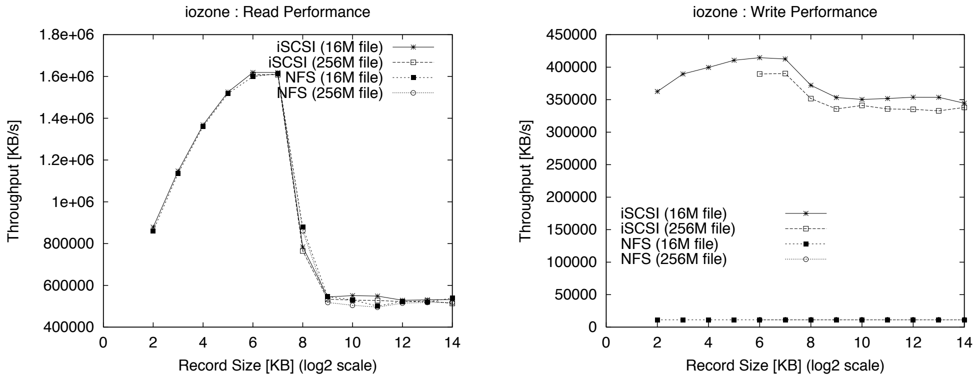
Figure 7(b) shows a significant difference between iSCSI and NFS v3 for sequential writes. An analysis of the NFS v3 protocol traffic shows predominantly write data messages being exchanged between the client and the server. Since both iSCSI and NFS v3 support asynchronous writes, one would have expected the two systems to exhibit similar write performance.

However, as explained in Section 4.5, the Linux NFS client imposes a maximum limit on pending asynchronous writes [19]. Once this limit is exceeded, the write-back cache at the client degenerates to a write-through cache due to the need to flush pending writes. Consequently, write-intensive workloads see a pseudo-synchronous write behavior, and a corresponding degradation in performance. No such limitation is imposed by the ext3 file system in iSCSI, and consequently, write-intensive applications benefit from a true write-back cache and asynchronous writes.

5.3 TPC-C and TPC-H Results

TPC-C is an On-Line Transaction Processing (OLTP) benchmark that issues small random I/Os. The benchmark consists of highly-multithreaded 4 KB block transfers where two-thirds of the transfers are reads. We set up TPC-C with 300 warehouses and 30 clients. We use IBM's DB2 database for Linux (version 8.1 Enterprise Edition). The metric for evaluating TPC-C performance is the number of transactions completed per minute (tpmC).

Table 6 shows the TPC-C performance and the network



(a) Read throughput

(b) Write throughput

Figure 7: Throughput of read and write operations in IOZone

Table 6: TPC-C Results. Reported throughput (tpmC) is normalized by a factor α equivalent to the throughput obtained with NFS v3.

Peak Throughput (TpmC)		Messages	
NFS v3	iSCSI	NFS v3	iSCSI
α	$1.08 *\alpha$	517,219	530,745

Table 7: TPC-H Results. Reported throughput (QphH@1GB) is normalized by a factor β equivalent to the throughput obtained in NFS v3.

Throughput (QphH@1GB)		Messages	
NFS v3	iSCSI	NFS v3	iSCSI
β	$1.07 *\beta$	261,769	62,686

Table 8: Completion times for other benchmarks.

Benchmark	NFS v3	iSCSI
`tar -xzf`	60s	5s
`ls -lR > /dev/null`	12s	6s
`kernel compile`	222s	193s
`rm -rf`	40s	22s

message overhead for NFS and iSCSI. Since these are results from an unaudited run, we withhold the actual results and instead report normalized throughput for the two systems.[4] As shown in the table, there is a marginal difference between NFS v3 and iSCSI. This is not surprising since TPC-C is primarily data-intensive and as shown in earlier experiments, iSCSI and NFS are comparable for data-intensive workloads. An analysis of the message count shows that the vast majority of the NFS v3 protocol traffic (99%) is either a data read or a data write. The two systems are comparable for read operations. Since data writes are 4KB each and less-intensive than in other benchmarks, NFS is able to benefit from asynchronous write support and is comparable to iSCSI.

The TPC-H benchmark emulates a decision support systems that examines large volumes of data, executes queries with a high degree of complexity, and gives answers to critical business questions. Our TPC-H experiments use a database scale factor of 1 (implying a 1 GB database). The page size and the extent size for the database were chosen to be 4 KB and 32 KB, respectively. We run the benchmark for iSCSI and NFS and report the observed throughout and network message overheads in Table 7. Again, we report normalized throughputs since our results are unaudited. The reported throughput for TPC-H is the number of queries per hour for a given database size (QphH@1GB in our case).

We find the performance of NFS and iSCSI is comparable for TPC-H. Since the benchmark is dominated by large read requests—an analysis of the traffic shows that the vast majority of the messages are data reads—this result is consistent with prior experiments where iSCSI and NFS were shown to have comparable performance for read-intensive workloads.

5.4 Other Benchmarks

We also used several simple macro-benchmarks to characterize the performance of iSCSI and NFS. As explained earlier, these benchmarks include extracting the Linux kernel source tree from a compressed archive (tar xfz), listing the contents (ls -lR), compiling the source tree (make) and finally removing the entire source tree (rm -rf). The first, second and fourth benchmarks are met-data intensive and amenable to meta-data caching as well as meta-data update aggregation. Consequently, in these benchmarks, iSCSI performs better than NFS v3. The third benchmark, which involves compiling the Linux kernel, is CPU-intensive, and consequently there is parity between iSCSI and NFS v3. The marginal difference between the two can be attributed to the impact of the iSCSI protocol's reduced processing length on the single-threaded compiling process.

---

[4] The Transaction Processing Council does not allow unaudited results to be reported.

Table 9: Server CPU utilization for various benchmarks. The 99^{th} percentile of the CPU utilization at the server is reported for each benchmark.

	NFS v3	iSCSI
PostMark	77%	13%
IOZone	34%	15%
TPC-C	13%	7%
TPC-H	20%	11%

5.5 Server CPU utilization

A key performance attribute of a protocol is its scalability with respect to the number of clients that can be supported by the server. If the network paths or I/O channels are not the bottleneck, the scalability is determined by the server CPU utilization for a particular benchmark.

Table 9 depicts the 99^{th} percentile of the server CPU utilization reported every 2 seconds by `vmstat` for the various benchmarks. The table shows that, the server utilization for iSCSI is lower than that of NFS. The server utilization is governed by the processing path and the amount of processing for each request. The lower utilization of iSCSI can be attributed to the smaller processing path seen by iSCSI requests. In case of iSCSI, a block read or write request at the server traverses through the network layer, the SCSI server layer, and the low-level block device driver. In case of NFS, an RPC call received by the server traverses through the network layer, the NFS server layer, the VFS layer, the local file system, the block layer, and the low-level block device driver. Our measurements indicate that the server processing path for NFS requests is twice that of iSCSI requests. This is confirmed by the server CPU utilization measurements for data intensive TPC-C and TPC-H benchmarks. In these benchmarks, the server CPU utilization in for NFS is twice that of iSCSI. A similar observation holds for the IOZone benchmark

The difference is exacerbated for meta-data intensive workloads. A NFS request that triggers a meta-data lookup at the server can greatly increase the processing path—meta-data reads require multiple traversals of the VFS layer, the file system, the block layer and the block device driver. The number of traversals depends on the degree of meta-data caching in the NFS server. The increased processing path explains the large disparity in the observed CPU utilizations for PostMark. The PostMark benchmark tends to defeat the meta-data caching on the NFS server because of the random nature of transaction selection. This causes the server CPU utilization to increase significantly since multiple block reads may be needed to satisfy a single NFS data read.

6 Summary and Discussion of Our Results

In this section, we summarize our micro- and macro-benchmarking results and discuss their implications for IP-networked storage.

6.1 Data-intensive applications

Overall, we find that iSCSI and NFS yield comparable performance for data-intensive applications. Whereas this result is broadly true for read-intensive workloads, it is true with a few caveats for write-intensive or mixed workloads.

In particular, we find that any application that generates predominantly read-oriented network traffic will see comparable performance in iSCSI and NFS v3. Since NFS v4 does not make significant changes to those portions of the protocol that deal with data transfers, we do not expect this situation to change in the future. Furthermore, the introduction of hardware protocol acceleration is likely to improve the data transfer part of both iSCSI and NFS in comparable ways.

In principle, we expect iSCSI and NFS to yield comparable performance for write-intensive workloads as well. However, due to the idiosyncrasies of the Linux NFS implementation, we find that iSCSI significantly outperforms NFS v3 for such workloads. We believe this is primarily due to the limit on the number of pending asynchronous writes at the NFS client. We find that this limit is quickly reached for very write-intensive workloads, causing the write-back cache at the NFS client to degenerate into a write-through cache. The resulting pseudo-synchronous write behavior causes a substantial performance degradation (by up to an order of magnitude) in NFS. We speculate that an increase in the pending writes limit and optimizations such as spatial write aggregation in NFS can eliminate the gap between the two protocols.

Although the two protocols yield comparable application performance, we find that they result in different server CPU utilizations. In particular, we find that the server utilization is twice as high in NFS than in iSCSI. We attribute this increase primarily due to the increased processing path in NFS when compared to iSCSI. An implication of the lower utilization in iSCSI is that the server is more scalable (i.e., it can service twice as many clients with the caveat that there is no sharing between client machines).

6.2 Meta-data intensive applications

NFS and iSCSI show their greatest differences in their handling of meta-data intensive applications. Overall, we find that iSCSI outperforms NFS for meta-data intensive workloads—workloads where the network traffic is dominated by meta-data accesses.

The better performance of iSCSI can be attributed to two factors. First, NFS requires clients to update meta-data synchronously to the server. In contrast, iSCSI, when used in conjunction with modern file systems, updates meta-data asynchronously. An additional benefit of asynchronous meta-data updates is that it enables update aggregation—multiple meta-data updates to the same cached cached block are aggregated into a single network write, yielding significant savings. Such optimizations are not possible in NFS v2 or v3 due to their synchronous meta-data update requirement.

Second, iSCSI also benefits from aggressive meta-data caching by the file system. Since iSCSI reads are in granularity of disk blocks, the file system reads and caches entire blocks containing meta-data; applications with meta-data locality benefit from such caching. Although the NFS client can also cache meta-data, NFS clients need to perform periodic consistency checks with the server to provide weak consistency guarantees across client machines that share the same NFS namespace. Since the concept of sharing does not exist in the SCSI architectural model, the iSCSI protocol also does not pay the overhead of such a consistency protocol.

6.3 Applicability to Other File Access Protocols

An interesting question is the applicability of our results to other protocols such as NFS v4, DAFS [4] and SMB.

The SMB protocol is similar to NFS v4 in that both provide support for strong consistency. Consistency is ensured in SMB by the use of opportunistic locks or oplocks which allow clients to have exclusive access over a file object. The DAFS protocol specification is based on NFS v4 with additional extensions for hardware-accelerated performance, locking and failover. These extensions do not affect the basic protocol exchanges that we observed in our performance analysis.

NFS v4, DAFS and SMB do not allow a client to update meta-data asynchronously. NFS v4 and DAFS allow the use of compound RPCs to aggregate related meta-data requests and reduce network traffic. This can improve performance in meta-data intensive benchmarks such as PostMark. However, it is not possible to speculate on the actual performance benefits, since it depends on the degree of compounding.

Thus, we believe that our results are generally applicable to these other widely used file protocols with the above caveats.

6.4 Implications

Extrapolating from our NFS and iSCSI results, it appears that block- and file-access protocols are comparable on data-intensive benchmarks and the former outperforms the latter on the meta-data intensive benchmarks. From the perspective of IP-networked storage, this result favors a block-access protocol over a file-access protocol. This is especially true when sharing data across multiple machines is not an important consideration. When server data needs to be shared across client machines, two possibilities exist. One is to use a block protocol between a local "proxy" and the server and have the proxy rexport the data to clients via a protocol that permits sharing (e.g., NFS). A second option is to employ a file-access protocol that somehow eliminates the limitations on the meta-data path and yet permits sharing. We propose two enhancements to NFS in Section 7 to achieve such the latter goal.

7 Potential Enhancements for NFS

Our previous experiments identified three factors that affect NFS performance for meta-data-intensive applications: (i) consistency related messages (ii) synchronous meta-data update messages and (iii) non-aggregated meta-data updates.In this section, we seek to explore enhancements that eliminate these overheads and measure the impact of the enhancements.

7.1 Impact of aggressive caching on NFS v3 performance

The first enhancement for NFS examines whether aggressive caching of file system meta-data would reduce the number of consistency related messages in NFS v3 and thereby improve performance. To verify this hypothesis, we modify the default NFS client parameters so that the client checks for consistency of data and meta-data only every 60 seconds or more. Then, we run the four macro-benchmarks: PostMark, IOZone, TPC-C and TPC-H on this version of NFS v3 with aggressive caching. As shown in Table 10, the results show that there is a small reduction in number of consistency messages, particularly in the meta-data intensive PostMark benchmark where the number of messages drops by 6%. However, the performance impact of this reduction is negligible. This implies that asynchronous meta-data updates as well as meta-data update aggregation are more important in getting performance improvements.

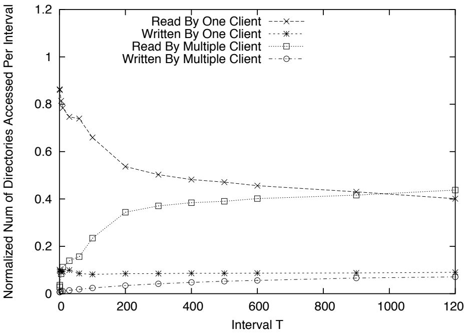
Table 10: Impact of aggressive caching in NFS v3. The table shows the ratio of performance and messages of NFS v3 with aggressive caching compared to NFS.

Benchmark	Ratio (NFS_aggressive/NFS)	
	Performance	Messages
PostMark	1.01	.94
IOZone	0.97	1
TPC-C	1.01	0.97
TPC-H	1	1

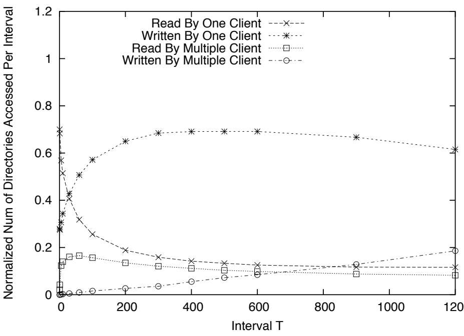
7.2 Directory Delegation

The second enhancement we consider is *directory delegation*, where a NFS client holds a lease on meta-data and can read and update the cached copy without server interaction. Since NFS v4 only supports file delegation, directory delegation would be an extension to the NFS v4 protocol specification. Observe that directory delegation allows a client to asynchronously update meta-data in an aggregated fashion. This in turn would allow NFS clients to have comparable performance with respect to iSCSI clients even for meta-data intensive benchmarks. A strongly-consistent meta-data client cache can be implemented using leases and callbacks [9, 12].

Since only one client can hold a lease on a directory at any one time, the effectiveness of this approach depends on the amount of meta-data sharing across client machines. We determine the characteristics of meta-data sharing in NFS by analyzing two real-world NFS workload traces from Harvard University [5]. We randomly choose one day (09/20/2001)



(a) EECS Trace



(b) Campus Trace

Figure 8: Sharing Characteristics of Directories

trace from the EECS traces (which represents a research, software development, and course-based workload) and the home02 trace from the Campus traces (which represents a email and web workload). Roughly 40,000 file system objects were accessed for the EECS traces and about 100,000 file system objects were visited for the Campus traces.

Figure 8 demonstrates that the read sharing of directories is much higher than write sharing in the EECS trace. In Campus trace, we find that although the read-sharing is higher at smaller time-scales, it is less than the read-write sharing at larger time-scales. However, in both the traces, a relatively small percentage of directories are both read and written by multiple clients. For example, at time-scale of 300 seconds only 4% and 3.5% percentage of directories are read-write shared in EECS and Campus traces, respectively. This suggests that contention for leases should not be significant, and it should be possible to implement strongly consistent meta-data caching with low overhead.

Strongly-consistent meta-data caching comes at the cost of network message overhead due to callback messages. Fortunately, this cost of network overhead messages is not significant for the EECS and campus traces and would not be adversely affect performance. The callback ratio, defined as ratio of callback messages and number of meta-data messages for different cache sizes, is less than 0.4% for a directory cache size of 5 (details omitted due to space constraints).

The above preliminary results indicate that implementing a strongly consistent meta-data cache is feasible and would enable a NFS v4 client with directory delegation extensions to have comparable performance with respect to an iSCSI client even for meta-data intensive benchmarks. A detailed design of directory delegation policies and their performance is beyond the scope of this paper and is the subject of future research.

8 Related Work

Numerous studies have focused on the performance of network file-access protocols[12, 13, 14, 22, 24], as well as studies on cache consistency for such protocols[20, 29]. In particular, the benefits of meta-data caching in a distributed file system for a decade old workload were evaluated in [28].

The concept of block storage over IP was initially proposed in [11, 18]. Around the same time, a parallel effort from CMU also proposed two innovative architectures for exposing block storage devices over a network for scalability and performance [6, 7]. A distributed block storage architecture was proposed and evaluated in [16].

Several studies have focused in the performance of the iSCSI protocol from the perspective of on data path overheads and latency[1, 17, 25]. The feasibility of deploying block storage as a service in WAN environments was demonstrated in [21]. With the exception of [17], which compares iSCSI to SMB, most of these efforts focus solely on iSCSI performance. Our focus is different in that we examine the suitability of block- and file-level abstractions for designing IP-networked storage. Consequently, we compare iSCSI and NFS along several dimensions such as protocol interactions, network latency and sensitivity to different application workloads.

A comparison of block- and file-access protocols was first carried out in the late eighties [23]. This study predated both NFS and iSCSI and used analytical modeling to compare the two protocols for DEC's VAX systems. Their models correctly predicted higher server CPU utilizations for file access protocols as well as the need for data and meta-data caching in the client for both protocols. Despite Moore's Law and numerous technological improvements in the ensuing decade, we find that these analytical results are still applicable today. Our experimental study complements and corroborates these analytical results for modern storage systems.

9 Concluding Remarks

In this paper, we considered two fundamentally different abstractions—files and blocks—for designing IP-networked storage. We conducted an experimental study to determine whether a file-level or a block-level abstraction if better suited for IP-networked storage. Using NFS and iSCSI as representative examples of the file- and block-access protocols, we compared their performance using a various micro- and macro-benchmarks. Our results showed that block- and file-access protocols are comparable for data-intensive workloads, while the former outperforms the latter by a factor of 2 or more for meta-data intensive workloads. We identified aggressive meta-data caching and update aggregation in iSCSI to be the primary reasons for this performance difference. We proposed two enhancements—aggressive caching and directory delegation—to improve meta-data performance of the NFS protocol. Our preliminary results for directory delegation are promising, and a detailed design of delegation policies and their performance is the subject of future research.

References

[1] S Aiken, D. Grunwald, A. Pleszkun, and J. Willeke. A Performance Analysis of the iSCSI Protocol. In *Proceedings of the 20th IEEE Symposium on Mass Storage Systems, San Diego, CA*, April 2003.

[2] G. Alvarez, E. Borowsky, S. Go, T. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: An Automated Resource Provisioning Tool for Large-scale Storage Systems. *ACM Transactions on Computer Systems*, 2002.

[3] Brent Callaghan. *NFS Illustrated*. Addison Wesley, 1999.

[4] Direct Access File System Protocol, Version 1.0. http://www.dafscollaborative.org, September 2001.

[5] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer. Passive NFS Tracing of Email and Research Workloads. In *Proceedings of USENIX FAST'03*, San Francisco, CA, March 2003.

[6] G A. Gibson et. al. File Server Scaling with Network-Attached Secure Disks. In *Proceedings of the ACM Sigmetrics'97, Seattle, WA*, pages 272–284, June 1997.

[7] G A. Gibson et. al. A Cost-Effective, High-Bandwidth Storage Architecture. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII), San Jose, CA*, pages 92–103, Oct 1998.

[8] S. Shepler et. al. Network File System Protocol (NFS) Version 4 Specification. *IETF Request for Comments, RFC 3530*, April 2003.

[9] C. Gray and D. Cheriton. Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 202–210, 1989.

[10] J. Hartman and J. Ousterhout. The Zebra Striped Network File System. *ACM Transactions on Computer Systems*, August 1995.

[11] S. Hotz, R. Van Meter, and G. Finn. Internet Protocols for Network Attached Peripherals. In *Proceedings of the Sixth IEEE/NASA Conference on Mass Storage Systems and Technologies*, 1998.

[12] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[13] C. Juszczak. Improving the Performance and Correctness of an NFS Server. In *Proceedings of the Winter USENIX Conference, San Diego, CA*, pages 53–63, 1989.

[14] C. Juszczak. Improving the Write Performance of an NFS Server. In *Proceedings of the Winter USENIX Conference, San Francisco, CA*, pages 247–259, 1994.

[15] J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[16] E. K.Lee and C. A.Thekkath. Petal Distributed Virtual Disks. In *Proceedings of the Conference on Architectural For Programming Languages and Operating Systems (ASPLOS)*, pages 84–92, 1996.

[17] Y. Lu and D. Du. Performance Study of iSCSI-Based Storage Subsystems. *IEEE Communications Magazine*, August 2003.

[18] R. Van Meter, G. Finn, and S. Hotz. VISA: Netstation's Virtual Internet SCSI Adapter. In *Proceedings of ASPLOS-VIII, San Jose, CA*, pages 71–80, 1998.

[19] T. Myklebust. Status of the Linux NFS Client. Presentation at Sun Microsystems Connectathon 2002, http://www.connectathon.org/talks02, 2002.

[20] M. Nelson, B. Welch, and J. Ousterhout. Caching in the Sprite Network File System. *ACM Transactions on Computer Systems*, 6(1), February 1988.

[21] W. Ng, B. Hillyer, E. Shriver, E. Gabber, and B. Ozden. Obtaining High Performance for Storage Outsourcing. In *Proceedings of Usenix Symposium on File and Storage Technologies (FAST), Monterrey, CA*, pages 144–158, January 2002.

[22] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz. NFS Version 3 Design and Implementation. In *Proceedings of the Summer 1994 USENIX Conference*, June 1994.

[23] K K. Ramakrishnan and J Emer. Performance Analysis of MAss Storage Service Alternatives for Distributed Systems. *IEEE Trans. on Software Engineering*, 15(2):120–134, February 1989.

[24] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and Implementation of the Sun Network Filesystem. In *Proceedings of the Summer 1985 USENIX Conference*, pages 119–130, June 1985.

[25] P Sarkar and K Voruganti. IP Storage: The Challenge Ahead. In *Proceedings of the 19th IEEE Symposium on Mass Storage Systems, College Park, MD*, April 2002.

[26] J. Satran, K. Meth, C. Mallikarjun, C. Sapuntzakis, and E. Zeidner. iSCSI Internet Draft. *IETF Work in Progress*, 2003.

[27] S. Shepler. NFS Version 4 Design Considerations. *IETF Request for Comments, RFC 2624*, June 1999.

[28] K. Shirriff and J. Ousterhout. A Trace-Driven Analysis of Name and Attribute Caching in a Distributed System. In *Proceedings of the Winter 1992 USENIX Conference*, pages 315–331, January 1992.

[29] V. Srinivasan and J. Mogul. Spritely NFS: Experiments with Cache Consistency Protocols. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 45–57, December 1989.