Performance Evaluation of Commodity iSCSI-based Storage Systems

Dimitrios Xinidis†, Michail D. Flouris and Angelos Bilas†
Institute of Computer Science (ICS)
Foundation for Research and Technology - Hellas (FORTH)
Vassilika Vouton, P.O.Box 1385,
GR 711 10 Heraklion, Greece
{dxinid, flouris, bilas}@ics.forth.gr

Abstract

iSCSI is proposed as a possible solution to building future storage systems. However, using iSCSI raises numerous questions about its implications on system performance. This lack of understanding of system I/O behavior in modern and future systems inhibits providing solutions at the architectural and system levels. Our main goals in this work are to understand the behavior of the application server (iSCSI initiator), to evaluate the overhead introduced by iSCSI compared to systems with directly-attached storage, and to provide insight about how future storage systems may be improved.

We examine these questions in the context of commodity iSCSI systems that can benefit most from using iSCSI. We use commodity PCs with several disks as storage nodes and a Gigabit Ethernet network as the storage network. On the application server side we use a broad range of benchmarks and applications to evaluate the impact of iSCSI on application and server performance. We instrument the Linux kernel to provide detailed information about I/O activity and the various overheads of kernel I/O layers.

Our analysis reveals how iSCSI affects application performance and shows that building next generation, network-based I/O architectures, requires optimizing I/O latency, reducing network and buffer cache related processing in the host CPU, and increasing the sheer network bandwidth to account for consolidation of different types of traffic.

1. Introduction

Future storage systems are required to scale to large sizes due to the amount of information that is being generated and the increasing capacities and dropping prices of magnetic disks. Network-attached, block-level storage, also called SAN-approach due to the use of storage area networks (SANs), is proposed as one method for addressing these issues. In this approach, large numbers of magnetic disks are attached to a network through custom storage controllers or general-purpose PCs and provide storage to application servers. One of the main issues in this approach is the protocol used for gaining access from application servers to remote storage over the network. Traditionally, specialized interconnection networks and protocols have been developed for this purpose. For instance, SCSI [7] and Fiber Channel [6] are among the most popular such interconnects and associated protocols.

Although these approaches have been used and are still used extensively for building storage area networks, many problems have emerged due to changes in underlying technologies. First, these interconnects use custom network components and thus are not able to take advantage of the steep technology curves and dropping costs of commodity, IP-based networks. Moreover, the fact that they require specialized equipment leads to building storage systems and data centers with multiple interconnects. This on one hand does not allow for dynamic sharing of resources since they need to be partitioned statically based on the type of interconnect servers are attached to and, on the other hand, increases significantly management overhead, since multiple interconnects need to be maintained and optimized.

Thus, there is recently a lot of interest in examining alternative solutions that would both be able to reduce the cost of the underlying equipment and management as well as better follow the technology curves of commodity networking components. One such approach is using IP-type networks and tunneling storage protocols on top of IP to leverage the installed base of equipment as well as the increased bandwidth available, especially of local area networks. It is currently projected that 10 Gigabit Ethernet interconnects will soon become commodity providing at least as much bandwidth as is (and will be available) in storage area networks.

iSCSI [13] is a storage networking standard that provides a transport layer for SCSI [7] commands over TCP/IP. The main premise of iSCSI is that it can provide a familiar API and protocol (SCSI) to the application and storage nodes utilizing low-cost, commodity IP infrastructure and taking advantage of the technology curves. Thus, it functions as a bridge between the popular SCSI storage protocol and the popular TCP/IP local area network family of protocols. With the advent of 1 and 10 Gigabit Ethernet networks, there is increased interest in using iSCSI for local access to storage in multi-tier server architectures in data centers.

Although adoption of iSCSI appears appealing, it is not clear at this point what is its impact on system performance. The introduction of both a new type of interconnect as well as a number of protocol layers in the I/O protocol stack may introduce significant overheads. Previous studies on iSCSI performance issues have revealed the adverse effect of TCP processing, however, do not provide a detailed breakdown of kernel overheads in the various components of the I/O path. This lack of understanding of how iSCSI impacts I/O path overheads inhibits solutions at the system and architectural levels.

†Also, with the Department of Computer Science, University of Crete, P.O. Box 2208, Heraklion, GR 71409, Greece.

Our work aims to examine iSCSI-related overheads in the I/O path. We investigate the iSCSI overheads associated with storage systems built out of commodity PCs and commodity Gigabit Ethernet interconnects. Although iSCSI may be used with customized systems as well, our storage nodes are commodity PCs with multiple (5-8) disks. Given today's technologies such nodes may host about 1.5-2.5 TBytes of storage and in excess of 10 TBytes in the near future. Storage nodes are connected with application servers and application clients through a Gigabit Ethernet network. In our evaluation we use both microbenchmarks as well as real applications to examine the impact of iSCSI-based storage compared to directly-attached disks. We instrument the Linux kernel and in particular the block-level I/O paths to gain detailed information about I/O activity and assess both application- and system-level statistics.

We find that the significant overheads introduced by iSCSI has different effect on different applications. Postmark is sensitive to increased latencies and is not affected by the reduction in I/O throughput or the increase in CPU cycles. TPC-H is sensitive to I/O throughput and CPU cycles. Finally, Spec-SFS is hurt by reduced network bandwidth (I/O throughput) and the fact that the same network is used to carry both client-server NFS traffic as well as iSCSI I/O traffic. On the other hand, iSCSI allows us to easily scale the amount of storage attached to a server and thus, recover most of the performance loss. Increasing the number of disks in TPC-H is able to outperform the base, direct configuration. Similarly, increasing the I/O cache in iSCSI has similar results for Postmark performance. Finally, our examination of kernel I/O path overheads shows that applications can benefit significantly from offloading not only TCP processing and interrupt cost as previous work has shown, but system buffer cache processing as well.

The rest of this paper is organized as follows. In Section 2 we present our experimental platform and in Section 3 we discuss our methodology. Section 4 presents our results and analysis of system behavior. Finally, Section 5 presents related work and Section 6 draws our conclusions.

2. Platform

In this section we present our testbed and we introduce the workloads we use in our work.

2.1. Experimental testbed

Our iSCSI testbed consists of 16 dual-processor (SMP) commodity x86 systems. Each system is equipped with two Athlon MP2200 processors at 1.8GHz, 512 MBytes of RAM and a Gigabyte GA-7DPXDW motherboard with the AMD-760MPX chipset. The nodes are connected both with a 100MBit/s (Intel 82557/8/9 adapter) and a 1GBit/s (D-Link DGE550T adapter) Ethernet network. All nodes are connected on a single 24-port switch (D-Link DGS-1024) with a 48 GBit/s backplane. The 100MBit network is used only for management purposes. All traffic related to our storage experiments uses the GBit Ethernet network.

The AMD-760MPX chipset supports two PCI 64-bit/66 MHz bus slots, three PCI 32-bit/33 MHz slots and two on board IDE controllers, a system IDE controller with two ATA-100 channels for up to four devices and an IDE Promise PDC20276 RAID controller with two ATA-100 channels for up to four devices. It is important to note that the IDE controllers are connected through a *single PCI 32-bit/33 MHz link* to the memory, which limits the aggregate IDE bandwidth to about 120 MBytes/sec. Each node has an 80-GByte system disk (Western Digital WD800BB-00CAA1, 2MB cache) connected to the system IDE controller. Three of the system nodes are equipped with five additional disks of the same model. Three of the disks are connected to the system IDE controller and the other two to the Promise IDE RAID controller. All disks (except the system disk) are configured in RAID-0 mode using the Linux MD driver (software RAID). The hardware RAID functionality of the Promise controller is not used. Since we are interested in examining commodity platforms, we use IDE/ATA disks. However, in the case of iSCSI we need to use the SCSI I/O hierarchy in the Linux kernel to generate SCSI commands for the iSCSI modules.

The operating system we use is Linux RedHat 9.0, with a kernel version 2.4.23-pre5 [2]. Not all Linux kernel versions provide support for fast disk I/O. In versions 2.4.19 to 2.4.22 a bug in the kernel driver for the AMD IDE chipset disables support for fast data transfers and results in low raw disk throughput. According to our measurements, versions up to 2.4.18 or later than version 2.4.22 offer the expected disk I/O performance. In our work we use version 2.4.23-pre5 for all the experiments. Furthermore, we use the Linux `hdparm` utility to set each disk to 32-bit I/O and to UDMA100 mode. These result in an increase of maximum disk throughput from 25 to 45 MBytes/s.

2.2. iSCSI implementation

During the course of this project, we have experimented with various Linux iSCSI implementations [19, 3, 8]. [8] suffers from low performance, as we verified through several microbenchmark experiments. On the other hand, [19] requires SCSI disks on the target side. This makes it unsuitable, since our goal is to build commodity storage nodes based on inexpensive IDE disk technology.

The Intel iSCSI implementation [3] we choose for our work has the fewest limitations. Mainly, that it supports only non-SMP kernels, so our kernel is built with no SMP support (i.e. only one processor is used). Secondly, we found that the Intel iSCSI target was originally developed for block devices up to 4 GBytes (32-bit byte addressing). Since we would like to build storage nodes with significantly higher capacity (currently 5x80 GBytes/node) and to experiment with datasets that exceed 2 GBytes, we have modified the iSCSI target to support 32-bit *block* addressing, which is adequate for our purposes[1]. In the Intel implementation, the iSCSI target runs at user level, whereas the iSCSI initiator runs in the kernel.

2.3. Workload

To examine system behavior we use a set of microbenchmarks and applications: IOmeter [14], a workload generator that has been used extensively for basic evaluation of I/O subsystems, Postmark [15], a benchmark that emulates the I/O behavior of an e-Mail server, the TPC-H [22] decision support workload on MySQL [23], a popular open-source database, and Spec-SFS [20], a widely-accepted NFS server benchmark. Next we examine each benchmark in more detail.

[1] The main issue is that the iSCSI target, which is implemented in user space, opens the target devices with an `lseek` call that repositions the file offset. The specific call used, allows only 32-bit offsets and needs to be replaced with the `_llseek` call that handles 64-bit offsets.

IOmeter: IOmeter [14] is a configurable workload generator. The parameters we vary are, access pattern, mix of read and write operations, number of outstanding requests, and block size. We choose four workloads that represent extreme access patterns, all sequential or all random, 100% reads or writes, and two mixed workloads with 70-30% reads and writes. Finally, for each workload we vary the number of outstanding requests between 1 and 16 and the block size between 512 Bytes and 128 KBytes. The results we report are with 16 outstanding I/O requests, unless stated otherwise. In our discussion we use the average throughput, the average response time for each I/O request, and the total CPU utilization.

PostMark: PostMark [15] simulates the behavior of an Internet mail server. PostMark creates a large number of small files (message folders) that are constantly updated (as new mail messages arrive). PostMark can be configured in two ways [15]; The number of files to create and the total number of transactions to perform on this set of files. In our experiments we use inputs 50K/50K, 50K/100K, 100K/100K, and 100K/200K. A new filesystem is created with "mkfs" before each experiment to ensure that filesystem state does not affect the results.

MySQL: MySQL is a popular open-source database [23]. To examine the behavior of this important class of applications we use the TPC-H workload [22]. The TPC-H benchmark models a decision support system and consists of a suite of 22 business-oriented ad-hoc queries and concurrent data modifications. In our work we use TPC-H queries 1-3,5-8,11,12,14 and 19. Since we need to perform multiple runs for each query, we omit the rest of the queries that take a long time to complete (in the order of hours). A new filesystem is also created with "mkfs" and the database is reloaded before each TPC-H experiment (each experiment includes all the queries we use).

Spec-SFS: Spec SFS97_R1 V3.0 [20] measures the throughput and response time of an NFS server. The first iteration starts with a total NFS server load of 500 operations/sec, while each consecutive iteration increases the load by 100 operations/sec until the server is saturated. In our work we use total response time and response rate for NFS operations.

3. Methodology

In our work we are interested in examining both application-specific as well as system-wide metrics. In particular, we are interested in presenting detailed breakdowns of system I/O activity in each kernel layer.

3.1. Kernel Instrumentation

To obtain the breakdown of kernel time in various layers we instrument the Linux kernel source with our own stop-watch timers. More details about our kernel-level instrumentation can be found in [24]. Our timer subsystem uses the processor cycle counters to get accurate and fast timing measurements at clock-cycle granularity. The instrumented kernel source contains several stop-watch timers that measure processor cycles between specific points in the kernel source.

In our instrumentation we have placed timers in the borders of consecutive layers of the I/O path. Thus, starting from the high-level `read()` and `write()` system calls we are able to measure times spent in each layer as an I/O call is forwarded from the user

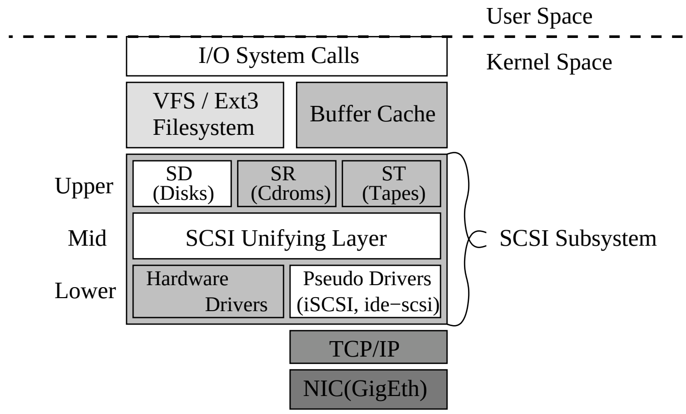


Figure 1. Kernel Layer Hierarchy.

process to the actual disk device driver (or network device driver in the case of an iSCSI device).

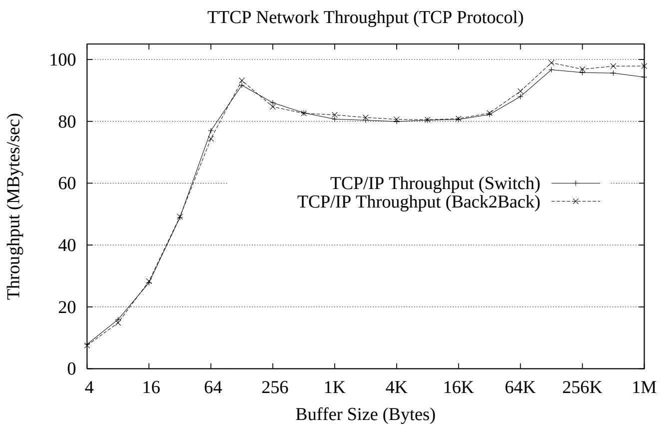
Figure 1 illustrates the kernel layers that are of interest for the iSCSI call path. The "system call" layer is the interface to the user-space world. The I/O system calls use VFS (or generic filesystem) calls to perform their tasks, while individual filesystems, such as ext3, plug their code into the VFS handlers. The buffer cache code is used for managing memory pages that cache file inodes or disk blocks. The buffer cache functions are used both by VFS and the lower-level filesystem code sometimes in a recursive manner. Thus, it is generally hard to distinguish system time spent in VFS, Ext3 and buffer cache code since their calls are interdependent. However, in our analysis we do not aim at a detailed analysis of the filesystem time, but instead on understanding iSCSI-related overheads. For this reason we measure two separate times for the VFS/Ext3 layer: (a) the time for read/write calls, labeled as *"FS: read/write"* in our breakdown graphs and (b) the time for performing file and directory operations (e.g. create, delete, locate), labeled as *"FS: File Mgmt"*. These two components are possible to distinguish because they initiate through different system calls.

Below the filesystem level, is the SCSI hierarchy that consists of three layers: upper, middle, and lower SCSI layers. We combine the time spent by the upper and middle layers in a single component, labeled as *"SCSI"*. The low-level SCSI driver is the iSCSI module, which we time separately. We also quantify what happens below the iSCSI layer, namely in the TCP/IP layer and the network device driver and the associated interrupt handler (the DL2k module in our systems). Our measurements of the TCP/IP stack, labeled *"TCP"*, contain all the time spent in both the send and receive paths only accounting for the iSCSI traffic. Finally, we measure the time spent in the network device interrupt handler separately (labeled *"NIC IRQ"*).

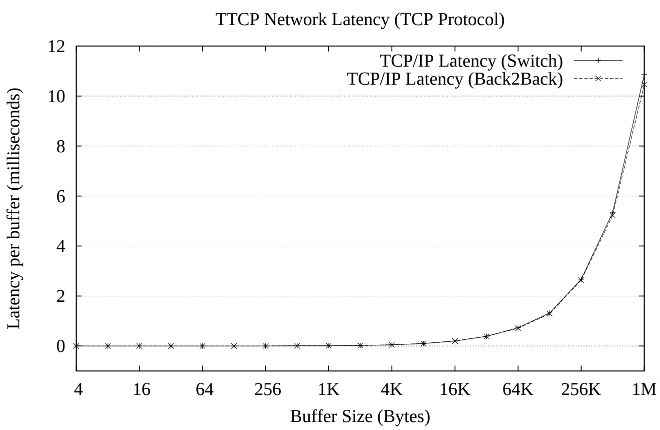
It is important to note where in the kernel layers we have memory copies of data related to iSCSI traffic. There are two such cases: one in the "filesystem read/write" component, where data is copied between the kernel's buffer cache and the application buffer, and secondly in the "TCP" layer, where data is copied between the NIC's buffer and kernel memory. These two copies occur for all data stored on iSCSI storage. In the directly-attached disk case, the NIC copy does not occur and is replaced with a DMA to the hard disk buffer.

3.2. System Configurations

For our evaluation we have examined three system configurations.



(a) TCP Throughput.



(b) TCP Latency.

Figure 2. TCP/IP throughput and latency over D-Link DGE550T Gigabit Ethernet NIC, measured with ttcp.

(A) *Direct-attached (or Local) Disk.* Five disks directly attached to the application server.

(B) *iSCSIx1.* One storage node with five disks connected to the application server through Gigabit Ethernet. The storage node (iSCSI target) exports a single RAID-0 volume through iSCSI.

(C) *iSCSIx3.* Three storage nodes (iSCSI targets) with five disks each connected to the application server through Gigabit Ethernet. Each storage node exports a single RAID-0 volume through iSCSI. The three iSCSI volumes are concatenated with software RAID-0 on the application server.

4. Results and Analysis

Next we present our experimental results and analysis.

4.1. Basic measurements

Figure 2 shows the throughput and latency of the Gigabit Ethernet network we use, measured with `ttcp`. The system achieves a maximum bandwidth of about 800 MBit/s at 128K packet sizes. Our system configuration does not use jumbo frames since many commodity network interfaces and switches do not support this feature.

Figure 3 shows the basic throughput for the disks and controllers we use, measured with IOmeter. We see that each disk is capable of 45 MBytes/s throughput for sequential read accesses and about 20 MBytes/s for sequential write accesses. A mix of 70% read and 30% write operations achieves also about 20 MBytes/s throughput. Each IDE controller can transfer about 120 MBytes/s. As mentioned in Section 2, the PCI bus in our systems is a 33MHz/32Bit bus, resulting in a theoretical peak of 125 MBytes/s. Given that we use two IDE controllers in each system we expect that the maximum I/O throughput in each node is limited by the PCI bus. CPU utilization and response time for a single disk are depicted in Figure 3(a).

Finally, Figure 3 show throughput, CPU utilization and response time for iSCSI when using a RAM-disk on the iSCSI target with one outstanding I/O request. We see that the initiator reaches 90% CPU utilization for reads and about 80% for writes for 1K or larger I/O request sizes. Thus, when using iSCSI I/O throughput is limited by network bandwidth to about 50 MBytes/s (with 16 KByte requests) at a system utilization between 80-90%. Minimum response time (512 Byte requests) is less than a 100 μs.

4.2. Microbenchmarks

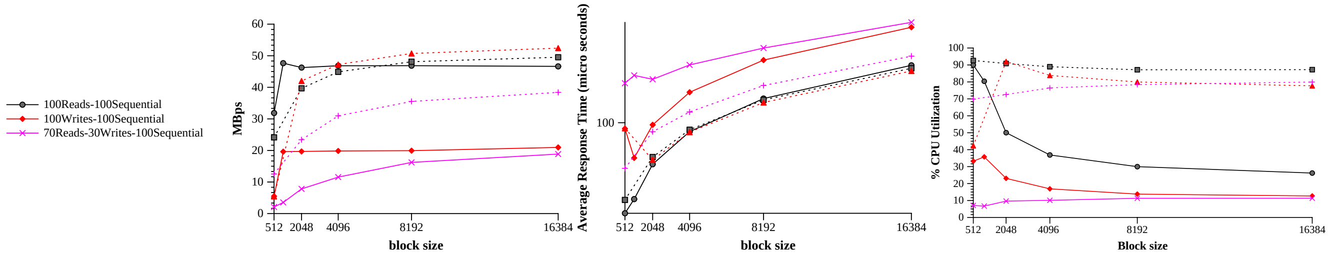
First, we look at IOmeter to understand basic aspects of the local and iSCSI configurations. Figure 4 shows that for directly-attached disks and sequential read requests, maximum throughput approaches 120 MBytes/s, limited by the PCI bus in our systems. For write requests, maximum throughput is about 50 MBytes/s. The read performance is higher than write due to the aggressive prefetching (read-ahead) the software RAID driver performs. In the iSCSIx1 configuration, maximum throughput is limited to 40 MBytes/s and maximum read throughput is limited to about 25 MBytes/s. The writes in the iSCSI configuration are faster because of the write-back cache on the iSCSI target nodes. Reads, however, are slower because data must be read from the remote disk. In both cases, maximum throughput is achieved at 4 or 8KByte I/O requests. When using 70-30 read-write mix, throughput is about 50 MBytes/s for the local case and about 20 MBytes/s for the iSCSIx1 configuration. iSCSIx3 achieves about the same write throughput as iSCSI, but does much better in reads. This is due to the larger buffer cache that three nodes have compared to one node in the iSCSIx1 case. Random access patterns exhibit a significantly lower throughput in all cases.

In terms of response time (Figure 5), we see that the local and iSCSIx1 configurations have similar response times. In contrast, the iSCSIx3 setup shows higher latencies for larger block size, mainly due to network congestion from the three iSCSI connections. Finally, Figure 6 shows the CPU utilization in the initiator (application server). Maximum CPU utilization with sequential requests is between 75-90% in the local case and between 60-70% in the iSCSIx1 case depending on I/O request size. We note that, given the achievable throughput, iSCSIx1 utilization is higher compared to the local configuration. iSCSIx3 shows very high CPU usage, especially in the case of sequential reads. This is due to increased network throughput, which results in high network processing times (including memory copies of data from the NIC).

4.3. Application Performance

4.3.1. PostMark

Overall, we find that Postmark is sensitive to I/O latency and iSCSI reduces performance up to 20%. For the same reason, however, in iSCSIx3, performance improves up to 80% due to

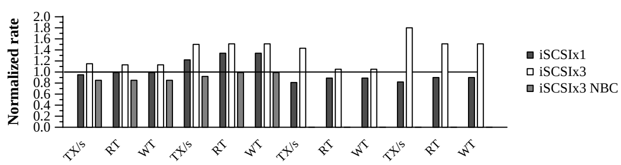


(a) Throughput (b) Average Response Time (c) Cpu Utilization

Figure 3. IOmeter statistics for a single disk vs. an iSCSI RAM-disk. Dotted lines denote the single disk and solid the iSCSI RAM-disk

Table 1. Postmark results. Throughput is in KBytes/sec.

Input Size (#files/#trans)	Tx/sec	Read Throughput	Write Throughput
50K/50K	204	272.54	855.81
50K/100K	165	302.93	627.81
100K/100K	81	126.52	394.30
100K/200K	74	150.61	312.88



(a)

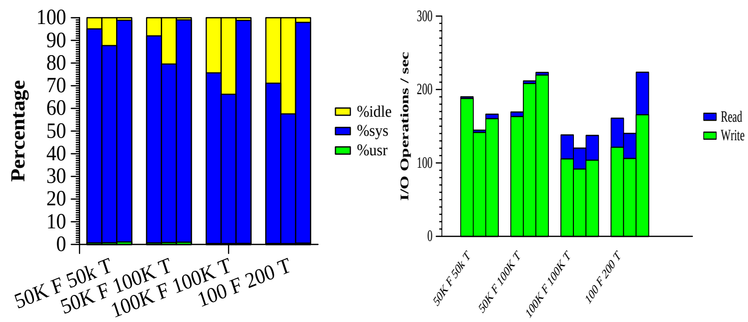
Figure 7. Postmark results normalized to the direct configuration. Each graph represents one input size. Each group of bars refers to one application metric: Transactions/s, Read Throughput, and Write Throughput. Each bar refers to one system configuration: direct, iSCSIx1, iSCSIx3 and iSCSIx3 without buffer cache (left to right).

the increased target buffer cache that reduces I/O latency. Also, iSCSIx3 is able to saturate the host CPU, something that is not possible in the direct and iSCSI configurations. Next we present a more detailed analysis.

Table 1 shows the performance of PostMark in each of the three system configurations. We notice that using iSCSI reduces transaction rate between 0% and 20% compared to the direct configuration. In iSCSIx3 all application metrics improve compared to direct by 5% to 80%. Moreover, we note that the largest improvement occurs in the configurations that use a larger number of transactions for a given number of files. This is due to the fact that when increasing the number of transactions and keeping the number of files constant the larger iSCSIx3 target buffer cache becomes more effective.

Figure 8 shows the execution time breakdown for PostMark. We see that system time drops by 10-15% in iSCSIx1, compared to the direct case. This is due to the increased response time in iSCSIx1. PostMark performs synchronous I/O and thus, is sensitive to I/O response time. In iSCSIx3, response time improves due to the larger target cache, which results in higher system utilization that reaches almost 100% for all input sizes. To verify this we disable the buffer cache in the Linux kernel in all iSCSI targets.

Figure 7 shows that PostMark performance drops dramatically, even below the direct configuration (Table 1). Thus, PostMark benefits mostly from the presence of the increased I/O subsystem cache in the iSCSI and iSCSIx3 configurations, demonstrating one of the advantages of using iSCSI.



(a) (b)

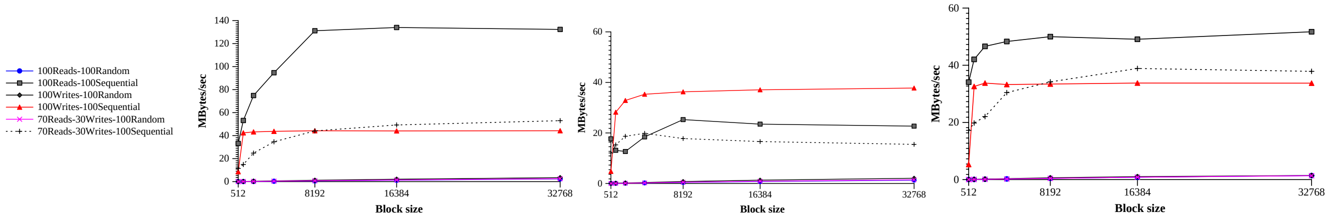
Figure 8. Postmark execution time breakdown and I/O rate. The left bar in each pair refers to the direct configuration and the middle bar to the iSCSIx1 configuration and the right to the iSCSIx3.

Figure 8 shows the I/O rate at the initiator (application server) for each system configuration and input size. This rate shows the number of requests that reach the physical disks in the direct configuration and the iSCSI layer in iSCSIx1 and iSCSIx3. The I/O rate in PostMark follows the same pattern as the transaction rate reported by the application.

4.3.2. MySQL

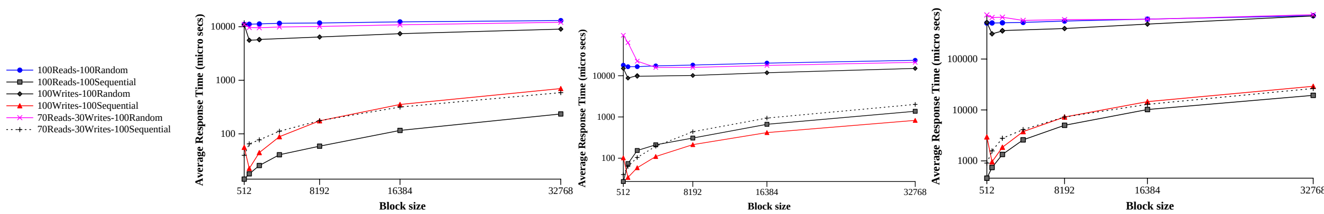
Overall, TCP-H is sensitive to disk I/O throughput and CPU utilization. Thus, iSCSI reduces performance by up to 95%. However, iSCSIx3 is able to scale the number of disks (and the total size of I/O cache). This improves disk I/O throughput and reduces the gap with the direct configuration, in some cases even improving performance up to 25% (Q14). Next we discuss our results in more detail.

Table 2 and Figure 9 show the execution time for the TPC-H queries we use. We see that using iSCSI increases execution time between 1% (Q8) and 95% (Q6). In the iSCSIx3 configuration, execution time reduces significantly compared to the iSCSIx1 case and is within 40% of the direct case (and usually within 20%). Similarly to PostMark, iSCSIx3 is able to recover the performance degradation of using iSCSI with a significant increase in resources.



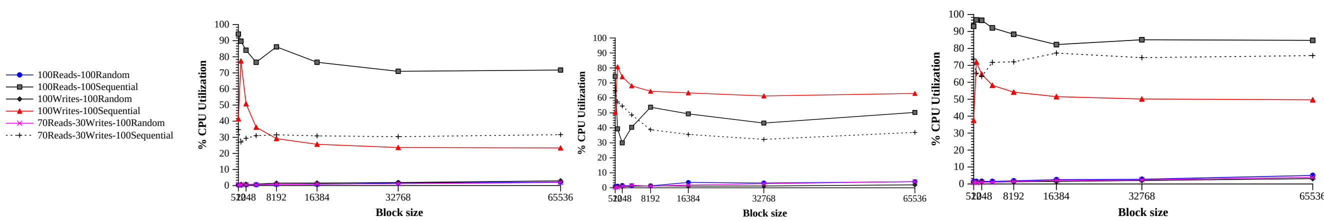
(a) Directly-attached disks (b) iSCSIx1 (c) iSCSIx3

Figure 4. IOmeter throughput.



(a) Directly-attached disks (b) iSCSIx1 (c) iSCSIx3

Figure 5. IOmeter average I/O response time.



(a) Directly-attached disks (b) iSCSIx1 (c) iSCSIx3

Figure 6. IOmeter CPU utilization.

Table 2. TPC-H query execution time in MySQL (in seconds).

Query	q1	q3	q5	q6	q7
Exec time (s)	77.34	54.91	60.73	10.91	56.89
Query	q8	q11	q12	q14	q19
Exec time (s)	147.97	26.92	15.75	24.30	17.18

To distinguish whether the performance improvement with iSCSIx3 is due to the increased target cache or the larger number of disks we also run experiments with the iSCSIx3 configuration where the target buffer is disabled (Figure 9). We see that, except for query Q14, in all cases the performance degradation is fairly small (within 7%) which suggests that TPC-H benefits mostly from the increased number of disks under iSCSIx3.

Figure 10 shows the execution time breakdown for each query. First, we note that between direct and iSCSIx1 system time increases by up to 200% (Q14) which indicates that iSCSI introduces a significant overhead. Moreover, user time reduces in all queries and by up to 50% (Q6) indicating that either system time takes up useful cycles from the host CPU or that the increase in iSCSI response time results in lower CPU utilization.

Similarly to iSCSIx1, iSCSIx3 exhibits a higher user time compared to the direct configuration for all queries where it performs better. Idle time in iSCSIx3 is almost 0% in all queries.

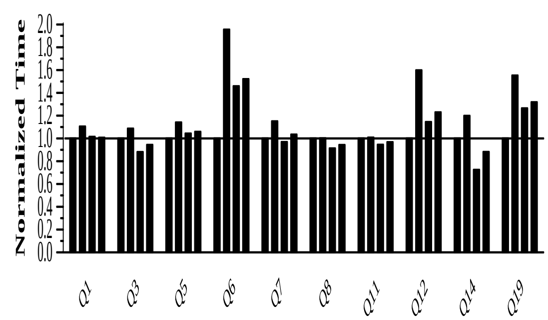
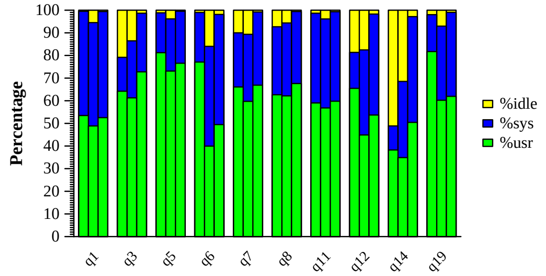


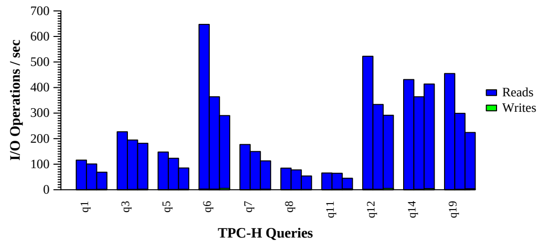
Figure 9. TPC-H execution time normalized to the direct configuration. Each group of bars refers to one query; Each bar refers to one system configuration: direct, iSCSIx1, and iSCSIx3, and iSCSIx3 without target cache (left to right).

Thus, application server CPU is saturated and further improvements in application performance may only be achieved with lowering CPU utilization.

Finally, Figure 10 shows the I/O rate for each configuration. We see that the large differences in user time, especially in queries Q6, Q12, and Q19 are reflected to differences in the I/O rate.



(a) Execution Time Breakdown



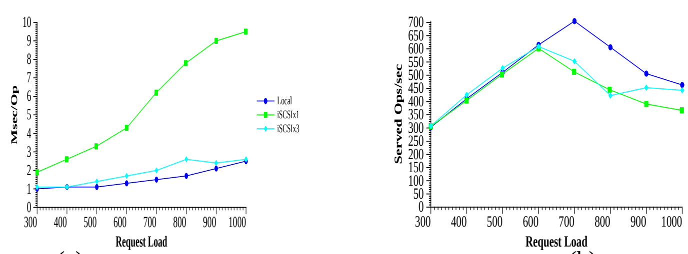
(b) I/O rate (IOs/s)

Figure 10. TPC-H I/O rate and execution time breakdown. The left bar in each pair refers to the direct configuration, the middle bar to the iSCSIx1 configuration and the right to iSCSIx3.

4.3.3. Spec-SFS

Overall, we find that Spec-SFS is hurt by mixing client-server and iSCSI traffic over the same network. Although consolidating all traffic on top of a single network is considered an advantage of iSCSI, this has an adverse effect on Spec-SFS performance.

Figure 11 shows that the iSCSIx1 configuration saturates faster than the direct configuration at 600 I/O requests/s, as opposed to 700 I/O requests/s (14% difference). Moreover, I/O response time is larger in the iSCSIx1 configuration for all I/O request loads by about 25% to 40%. However, the I/O rate is similar in both the direct and iSCSI cases for the the loads that do not saturate iSCSI (up to 600 IOs/s). The iSCSIx3 configuration behaves similarly to iSCSIx1, however, I/O response time improves and is within 25% of the direct case in most cases.



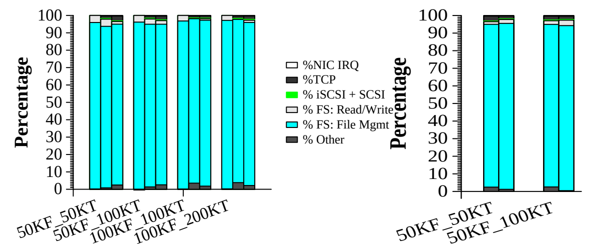
(a) (b)

Figure 11. Spec-SFS results.

The execution time breakdown for Spec-SFS (Figure 11) shows that the Spec-SFS server is idle most of the time, which suggests that the network bandwidth of the Spec-SFS server is limiting system performance. Note that all systems in this experiment are attached to the same Gigabit Ethernet switch, meaning that both Spec-SFS and iSCSI traffic traverse on the same link that connects the Spec-SFS server (iSCSI initiator) to the switch. For this reason, in the iSCSIx1 and iSCSIx3 configurations the system saturates at a lower number of I/O requests, since using iSCSI increases the traffic on the network.

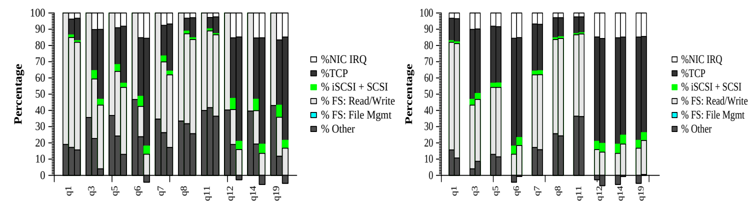
4.4. System Overhead Breakdown

Next we examine the overhead introduced by iSCSI in the I/O protocol stack.



(a) (b)

Figure 12. Postmark system time breakdowns with and without (NBC) buffer cache in the storage targets. Each group of bars refers to one system configuration: direct (left), iSCSIx1 (middle), and iSCSIx3 (right).



(a) (b)

Figure 13. TPC-H system time breakdowns with and without (NBC) buffer cache in the storage targets. Each group of bars refers to one system configuration: direct (left), iSCSIx1 (middle), and iSCSIx3 (right).

Figure 12(a) and 13(a) shows the breakdown of system time in the major components of the I/O stack (Figure 1)[2]. We see that in PostMark most (90%-95%) of the system time is spent in the filesystem component. This is due to the fact that PostMark represents mail folders as directories and writes mail messages to separate files. Thus, each mail operation results in (multiple) file and directory operations (`open, close, search, delete, create`) that account for most of the system overhead.

In TPC-H queries can be divided in two categories: (i) Queries where overheads related to the block-level I/O stack are between 30%-80%. In this category belong Q3, Q5, Q6, Q7, Q12, Q14, and Q19. (ii) Queries where block I/O-related overheads are low (Q1, Q8, Q11). In all cases, the file management overhead is minimal, because during query execution the only file operations that take place are read and write operations that are passed directly to the block I/O hierarchy. On the other hand, buffer cache management is in many cases a significant component of system time (up to 80%).

In terms of the block I/O stack, the overhead is divided almost equally between block I/O and network components. The ma-

[2]The negative components in some bars are due to errors in our measurements, which, however, do not affect our results.

jor components of the block I/O stack involved in the I/O path is the buffer cache management (between 15% and 70% of system time). The major network components are the send and receive paths of the TCP/IP stack (between 10% and 65% of system time) and the NIC interrupt handler (between 5% and 20% of system time).

Thus, we see that a significant part of the I/O overhead for TCP-H is in the block-level I/O hierarchy and that using iSCSI has a significant impact due to TCP and NIC interrupt handler overheads. Also, when comparing iSCSIx3 to iSCSIx1 we see that block-level I/O overheads increase significantly and up to 20%.

When disabling the buffer cache in all targets in the iSCSIx3 configuration (Figure 13(b)) trends remain the same as before, since TPC-H is not affected significantly by the available target cache size.

Finally, Spec-SFS exhibits very little system time in the initiator (Spec-SFS server) and thus, we do not examine the related breakdowns.

Overall, we see that the most significant kernel overheads in the I/O path are not only TCP and interrupt processing, as previous work has shown, but buffer cache processing as well. This suggests that novel I/O architectures should not only consider TCP-related costs, but buffer cache processing as well.

5. Related Work

Recently and due to the increasing importance of scaling storage subsystems and reducing system costs, there have been a number of efforts to build efficient iSCSI implementations and also to evaluate various aspects of iSCSI.

A number of iSCSI implementations are currently available [3, 8, 19, 12, 1, 9, 4, 25]. As mentioned in our work we use [3] after examining most of the publicly available systems.

The authors in [18] evaluate the performance of iSCSI when used for storage outsourcing. Similarly to our work, the authors use both microbenchmarks and real applications (TPC-C and Postmark). However, unlike our work they focus on network issues. They examine the impact of latency on application performance and how caching can be used to hide network latencies. The authors in [5] examine the performance of iSCSI in three setups: An optimized, commercial-grade back-end in a SAN environment, a commodity-type back-end in SAN environment, and a commodity-type back-end in a WAN environment. They perform high-level experiments and examine system throughput with microbenchmarks. The authors in [16] use simulation to examine the impact of iSCSI and network parameters on iSCSI performance. They only examine throughput of a simple test and the consider iSCSI PDU size and network Maximum Segment Size, TCP Window Size, and Link Delay. The authors in [10] present the design and implementation of iSCSI for Linux and perform preliminary evaluation of their system with a simple microbenchmark. The authors in [11] examine the impact of TCP Window Size and iSCSI request size for LAN, MAN, and WAN environments. The authors find that the default TCP parameters are inappropriate for high-speed MAN and WAN environments and that tuning of these layers is required.

In contrast, our work focuses on the impact of iSCSI on application server performance and we examine in detail, by instrumenting the Linux kernel, system (kernel) overheads introduced by iSCSI. We also examine how adding system resources in an iSCSI configuration impacts application and server performance.

The authors in [17] also examine the impact of iSCSI on application server performance. They use a simple microbenchmark directly on top of block-level storage or through NFS. Similarly to our results they find that iSCSI impacts system behavior significantly and that Ethernet interrupt cost is the most significant source of overhead. Our results show similar behavior for the network-related costs. However, we find that other, iSCSI-related costs can also be very high when using real applications. The authors in [17] conclude that using jumbo frames reduces interrupt overhead by about 50% but state that this may not be a practical option in real systems where not all components in the network path may support jumbo frames. In our work, we do not consider Jumbo frames, since we also feel that this may not be representative of practical setups.

The authors in [21] discuss an implementation of iSCSI that avoids a copy on commodity network adapters. They use simple microbenchmarks to examine the performance of their implementation and find that it reduces CPU utilization from 39.4% to 30.8% for reads and that it does not have a significant impact for reads.

6. Conclusions

Storage systems are currently undergoing major transformations, mainly due to the increasing disk capacities and network bandwidth as well as due to dropping component prices. Various forms of networked storage are proposed as candidates for future storage systems. One issue in this direction is the storage protocol that will be used for accessing remote storage from application servers. iSCSI emerges as one of the key protocols. Although it has numerous advantages, such as the usage of commodity IP-based infrastructure and thus, reduced cost and management effort, there are numerous questions associated with the impact of iSCSI on system performance.

In this paper we evaluate the performance of commodity iSCSI storage systems and compare it with directly attached storage. We use a public domain iSCSI implementation [3] and a set of real-life applications to examine the impact of iSCSI on end-application performance.

In summary, we see that using iSCSI without increasing system resources compared to a local configuration has a significant impact in all applications we examine. However, the impact of iSCSI differs in each case. Postmark is sensitive to increased I/O latency, TPC-H is affected by reduced I/O throughput and increased CPU cycles, and Spec-SFS by the sharing of one network between both client-server as well as iSCSI I/O traffic.

iSCSIx3 is able to scale system resources and recover and in some cases improve system performance. Postmark benefits from the increased target buffer cache, TPC-H by the increased number of disks (and to a lesser extend by the increased buffer cache), whereas Spec-SFS remains limited by the available network bandwidth in the unified interconnect. Finally, our examination of kernel-level overheads shows that improving I/O path performance requires dealing not only with TCP and interrupt processing costs, but buffer cache management as well.

Thus, building next generation, network-based I/O architectures, requires optimizing I/O latency, reducing network and buffer cache related processing in the host CPU, and increasing the network bandwidth to account for consolidation of different types of traffic. Overall, our work provides valuable insight on the impact of iSCSI on application server behavior and shows that there is significant room for improvements in future storage subsystems.

7. Acknowledgments

We would like to thank the members of the CARV laboratory at ICS-FORTH for useful discussions. Also, we thankfully acknowledge the support of the European FP6-IST program through the SIVSS project.

REFERENCES

[1] Linux-iSCSI project. http://linux-iscsi.sourceforge.net.
[2] Linux kernel sources version 2.4.23-pre5. http://www.kernel.org.
[3] Project: Intel iSCSI reference implementation. http://sourceforge.net/projects/intel-iscsi.
[4] Project: iSCSI enterprise target. http://sourceforge.net/projects/iscsitarget.
[5] S. Aiken, D. Grunwald, and J. W. Andrew R. Pleszkun. A performance analysis of the iSCSI protocol. Apr. 2003.
[6] ANSI. Fibre Channel Protocol (FCP), X3.269:1996. In *11 West 42nd Street, 13th Floor, New York, NY 10036*.
[7] ANSI. SCSI-3 Architecture Model (SAM), X3.270:1996. In *11 West 42nd Street, 13th Floor, New York, NY 10036*.
[8] M. F. Brown, J. Hawkins, M. Ostman, and W. Moloney. UMass Lowell iSCSI Project. http://www.cs.uml.edu/ mbrown/iSCSI.
[9] A. T. BV. Linux iSCSI target implementation. http://www.ardistech.com/iscsi.
[10] A. Chadda, A. Palekar, R. Russell, and N. Ganapathy. Design, implementation, and performance analysis of the iSCSI protocol for SCSI over TCP/IP. In *Internetworking 2003 International Conference*, June 2003.
[11] I. Dalgic, K. Ozdemir, R. Velpuri, and U. Kukreja. Comparative performance evaluation of iSCSI protocol over metro, local, and wide area networks. Apr. 2004.
[12] IBM. iSCSI initiator project. http://www-124.ibm.com/developerworks/projects/naslib.
[13] I. E. T. F. (IETF). iSCSI, version 08. In *IP Storage (IPS), Internet Draft, Document: draft-ietf-ips-iscsi-08.txt*, Sept. 2001.
[14] Intel Server Architecture Lab. Iometer: The I/O Performance Analysis Tool for Servers. http://developer.intel.com/design/servers/devtools/iometer/.
[15] J. Katcher. PostMark: A New File System Benchmark. http://www.netapp.com/ tech_library/3022.html.
[16] Y. Lu, Farrukh, Noman, and D. H. Du. Simulation study of iSCSI-based storage system. Apr. 2004.
[17] Mike Brim and George Kola. An analysis of iSCSI for use in distributed file system design. http://www.cs.wisc.edu/m̃jbrim/uw/740/paper.pdf.
[18] W. T. Ng, H. Sun, B. Hillyer, E. Shriver, E. Gabber, and B. Ozden. Obtaining high performance for storage outsourcing. pages 145–158, Jan. 2002.
[19] A. A. Palekar and R. D. Russell. Design and implementation of a SCSI target for storage area networks. Technical Report TR 01-01, University of New Hampshire, May 2001.
[20] S. P. E. C. (SPEC). SFS 3.0. http://www.spec.org/sfs97r1/docs/sfs-3.0.pdf, 2001.
[21] F. Tomonori and O. Masanori. Performance optimized software implementation of iSCSI. In *SNAPPI03*, Sept. 2004.
[22] T. P. P. C. (TPC). *TPC BENCHMARK H, Standard Specification, Revision 2.1.0*. 777 N. First Street, Suite 600, San Jose, CA 95112-6311, USA, Aug. 2003.
[23] M. Widenius and D. Axmark. *MySQL Reference Manual*. O'Reilly & Associates, Inc., June 2002.
[24] D. Xinidis, M. D. Flouris, and A. Bilas. Virtual timers: Using hardware physical timers for profiling kernel code-paths. Feb. 2005.
[25] H. Xiong, R. Kanagavelu, Y. Zhu, and K. L. Yong. An iSCSI design and implementation. Apr. 2004.