# Analysis of iSCSI Target Software

FUJITA Tomonori and OGAWARA Masanori
*NTT Cyber Solutions Laboratories*
*tomof@acm.org, ogawara.masanori@lab.ntt.co.jp*

## Abstract

We analyzed the design and performance of iSCSI storage systems, built into general purpose operating systems. Our experiments revealed that a storage system that uses specialized functions, in conjunction with the modified operating system, outperforms a storage system that only uses the standard functions provided by the operating system. However, our results also show that careful design enables the latter approach to provide a comparable performance to that of the former, in common workloads.

## 1 Introduction

Many companies have started to consider the iSCSI protocol [1] used to build inexpensive Storage Area Network (SAN) environments by using Ethernet networks. The iSCSI protocol encapsulates the SCSI protocol into the TCP/IP protocol, and carries packets over IP networks.

Several studies have conducted the performance evaluation of the iSCSI protocol. However, all of them evaluated the performance a host computer, called an initiator, issuing SCSI commands. The performance of storage system, known as a target, providing services to initiators have not been investigated.

We evaluated the design of iSCSI storage systems using two open-source software packages to build an iSCSI storage system on a general-purpose operating system.

This paper focus on storage systems build by using commodity hardware and an open-source general-purpose operating system. Commercial iSCSI storage systems adopt specialized network adapters that directly support the iSCSI protocol and optimized operating systems to deliver high performance. The latter approach has a clear advantage from the perspective of performance. But the former approach can cut the costs and benefit from the rapid progress of commodity hardware and open-source software.

The iSCSI target software packages that we invesitigated have different design philosophies: one uses standard kernel interfaces to provide rich functionality; the other implements and uses functions specialized for iSCSI storage systems by using low-level interfaces that the modified kernel provide.

Our experiments confirm that the latter approach outperforms the former. However, we wish to point out that careful design enables the latter approach to provide a similar performance to what the former approach can do in common workloads. We also confirm it is comparable to that of an entry-class storage system.

We used two iSCSI target software packages for Linux, UNH iSCSI target software [2] version 1.5.03 and Ardis target software [3] version 20040211.

The outline of the rest of this paper is as follows. Section 2 summarizes related work. Section 3 analyses the design of iSCSI target software. Section 4 presents our performance results.

And then Section 5 summarizes the main points to conclude the paper.

# 2 Related work

While some performance studies of the iSCSI protcol have been conducted in the past, none has evaluated iSCSI target systems.

Peter Radkov et al. [4] have compared the performances of the iSCSI protocol and NFS (Network File System) [5].

Wee Teck Ng et al. [6] investigated the performance of the iSCSI protocol over wide area networks with high latency and congestion.

Sarkar et al. [7] compared an initiator implementation using a Gigabit Ethernet adapter to implementations using specialized network adapters, known as TOE and HBA adapters.

Ashish Palekar et al. [8] described the design of UNH iSCSI target software.

There are a number of papers that point out that on high-speed links is limited by the host system and study several optimization approaches.

Chase et al. [9] outline a variety of approaches to optimize TCP performance, which operating systems specialized for commercial storage systems possibly adopt.

# 3 iSCSI target software design

The UNH target provides two different modes; *fileio mode* and *diskio mode*. We call UNH fileio target and UNH diskio target respectively.

We compare four iSCSI targets; UNH fileio target, UNH diskio target, Ardis target, and our iSCSI target, called *Threaded-Ardis* target.

Our iSCSI target is based on the Ardis code. We chose an Ardis iSCSI target software as the starting point because it is easy to reuse the code due to the simplicity of its design.

## 3.1 Assessment criteria

Our assessment criteria for iSCSI target designs is as follows:

**Kernel modification** Modifying the source code of Linux kernels, which are not specialized for storage systems, can lead to the better performance. However, such modifications makes it difficult to utilize future functionality developed by Linux kernel developers.

**Interoperability** A storage system would be disqualified if it were to require software changes in the operating system or application. That is, iSCSI storage systems need to work in the exact same way as traditional direct-attached storage systems do.

**Disk management** It is important for storage systems in production use to utilize disk management features like Logical Volume Manager (LVM) [10] and software RAID, which are provided by Linux kernels.

**Performance** Target software must be designed to support workloads in a production environment. In Section 4, we compare open-source iSCSI target systems to a commercial iSCSI target system, which is expected to support such workloads, by using benchmark software.

## 3.2 Architecture overview

As shown in Figure 1, all targets use three kernel threads to provide the major part of the iSCSI target functionality: a *read thread* receives data from an initiator; an *I/O thread* performs I/O operations; and a *write thread* transmits the response and data to the initiator.

## 3.3 I/O thread

In I/O thread design, there are many design differences among the targets.

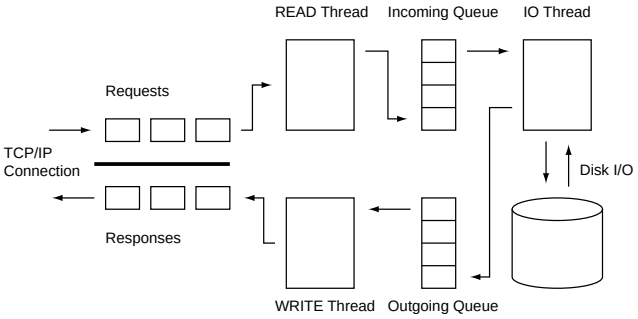


Figure 1: Architecture overview of iSCSI target

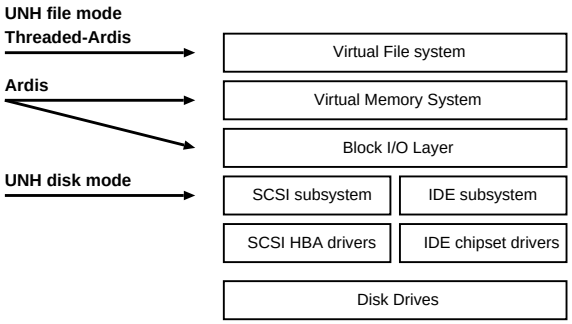


Figure 2: I/O interfaces

A Linux kernel provides several different interfaces for I/O operations. As shown in Figure 2, each I/O thread uses a different interface.

UNH fileio and Threaded-Ardis targets use the virtual file system (vfs) interface to perform I/O operations. UNH fileio target can only support regular files, although the vfs interface also provides access to block devices. On the other hand, Threaded-Ardis target can support both regular files and block devices.

UNH diskio target uses an SCSI subsystem interface. The SCSI command data structure, included in the iSCSI command, is passed to the SCSI subsystem interface. As a result, a diskio I/O thread can only handle SCSI disk drives.

Ardis target uses original functions implemented by using the interfaces provided by the modified Virtual Memory (VM) subsystem and the block I/O layer. Ardis target can handle block devices.

### 3.3.1 Kernel modifications

Ardis target requires modifications to the source code of Linux kernels, although the others don't.

UNH fileio, UNH diskio, Threaded-Ardis targets uses only exported kernel functions and data objects, that is, standard kernel interfaces.

Ardis target uses unexported kernel functions and data objects by modifying the kernel source code in order to implement specialized functions.

Such modifications increase costs to utilize future bug fixes and features produced by Linux kernel developers.

There is one more issue to consider about the cost of maintaining the code.

UNH fileio, UNH diskio, Threaded-Ardis targets keep the design simple by choosing as many high-level interfaces as possible [1].

On the other hand, the Ardis target uses many low-level kernel interfaces in order to implement specilized functions.

These different design philosophies lead to tradeoffs. The usage of well-tested, high-level, standard kernel interfaces has clear advantages in terms of stability, implementation and maintenance costs, and portability. On the other hand, the Ardis philosophy has performance advantages, due to its specialized functions.

### 3.3.2 Interoperability

About handling SCSI write commands, the UNH file target works in the different way as traditional direct-attached storage systems do.

When an initiator receives a SCSI write command response from a target, the initiator expects the target to finish writing the data to a physical disk drive. We call this *write durability*. File systems and database programs depend on this to maintain data integrity against system failures [11].

[1] In this paper, we call an interface providing rich functionality a high-level interface.

However, the UNH fileio target does not provide write durability. When copying the received data to the page cache is completed, the target transmits its response. The modified cached data are written to disk at some future time by the Linux kernel. When a system crashes before these data are written, the initiator finds the unwritten data even if the initiator received the completion response of the data.

Due to the lack of write durability, the UNH fileio target is not practical for production purposes.

#### 3.3.3 Disk management

UNH fileio, Ardis, and Threaded-Ardis targets have the advantage of device virtualization. That is, the Linux kernel provides a generic interface to various block devices, allowing all block devices (SCSI, IDE, Serial-ATA disk drives, etc) to be used easily. In addition, these targets also use virtual block devices that provide flexible storage management (LVM) and redundancy (software RAID).

In contrast, UNH diskio target cannot use such features because it bypasses the block I/O layer, which provides such features. Thus, as described, UNH diskio target can support only SCSI disk drives.

#### 3.3.4 Performance

UNH fileio, Ardis, and Threaded-Ardis targets can use the page cache, which minimizes disk I/O by storing data in the physical memory, whereas UNH disk target cannot. This is because the UNH diskio target bypasses the VM subsystem, which provides the page cache feature, and directly accesses the SCSI subsystem. As a result, every SCSI read command invokes a disk I/O, which causes a poor read performance. This is confirmed in section 4.

To achieve a high performance, an I/O thread needs to perform I/O operations required by SCSI commands simultaneously. Thus, I/O threads need an I/O interface to work asynchronously.

UNH diskio target can easily achieve this because the SCSI subsystem interface works asynchronously.

The vfs interface, which UNH fileio and Threaded-Ardis targets use, works synchronously. That is, the vfs interface makes an I/O thread block until an I/O operation finishes.

Threaded-Ardis target uses multiple I/O threads to handle multiple I/O operations simultaneously, although UNH fileio target can handle only one SCSI command at a time. We will examine how much this constraint effects performance in section 4.

Ardis target also has difficulty performing SCSI command I/O operations simultaneously, because the interface for the page cache works synchronously. To carry out the SCSI command operations, the Ardis I/O thread adopts a polling model by using VM subsystem interfaces which are not exported to external kernel modules.

# 4 Performance

## 4.1 System Comparison

We examined five iSCSI target systems: *Ardis*, which is the original version of the Ardis target; *Threaded-Ardis*, which a newly implemented target based on the Ardis target code; *UNH-disk*, which is the UNH target using diskio mode; *UNH-file*, which is the UNH target using fileio mode; and *UNH-file-sync* is the a modified UNH target using fileio mode that provides write durability by setting the `O_SYNC` flag.

There is no difference between UNH-file and UNH-file-sync about executing READ commands. Thus, we omit the results of UNH-file-sync about read workloads.

Table 4.1 lists our iSCSI configurations.

We report the averages of the five runs for all the experiments.

Table 1: iSCSI parameters

| Item | Value |
|---|---|
| InitialR2T | On |
| ImmediateData | On |
| MaxRecvDataSegmentLength | 128 KB |
| MaxBurstLength | 256 KB |
| FirstBurstLength | 64 KB |
| MaxConnections | 1 |
| MaxOutstandingR2T | 1 |

## 4.2 Experimental infrastructure

The iSCSI target host uses one 2.0 GHz Xeon processor, with 2 GB main memory, and runs the Linux kernel version 2.4.25. Maxtor Atlas 10K, 36.7 GB 10,000 RPM SCSI disks are directly connected to the host via LSI Logic 53C1030 Ultra320 SCSI chip.

The iSCSI initiator host uses one 2 GHZ Xeon processor with 1 GB main memory, and runs the Linux kernel version 2.6.4. It uses the version 4.0.1.1 of a Cisco iSCSI initiator [12].

Both the hosts use an Intel Pro/1000 MT Server Adapter connected to a 66MHz 64-bit PCI slot. They are connected by an Extreme Summit 7i gigabit Ethernet switch.

The UNH-file provides a regular file, stored on the ext2 file system, as a logical unit (LU) to the initiator.

## 4.3 Microbenchmark results

We performed microbenchmarks that run in kernel mode and directly interact with the block I/O layer. The benchmarks read or wrote data 50,000 times, with I/O sizes ranging from 2 to 32 KB bytes.

The benchmarks began with a cold cache on both the initiator and target hosts.

With Ardis, we were not able to perform sequential writes with 2KB or all tests with 32KB because of its bugs.

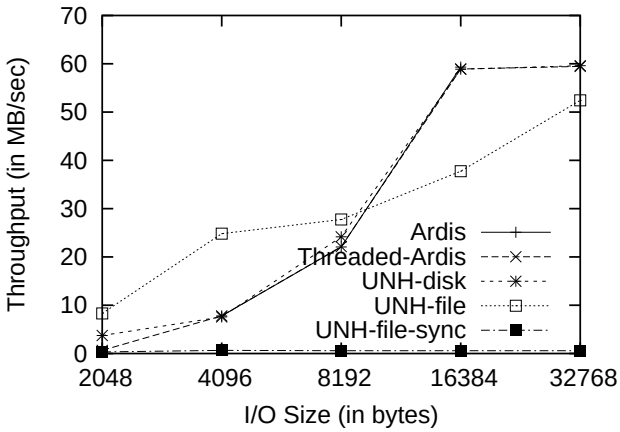


Figure 3: Sequential write results

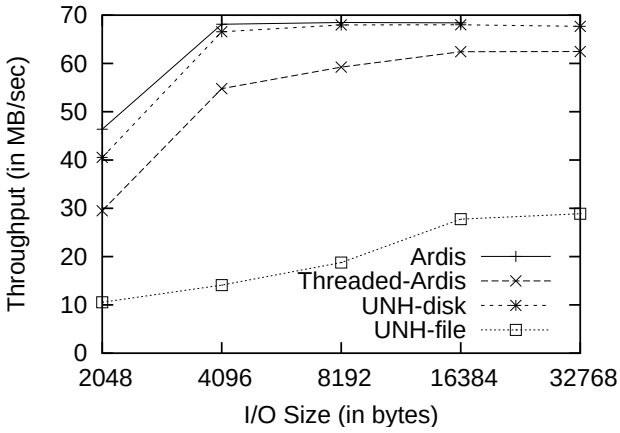


Figure 4: Sequential read results

Figure 3 shows the results of the sequential writes. As expected, the UNH-file-sync performance was very poor because it cannot handle I/O operations simultaneously.

While the UNH-file cannot handle I/O operations simultaneously, it can provide performance comparable with others. This is because the target host rarely performed disk IOs. That is, the overhead of disk operations has little effect on the performance. The ext2 file system, on which the UNH-file stores the file provided for the initiator as a LU, delays writing modified cached data to disk. Especially, it outperformed others for I/O sizes of less than 8 KB because the oper-

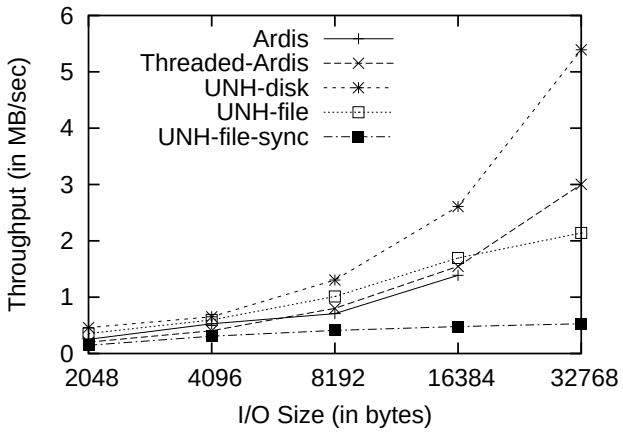


Figure 5: Random write results

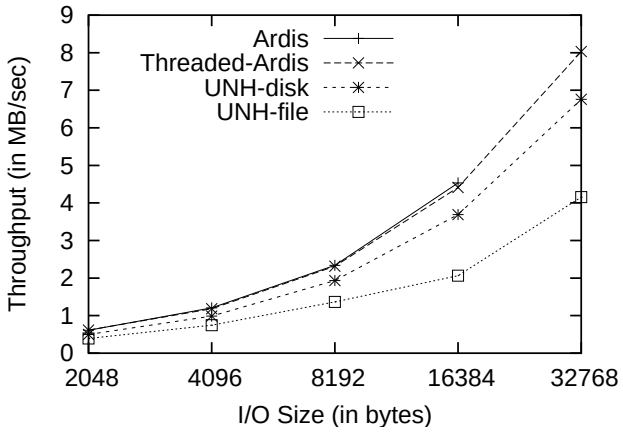


Figure 6: Random read results

ating system performed no disk operations during the benchmarks in these conditions.

Ardis, Threaded-Ardis, and UNH-disk had comparable performances. This indicates that the multi-threaded design overheads are negligible.

Figure 4 shows the results of sequential reads. As expected, the UNH-file was outperformed by the others, because it cannot handle I/O operations simultaneously. Unlike sequential writes, completing a SCSI command takes time due to disk operations. Thus, we could see the large performance drop results from the lack of ability to handle multiple SCSI commands at a time.

With 2 KB, Ardis is faster than UNH-disk. This is because Ardis always performs I/O operations in 4 KB units.

Threaded-Ardis performs at 63%~91% of Ardis. This is due to the performance difference between the specialized functions and the vfs interface. The vfs interface for reads is more complicated than specialized functions because it performs extra operations like read-ahead (prefetching of the next disk blocks).

Figure 5 and 6 shows random writes and reads respectively. The large performance drop results from the overheads of disks.

While most of the results are as expected, there is one point to note. The write performance of Ardis is slower than UNH-disk, UNH-file, and Threaded-Ardis. This is because Arbre cannot effectively handle lots of pending SCSI commands due to the adoption of a polling model. With random writes and reads, it takes longer to complete I/O operations. Thus, targets tend to handle more SCSI commands simultaneously.

## 4.4 Macrobenchmark results

To evaluate the performance in common workloads, we performed two macrobenchmarks. With these macrobenchmarks, we also evaluated an entry-class commercial iSCSI target system. The product details are confidential.

With macrobenchmarks, the initiator host creates an ext2 file system with a 4 KB block size on a block device that the target provides and mounted it with the default option.

### 4.4.1 Postmark benchmark

To evaluate the performance in write intensive workloads, we ran the PostMark benchmark [13], designed to measure performance in the ephemeral small-file workloads seen by Internet service providers. We set the benchmark to run with 20,000 files, 50,000 transactions, and file sizes of between 512 bytes and 16 KB.

The benchmarks were run with a cold cache on both the initiator and target hosts.

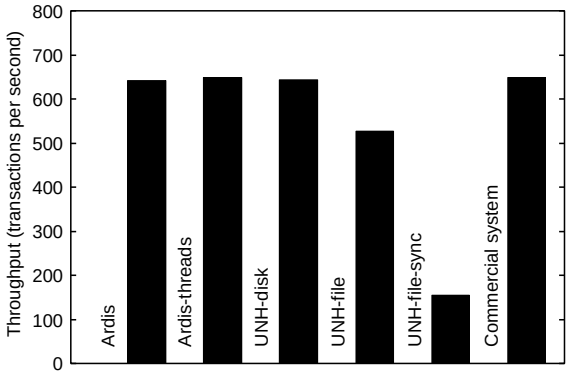


Figure 7: Postmark results

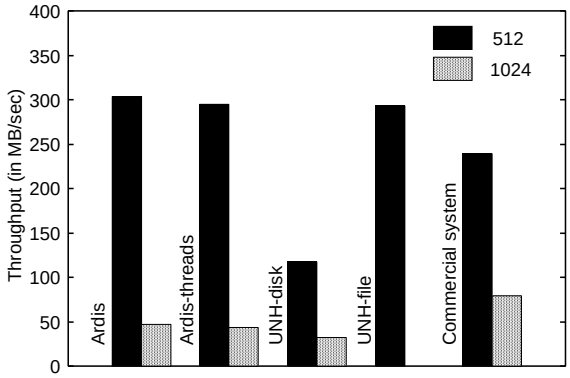


Figure 8: Network server benchmark results

Figure 7 shows the results. As expected, the UNH-file-sync performance was the worst. All targets except for the UNH-file-sync and UNH-file targets provided comparable throughputs.

### 4.4.2 Network server benchmark

We ran a read intensive benchmark, Network Server benchmark, designed to produce the workloads seen by HTTP or FTP servers. The benchmark repeatedly reads many files, which were placed in one directory, in random order. The files are subjected to repetitious reads. Thus, the page cache has large effects on the performances.

We configured the benchmark to read 512, 2 MB files 4,096 times in total and 1,024, 2 MB files 4,096 times in total. The maxium size of disk that UNH-disk can provide to the initiator is 2 GB. Thus, it cannot handle 1,024, 2MB files.

The goal of the benchmark was to evaluate how much the page cache effected the performance. Therefore, we started the benchmark with both the initiator and target in a hot cache.

Figure 8 shows the results. The page cache on the target host achieved a hit rate of 85.4% and 25.4% respectively.

When the number of files is 512, we could see that the three targets using the page cache, greatly outperformed UNH-disk that cannot use the page cache.

When the number of files is 1,024, the hit rate decreased. Thus, the page cache did not have much effect on the performances. As a results, we could see the smaller performance margin between the three targets using the page cache and UNH-disk. With taking account of 2 GB memory of the target host, the hit rate is lower than expected. This is because Linux kernels use only the first 896 MB of physical memory as page cache for block devices.

# 5 Conclusion

This paper presents an analysis of the iSCSI target software designs, using two open source software packages. Our experiments showed that, without any modification to the general-purpose operating system kernel, careful design techniques enable to achieve the comparable performances of the storage system that uses specialized functions with the modified kernel on both the read and write intensive workloads. We also found that an iSCSI storage system build by using open-source software and commodity hardware can provide comparable performance with an entry-class storage system.

Threaded-Ardis source is available from **http://sourceforge.net/projects/iscsitarget/**

# References


[1] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner, "Internet Small Computer Systems Interface (iSCSI)," April 2004, RFC 3720.

[2] InterOperability Laboratory: the iSCSI consortium, http://www.iol.unh.edu/consortiums/iscsi/.

[3] Ardis Technologies iSCSI target implementation, http://www.ardistech.com/iscsi/.

[4] P. Radkov, L. Yin, , P. Goyal, and P. Sarkar, "A Performance Comparison of NFS and iSCSI for IP-Networked Storage," in *the USENIX Conference on File and Storage Technologies*, San Francisco, CA, March 2004, pp. 101–114.

[5] R. Sandberg, D. Coldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network Filesystem," in *the USENIX Summer Conference*, Portland, OR, June 1985, pp. 119–130.

[6] W. T. Ng, B. Hillyer, E. Shriver, E. Gabber, and B. Özden, "Obtaining High Performance for Storage Outsourcing," in *the USENIX Conference on File and Storage Technologies*, Monterey, CA, January 2002, pp. 145–158.

[7] P. Sarkar, S. Uttamchandani, and K. Voruganti, "Storage over IP: When Does Hardware Support Help?" in *the Conference on File and Storage Technologies (FAST 03)*. San Francisco, CA: USENIX, January 2003, pp. 231–244.

[8] A. Palekar, N. Ganapathy, A. Chadda, and R. D. Russell, "Design and implementation of a Linux SCSI target for storage area networks," in *5th Annual Linux Showcase & Conference*. Atlanta, GA: USENIX, November 2001.

[9] J. S. Chase, A. J. Gallatin, and K. G. Yocum, "End-System Optimizations for High-Speed TCP," *IEEE Communications*, vol. 39, no. 4, pp. 68–74, 2001.

[10] D. Teigland and H. Mauelshagen, "Volume Managers in Linux," in *the USENIX Annual Technical Conference*, Boston, MA, June 2001, pp. 185–198.

[11] M. Seltzer, G. Ganger, M. K. McKusick, K. Smith, C. Soules, and C. Stein, "Journaling Versus Soft Updates: Asynchronous Meta-data Protection in File Systems," in *the USENIX Annual Technical Conference*, San Diego, CA, June 2000, pp. 71–84.

[12] Cisco iSCSI initiator Implementation, 2001, http://sourceforge.net/projects/linux-iscsi/.

[13] J. Katcher, "PostMark: A New File System Benchmark," Network Appliance, Tech. Rep. TR3022, October 1997.