Trace System of iSCSI Storage Access ver. 773

Saneyasu Yamaguchi
Institute of Industrial Science,
The University of Tokyo
4-6-1 Komaba Meguro-ku,
Tokyo 153-8505, Japan
sane@tkl.iis.u-tokyo.ac.jp

Masato Oguchi
Department of
Information Sciences,
Ochanomizu University
2-1-1 Otsuka Bunkyo-ku,
Tokyo 112-8610, Japan
oguchi@computer.org

Masaru Kitsuregawa
Institute of Industrial Science,
The University of Tokyo
4-6-1 Komaba Meguro-ku,
Tokyo 153-8505, Japan
kitsure@tkl.iis.u-tokyo.ac.jp

Abstract

In this paper, an IP-SAN access trace method is proposed and its implementation is presented. IP-SAN and iSCSI are expected to remedy problems of Fibre Channel (FC)-based SAN. Because servers and storage cooperatively work with communications through TCP/IP layer in IP-SAN system, an integrated analysis of both sides is considered to be significant for achieving better performance.

Our proposed system can precisely point out the cause of performance degradation when IP-SAN is used for a remote storage access. In the experiment of parallel iSCSI access in a long-delayed network, the total performance is limited by a parameter in an implementation of the SCSI layer in the iSCSI protocol stack. Based on the result obtained with our IP-SAN access trace system, the parameter in the layer is modified. As a result, four times performance improvement is achieved compared with the default value case. Thus it is effective to monitor all the layers in the iSCSI protocol stack and execute an integrated analysis, using our system.

1. Introduction

Recently, storage management cost is one of the most important issues of computer systems [12, 13]. Since periodical backup is required for management of storage, if the storage is distributed among many servers, its management cost is extremely high. Storage Area Network (SAN), a high speed network for storage, is introduced to resolve this issue. Each server is connected to consolidated storage through SAN. Management cost can be significantly decreased by the consolidation of storage, thus SAN has already become an important tool in the business field. However, current generation SAN based on FC [6, 7] has some demerits; for example, (1) the number of FC engineers is small, (2) installation cost of FC-SAN is high, (3) FC has distance limitation, (4) the interoperability of FC is not necessarily high.

The next generation SAN based on IP (IP-SAN) is expected to remedy these defects. IP-SAN employs commodity technologies for a network infrastructure, including Ethernet and TCP/IP. One of the promising standard data transfer protocol of IP-SAN is iSCSI [2, 18], which was approved by IETF [1] in February 2003 [4]. IP-SAN has following advantages over FC-SAN [8, 16, 17]: (1) the number of IP engineers is large, (2) initial cost of IP-SAN is low, (3) IP has no distance limitation, (4) Ethernet and IP have no interoperability problem. However, the problem of low performance and high CPU utilization is pointed out as demerits of IP-SAN [16, 17, 20]. Thus improving its performance and keeping CPU utilization at low rate [11, 16] are critical issues for IP-SAN.

We evaluate the performance issue of iSCSI in this paper. As an instance of evaluation, an iSCSI access in a long-delayed network is investigated. This is because performance decline caused by network latency is pointed out in IP-SAN [13, 15], while iSCSI achieves almost comparable performance with that of FC-SAN in a LAN environment [8, 10]. In addition, although a SCSI access over a long-delayed network is an important case, which can be realized since iSCSI has no distance limit, an iSCSI access over a long-delayed network is not discussed enough.

Studies for CPU utilization during communications are found in the literature [9, 11, 16, 17, 19]. We do not discuss hardware supported TCP processing in this paper, because [8, 16, 17] concluded that although such hardware is effective for reducing CPU utilization, it does not achieve better performance than that of the software-based approach. We also evaluated the hardware supported TCP processing by ourselves and obtained a similar result: In our experiments, TCP/IP communication throughput and CPU utilization with software TCP/IP implementation were 70.4

[MB/sec] and 27.5 %, respectively. Throughput and CPU utilization with TCP/IP offload engine (TOE) were 63.1 [MB/sec] and 10.4%, respectively.

An iSCSI storage access is composed of many protocols, that is "SCSI over iSCSI over TCP/IP over Ethernet". The protocol stack of iSCSI is complicated and the storage access is executed through all these layers, so that any layers can be a performance bottleneck of end-to-end performance communication. Consequently, all these layers should be observed for improving iSCSI storage access performance. In addition, integrated analysis of the behavior of both server computers and storage appliances is important because iSCSI is composed of the protocol stack in both sides. In IP-SAN system, although server computers and storage appliances work cooperatively via iSCSI protocol, since they have their own OSs, they are monitored separately in general. However, it is difficult to understand the whole system's behavior by monitoring only one side, thus integrated analysis is required.

In this paper, we propose an "IP-SAN trace system", which can monitor all the layers in IP-SAN protocol stack and show the integrated analysis of the whole IP-SAN system. Next, we apply the proposed system to parallel iSCSI accesses in a long-delayed network, and demonstrate the system can point out the cause of performance decline. In our experiment, significant iSCSI performance improvement is achieved using the system.

The rest of this paper is organized as follows. Section 2 provides a brief explanation of iSCSI. We propose "IP-SAN Trace System" in Section 3, and present an actual adaptation of the system in Section 4. Section 5 introduces related works. In Section 6, we conclude this paper.

2. iSCSI

In this section, an overview of iSCSI and IP-SAN is shown.

iSCSI [2, 18] is a block level data transfer protocol for IP-SAN. It was approved by IETF [1] in February 2003. In an iSCSI storage access, the SCSI protocol is encapsulated into the TCP/IP protocol and transferred over TCP/IP network for accessing remote SCSI storage. The protocol stack of an iSCSI storage access is "SCSI over iSCSI over TCP/IP over Ethernet", which is shown in Figure 1 ("Monitor" in the figure will be introduced in Section 3).

I/O requests issued by application programs in server computers are transferred through either files system or block device or character device at first, then through the SCSI layer, the iSCSI layer, the TCP/IP layer, and the Ethernet layer in a server computer. The requests are transmitted to storage appliances via Ethernet, and transferred through the Ethernet layer, the TCP/IP layer, the iSCSI layer and the SCSI layer, and finally received by the storage device.

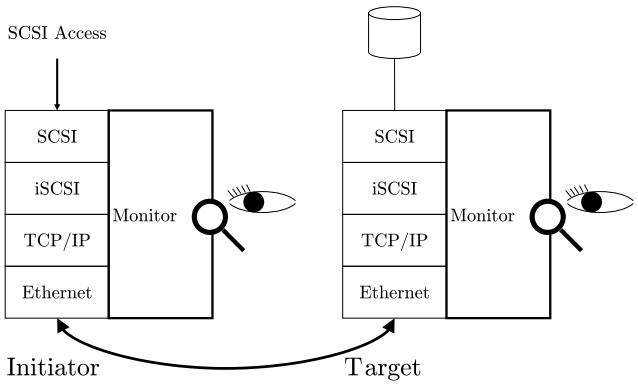

Figure 1. iSCSI Protocol Stack and Analysis System

Responses for the I/O requests are transferred to the applications through the same path inversely. Any of these layers may be the cause of end-to-end performance decline. For example, iSCSI protocol defines "MaxRecvDataSegmentLength", "MaxBurstLength" and "FirstBurstLength", the TCP/IP layer restricts output performances according to its flow controlling algorithm [23], and the number of Ethernet layer's packets descriptor is limited [23]. They can be significant issues of performance decline.

As we mentioned, an iSCSI storage access is executed through all these layers. However, because these layers are designed not for iSCSI storage accesses but for general purposes, some functions of these layers may degrade iSCSI performance. Consequently, detailed analyses of all layers are required for improving iSCSI performance. For example, we have already found that TCP's Nagle's algorithm and delayed ack algorithm severely decrease iSCSI performance [22]. A large block are divided into multiple small blocks by OS and they are synchronized in the SCSI layer, which severely decrease iSCSI performances [14, 21]. And the combination of TCP's flow controlling algorithm and SCSI synchronization heavily declines iSCSI performances [23].

3. IP-SAN Trace System

In this section, we present detailed explanation of the proposed "IP-SAN Trace System".

We have implemented a monitoring system, shown in Figure 1, which monitors all the layers in IP-SAN. We have also constructed an integrated trace system which comprehensively analyzes logs recorded in both server computes and storage appliances. For our experiments, open source codes of an OS implementation and an iSCSI driver are

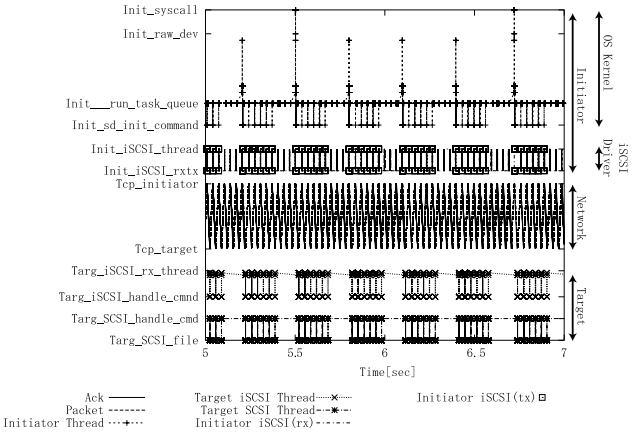

Figure 2. iSCSI Access Trace

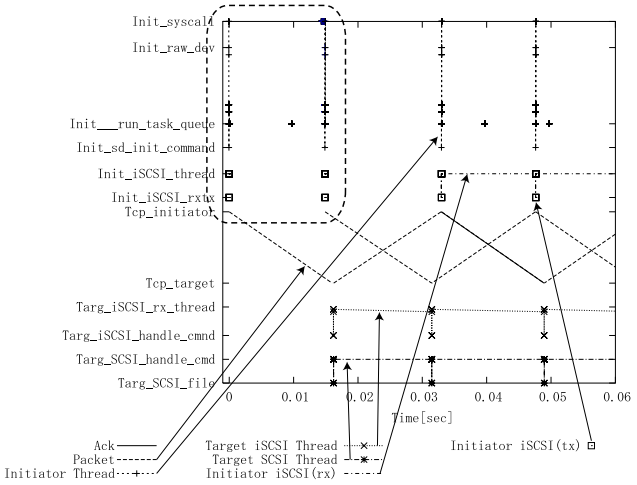

Figure 3. Visualized Trace of Parallel iSCSI Access (default): A

used. We have inserted monitoring codes into these implementations for recording IP-SAN system's behavior. Linux (kernel version 2.4.18) is adopted as an OS implementation and iSCSI reference implementation (version 1.5.02) [3] developed by University of New Hampshire's InterOperability Lab [5] (we call this implementation "UNH") is used as an iSCSI driver.

Figure 2 shows an example of a visualized iSCSI trace obtained by the proposed trace system. In the figure, Y-Axis stands for the state transition of iSCSI storage access. Each label beside Y-Axis indicates each layer in the iSCSI protocol stack. Meaning of these labels (`Init_syscall`, `Init_raw_dev` and so on) are; 1) system calls issued by applications, 2) the raw device layer, 3) the SCSI layer, 4) the iSCSI layer, 5) the TCP/IP layer, 6) Packet transmission by the Ethernet layer, 7) the TCP/IP layer, 8) the iSCSI layer, 9) the SCSI layer, 10) HDD device access from the top to the bottom respectively. The labels from 1) to 5) belong to processes in server computers (iSCSI initiator) and the labels from 7) to 10) belong to processes in storage appliance (iSCSI target). In this case, we have used raw device mode instead of file system mode in the iSCSI initiator. The iSCSI target works with "File Mode" of UNH implementation[1], thus the trace in the lowest layer is not that of HDD device access but that of file access in the target OS's file system. X-Axis stands for the time of each trace.

The figure helps to understand IP-SAN's behavior, for example, which process in iSCSI protocol stack dominantly consumes time, in which layer processes are waiting for I/O responses, and a block of the issued I/O requests are divided into small blocks by some layers. In the case of this figure, an application issues system calls `read()` with 2MB block size. The raw device layer divides it into 4 blocks of 512KB, then issues 512KB I/O requests one by one to the lower layer (the SCSI layer), finally it returns I/O responses to the upper layer (the system call layer) after completing 4 requests. After the SCSI layer receives 512KB I/O requests, it divides the requests into multiple 32KB SCSI read commands and transfers them to the lower layer (the iSCSI layer). iSCSI `tx_thread` is activated when requests are sent from the SCSI layer, and the iSCSI layer transferred the requests to the TCP/IP layer. The TCP layer sends data segments to the Ethernet layer, and the Ethernet layer sends them to the storage appliance (target computer).

Figure 3 shows an example of a visualized traces in the case of parallel iSCSI accesses (the area surrounded with the broken line will be mentioned in Section 4.3.2). Cooperation of the initiator and the target can be understood easily with this figure.

The proposed system monitors the IP-SAN's behavior by modifying the source codes, thus the system can observe the behavior of the kernel and the iSCSI driver at source code level. For example, this system records which way is selected in a branch like Figure 4. These figure will be mentioned in Section 4.3 again.

As shown above, the whole IP-SAN's behavior can be easily understood with the trace system.

4. Trace of Parallel iSCSI Accesses in Long-Delayed Network

In this section, we present how the proposed "IP-SAN access trace system" is actually applied to IP-SAN. In addi-

1 UNH implementation can export a local file to initiator as a storage image.

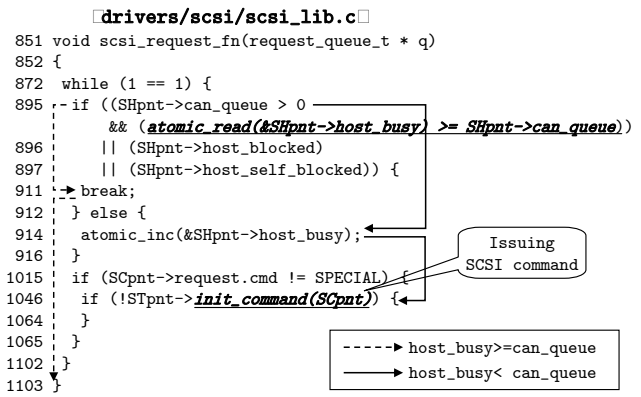

Figure 4. Trance of Linux SCSI Layer: "drivers/scsi/scsi_lib.c"

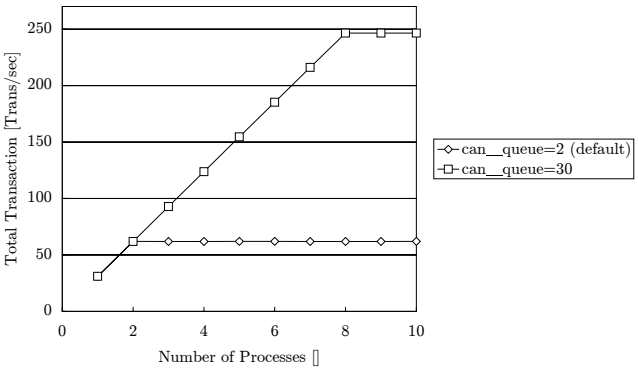

Figure 5. Experimental Result: Total performance of parallel I/O, 16ms

tion, we show that the system can point out the cause of performance decline. The proposed system is applied to a short block of parallel iSCSI accesses in a long-delayed network environment. It points out which layers restrict the number of parallel processing.

4.1. Experimental Setup

We have constructed a long-delayed IP-SAN environment. A network delay emulator is inserted between an iSCSI initiator (server computer) and an iSCSI target (storage appliance). The network delay emulator is constructed with FreeBSD Dummynet. The initiator and the target establish TCP connection over the delay emulator and an iSCSI connection is established over this TCP connection.

The UNH iSCSI implementation (refer to Section 3) is employed as iSCSI initiator and target implementation. Since the iSCSI target works with "File Mode", the following experiments do not include actual HDD device accesses. One way delay time is 16 ms.

The initiator, the Dummynet, and the target are built with PCs. Linux OS is installed to both initiator and target PCs. The detailed specifications of the initiator and the target PC are as follows: CPU Pentium4 2.80GHz, Main Memory 1GB, OS Linux 2.4.18-3, NIC Gigabit Ethernet Card Intel PRO/1000 XT Server Adapter. The detailed specifications of Dummynet PC are as follows: CPU Pentium4 1.5GHz, Main Memory 128MB, OS FreeBSD 4.5-RELEASE, NIC Intel PRO/1000 XT Server Adapter$\times$2.

We have executed the following benchmark in this experimental environment. The benchmark software iterates issuing system call `read()` to raw device which is established with an iSCSI connection. The block size of the read requests is 512 Bytes. SCSI Logical Block Addresses (LBAs) to be read is specified sequentially. The addresses do not have an impact on experimental performance because of target side file system cache (which will be mentioned later in this section). We executed multiple processes simultaneously and measured total performance of all processes. Each benchmark process iterates 2048 times system call `read()`.

In this environment, the issued system calls are always transmitted to the SCSI layer in the target side without any cache hit in the initiator side, because the benchmark processes issue system calls to the raw device. These experiments are executed when target storage image in file (the iSCSI target is executed with "File Mode") is stored in file system's cache on the target (worm cache). Consequently, all read requests issued from the initiator reach the SCSI layer in the target side and hit file system cache in the target side, thus it does not include HDD device access. We have employed the file mode iSCSI target in order to isolate the efficiency of the behavior of IP-SAN system from the behavior of the HDD device.

Experiments in Section 4.2 and Section 4.4 are executed without the monitoring system, thus the performances shown in Figure 5 are not effected by the monitoring system.

4.2. Experimental Results

The experimental results with default setup, which does not have any tuning, are shown as "`can_queue=2(default)`" in Figure 5. X-Axis in the figure stands for the number of processes executed simultaneously. Y-Axis stands for the number of total transactions of all processes per second. The number of transactions means the number of 512 Bytes system calls `read()`.

In the case of a single process, the transaction performance is 31.0 [Trans/sec]. This nearly equals to the reciprocal number of Round Trip Time (RTT), 32 [ms/transaction] in this experiment. The result is reasonable in this case. In the case of two concurrent processes, the total transaction performance is 62.0 [Trans/sec]. Nearly doubled performance improvement is achieved using two processes. This is also reasonable because short block accesses in a long-delayed network do not consume many computer resources, for instance the resources of network, CPU, and memory.

In the cases of more than three concurrent processes, the performance improvement is not obtained compared with that of two processes case. These results imply that a layer in iSCSI protocol stack restricts the number of I/O requests processed concurrently to two, which is a critical cause to hinder the improvement of total performance.

4.3. Trace Analysis of Parallel iSCSI Access

In this subsection, we present trace analyses of parallel accesses and demonstrate that the proposed system can point out the cause of performance decline.

4.3.1. Trace of I/O requests Figure 3 is obtained by analyzing traced logs of the experiment in Section 4.1 and Section 4.2. The number of processes is three. The trace lines of "Initiator Thread" are drawn discontinuously in the figure. This is because context switches are issued by OS's process scheduler, and the processes suspended and resumed on these lines. Plots at 0.010 [sec] and 0.040 [sec] also indicate context switches by the process scheduler. Although the scheduler allocates CPU resources to the processes at these points, the processes are waiting for I/O response at that time, thus they immediately invoke context switches and release CPU resources. According to the figure, only two I/O requests are sent from the initiator to the target within RTT (32ms). This indicates the number of I/O requests processed concurrently is restricted in the initiator side.

4.3.2. Analysis of the Trace across Multiple Layers An analysis of the trace to determine a cause of the restriction for parallel processing is shown in this subsection. The trace is analyzed across multiple layers of iSCSI at first.

Figure 6 is obtained by magnifying the area surrounded with the dotted line in Figure 3. Figure 7 shows magnified view of the area surrounded with the dotted line in Figure 6.

In these figures, three processes running concurrently are drawn, labeled as "I/O(A)", "I/O(B)", and "I/O(C)". Traced lines are shown discontinuously like Figure 3. The lines terminate when context switch occurs and processes resume. Figure 6 shows that I/O(A), (B), and (C) issue a system call

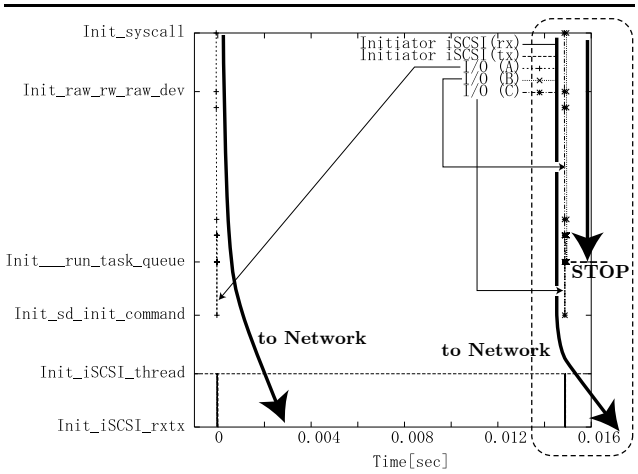

Figure 6. Visualized Trace of Parallel iSCSI Access (default): B

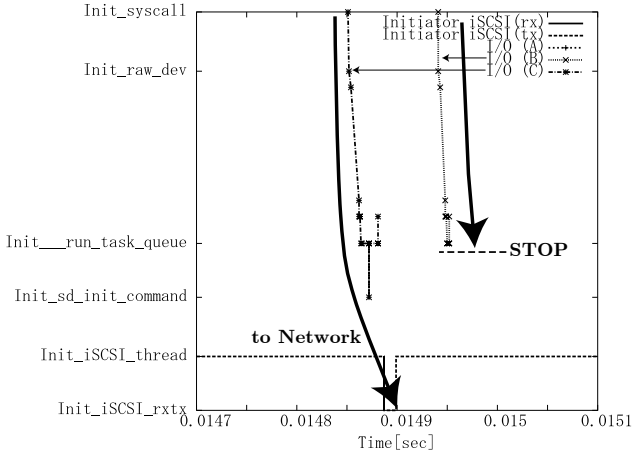

Figure 7. Visualized Trace of Parallel iSCSI Access (default): C

at 0.000 [sec], 0.015 [sec] and 0.015[sec] respectively. According to the figure, three system calls can be issued within one RTT, receiving no response from the target.

The trace of "I/O(A)" shows that the request by "I/O(A)" is transferred through the raw device layer, the SCSI layer, and the iSCSI layer, then the iSCSI layer issues a request to the TCP/IP layer. Figure 7 shows that "I/O(C)" issues a system call at 0.01485 [sec], and the request is transferred up to the iSCSI layer and sent to the network.

On the other hand, in the case of "I/O(B)", the issued re-

quest is not sent to the iSCSI layer. A system call is issued by "I/O(B)" at 0.01494 [sec], and the raw device layer also issues the I/O request to the lower layer (the SCSI layer) after the issue of the system call. However, the SCSI layer returns without issuing a SCSI command even though the layer has received the request. This result indicates that the maximum number of SCSI commands issued simultaneously is restricted to two in the SCSI layer, which is considered to be the cause of the upper limit of total performance.

4.3.3. Analysis of the Trace inside a Layer The trace is analyzed more precisely, focusing on a particular layer, which is determined in the previous analysis. The proposed system can trace the behavior inside an IP-SAN system in source code level.

The branch point of the first two requests (I/O(A) and (C)) and the third request (I/O(B)) is in "`drivers/scsi/scsi_lib.c`" in the implementation of Linux SCSI layer, as shown in Figure 4. This part in Linux SCSI implementation compares "`host_busy`" [2], the number of active commands and "`can_queue`" [3], the maximum number of SCSI commands the lower layer (iSCSI driver implementation in our case) can receive simultaneously. The default value of "`can_queue`" in the UNH iSCSI implementation is 2.

At the beginning, "`host_busy`" is 0. In the cases of the first two I/O requests (I/O(A) and (C)), "`host_busy`" are 0 and 1 respectively, thus the route labeled as "`host_busy<can_queue`" in the figure is traced. In this route, incrementing "`host_busy`" at line `914` and issuing a SCSI command at line `1046` are recorded. In the case of the third I/O request (I/O(B)), "`host_busy`" was 2, thus the route labeled as "`host_busy>=can_queue`" in the figure is traced. In this route, a SCSI command is not issued as shown in the figure.

These analyses of the SCSI layer, which are determined in the proposed analysis system, point out that the upper limit of the total performance of parallel I/O requests is decided by the iSCSI implementation's default value of "`can_queue`".

4.4. Resolving the Pointed Out Issue and Performance Improvement

We measure the total performance of concurrent iSCSI accesses with "`can_queue`" = 30 and obtained "`can_queue=30`" in Figure 5. The total performance of all processes increases linearly from single process to eight processes. Four times performance improvement is achieved when the number of processes is greater than 8 by removing the cause of the performance decline pointed out by the proposed analysis system.

As we have shown, the integrated analysis of both server computers and storage appliances, monitoring all layers from application's system calls to HDD device access in the iSCSI protocol stack, is an effective method for improving iSCSI performance. We have demonstrated it by applying the system to an actual IP-SAN system so that the proposed system can properly point out the cause of performance limit, and the performance is significantly improved by resolving the pointed out issue.

5. Related Work

Some studies present performance evaluation of IP-SAN using iSCSI [8, 10, 13, 15, 16, 17].

Ng *et al.* [13] is a pioneering work. They early presented detailed performance evaluations and discussions of SCSI over IP. They showed experimental results of both microbenchmarks and macrobenchmarks, and showed results in various network delays and congestion environment. Their analysis of identifying bottlenecks includes discussions of file system, OS, TCP/IP and SCSI. They suggested that caching in the initiator side is effective in increasing SCSI over IP performance.

Sarkar *et al.* [17] prevented iSCSI performance evaluation. The study especially paid attention to CPU utilization of iSCSI storage access. It is very important work because high CPU utilization is one the most essential issue of IP-SAN. They experimented with performances of iSCSI storage accesses with various block sizes in a LAN environment. The work demonstrated that TCP/IP processing consumed much CPU resources. They showed the CPU utilization reached 100 % at the peak throughput, 64 KB block size. Sarkar *et al.* also published the study for effect of a current generation of TOE and a current generation of iSCSI HBA in [16]. It is also important work because using a TOE and an iSCSI HBA attract attention for resolving the iSCSI's CPU utilization issue. In order to compare iSCSI performances with a software approach, an approach using a representative TOE and an approach using a representative HBA approach, they executed micro-benchmarks and macro-benchmarks with various block size, with various I/O and with various CPU frequencies. They showed hardware approaches (TOE and HBA) were effective in decreasing CPU utilization but hardware approaches were not effective in improving iSCSI performances.

Fujita *et al.* [20] presented an analysis of iSCSI targets. The work gave not only performance evaluations but also

2 "`host_busy`" is explained as "commands actually active on low-level" in Linux SCSI implementation "`drivers/scsi/hosts.h`".

3 "`can_queue`" is explained as "max no. of simultaneously active SCSI commands driver can accept" in the UNH iSCSI implementation "`initiator/iscsi_initiator.c`".

discussions including the iSCSI target implementation and kernel implementation. This discussion is also mention behaviors inside IP-SAN system.

6. Conclusion

In this paper, we proposed an integrated IP-SAN trace method, implemented a system based on the idea, and demonstrated that the system could precisely point out the cause of performance decline. It was confirmed that iSCSI performance could be significantly increased by resolving the pointed out issue. In the case of our experiments, four times performance improvement has been obtained. Thus we found that monitoring all the layers in the iSCSI protocol stack and executing an integrated analysis including both server computers and storage appliances are effective for improving iSCSI performance.

We plan to explore the following matters as a future work. In this paper, we selected raw device as an upper layer of the SCSI layer. The iSCSI target driver works with "File Mode". We plan to analyze IP-SAN's behavior using a file system and actual HDD devices. We also plan to evaluate an overhead of the proposed analysis system. Since the cause of performance upper limit which restricts up to eight parallel processes is not mentioned in this paper, we will analyze and determine the cause.

References

[1] IETF Home Page. `http://www.ietf.org/` .

[2] IETF IPS. `http://www.ietf.org/html.charters/ips-charter.html` .

[3] iSCSI reference implementation. `http://www.iol.unh.edu/consortiums/iscsi/downloads.html` .

[4] Storage Networking Industry Assocation. `http://www.snia.org/` .

[5] University of new hampshire interoperability lab. `http://www.iol.unh.edu/` .

[6] "Fibre Channel - Arbitrated Loop," Standard X3.272-1996, 1996.

[7] "Fibre Channel - Switch Fabric," Standard NCITS 320-1998, 1998.

[8] S. Aiken, D. Grunwald, and A. Pleszkun. A Performance Analysis of the iSCSI Protocol. In *IEEE/NASA MSST2003 Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2003.

[9] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An Analysis of TCP Processing Overhead. *IEEE Communications Magazine*, 27(6):94–101, June 1989.

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

[11] J. C. Mogul. Tcp offload is a dumb idea whose time has come. In *9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, May 2003.

[12] F. Neema and D. Waid. Data Storage Trend. In *UNIX Review, 17(7)*, June 1999.

[13] W. T. Ng, B. H. E. Shriver, E. Gabber, and B. Ozden. Obtaining High Performance for Storage Outsourcing. In *Proc. FAST 2002, USENIX Conference on File and Storage Technologies*, pages 145–158, January 2002.

[14] M. Oguchi, S. Yamaguchi, , and M. Kitsuregawa. Performance Improvement of Sequential Access to IP-Storage using IP-SAN analysis tools. In *In Proceedings of the International Symposium of Santa Caterina on Challenges in the Internet and Interdisciplinary Research (SSCCII-2004), No.21*, January 2004.

[15] P. Radkov, L. Yin, P. Goyal, P. Sarkar, and P. Shenoy. A performance Comparison of NFS and iSCSI for IP-Networked Storage. In *Proc. FAST 2004, USENIX Conference on File and Storage Technologies*, March 2004.

[16] P. Sarkar, S. Uttamchandani, and K. Voruganti. Storage over IP: When Does Hardware Support help? In *Proc. FAST 2003, USENIX Conference on File and Storage Technologies*, March 2003.

[17] P. Sarkar and K. Voruganti. IP Storage: The Challenge Ahead. In *Proc. of Tenth NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2002.

[18] J. Satran et al. Internet Small Computer Systems Interface (iSCSI). `http://www.ietf.org/rfc/rfc3720.txt` , April 2004.

[19] P. Shivam and J. S. Chase. On the Elusive Benefists of Protocol Offload. In *Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: Experience, Lessons, Implications*, pages 179–184, 2003.

[20] F. Tomonori and O. Masanori. Analisys fo iSCSI Target Software. In *SACSIS (Symposium on Advanced Computing Systems and Infrastructures) 2004*, April 2004. (in Japanese).

[21] S. Yamaguchi, M. Oguchi, and M. Kitsuregawa. Performance Evaluation and Improving of Sequential Storage Access using the iSCSI Protocol in Long-delayed High throughput Network. *DBSJ Letters Vol.2 No.1*, 2003. (in Japanese).

[22] S. Yamaguchi, M. Oguchi, and M. Kitsuregawa. Analysis of iSCSI Storage Access with Short Blocks. In *IEICE the 15th Data Engineering Workshop*, March 2004. (in Japanese).

[23] S. Yamaguchi, M. Oguchi, and M. Kitsuregawa. iSCSI Analysis System and Performance Improvement of Sequential Access. *The IEICE Transactions on Information and Systems(Japanese Edition)*, 87:216–231, February 2004. (in Japanese).