Design and Implementation of an SAN System Based on the Fiber Channel Protocol

Jiwu Shu, Bigang Li, and Weimin Zheng

Abstract—With the increasing demand for vast storage repositories, network storage has become important for mass data storage and processing, telescopic addressing and availability, and the quality of service and security of data storage. This situation demands the emergence of new technology in the data storage field. In this paper, TH-MSNS, a SAN system, is introduced. This system was designed and implemented based on the Fiber Channel Protocol and its I/O route was tested. This paper introduces some of the key techniques in the network storage system, including an SCSI simulating target, intelligent and uniform storage management architecture, and the processing flow of the read/write commands. The software for the new storage area network system was implemented as a module in the kernel mode to improve its efficiency. The SCSI target adopts a layered design and standardized interface, which is compatible with various types of SCSI devices and can use different network protocols. The storage management software adopts distributed architecture, which enables higher interoperability and compatibility with various kinds of management protocols. TH-MSNS boasts characteristics such as high adaptability, high efficiency, high scalability, and high compatibility and is easy to maintain.

Index Terms—Network architecture and design, mass storage, information storage, FCP.

1 Introduction

Mass data storage has become one of the main problems in the development of networks because of a sharp increase in data storage. With a large capacity, high transfer speed, and high system availability, a network storage device can serve for information access and data sharing. In past years, the C/S computational model has been widely used in networks, while, at the same time, information and data have been scattered because every server in this model has its own storage system. This kind of inconvenient architecture may easily lead to a phenomenon known as "information islands," which goes against the aim of information integration and data sharing. As new storage architecture emerges, the storage area network technique provides a solution to the problem of how to achieve information integration and data sharing; it also offers easy manageability and high security. Because the storage area network introduces a network-oriented storage structure and complete separation between data storage and computing, it has many desirable features, such as flexible addressing ability, long-distance transmissibility, high I/O speed, and the ability to share data.

In a topological study of storage networks, Molero et al. analyzed the efficiency and reliability of some topological structures, such as their hierarchy, star, and ring [1]; they also proposed performance models for different topological structures [2]. In their study of network storage protocols, Voruganti and Sarkar made a comparison among three typical protocols in the aspects of efficiency, compatibility, and scalability [3]; they then gave the most appropriate working conditions for each protocol. In design and implementation, Palekar and Russell provided an implementation of network storage systems based on the ISCSI protocol [4], which is modularized in a Linux kernel. Namgoong developed a disk array system, which was based on the Fiber Channel Protocol (FCP), and then evaluated its performance [5]. In the application of storage networks, Milanovic proposed several methods to set up an enterprise network storage system [6].

In this paper, a storage area network (SAN) system, the TH-MSNS (TsingHua Mass Storage Network System) [7], [8], [9] is designed and implemented. The system is based on Linux SCSI and FCP and its storage node has cluster or multiprocessor architecture. The storage nodes in our system can easily be extended in its storage capacity and data processing ability. As the user would demand, our system can extend its storage capacity and data processing ability with a maximum capacity of 45TB for each 16-port FC Switch. It supports multiuser connection and has high throughput. In order to improve its performance, we designed an embedded operating system, working on the storage nodes to make an optimized runtime environment for the software we developed. All of the drivers and our software run in the kernel space with multithread programming technology, which greatly reduces the resource cost. An intelligent and uniform management software was designed, which adopted distributed architecture and took the entire storage system as a set of interrelated, layered objects to set up the object mode of the storage system. The modes are consistent with the CIM/WBEM specifications, so it has high compatibility and portability. The management system provides two access modes, one is the GUI mode and the other is a WEB page mode. Optimized

• *The authors are with the Computer Science and Technology Department, 8-208, East Main Building, Tsinghua University, Beijing 100084, PRC. E-mail: {shujw, Zwm-dcs}@tsinghua.edu.cn, Lbg01@mails.tsinghua.edu.cn.*

Manuscript received 24 Feb. 2004; revised 23 Sept. 2004; accepted 6 Oct. 2004; published online 15 Feb. 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0061-0204.

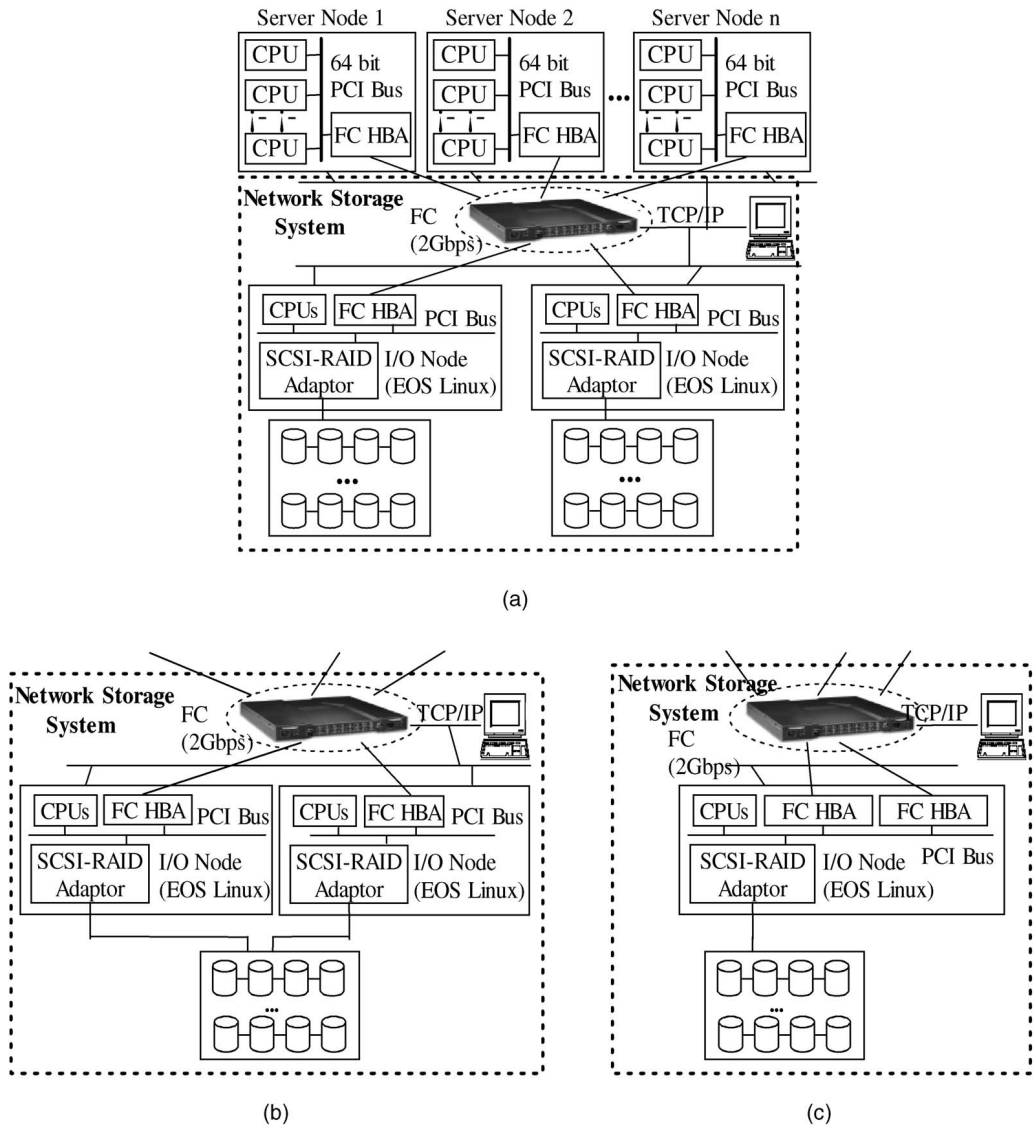

Fig. 1. Architecture of the TH-MSNS system on a Linux SCSI subsystem.

techniques in both hardware and software are used together to optimize the I/O route of network storage and to increase the transfer speed and efficiency.

In contrast with other systems in this area, the TH-MSNS has a unique flexible hardware and software architecture. It can connect to fiber hard disks or SCSI hard disks, forming a network storage system with different functions and different levels of performance. The storage node has an SMP main board, on which multithread I/Os are scheduled to improve I/O efficiency. The design of the storage network software tools of the system, such as a virtualized storage management system and a fault-tolerant system, provides convenient performance at the node level and, thus, the adaptability of the system is improved. The technology of a multientry I/O processor and a cooperative multi-I/O processor is also provided to ensure high availability and high reliability.

A detailed description of the system will be introduced in later sections of the paper. First, we will introduce the hardware configuration and network connection of our storage area network system. Second, the software framework and I/O route are proposed with the key technology and flow description of main commands and the validity of the system is confirmed. Then, the architecture of the intelligent and uniform storage management software is proposed. Third, we draw conclusions from some test results and introduce future work based on this study.

2 The Design of the TH-MSNS

The TH-MSNS is a unique storage area network system, with its storage I/O operation controlled by a software system rather than by hardware, as in a conventional storage area. This avoids the high cost of the hardware and, at the same time, gains the maximum performance and flexibility by the efficiency of the software.

2.1 Hardware Configuration and Structure

Fig. 1 illustrates the hardware configuration and connection structure of the TH-MSNS, which is connected with a

cluster of server nodes by a fiber switch. The hardware architecture of the storage area network system consists of consoles, I/O process nodes, a dense RAID disk array, fiber channel connection devices, and a TCP/IP connection network. Consoles, which act as centers for management and control the storage system, are not involved with the data transfer and storage services. They are used for the management of the storage system. I/O process nodes are the most essential and important part of the system. They provide the function of data access and protocol interpretation, such as the FCP and RAID Protocols. The I/O process node has two kinds of processors: the FCP processors, which manage the data transfer, and the storage processors, which manage the data storage. Each storage system can include multi-I/O process nodes to provide even greater capacity. The connection device connects the I/O process nodes with the consoles. This connection device can be used to increase the scalability. A dense RAID disk array is directly attached to the system via a RAID controller (e.g., a four-channel SCSI-RAID adapter). Our storage system can support up to 15 I/O process nodes, and each I/O process node can support 2-3 dense RAID disk arrays. Each disk array can include at least 30 hard disks, so the total capacity of our system is about 45 TB.

In order to gain high reliability and high availability and to keep an eye on the system load balance, multiple I/O paths are enabled for each storage device in the design of the system. The paths are kept consistent when accessing the storage device by a set of policies. There are two kinds of techniques: One is a cooperative multiple I/O processor technique—see Fig. 1b—and the other is a multientry processor technique—see Fig. 1c. With the multientry processor technique, one or more HBA cards are added to the original I/O processor and, therefore, provide multiple I/O paths for a single processor, as shown in Fig. 1a and Fig. 1b. The multiple I/O paths typically take on parts of the network load and, thus, the bandwidth is effectively expanded. Once one of their I/O paths fails, the remaining paths will provide the access path for all of the data and, so, the SAN availability is improved by reducing the single node failure on the I/O path. With the cooperative multiple I/O processors, multiple I/O processor techniques are applied in a network storage system, not only extending the capacity of the storage system, but also improving the availability and performance by ensuring the cooperation of multiple I/O processors. The I/O processors might connect to the same storage device, providing multiple I/O paths for the device.

2.2 Software Architecture

If classified by the location, the storage area network system includes three kinds of software, as shown in Fig. 2: the storage software in the server node, the storage management software in consoles, and the software in the network storage system. The storage software in the server node consists of the driver of the fiber channel adapter, the SCSI protocol driver, and the storage management agent. The driver of the fiber channel adapter implements storage transfer protocols. It acts as an SCSI client. It encapsulates the SCSI commands from the SCSI mid-level driver, transfers them to the fiber target device, and returns the results. The storage management agent accepts commands from the management software in consoles, executes the commands, and returns the results. The storage system management software consists of four components:

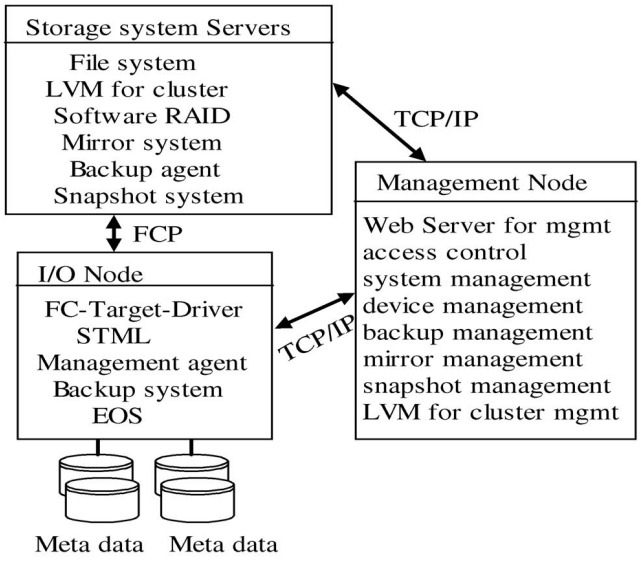

Fig. 2. Software architecture of TH-MSNS.

1. Storage management software based on a Browser/Master/Slave structure.
2. A storage management agent on each server.
3. Storage management interface software in consoles, such as device management, system management, backup management, mirroring management, snapshot management, and virtualized management.
4. An I/O agent in the I/O process nodes. The software in the I/O process nodes is data accessing, Fiber Channel protocol processing, RAID protocol processing, and several others, including a Fiber Channel target driver, an object simulator, an embedded operating system, and function agent software.

The software and hardware architecture described above not only facilitates the expansion and management of the storage capacity, as well as the design of the backup subsystem, the mirror subsystem, and the snapshot subsystem within the storage networks, but also contributes to the design of reliable, high-performance, and special-purpose storage network protocols and application interface systems, and it allows storage nodes of the TH-MSNS system to be connected across a special-purpose network. Compared with other storage area network products, the benefits of the TH-MSNS system can be described as follows:

1. The storage node is implemented on an SMP mainboard, which not only provides the precondition for implementing the multithreaded I/O scheduling and improving the I/O efficiency, but also enhances the processing capability of I/O nodes.
2. A newly designed flexible and specified software/hardware architecture is implemented,which enables the connection of fiber disks and SCSI disks

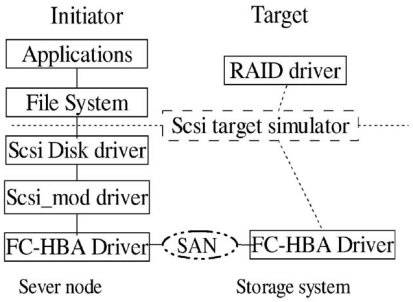

Fig. 3. The I/O route of the storage system.

and the composition of network storage systems with various functions and performances.

3. The software modules of our storage network, such as the storage virtualization management module and the disaster recovery module, were designed on the storage node level. Such a mechanism truly achieves irrelevancy with the operating systems and even the file systems of servers and, thus, greatly enhances the systems' generality.
4. An advanced architecture with a multientry I/O node and cooperative multiple I/O nodes is provided which efficiently enlarges the data accessing bandwidth and enhances the reliability and availability as well.

2.3 I/O Route in TH-MSNS

The key problem with the success of a storage system is the selection and the validation of the I/O route. A special I/O route should be used to ensure that the server node finds such a "local disk" when it searches its hardware, even though such a space has been put somewhere else on the network. The difference here is that SCSI target simulator architecture was adopted to extend the SCSI bus on the network through the Fiber Channel protocol. So, the SCSI target simulator must be able to analyze and process the SCSI commands and messages. In fact, the target simulator in the I/O node must receive the SCSI command from the network and then pass the command to the local SCSI device to be executed and, after that, deliver the result to the front server. The SCSI-RAID function is provided to present multiple SCSI disks as a single RAID disk such as RAID 5, RAID 1, and so on. The I/O path can map the logical RAID block device to the server as the server's local device. Applications on the server node can perform every disk operation on this "local" disk such as FDISK, FORMAT, COPY, and READ/WRITE [10].

Because TH-MSNS refers to the multientry I/O machine and cooperative multiple I/O node machines, its I/O paths are complex. To explain the whole system clearly, we provide an introduction to the most fundamental and important techniques of our system in the following sections, excluding the I/O paths and technologies with the multientry I/O nodes and cooperative multiple I/O nodes.

With the technology described above, the SCSI target can easily implement the network-based I/O route of a storage area network system, as Fig. 3 illustrates. On the server nodes, the users' I/O requests are delivered to the SCSI mid level by the file system. The SCSI mid level transforms these

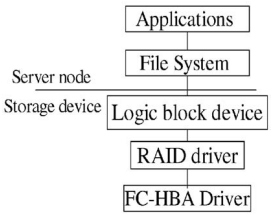

Fig. 4. The I/O virtual route.

I/O requests into SCSI commands and detaches them to I/O storage nodes by using the interface provided by the FC adapter driver. Then, the FCP adapter sends the SCSI commands to I/O storage nodes via a switch or another FC connection device. In this way, to the server node, the FC adapter driver performs the role of a local SCSI disk. Generally, applications on server nodes can send SCSI commands to the SCSI driver as a local SCSI disk. The SCSI driver transfers the commands to the transfer protocol driver. An FC adaptor driver or an I-SCSI driver encapsulates the command into packages and sends the packages to storage nodes through the storage network. The transfer protocol driver on the storage nodes deals with the packages they have received and transmits them to the SCSI target. Then, the SCSI target handles all SCSI commands. In the implementation, we chose the SCSI-RAID subsystem to provide an SCSI disk pool to consolidate storage, which passes SCSI commands on to its firmware to complete the final step of the I/O request [7], [11].

By using a certain mapping method, a connection was established between the logical block device on the storage node and on the server node. In general, although the logical device is a virtual device, it seems to act as the real local storage device of the server nodes and can perform any block device operations. For a more systematic illustration, the I/O route of the storage system is broken down in Fig. 4.

In a traditional I/O system, commands are sent to an SCSI controller; in our system, they are sent to the function *queuecommand( )* of the Qlogic Fibre Channel Adapter driver, encapsulated, and then sent to the FC adapter on the storage node. So, the local I/O route is extended to the fiber channel-based network I/O route.

3 Key Technology in the TH-MSNS

3.1 SCSI Target Simulator

3.1.1 Two-Layered Structure of FETD and STML

The SCSI target is the kernel that implements the network storage. In the I/O storage node, the target receives an SCSI command from the network and then passes the command to the local SCSI device to be executed and, finally, passes the result back to the server. Considering the scalability and standardization of the target and referring to the layered method of the SCSI specification, we divided the target into two layers: the Front End Target Driver (FETD) and the SCSI Target Middle Level (STML) to implement the target. The FETD layer functions as a lower SCSI layer that is responsible for receiving the SCSI command from the network and then passing it to the STML layer. On the other hand, the STML layer is functionally like the SCSI mid-layer, which has the

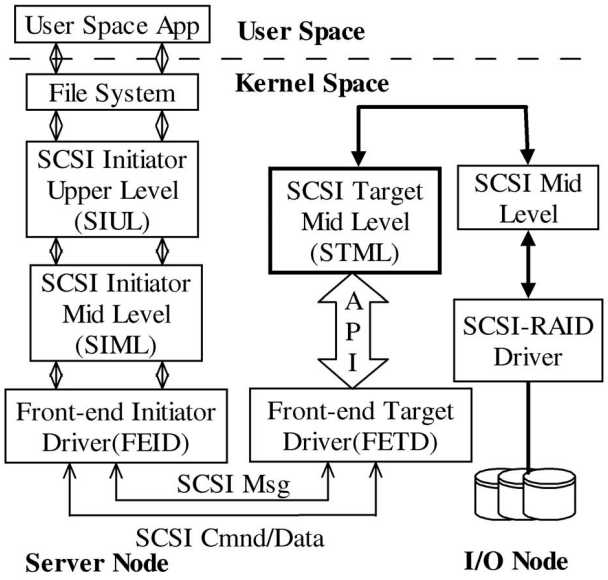

Fig. 5. The I/O path of the two level structure, FETD and STML, based on the TH-MSNS system.

task of receiving the incoming SCSI command and then passing it to the local node to be executed. The details of the two-layer structured TH-MSNS system are shown in Fig. 5.

The FC adapter generates an interrupt signal when it receives a frame. Then, the FETD level uses the interrupt handler to deal with the received frame. If nothing is wrong with the frame, the FETD sends it to the STML and continues to wait for other commands. The STML gives the SCSI COMMAND a unique ID after checking it and then sends responsive information to the FETD. If the received SCSI command is READ/WRITE, management for buffers and exceptions is also needed.

In our implementation, both the FETD and the STML work in kernel mode. When the FETD is loaded into the kernel, it registers to the STML first. The STML includes two threads: the STT (SCSI Target Thread), which performs the interaction with the FETD, and the SPT (SCSI Signal Processing Thread). During registration, the FETD submits an API function table to the STML through which the STML can return responsive data to the Initiator. On the other hand, the STML provides a function interface to the FETD by determining which FETD can transfer SCSI commands and data and activate the STT. When registration is completed, the FETD begins to wait for SCSI commands from the Initiator. Once a command has been received, the FETD sends it to the STML and activates the STT. Then, the STT begins to execute the command. The following description illustrates the process by which a READ command is executed: First, when the STT is ready, it notifies the FETD to read the data and then suspend itself again. Second, when the FETD finishes reading the data, it wakes up the STT. Third, when the STT completes its task, it notifies the FETD. Finally, when the FETD finishes interaction with the Initiator, it notifies the STT to release resources. Thus, the execution of the READ command is accomplished.

Because the interaction between the FETD and the STML is implemented through a standard API function call, the two levels can be designed separately, which greatly helps reduce the complexity. Only the FETD level must be modified to fit different network protocols, while the other parts of the SCSI Target can remain unchanged when different transfer protocols are applied. Generally speaking, the two-level framework not only is compatible with all kinds of servers, but also greatly reduces the cost of SCSI protocols.

The following section demonstrates the main data structure and the FETD and STML's control flow of read/write commands.

3.1.2 The Main Data Structure

The main data structure used in FETD/STML is as follows: The Target_Emulator is a global variable and is also the interface between the FETD and the STML. The template designed as Scsi_Target_Template is the data structure of the FETD/STML interface function and Target_Scsi_Cmnd provides the data structure that describes the SCSI command. The Target_Scsi_Message describes the data structure of the SCSI message state and the Scsi_Target_Device describes the data structure of the SCSI target device. By operating with these data structures, an entire network storage system is constructed.

Target_Emulator is the main data structure and its related variables are as follows:

- Target_Scsi_Cmnd*cmd_queue_start/cmd_queue_end: the global command queue, which stores all the SCSI commands and their states as well as other information.
- Target_Scsi_Message*msgq_start/msgq_end: the global information and command queue, storing all the target management commands and their states as well as other information. For management simplicity, we put the exception process command and management command in this queue for single processing.

In addition to the primary data structure described above, there is also a request queue, a response queue, and an iocb (IO Control Block) queue of the device command queue, as well as other queues for various functions.

3.1.3 Design of the Interface between STML and FETD

The target is divided into two layers in the system's architecture: the FETD layer and the STML (Fig. 5). They connect to each other by some API functions. The main functions of the target are defined as follows:

1. The main functions of the FETD provided by the STML:
   - **int register_target_template (Scsi_Target_Template*)**
   - **int deregister_target_template (Scsi_Target_Template*)**

   These two functions are the register interface of the FETD. The parameter *scsi_target_template* is the most important template and contains the API functions that the FETD provides the STML. By using these functions, the STML can call back the FETD for event handling.

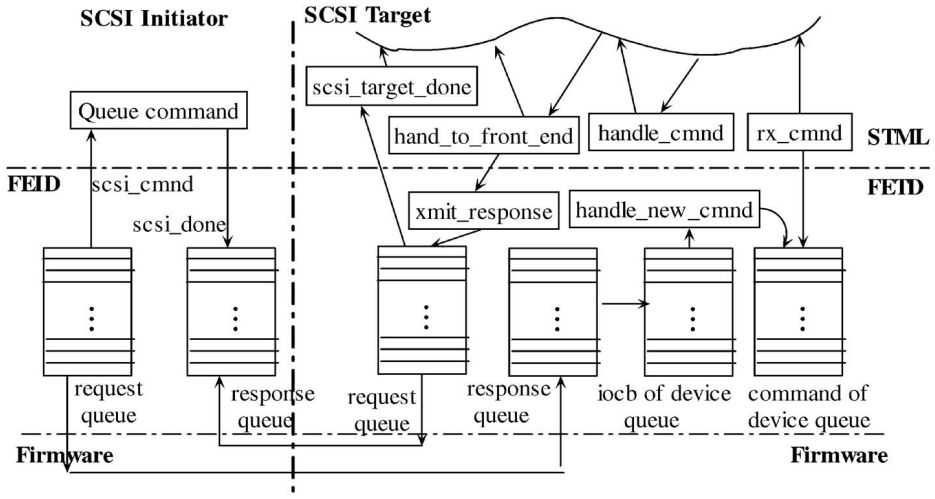

Fig. 6. Processing of a READ type command.

- **Scsi_Target_Device* register_target_front_end (Scsi_Target_Template*)**
- **int deregister_target_front_end (Scsi_Target_Device*)**

These two functions implement the FETD's registration into the STML.

- **Target_Scsi_Cmnd *rx_cmnd (Scsi_Target_Device *device, __u64 target_id, __u64 lun, unsigned char *scsi_cdb, int len)**

The most basic function is called when the FETD receives an SCSI command. It is used to insert an SCSI CDB command into the global SCSI command queue. When the FETD layer receives an SCSI command, it calls this function to deliver the SCSI command to the STML layer.

- **int scsi_rx_data (Target_Scsi_Cmnd *the_command)**

scsi_rx_data is used for the FETD layer to deliver the data to the STML layer. In the write operation, the command and the data are delivered separately. The FETD layer prompts this function to inform the STML layer that the data of *the_command* is ready and the STT thread will be resumed to execute this command.

- **int scsi_target_done (Target_Scsi_Cmnd* the_command)**

The FETD layer alerts this function to indicate that the command execution is over, modifying the command status, resuming the STT thread, and then deleting the command. When the function returns, all the information concerning the command is eliminated.

- **int scsi_release (Target_Scsi_Cmnd*)**

The FETD layer alerts this function when it wants to abort some command midway.

- **Target_Scsi_Message* rx_task_mgmt_fn (Scsi_Target_Device*,int,void*)**

The FETD layer notifies this function to insert the task management command and exception command into the corresponding queue.

2. The functions for the STML provided by the FETD.

   The functions for the STML provided by the FETD are defined in the *Scsi_Target_Template*, referring to the data type: *struct STT*.

3.1.4 Process of Executing the Read/Write Command

The functions for each SCSI command are based on the data structures and functions defined above to achieve network storage. The processing flows for read and write commands are listed as follows:

1. Read command: The processing flow is illustrated in Fig. 6. Read commands are divided into nine steps as follows:
   a. *Scsi mid-level* alerts the interface function *queue-command*, sending the Scsi_Cmnd formatted read command to the initiator. After the initiator driver receives the command, it encapsulates the command into a Command_Entry instance and inserts it into the request queue.
   b. The *iocb* is delivered from the initiator to the target through the network. The target driver alerts the interrupt service function, gets the *iocb* from the response queue, and inserts the *iocb* into the device's *iocb* queue.
   c. The interrupt service function notifies the *process_thread* thread to process the *iocb* queue. If the command is a new one, the function accesses the *handle_new_cmnd* function, sending the *Target_Scsi_Cmnd* as the parameter of function *rx_cmnd* to the STML layer.
   d. The *handle_new_cmnd* function returns and the *process_thread* thread is suspended.
   e. After the STML layer has processed the command, the target returns the struct *Target_Scsi_Cmnd* to FETD and sets the command

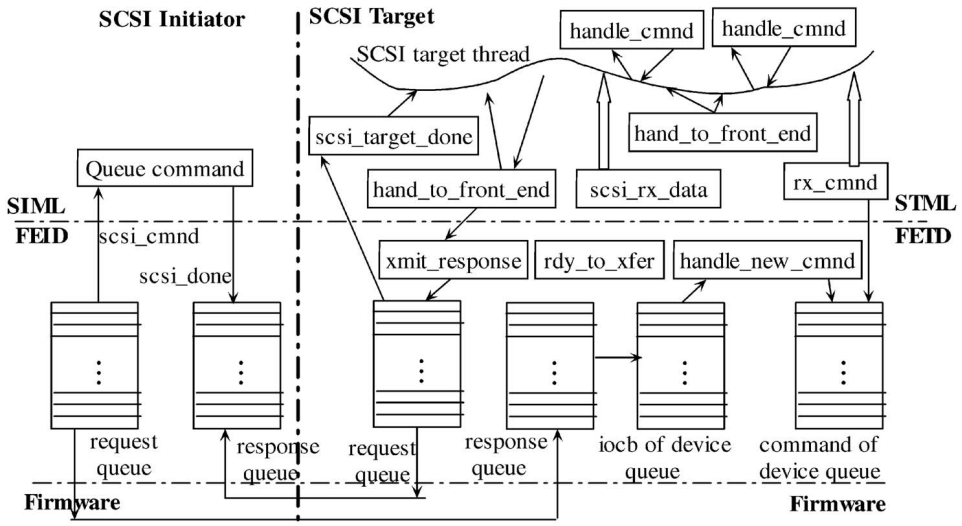

Fig. 7. Processing of a WRITE command.

status to PROCESSED in the device command queue. Then, the target creates an *iocb* instance, including the information that the data has been transmitted, and inserts it into the request queue.

f. The data transmission between the initiator and the target is processed through the firmware.

g. The initiator sends the *confirm* message or the *ACK* message to inform the target that the data is received.

h. After acquiring an asynchronous event by the interruptor, the target sets the command status to DEQUEUE in the device command queue, then initiates the STML interface function *scsi_target_done*. In the *scsi_target_done* function, the *hand_to_front_end* function is prompted to renew an *iocb* instance, including the command-completed status, and insert it into the request queue. At this point, the target's job is over.

i. The initiator's response queue gets the *iocb* instance that was sent by the target and calls the interrupt service function and the mid-level interface function indicated by the *scsi_done* of the *scsi_cmnd* object. At this point, the initiator's job is over.

2. Write command: The processing flow is illustrated in Fig. 7. Write commands are processed in nine steps as follows:

a. *Scsi mid-level* initiates the interface function *queuecommand*, sending the *Scsi_Cmnd* formatted write command to the initiator. After the initiator driver receives the command, it encapsulates the command into a *Command_Entry* instance and inserts it into the request queue.

b. The *iocb* is delivered from the initiator to the target through the network. The target driver accesses the interrupt service function, gets the *iocb* from the response queue, and inserts the *iocb* into the device's *iocb* queue.

c. The interrupt service function notifies the *process_thread* thread to process the *iocb* queue. If the command does not exist, the function initiates the *handle_new_cmnd* function, sending the *Target_Scsi_Cmnd* as the parameter of the *rx_cmnd* function to the STML layer.

d. The *handle_new_cmnd* function returns to the target and the *process_thread* thread is suspended.

e. After the STML layer has processed the command, the target prompts the interface function *rdy_to_xfer* to receive the data. Then, the target renews an *iocb* instance, including the information that informs the initiator that the target is ready to receive the data, and inserts it into the request queue.

f. The data transmission between the initiator and the target is processed through the firmware. The initiator sends the data to the FETD layer. The FETD layer sends the data to the STML layer using the *scsi_rx_data* function.

g. The initiator sends the *confirm* message or the *ACK* message to inform the target that the data has been received.

h. After acquiring an asynchronous event by the interruptor, the target sets the command status to DEQUEUE in the device command queue, then sends the STML interface function *scsi_target_done*. In the *scsi_target_done* function, the *hand_to_front_end* function is prompted to renew an *iocb* instance, including the command-completed status, and inserts it into the request queue. At this point, the target's job is over.

i. The initiator's response queue receives the *iocb* sent by the destination and performs the interrupt service function and the mid-level interface function indicated by the *scsi_done* of the *scsi_cmnd* object. At this point, the initiator's job is over.

The execution of the commands above demonstrates that the design and the implementation of the system is proper.

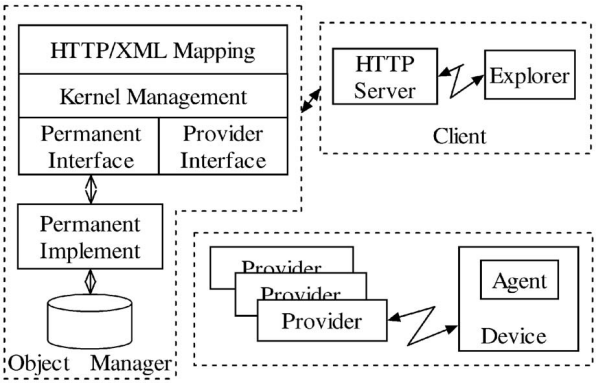

Fig. 8. The framework of the TH-MSNS storage management software.

Both the hardware connection and the software performance have achieved the expected objective.

3.2 Intelligent and Uniform Storage Management Software

The SCSI path and the data path are implemented entirely in the Linux Kernel and, at the same time, the TH-MSNS provided intelligent storage management software systems are at the application level to exchange information with the users. Today, some storage network-based storage management software is neither universal, they are specifically designed for certain storage networks or devices, nor are they standardized, so different management software is not interoperable [12], [13], [14]. In order to standardize system management, the Distributed Management Task Force (DMTF) proposed a management specification proposal of software design: the Common Information Model (CIM) and the Web-Based Enterprise Management (WBEM) Initiative [15], [16], [17]. TH-MSNS management software adopts distributed architecture and consists of five layers, including the browser, HTTP server, object manager, provider, and agent. It implements several management functions, such as object management, device automated discovery, access control, and log functions. The management software is independent of the operating system, meets the CIM/WBEM specifications, is compatible with various kinds of protocols, and has strong interoperability with other management software that meets the CIM/WBEM specifications. The framework of the TH-MSNS storage management software is illustrated in Fig. 8.

In the three components of the storage management software, the object manager is the kernel component. The object manager consists of four modules: the kernel manager, the provider interface, the solid storage module, and the HTTP/XML mapping modules.

From the view of the storage management software, the entire environment that is managed consists of a set of interrelated objects and their definition and management is the kernel manager's main task. In the TH-MSNS system, the CIM mode is used to define and manage the basic class of the various related objects. When the users apply the management software for a certain storage network, various managed objects can derive from the CIM basic class; thus, the various kinds of management software can be aware of each other. By determining the managed objects and their attributes and building associations among the objects, the whole intelligent storage system is abstracted as a set of interrelated, layered objects and, then, the object mode of the intelligent storage system is constructed. In addition to the object management function, the function of the kernel manager is also expanded and some other functions are also added to the system, including access control, log, report, warning, and automated discovery of the device connected to the intelligent storage system. With the intelligent management system, the TH-MSNS system is easier to manage and maintain. The intelligent storage network consists of a network link device, a storage device, and management software. In contrast to traditional storage management software, the TH-MSNS system provides two kinds of management modes: the GUI and the WEB page mode. The management interface is able to manage the storage resources, Fiber Channel card, Fiber Channel Switch, and SCSI-RAID card uniformly and easily. Through the Web service, the administrator can directly manage the storage resources and monitor the storage system on a common console. The storage software regards the device on the storage network as its own managed object and, thus, the system meets the CIM/WBEM specifications, is easy to transplant, and is compatible with other systems.

TABLE 1
Main Configuration of the System Test Environment

Server TH-Cluster	Processor	4×Xeon700MHz
	Memory	1GB
	Fiber Channel card	Emulex LP982
	Operating system	Linux 2.4.18-5smp
	Fiber Channel switch	Brocade Silkworm 3200
I/O processor	processor	2×Xeon2GHz
	memory	1GB
	Fiber Channel card	Qlogic qla2310F
	SCSI-RAID card	Adapter 2110s
	Operating system	Linux 2.4.18-3smp
	RAID disk array	14× 73G(10Krpm) SCSI Disks

4 Results of Testing and Analysis of the TH-MSNS

From the Fiber Channel-based TH-MSNS system implemented above, we constructed a server system, the TH-Cluster, for which a uniform data storage service was provided. Its performance in reading or writing data was tested by the TH-MSNS system.

The configuration of the system test environment is illustrated in Table 1.

We used an Intel IOmeteor 2003 to test the performance of the TH-MSNS. The testing data blocks were 4KB, 16KB, 32KB, 64KB, 128KB, and 512KB. The sequential read/write operations and the random I/O operations (read operations

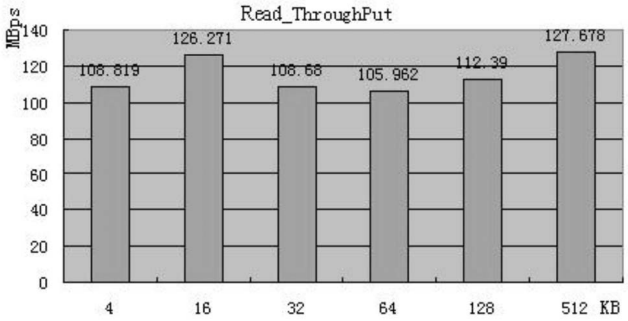

Fig. 9. Throughput results for sequential read operations.

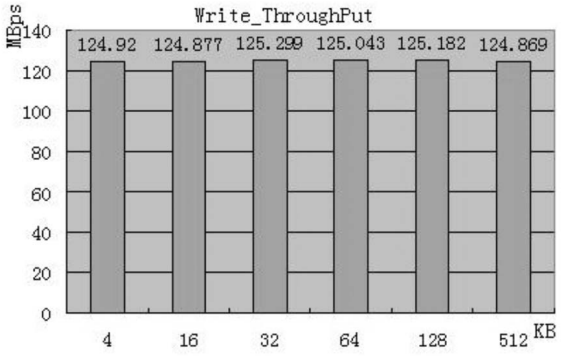

Fig. 10. Throughput results for sequential write operations.

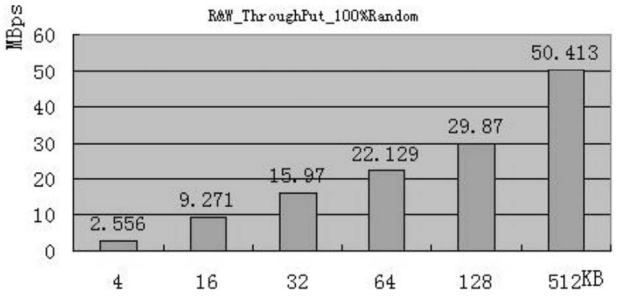

Fig. 11. Throughput result for random read/write operations.

constituted 80 percent of all operations) were generated by six 32-bit servers. Figs. 9, 10, 11, 12, and 13 show the throughput, I/O average response time, and the CPU utilization of our system.

Fig. 9 and Fig. 10 show the sequential read/write performance changes slightly with varying increments of the transfer size. The read throughput reaches more than 100MB/s and the write throughput exceeds 120MB/s and the best performance reaches 125.299MB/s. With the increase of the transfer size, the random read/write throughput continues to increase until it reaches 50.413MB/s, and the latency increases slightly. The CPU utilization changed slightly during the experiments, but it never reached 10 percent. These results show that the system performance is perfect.

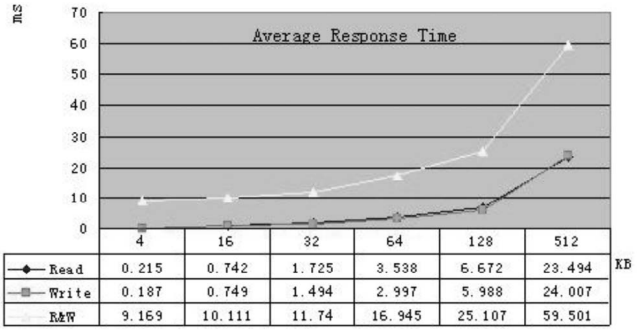

Fig. 12. The average response time.

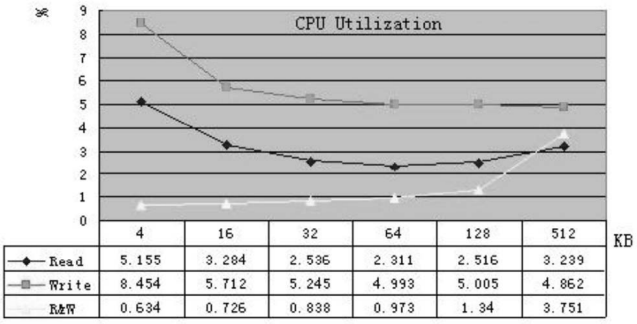

Fig. 13 CPU utilization.

5 Conclusions and Future Work

This study included the implantation of a SAN based on the FCP protocol. The system has the functions that a standard SAN would have, such as effective data backup and recovery, data sharing and virtualization, standard software for management, and support for various platforms. Because of the limitation of the length of this paper, these functions and their implementation are introduced in another paper [18].

Comparing this system with similar research and products, our system proved to have the following characteristics:

1. The software design introduces layer-based software architecture. This simplifies the system's ability to be extended for different SCSI devices and network communication protocols and makes the system flexible.
2. All of the software is run on the OS's kernel space, implemented by a core module. Thus, the number of memory copies between the user level and the kernel level is cut down, increasing the system's efficiency.
3. The system follows the mainstream protocol criterion. It can be connected and extended easily.
4. The nodes processing the I/O operations employ an SMP processor and several FC adaptors can be plugged in. Therefore, to increase the efficiency of transmission and I/O throughput, multithread technology should be imposed. A parallel scheduler and the optimization of I/O routes in an environment with many initiators and many targets are areas that require further research.

ACKNOWLEDGMENTS

The work described in this paper was supported by the National Natural Science Foundation of China (No. 60473101 and No. 60433040) and the National High-Tech Research and Develop Program of China (No. 2004AA111120). In addition, the authors would like to express their deep appreciation to Professor Wen Dongchan, Doctor Fu Changdong, Yao Jun, Liu Tianmiao, Wu Hao, and Pan Jiaming. As members of this projects team, the authors are grateful for their contributions to the work for the goal of the group and they are grateful for their contributions to the work.

REFERENCES

[1] X. Molero, F. Sills, V. Santonja, and J. Duato, "On the Interconnection Topology for Storage Area Networks," *Proc. 15th Int'l Parallel and Distributed Processing Symp.*, pp. 1648-1656, 2001.
[2] X. Molero, F. Silla, V. Santonja, and J. Duato, "Modeling and Simulation of Storage Networks," *Proc. Eighth Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems*, pp. 307-314, 2000.
[3] K. Voruganti and P. Sarkar, "An Analysis of Three Gigabit Networking Protocols for Storage Area Networks," *Proc. 20th IEEE Int'l Performance, Computing, and Communications Conf. (IPCCC)*, pp. 259-265, Apr. 2001.
[4] A. Palekar and R.D. Russell, "Design and Implementation of a SCSI Target for Storage Area Networks," Technical Report 01-06 Computer Science Dept., Univ. of New Hampshire, Durham, May 2001.
[5] J.-C. Namgoong, "Design and Implementation of a Fibre Channel Network Driver for SAN-Attached RAID Controllers," *Proc. Eighth Int'l Conf. Parallel and Distributed Systems (ICPADS 2001)*, pp. 477-483, 2001.
[6] S. Milanovic, "Building the Enterprise-Wide Storage Area Network," *Proc. EUROCON 2001, IEEE Int'l Conf. Trends in Comm.*, vol. 1, pp. 136-139, July 2001, http://www.ktl.elf.stuba.sk/EUROCON/program.htm.
[7] J. Shu, J. Yao, C. Fu, and W. Zheng, "A Highly Efficient FC-SAN Based on Load Stream," *Proc. Fifth Int'l Workshop Advanced Parallel Processing Technologies (APPT2003)*, X. Zhou, S. Jahnichen, M. Xu, and J. Cao, eds., pp. 31-40, Sept. 2003.
[8] C. Fu, J. Shu, and W. Zheng, "Intelligent Network FC-Disk Based on Autonomic Computing," *Proc. Int'l Conf. Software, Telecomm., and Computer Networks*, Oct. 2003.
[9] H. Wu, J. Shu, D. Wen, and W. Zheng, "Design and Implementation of a Fibre Channel Target Driver Supporting SCSI," *J. Tsinghua Science and Technology*, accepted, http://storage.cs.tsinghua.edu.cn, 2004.
[10] R.K. Khattar, M.S. Murphy, G.J. Tarella, and K.E. Nystrom, *Introduction to Storage Area Networks, SAN*, http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245470.pdf, Aug. 1999.
[11] Working Draft T10/Project 1561-D/Rev 10, Information Technology, "The SCSI Architecture Model-3 SAM-3)," Nov. 2002.
[12] "Viewing Your SAN with SANpoint Control," Aug. 2002, http://www.veritas.com/put/support/products/SANpoint.
[13] "Introduction to EMC Control Center/Open Edition," Mar. 2002, http://www.emc.com/pdf/products/controlcenter/ecc_ve_intro.pdf.
[14] "Tivoli Storage Network Manager," July 2001, http://www.tivoli.com/products/docments/datasheets/storage_network_mgr.pdf.
[15] "Specification for the Representation of CIM in XML, Version 2.1," May 2002, http://www.dmtf.org/standards/documents/WBEM/DSP201.html.
[16] "Specification for CIM Operations over HTTP, V.1.1," May 2002, http://www.dmtf.org/standards/documents/WBEM/DSP200.html.
[17] "CIM Schema Version 2.6.0," June 2002, http://www.dmtf.org/standards/documents/CIM/CIM_Schema26/schema26_mofs.zip.
[18] J. Yao, J. Shu, R. Yan, and W. Zheng, "Design and Implementation of Midwares for Disaster Tolerance on TH-MSNS," *Proc. China Nat'l Computer Congress (CNCC '03)*, Z. Weimin, ed., pp. 528-535, http://storage.cs.tsinghua.edu.cn, 2003.

Jiwu Shu graduated from the Computer Department at the National Defense University of Science and Technology in 1991 and received the PhD degree in computer science from Nanjing University in 1998. In 2000, he finished his postdoctoral position research at Tsinghua University and has been teaching at Tsinghua University since then. He is now an associate professor at the Institute of High Performance Computing Technology in the Department of Computer Science and Technology at Tsinghua University. His current research interests include storage area networks, parallel and distributed computing and networking, algorithm analysis and design and parallel processing techniques, and cluster systems and communication.

Bigang Li received the master's degree in 2003 and is now a doctoral student at the Institute of High Performance Computing Technology of Tsinghua University's Department of Computer Science. His current research interests include storage area networks, safety data storage, and parallel and distributed computing and networking.

Weimin Zheng received the master's degree from Tsinghua University in 1982. He is the director of the Institute of High Performance Computing Technology in the Department of Computer Science and Technology at Tsinghua University, China. Since 1982, he has been working at Tsinghua University in the area of parallel and distributed processing. His research covers parallel computer architecture, parallel and distributed computing, AI-oriented computer architecture, compiler techniques, and runtime system design for parallel processing systems and grid computing and network storage.