International Journal of Networked and Distributed Computing, Vol. 3, No. 1 (January 2015), 42-50

# A Distributed Storage System with Dynamic Tiering for iSCSI Environment

**Atsushi Nunome [1] , Hiroaki Hirata [1] , Kiyoshi Shibayama [2]**

[1] *Department of Information Science, Kyoto Institute of Technology, Matsugasaki Sakyo-ku, Kyoto, 606-8585, Japan*

*E-mail: {nunome, hrt}@kit.ac.jp*

[2] *Department of Information Science, Kyoto Institute of Technology, Matsugasaki Sakyo-ku, Kyoto, 606-8585, Japan*

*E-mail: shibayam@kit.jp*

**Abstract**

We propose a distributed storage system which relocates data blocks autonomously among the storage nodes. In order to optimize I/O performance with slight administrative workload, our system is aimed at the realization of the following two functions; (i) Run-time construction of the storage tiers from the heterogeneous storage devices, and (ii) Automated block migration among the tiers. We also show a technique to reduce management traffic in iSCSI environment.

*Keywords:* Distributed Storage System, Block Migration, Storage Tiering, iSCSI

## 1. Introduction

Recently, even a client computer has enough storage space owing to inexpensive mass storage devices. In small organizations such as a small office or a laboratory in a university, a client computer is used for various tasks as a single node, and it is often not configured to cooperate with other client computers. Such a computer is apt to remain lots of free space on the storage device. Therefore, the storage devices on the client computers should be organized as a distributed storage system in order to be utilized effectively.

Storage management is one of the most important tasks of system administrators. They have to monitor the performance of the devices whether they are operating at expected I/O performance. Access concentration or device overload will cause performance degradation of the whole system. The data files which are seldom accessed by the user in spite of locating on the high performance storage should be migrated to another storage device.

However, this relocation task is a heavy burden on the administrators because file access pattern and storage workload are dynamically changed. If they statically decide the location of a file and never relocate it, the system may not be able to utilize the potential performance of the storage devices. On the other hand, excessive dynamic tuning brings considerable overhead.

We propose a distributed storage system which migrates file data blocks into the suitable storage device for their access pattern.

Published by Atlantis Press

## 2. Related Works

Virtualization generally helps to utilize various types of storage devices on the basis of a single principle. The modern file system has ability of storage virtualization. ZFS[1] file system which has been originally designed by Sun Microsystems is widely used for virtualizing storage. The virtualized storage which is called a *storage pool* may be composed of multiple storage devices. In ZFS, an administrator can manually attach a high performance storage, such as an SSD, as a cache or a log device to the storage pool. Once it is attached to the storage pool as such functions, it can not be used for data storage. Thus the purpose of the storage device is statically fixed. Also, even if the storage pool is composed of several types of devices and there is a large performance gap between the heterogeneous devices, we can not explicitly designate the location for storing a file.

Lustre[2] file system is one of the cluster file system, and it consists of the following two types of servers; the meta data server (MDS) and the object storage server (OSS). The former manages the metadata of the files, and the latter provides the contents of the files. Actual storage devices are connected to the OSS, and the physical devices are hidden from Lustre clients. Although many storage devices can be used through multiple OSSs in the network, the administrators have to be aware of the existence of the performance differences between the storage devices. Thus, they might need to adjust the block allocation manually.

The simplest way to manage multiple storage devices is to unify the type of all of them. Recently, the combination of different types of storage device (*storage tiering*) is widely used in the environment where used various types of storage devices. It aims at optimizing the balance of cost and performance.

*Easy Tier*[3] and *FAST* (Fully Automated Storage Tiering)[4] are commercial storage tiering systems implemented for storage server products. They support up to three storage tiers which are classified by device technologies. Because no effective performance evaluation is needed, administrators can easily assign the storage devices to the appropriate storage tiers. If there are large performance gap between the devices in the same tier, the actual effect of block migration is difficult to estimate.

Btier,[5] formerly named *tier*, is a block device with automatic migration for Linux kernel. It scans storage devices in the statically defined interval in order to find the blocks which should be migrated to another tier. The number of tiers can be controlled in accord with performance differences in the storage devices. However, too many tiers bother an administrator for configuring migration thresholds. Actually, most systems use only two tiers such as an SSD tier and an HDD tier. Because btier is designed for a stand-alone Linux box, the block migration over the network is not supported.

For network environment, *autonomous disks*[6] is proposed as clusters of disks, which aims at achieving of data distribution, fault tolerance, and heterogeneity. Although autonomous disks itself does not provide the function of storage tiering, an attempt to combine an autonomous disk cluster of magnetic disks (HDDs) with a solid state autonomous disk cluster has been proposed.[7,8] The system uses the solid state cluster as a cache of the magnetic disk cluster, and does not perform block migration.

In conventional tiering techniques, the storage devices are statically classified according to their theoretical performance or device technologies. Although such static classification helps to reduce the run-time overhead, it cannot respond to fluctuation of effective I/O performance. Because of a high level of the management overhead, it is hard to decide dynamically the storage tiers over the network.

## 3. System Assumptions

In this paper, we assume the following system as a main target to examine.

First, the target system is assumed to be constructed with heterogeneous components. This heterogeneity is not the system's intention, but a result of components' partial replacement. Hence, the performance of nodes and storage devices might vary widely. Since we assume that storage servers and client PCs coexist on the same network. So, they can directly transfer network frames to destination nodes.

Furthermore, the storage server could simultane-

ously act as a client for the other server. In this paper, we classify network nodes in this environment into the following three types.

**Storage node** The node which has a storage device and shares it with clients.

**Client node** The node which mounts a remote storage to use.

**Bi-functional node** The node which has a public storage device and also accesses another remote storage.

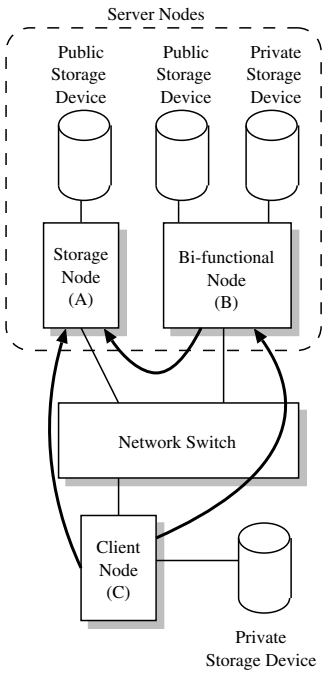


Fig. 1. An example of network nodes' classification.

An example of network nodes' classification is shown in Fig. 1. In this figure, three network nodes (A), (B), and (C) are connected via a network switch. Both the storage node (A) and the bi-functional node (B) attach their own storage device to share. Hereinafter, such the node which provides public storage service is also referred to as a *server node*. The client node (C) without shared storage device accesses two server nodes over the network. The node (B) accesses the remote storage on the node (A) as the role of a client.

There are situations of which a server node not only performs I/O operation but also runs some kind of application software. For example, some of recent NAS products also provide various network services, such as database management and media streaming with dynamic transcoding. Because each node in an assumed environment could independently use various software or network services, such a complex task brings unbalanced I/O workload. Thus the apparent throughput and latency of the nodes will fluctuate continuously. The target of our scheme is such environment which is composed of nodes with dynamical heterogeneity. This situation is ordinarily observed in environments where obsolete nodes are gradually replaced with the new.

The free space of a storage device is provided as a shared block device. Other nodes which need more storage area use the device over the network. The client node constructs an arbitrary file system on it. If a server node has sufficient free area on multiple storage devices, it first integrates their devices as a logical storage. After that the server node shares the logical storage for client nodes with iSCSI.[9] iSCSI is one of the major protocol to use in storage area network (SAN). In iSCSI, a SCSI command block is encapsulated within an IP packet, and it is sent over ordinary IP networks. The node which provides storage is called an *iSCSI target*, and the node which connects to the remote storage is called an *iSCSI initiator*.

Modern operating systems, such as Windows or Unix-like systems, have ability to be the iSCSI initiator. Because the iSCSI target simply provides a raw storage device, the iSCSI initiator can construct and use arbitrary file system independent of the target's OS. Therefore, our scheme targets the environment where iSCSI is used as the protocol for accessing the storage in order to make efficient use of free storage.

## 4. Dynamic Storage Tiering Scheme

In this section, we describe the details of a scheme to dynamically construct storage tiers and migrate data blocks across the tiers.

### *4.1. Overview*

In our scheme, each server node scans data blocks for migrating to the appropriate storage. Data blocks which are frequently used are preferentially located on high performance storage devices (the *upper storage tier*). On the other hand, data blocks which are seldom used are gathered on relatively poor performance storage devices (the *lower storage tier*). This makes a contribution to maintain enough size of free area on a fast storage device for frequently used data blocks.

The destination node for block migration is decided in consideration of the run-time performance gap between the server nodes. A maximum I/O throughput described in the specification sheet does not reflect the actual transfer speed. In our scheme, the server nodes relocate data blocks based on the dynamic aspects.

In order to obtain the dynamic aspects of the server nodes, we propose the method to improve utilization of packet payload for iSCSI protocol. Short length packets are exchanged frequently in iSCSI protocol. On the other hand, in order to improve efficiency of transmission and reduce overhead, the frame size tends to be long in recent network standards. Even a short frame costs constant time for processing its header on the network nodes or switching equipment. Hence, we propose the scheme which merges some small data into one frame in order to utilize the space in the iSCSI packet.

### *4.2. Migration strategy*

The server nodes perform an actual migration when I/O workload is estimated to be low. Because there is no explicit migration initiator in the fully decentralized environment, the inquiry for starting the migration may occur continuously. There is a possibility that random and complex inter-node communications cause temporary network congestion. Accordingly, each server node exchanges information about storage utilization in a controlled manner.

There are two points of examination to determine a destination of data blocks. One is the size of the free storage area of the destination, the other is the effective data transfer rate of the destination node. The concentration of frequently used data blocks on a specific server node might degrade its effective data transfer rate. However, it is possible to avoid such problems by relocating the data blocks onto the different nodes in order to prevent the simultaneous access. Accordingly, our scheme exchanges the status of each node using iSCSI messages which are sent at frequent intervals.

In our scheme, an iSCSI target is responsible for block migration. Generally, the migration can be initiated in the following two manners. One is receiver initiation, the other is sender initiation. In case of the former, idle server nodes randomly seek for data blocks which should be received. This task tends to obstruct proper I/O operations on the network. The server node which performs heavy I/O transactions knows which data block should be migrated to more powerful server node. Furthermore, it is also sufficiently functional in case to begin migration of a seldom accessed data block when the need for storage space arises. Consequently we choose the latter as a suitable policy for block migration.

### *4.3. Storage information*

The migration initiator determines the destination on the basis of information about other server nodes. This information (hereinafter referred to as *storage information*) is actually the state of block utilization on each server node. However, keeping track of all data blocks will be a hard task for the nodes. For that reason, we define a bunch of several data blocks as a *migration unit* which has fixed length of data. It is made so as to contain continuous data blocks which belong to the same file as far as possible. In order to manage the utilization of each migration unit, the server node uses a table named a *migration management table* (MMT). The fields of each MMT entry are the time of the last migration and a degree of migration priority. The MMT is located on the local storage area which is removed from the list of blocks to be migrated.

The number of blocks included in the migration unit should be defined by considering the sender's overhead. Because of the performance differences among the server nodes, the number of blocks in the

migration unit will not be constant over the whole system.

While larger size of migration unit helps reducing overhead, there is a strong possibility that data blocks of different files coexist on the same unit.

The storage information includes the following data.

(i) Storage ID:
This is used for identifying the storage device. In a simple way, it can be set same as iSCSI qualified name[9] which has the maximum length of 223 bytes. This ID is unique in the whole system.

(ii) The amount of free storage in megabytes:
This is used for notification of how much data the server node can receive. If the value is zero, the node will reject incoming migration units. When this field has two bytes of data width, it can express the size of free storage up to 64 gigabytes. If it is insufficient to represent the actual size of free area, the maximum number can be used for that.

(iii) A degree of access concentration:
This is used for decision whether the storage device is in overloaded condition. Each server node compares these values which are collected from other server nodes. If a server node finds that it is relatively in overloaded, it begins to migrate some of the blocks which are accessed frequently.

If the third field is represented in four bytes, each storage information can be composed in 230 bytes.

This storage information should be exchanged frequently in order to observe the precise condition of other server nodes. Therefore, it has to be exchanged efficiently for reducing overhead.

### *4.4. Storage information exchange scheme*

In order to migrate a data block onto proper node, each node should collect fresh information about candidate destinations. We propose the method for reducing the number of individual packets which deliver storage information.

A network node analyzes the header of every incoming packet for the purpose of accepting it. Therefore, a number of short packet cause great processing overhead of analysis. Our scheme merges several short packets into one long packet in order to reduce such overhead. We attempt to utilize lots of short packets which are exchanged in iSCSI protocol.

iSCSI is the protocol which provides the capability of transferring SCSI command and data over IP networks. In traditional SCSI, a command is sent in a *command descriptor block* (CDB).[10] In iSCSI, it is transferred as a part of an iSCSI *protocol data unit* (PDU). The most CDB has short length as compared with a network frame. For example, the so-called *Jumbo Frame*[11,12] which has the capacity of payload over 1,500 octets is often used in recent high-speed Ethernet. *IP over InfiniBand* also supports larger size of payload than traditional Ethernet.[13,14]

Because there are great differences between the actual size of the SCSI CDB and the transfer capacity of the network frame, the iSCSI packet still has the space to attach some useful information. In the SAN using iSCSI protocol, a large number of small packets are observed between an iSCSI initiator and target. In general, the initiator and target send a message to each other in order to verify whether or not its counterpart is alive. This ping-like mechanism using a pair of NOP-Out request and NOP-In response is defined in RFC.[9] An iSCSI initiator sends the NOP-Out request every tens of seconds.

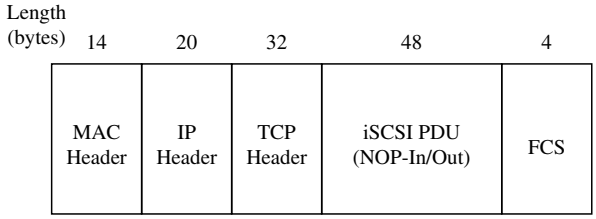


Fig. 2. An Ethernet frame structure of iSCSI NOP-In/Out.

As an example, we present the actual structure of the network frame. Fig. 2 shows an Ethernet frame structure of the iSCSI NOP-In or NOP-Out. The frame is constructed with 48 bytes of iSCSI PDU, 32 bytes of TCP header including optional field, 20 bytes of IP header, 14 bytes of media ac-

cess control (MAC) header, and trailing four bytes of frame check sequence (FCS). The total length of the frame reaches 118 bytes. The part of the frame except MAC header and FCS is included in Ethernet payload. Because the maximum transmission unit (MTU) of Jumbo Frame reaches 9,000 bytes or more, it is able to contain not only one NOP-In/Out PDU but also more additional information. In this way, our scheme uses the space of the iSCSI packet to exchange the storage information.

The storage information is appended to the iSCSI NOP packets by the *extended TCP layer* (hereinafter referred to as ExTCP). The ExTCP layer observes a communication of iSCSI. When a NOP PDU is about to be sent to the destination, the layer appends the appropriate amount of the storage information to the iSCSI NOP PDU according to the MTU size.

As mentioned above, the length of the normal iSCSI NOP packet is 100 bytes. For example, in the environment using 9,000 bytes of MTU, the remain payload of iSCSI NOP can contain 38 pieces of the storage information. Hereinafter, the NOP packet which is appended such storage information is referred to as the *extended NOP packet*. The storage information is stripped off from the extended NOP packet by the ExTCP layer on the receiving node.

The storage information which is collected by the extended NOP packets is stored into an on-memory table by each iSCSI node. The ExTCP layer picks a certain number of the storage information from the table for making the extended NOP packet. Therefore, each iSCSI node can obtain information about the node with which it has relations by the iSCSI sessions.

### *4.5. Block migration procedure*

Each server node can acquire information about the state of block utilization on neighbor server nodes by the scheme described in the previous section. It periodically checks whether there are data blocks which have been accessed consecutively. When it detects intensive access to a particular data block, it marks the migration unit which includes the block for relocating later.

The *migration reservation queue* (MRQ) is located on every server node for management of future migration. The MRQ is implemented as a priority queue and it stores information about which migration unit should be migrated preferentially. Hereinafter, the priority of the MRQ is referred to as a *migration priority*.

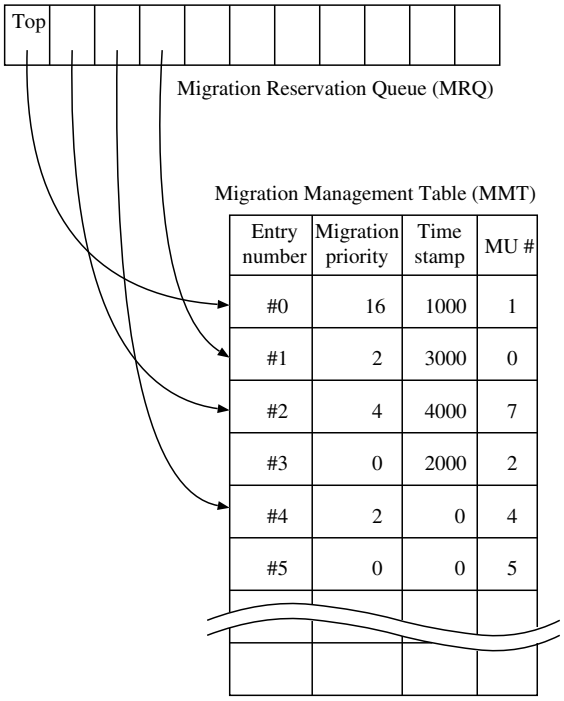


Fig. 3. An example of the migration reservation queue.

An example of the MRQ is shown in Fig. 3. Each entry of the MRQ points a record of the MMT. This example shows four entries queued in the MRQ. The top entry of the MRQ points the entry #0 which has the highest migration priority in the MMT. In this MMT, both the entry #1 and #4 have the same migration priority. To prevent contiguous migration, the entry #4, which has longer elapsed time than the entry #1 since the last migration, is given priority over the entry #1 in the MRQ. The MMT entry #3 and #5 are not pointed from the MRQ because they have no migration priority.

Because the migration priority must be calculated for every migration units, it should be found by simple and easy way. If there is no concurrent access to a storage device even if the number of ac-

cess to the device per a unit time is high, degradation of the I/O performance will not occur. Therefore, we define the migration priority as the influence on the other accesses to the same device, not a simple total of the access. In concrete terms, each server node calculates the average queue length during the access to a data block. At both the time of receiving and completing the access request, the server node logs the number of requests waiting in the queue. The average queue length can be found by the simple average of the two numbers. If this value is zero, it indicates that the access to the data block does not disturb any other accesses. Each server node sums up these values per data block, and considers the maximum value in the migration unit as the migration priority. This indicator, the migration priority, is utilized for decision of the migration unit to be transferred when it is necessary to migrate some data blocks from the server node.

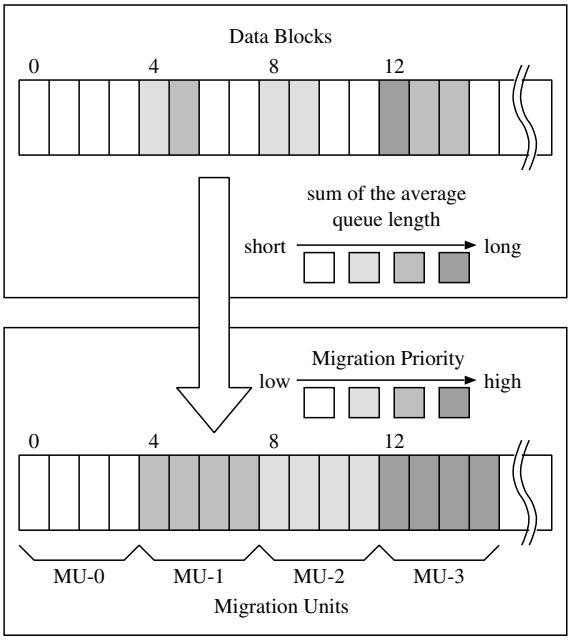


Fig. 4. An example of getting the migration priority.

Fig. 4 shows an example of getting the migration priority of the migration units. The upper side of this figure shows the sum of the average queue length per data block. Darker block indicates that the sum of length is long, and should be migrated to faster storage device. The data block number 12 has the longest sum of the average queue length in this figure. This means that a number of access has been disturbed by the access to the block number 12.

Each migration unit consists of four data blocks in this example. The lower side of Fig. 4 shows four migration units from MU-0 to MU-3. Each of them is composed of four data blocks. The migration priority of the fourth migration unit (MU-3) is represented by the sum of the average queue length of the block number 12. In this example, the server node arranges the pointer to the migration unit in the MRQ by the migration priority, in the order of MU-3, MU-1, MU-2 and MU-0.

The server node periodically evaluates a degree of access concentration for decision whether some blocks should be migrated or not. The degree of access concentration is calculated from the average time required to complete access requests. The time for waiting access completion is logged for every request in order to obtain the average completion time.

When the server node takes the longest average completion time and the time exceeds a threshold, it becomes the migration initiator. This means that relatively the storage device has been exposed to intense access or it has poor I/O performance, i.e., insufficient throughput or high latency. Hence, we consider the node which has long average completion time as the member of the lower storage tier in spite of its potential performance. The upper storage tier is also defined in the same way.

The migration initiator begins to transfer the blocks included in the migration unit which is located at the top of the MRQ. The server sends the blocks included in the migration unit as a unit of transfer. However, there is no guarantee that the effect of the migration appears without delay. If a server node performs contiguous block migrations for the reason that the effect has not been clear, a number of useless migration might obstruct other network traffic. Therefore, the block migration should be performed in sufficient intervals in order to prevent burst migration.

On the other hand, when the size of the free storage area is below the threshold defined statically,

the server node picks up the infrequently accessed blocks and recognizes them as the candidacy of migration. The migration initiator looks for the migration units which have not been accessed in a certain period of time. We prepare two threshold levels in order to prevent contiguous migration. The threshold levels are based on the amount of free storage, and used for the determination of the beginning and the end of the migration. When the size of free space is below the lower threshold, the server node starts the block migration to the lower storage tier. It continues until the size of free space reaches the upper threshold.

Since our method is fully decentralized, no specific node manages imbalance of I/O workload in the whole system. Therefore, each migration is performed based on local information only. Because each server node begins the block migration in order to mitigate the performance degradation accompanying access concentration to a storage device, the effect of the migration cannot be confirmed immediately. As the migration effect appears after a certain time period, drastic block migration should be avoided.

In order to prevent excessive migration, a server node initiates the migration procedure only when enough performance gap is observed between the server node and the destination node. As the result of the aforementioned migration procedure, storage tiers are organized dynamically. The number of storage tiers is decided upon by the performance differences among the nodes in the system.

### *4.6. Accessing to migrated blocks*

To shorten the time required for the block migration, the scheduling priority of the migration should be set at highest. When a client node attempts to read a block which is being migrated, the migration source node continues the process and transfers a required data block to the client node from its own storage device. On the other hand, when a migration source node receives a request to write data onto a migrating block, it first aborts the migration and then restarts it from the beginning after completion of the write operation.

After the block migration, the migration source node keeps the original data blocks. If the node receives a request to read or write the block and it has been already updated by the destination node, the migration source node discards the original data block and forwards the request to the destination node.

If the migrated block has not been updated yet, the migration source node which is received the request follows following procedure. When the migration source node receives a read request, it replies the requested block immediately. The reply message also contains information about the destination node of the migrated data block. After that, the client node directly sends a request for access to the migration destination node. When the migration source node receives a request to write the migrated block, it first updates the original block and then it re-transmits the updated block to the destination node. The migration source node notifies the client node about the migration destination node after updating the data block. Afterward the client accesses directly to the destination for the block. After updating the migrated block, the migration destination node notifies the source node to invalidate the migrated source block.

## 5. Conclusion

In this paper, we proposed a distributed storage system which provides an automated block migration mechanism.

Our system aims to utilize various types of storage devices effectively in a network of heterogeneous computers. Detailed and fresh information about the devices is needed for appropriate data block migration. However, exchanging such information in short interval generally causes unignorable overhead. In order to reduce the overhead, we show a scheme to append such information on periodic iSCSI communication. Our system configures dynamically storage tiers according to effective performance of the device. Data blocks are migrated among the tiers.

We plan to simulate our system in order to fix parameters and thresholds. Afterward, we would like to implement an experimental system for evaluating

its performance and actual overhead.

## Acknowledgment

This work was supported in part by JSPS KAKENHI Grant Number 25330058.

## References

1. "OpenZFS," http://www.open-zfs.org/, OpenZFS Project.
2. "Lustre community," http://lustre.opensfs.org/.
3. B. Dufrasne, B. A. Barbosa, P. Cronauer, D. Demarchi, H.-P. Drumm, R. Eliahu, X. Liu, and M. Stenson, *IBM System Storage DS8000 Easy Tier*. IBM Corporation (2013).
4. "EMC VNX FAST VP — A detailed review," http://www.emc.com/collateral/software/white-papers/h8058-fast-vp-unified-storage-wp.pdf, EMC Corporation (2013).
5. "btier," http://sourceforge.net/projects/tier/.
6. H. Yokota, "Autonomous disks for advanced database applications," *Proc. Intl. Symp. on Database Applications in Non-Traditional Environments (DANTE '99)*, 435–442 (1999).
7. T. Hanai, A. Watanabe, M. Yamaguchi, R. Taguchi, N. Hayashi, T. Uehara, and H. Yokota, "Hierarchical architecture of autonomous storage with solid state disks," *DBSJ Letters*, **2**(3), 41–44 (2003).
8. T. Hanai, A. Watanabe, D. Kobayashi, M. Yamaguchi, R. Taguchi, N. Hayashi, T. Uehara, and H. Yokota, "A performance improvement method of presuming a request-forwarding target in a hierarchical storage cluster," *DBSJ Letters*, **3**(1), 25–28 (2004).
9. J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner, "Internet small computer systems interface (iSCSI)," RFC 3720, Internet Engineering Task Force (2004).
10. *SCSI commands reference manual, rev. C*, Seagate Technology LLC (2010).
11. "Extended frame sizes for next generation ethernets," http://staff.psc.edu/mathis/MTU/AlteonExtendedFrames_W0601.pdf, Alteon Networks White Paper.
12. "Ethernet jumbo frames version 0.1," http://www.ethernetalliance.org/wp-content/uploads/2011/10/EA-Ethernet-Jumbo-Frames-v0-1.pdf, Ethernet Alliance (2009).
13. J. Chu and V. Kashyap, "Transmission of IP over InfiniBand (IPoIB)," RFC 4391, Internet Engineering Task Force (2006).
14. V. Kashyap, "IP over InfiniBand: connected mode," RFC 4755, Internet Engineering Task Force (2006).