A Study on High-performance and Reliable Distributed Storage System

March 2016

Hiroki Ohtsuji

A Study on High-performance and Reliable Distributed Storage System

Graduate School of Systems and Information Engineering

University of Tsukuba

March 2016

Hiroki Ohtsuji

Abstract

As we are heading towards an era of exa-scale computing, the amount of data handled by distributed storage systems is increasing rapidly. Exa-scale storage systems should provide high-performance and reliability. However, these two performance metrics are in a trade-off relationship. In particular, to handle large-scale data, bandwidth is the most important performance metric. Existing studies show that there is performance degradation when the system enables mechanisms for improving the reliability. This study proposes two methods and their designs for reliable distributed storage systems with minimized overheads. In addition, this study proposes a mechanism for wide-area distributed environments, where the network latency greatly affects the performance.

The first method is a pipelined parity-generation method with an active-storage mechanism. The active-storage mechanism can utilize the computational capabilities of the storage nodes to generate parity blocks of redundant storage systems. The proposed method builds parity-generation pipelines using the active-storage mechanism. The pipelined parity-generation mechanism can eliminate bottlenecks caused due to increase in traffic owing to parity blocks. Thus, the performance improvement with the proposed method is about 32.6% compared to the naive implementation.

The second method is a parity-generation method that uses programmable network switches. This method utilizes the programmable capabilities of the future network switches to generate parity blocks. Conventional implementations cannot utilize these abilities of network switches. However, network switches can generate parity blocks and there is no increase in the traffic from the writer node of the storage system in the proposed method. As a result, the performance improvement is 44%.

The third method optimizes the remote file access using a high-bandwidth network with long latency. When a distributed storage system is deployed in a wide-area environment, the network latency delays the file-access requests and responses between the clients and the servers. This proposal intends to solve the performance degradation caused by network latency with an adaptive strategy, which considers the data-access pattern, network delay, and network bandwidth. This method successfully improves the performance of all access patterns: sequential access, stride access, and random access. It can be applied to different layers of the storage systems so that it can help the system in providing both high-performance and reliability.

This study describes a form of a reliable storage system which can leverage the abilities of the storage nodes and network switches through two different methods and one additional mechanism. Each method proves the advantages of utilizing the programmable network and the computing capabilities of the storage nodes. In light of the rapid growth of software-defined networking devices and new storage architectures, which are typified by burst buffer, the proposals presented in this study can become important parts of the next-generation storage systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Storage systems play an important role in the development of data-intensive computing. The amount of data handled by supercomputers is increasing rapidly, and is about to reach exa-scale. There are multiple data sources such as sensors, space observatories, simulations, and so on. In order to obtain new information from the huge amounts of data, the computing systems have to provide better capabilities for data capture, curation, and analysis [2].

Storage systems also have to provide both reliability and high performance, for data-intensive computing, because the system consists of multiple storage nodes with possibilities of failure. However, there is a trade-off between reliability and performance [3]. If we add more redundancy to the storage system, the performance will degrade if there is no special mechanism to prevent this degradation.

This study proposes two methods for designing reliable and high-performance storage systems, which utilize the abilities of the storage nodes and network devices and an additional parameter optimization mechanism for wide-area environments.

1.1 Background

1.1.1 Architecture of distributed storage system

A distributed storage system consists of multiple storage nodes. There are different levels of redundancy. The first level ls the disk-level redundancy, which is provided by conventional RAID [4, 5] systems. However, typical RAID hardware cannot cover the node failures and rack failures; hence, the distributed storage systems have to cover node/rack-level failures. The details of the node-level redundancy are described in Section 1.1.2. The target of this study is a method to provide efficient mechanisms for the node-level and rack-level redundancy of distributed storage systems.

In addition to the conventional network storage architecture, the burst buffer mechanism [6] is used for an improved I/O performance. Burst buffer uses large amounts of flash memory (non-volatile memory) to accelerate the I/O performance. An existing product of the burst buffer mechanism [7] provides reliability by client-side coding, as it claims that erasure coding is not scalable. This study also helps the burst buffer mechanism to ensure the reliability by providing redundancy with a scalable design.

1.1.2 Reliability of the storage system

Reliability of the storage system consists of two metrics: availability and durability. In order to achieve these two metrics, the system has to cover multiple node failures. Previously, replication [8] was used for these purposes. The replication method creates multiple copies of the original data; therefore, it requires at least the same amount of space as required by the original data. This behavior is inefficient in the large-scale storage systems in terms of cost. Moreover, replica-creation processes require additional data transfers.

Several studies [9, 10] have applied erasure coding [11] to storage systems, instead of the replication method. Their results show that they successfully improved the space efficiency; however, there is performance degradation in the throughput of data-write. This performance degradation is caused by the increased amount of the data stored in the actual storage devices. The increase in the data size is inevitable; however, the performance can be improved by using an optimal data-processing mechanism, which is the main proposal in this study.

1.1.3 Performance degradation of redundant storage systems

If we simply add redundancy to the distributed storage systems, the performance of these systems will degrade because of overheads. These overheads can be classified as the cost of parity calculation, and an increase in network traffic. Owing to the calculation speed of the recent CPUs (Fig. 1.1), as long as the system uses the simple XOR parity, the parity-calculation cost is not a bottleneck. However, the increase in network traffic is a dominant bottleneck because the throughput of the flash-memory-based storage devices reaches 4GB/s (as on January 2016 [1]) and will increase rapidly owing to internal parallelism of the device [12]. In addition, the target system of this study has the same storage throughput as the network. The increase in network traffic is caused by the additional data for redundancy,

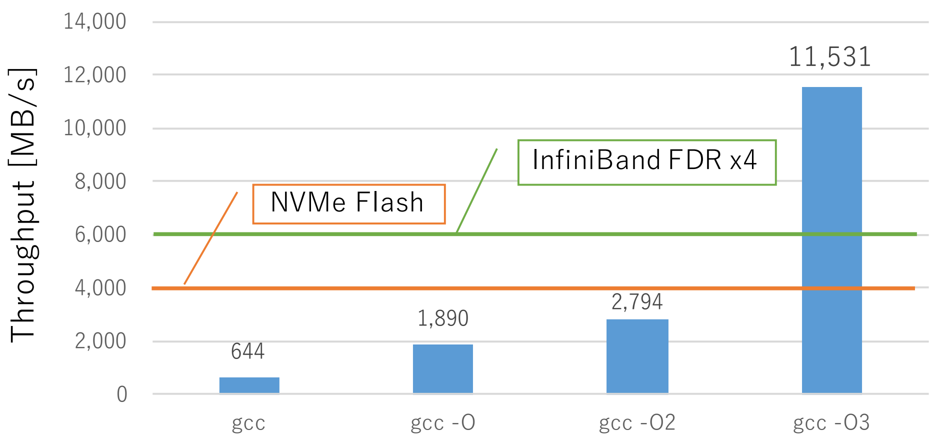

Figure 1.1: Comparison of XOR (A = A xor B) calculation throughput, the bandwidth of an InfiniBand FDR x4 network, and the write throughput of the latest flash memory device [1]. -O, -O2, and -O3 are optimization levels of gcc.

which is inevitable [13].

1.2 Contributions

The aim of this study is to figure out the optimal architecture for a distributed storage system, which utilizes the capabilities of storage servers and network devices. In particular, this paper proposes three different methods: an active-storage mechanism, a method to utilize programmable network switches for distributed storage systems, and an optimization method for remote file access in wide-area distributed environments.

These proposals are methodologies and can be applied to any layers of the storage systems. This study mainly focuses on object-based distributed storage systems, which separate the metadata management and the actual data I/O. The components of the systems are connected to the network. Each storage element is a computing node, which has various types of storage devices: disks, flash-memory-based storage devices, and conventional RAID systems. The proposed methods intend to improve the reliability and performance of the distributed storage systems, regardless of the type of underlying storage devices.

The active-storage mechanism utilizes storage nodes to process data by building data-processing pipelines across multiple storage nodes. The second method proposes to build a data-processing mechanism, using network switches, for storage redundancy. The third method is for wide-area environments and can be applied to any communication layers that uses RPCs (remote procedure calls) for data transfer.

The first two methods achieved reliable storage systems with less overheads in terms of the write-throughput by eliminating the increased traffic from the writer nodes. The last method improves the performance of file access in wide-area environments. In addition, the method with the active-storage mechanism offers high-scalability, which is an important performance factor in large-scale storage systems.

1.3 Outline

This study consists of three main chapters.

- Chapter 2. Active-storage Mechanism
- Chapter 3. Network-based Data Processing Architecture for Reliable and High-performance Distributed Storage System
- Chapter 4. A Method to Optimize Remote File Access Adapting to Access Pattern and Network Delay

Chapter 2

Active-Storage Mechanism

2.1 Abstract

Reliable and high-performance network I/O mechanism is a critical part of exascale storage systems. In order to improve system reliability, some storage systems use replication and erasure coding. However, this is an additional cost and requires higher bandwidth. This chapter proposes a novel method to implement a zero-overhead network RAID-5 with an active storage mechanism. The proposed method focuses on the write operation of the cluster-wide (network based) RAID system, and it can be generalized to the recovery process. In general, write throughput of the storage system with erasure coding is degraded because of the additional coding computation and the increased amount of transferred data. The proposal solves this problem and enables us to implement a zero-overhead cluster-wide RAID system by avoiding client-side computation cost and network traffic. Furthermore, the proposed method enables the system to separate the traffic of the striped data blocks (source) and the parity generation traffic. Scalability of cluster-wide RAID can be highly increased with a dedicated network for the parity generation process. In addition, this chapter proposes an efficient implementation method that utilizes an InfiniBand remote direct memory access (RDMA) mechanism to minimize the number of memory copy operations. This implementation is built with zero-copy pipelines, and enables us to use a low-overhead inter-node data processing mechanism. The measured throughput gain was 32.6% compared to that of the naive method. The results show the same performance as RAID-0, which means that the proposed method achieves the zero-overhead cluster-wide RAID.

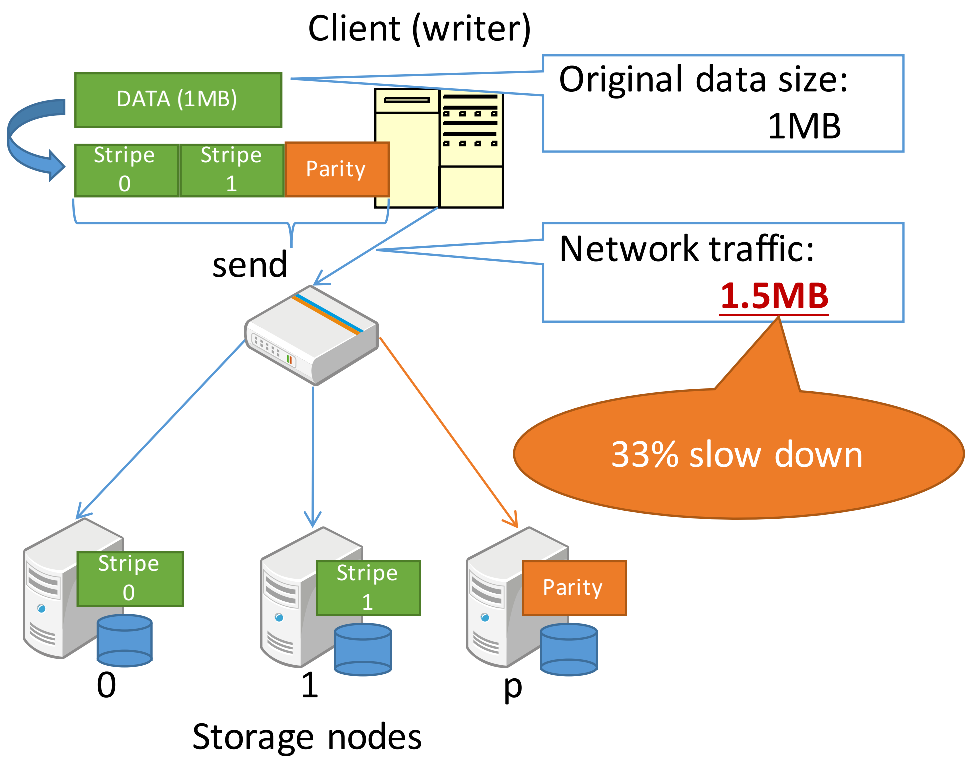

Figure 2.1: Naive implementation of cluster-wide RAID system. A client generates a parity block and transfers data and parity blocks to storage nodes.

2.2 Introduction

2.2.1 Background

Exa-scale storage system is not for a distant future. Many breakthroughs in technology are required to reach the post-petascale era. Large-scale storage systems consist of disks that are connected to each other via a network. Reliability is a critical issue for this kind of systems because the increase in the number of components increases the probability of system failure.

Replication [8] is a popular and simple technology to improve the reliability of large-scale storage systems; however, it requires twice the space of the original data or even more [14]. This problem is directly associated to the increase of the total system cost especially in large-scale systems.

Erasure coding including Reed-Solomon coding [11] and "exclusive or" (XOR) based parity systems are another way to improve the reliability. This requires less additional storage capacity than replication does; however, there are additional computational costs for erasure coding that result in poor I/O write performance. Existing storage systems that utilize erasure codes [15, 16], have an additional cost for reliability, which compromise the system performance. This study focused on both performance and reliability improvement.

Fig. 2.1 shows a naive implementation of a cluster-wide RAID system. A client generates parity blocks and sends both parity blocks and original striped blocks. Parity generation processes require additional computing cost and increase the amount of the outbound traffic from the writer node. They become bottlenecks and deteriorate the write I/O performance. This chapter proposes a way to reduce this additional cost from a client.

This study applied RAID [4, 5] like mechanism to distributed network storage systems. It is called "cluster-wide RAID". As described in the previous paragraph, the increased amount of traffic causes performance degradation when a client generates the parity blocks and sends them to the storage nodes. To eliminate this additional traffic, I design an inter-nodes zero-copy data processing mechanism. Each storage node works as "Active-Storage", which actively processes data blocks. The key idea is a brand-new topology of a data processing pipeline that is enabled using the Active-Storage mechanism. The optimized data processing pipeline off-loads the parity generation process to the storage nodes and eliminates the additional traffic of the client (writer node) network. This mechanism can be applied to emerging technologies such as Burst Buffer [6], which improve the I/O performance of exascale systems.

The proposed Active-Storage mechanism successfully achieves both performance and reliability in cluster-wide RAID systems.

2.2.2 Contributions

The proposed method reduce bottlenecks of critical paths of the redundant network storage system and offers a low-overhead scalable implementation for a cluster-wide RAID system. Networks became bottleneck in the system because the target system of this study should have the high-throughput storage devices which can provide sufficient bandwidth for the latest networks. There is no existing distributed storage system that offers a zero-overhead redundancy support. T herefore, the proposed method can change the stereotype: redundancy always comes with compromises on performance.

The active storage mechanism for the cluster-wide RAID adds redundancy to the storage system with minimal performance degradation. The performance of the cluster-wide RAID-5 is almost the same as RAID-0, which has no redundancy. In addition, the nature of the proposed method enables the system to add a dedicated network. It increases the scalability of the cluster-wide RAID implementation.

The maximum write throughput from a single client node was 5,969 MB/s using the cluster-wide RAID-5 (3D-1P). The total write throughput of the cluster-wide RAID-5 (3D-1P) scales up to 20,141 MB/s. The performance gain compared to that of the naive method was 32.6% and 11.9% respectively. The write throughput with the cluster-wide RAID-5 was almost the same as that of RAID-0 (only striped blocks). The performance was not changed although RAID-5 contains additional parity blocks and increases the network traffic. Hence, the proposed method is called "zero-overhead method".

2.3 Related Work

This chapter focuses on redundancy in storage systems; hence this section explains several existing technology and compare them with the proposed method.

Replication is a widely used technologies in distributed file systems. GlusterFS [17] and Gfarm [18] support replication for data redundancy.

Ceph [9] and HDFS [19, 10] support not only replication but also Reed-Solomon based erasure coding. The erasure coding in HDFS is supported by HDFS-RAID [20, 21] module, which asynchronously transforms from replication to erasure coding to save capacity using background tasks. In other words, it requires a replication process before a erasure coding process. The replication process is essentially a duplicated task so that it brings an additional overhead, which has grater impact than the synchronous coding in terms of the amount of the network traffic. In addition, it makes the system complicated. Xorbas [22] is also an extension for HDFS, which they claim that their new coding offers better performance than HDFS-RAID. However, they still uses a asynchronous encoding. Thus, it has the same problem.

Well-known commercial storage products do asynchronous encoding as well. It is believed that the synchronous coding degrades the performance and this is the reason why other studies and products use asynchronous encoding. The proposal realizes a zero-overhead on-the-fly cluster-wide RAID, which has benefits in terms of the implementation simpleness and the predictability of the performance.

NCCloud [23] is an optimized RAID-6 implementation for multiple-cloud storage system. The aim of this study is to reduce the overhead of code regeneration. In addition to distributed storage systems, there are several studies [24, 25] to improve the performance of conventional RAID-6 systems. These studies are not explicitly relevant to ours; however, they give us useful insights to optimize erasure coding in the storage systems. TickerTAIP [26] implements RAID-5 and uses a similar mechanism as the proposed method. However, their optimization only targets small data update. The proposal optimally writes large data that is much larger than the block size. In this case, the RAID-5 overhead can be avoided using the proposed parallel data pipeline processing among storage nodes. Evaluation results of TickerTAIP show that the performance of the RAID-5 is slower than the result with only stripes (RAID-0), while our proposal offers cluster-wide RAID-5 without performance degradation. TPT-RAID [27] also implements RAID-5 with a third party transfer mechanism. However, the design of the proposed method does not have a controller and does not only target iSCSI protocol. In addition, their binary tree XOR method can reduce the depth of data pipelines (n to $\log_2 n$) and to improve the performance in certain conditions as they claimed in their paper. However, the cluster-wide RAID implementation transfers the data in an optimal order (2.5.3) so that the effect of the depth of the pipeline is minimized in most case. The detail of the latency of the cluster-wide RAID will be discussed in 2.5.6 and 2.5.7.

Next, I focus on the network of distributed storage systems. PVFS [28] has the RDMA network implementation [29]. NFS [30] is a widely used remote file access mechanism and several studies exist in order to add the RDMA support to NFS. One example is NFS over RDMA [31]. RDMA is an imperative technology for this study because the proposed method leveraged the nature of RDMA functionality for the parity generation process.

With regard to the notion of “active-storage”, [32] proposed an architecture, which off-loads application specific data processing tasks to active disks. [33] proposed a similar architecture for DAG-structured workflows. This study focuses on the utilization of the active-storage mechanism to provide an efficient mechanism for storage redundancy.

For data redundancy (not limited to erasure codes), DRBD [34] is an example of on-the-fly data replication system. After the data is written to the source disk, DRBD transparently transfers the data and keeps the remote replica block device identical. Network Block Device [35] provides an interface to use block devices of the remote node. Network block devices are usable for software RAID; however they have a significant bottleneck. If a RAID system is used on these network block devices, the client needs to write parity data in addition to the original data. This results in a traffic increase in the client’s network. Network bandwidth within computers is narrower than the interconnection between a storage controller and disks. Therefore, implementing a cluster-wide RAID on top of the existing network storage systems is not recommended. That is why this study implements both storage and network layers for a cluster-wide RAID system.

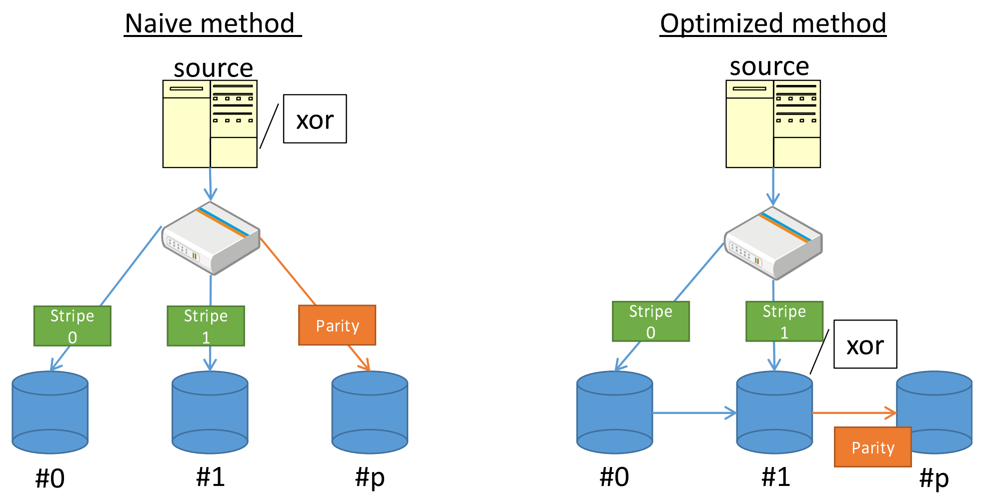

Figure 2.2: Naive and optimized transfer

Table 2.1: Amount of traffic for each node in the **naive** method when the client writes a 10 MB file

	Source	Node #0	Node #1	Parity
In	0 MB	5 MB	5 MB	5 MB
Out	**15 MB**	0 MB	0 MB	0 MB

2.4 Network-based RAID

2.4.1 Cluster-wide RAID architecture

Cluster-wide RAID, which is the main focus of this paper, provides cross-node redundancy by building a RAID-like structure on the network. In a simple term, cluster-wide RAID replaces disks with storage nodes. By replacing disks with storage nodes, cluster-wide RAID can adopt the distributed data processing approach. Each storage node can communicate with other nodes in cluster-wide RAID. This feature expands the possibility to optimize erasure coding generation in a distributed manner. This thesis focuses on the write phase of the cluster-wide RAID system and propose an optimized zero-overhead method.

2.4.2 Basic idea of proposed method

A cluster-wide RAID also brings us some possibility to optimize the data transfer method. Disks and storage nodes are completely different because storage nodes have network interfaces while disks do not. This difference enables us to exchange the data within storage devices whereas the conventional RAID delegates everything to the centralized controller. The proposal is based on this feature of the

Table 2.2: Amount of traffic for each node with the **optimized** method when the client writes a 10 MB file

	Source	Node #0	Node #1	Parity
In	0 MB	5 MB	10 MB	5 MB
Out	**10 MB**	5 MB	5 MB	0 MB

cluster-wide RAID.

For example, in a RAID-4 storage system with 4 drives (one of them is dedicated for parity data), the amount of data written by the RAID controller increases by 25%. RAID and other storage controller have enough bandwidth for each disk; hence, this increased traffic does not create any problem. However, if this method is applied naively to cluster-wide RAID systems, the source node (client / writer) sends an additional parity data via network. It will cause performance degradation. Fig.3.2 is an explanation of a proposed method to solve this problem. In this figure, there is an assumption that the storage system is configured as a RAID-4 array. The left figure is a naive implementation. The amount of traffic from the source node is 1.5 times more than the size of the original data. This causes 33% performance degradation. The right figure is the proposed method. The source node writes two stripes to storage node #0 and #1. Next, storage node #0 sends the received chunk to node #1. Then, node #1 calculates the XOR value of the data from node #0 and the source. Finally, node #1 sends the XOR block to the parity node. This method reduces the total amount of traffic from the source node. In the cluster-wide RAID environment, each network path is critical for I/O performance. Therefore, reducing the traffic of client node is effective to overall system throughput. Table.2.1 and Table.2.2 show the I/O traffic of each node when the client node writes a 10 MB file. If any part of the table exceeds the size of the original data, the performance is degraded. Using the naive method, output traffic of the client node exceeds 10 MB while using the optimized method, the I/O traffic never exceeds 10 MB. It means that there is no bottleneck in the optimized system.

2.5 Design of Cluster-wide RAID-5

2.5.1 Data layout

RAID-5 is a well known RAID configuration and is used in major existing systems. This study applies an "Active-Storage Mechanism" to the cluster-wide RAID-5. This section explains the RAID-5 data layout. Fig.3.3 shows a layout of RAID-5 configuration with four storage nodes. Each row corresponds to the number of a

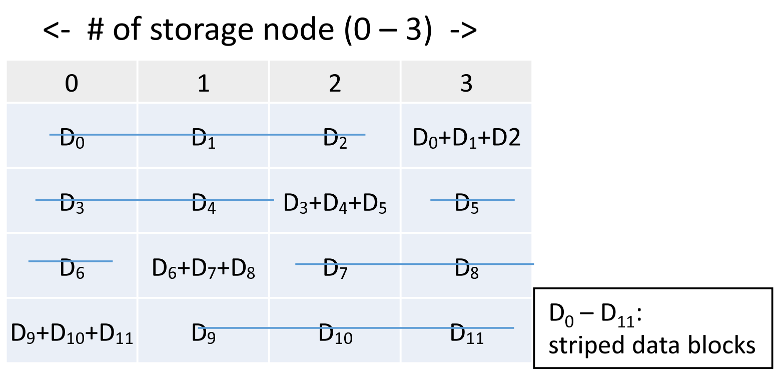

Figure 2.3: Data layout of RAID-5

storage node. The striped data blocks are presented as $D_0 \sim D_{11}$. Other blocks are parity data.

2.5.2 Pipeline topology

Cluster-wide RAID is an aggregation of data pipelines, which are processed by active storage nodes. This section describes the pipeline topology and data movement of the cluster-wide RAID-5. The topology of four nodes for a cluster-wide RAID-5 is shown in Fig. 2.4. Vertical lines correspond to the storage nodes (There are four storage nodes). The client node spreads stripes to storage zero to three. All storage nodes are connected with four pipelines that are separated with horizontal orange lines. All pipelines process data in parallel. The writer keeps the data until it receives ACK messages from storage nodes which store parity blocks. If there is no ACK message from storage nodes within the specified time, the writer resends data blocks.

2.5.3 Data movement

This section focuses on the data movement in a cluster-wide RAID-5 implementation. The data movement is shown in the Fig. 2.4; however, the matrices are shown for clarification. Matrices (2.1) to (2.3) show the data movement. Rows correspond to source nodes and columns correspond to destination nodes. For example, element (0,1) in matrix. (2.1) means that node #0 send D_0 to the node #1. These transfer operations compose three stage pipelines.

Pipeline connections are shown in Fig. 2.5. Each line corresponds to Peer(s). Striped blocks are saved to the disk and other data will be transferred to other nodes for parity generation. Fig. 2.8 describes the pipeline stage when four clients write the data to cluster-wide RAID-5 (3D-1P) simultaneously. Each row corre-

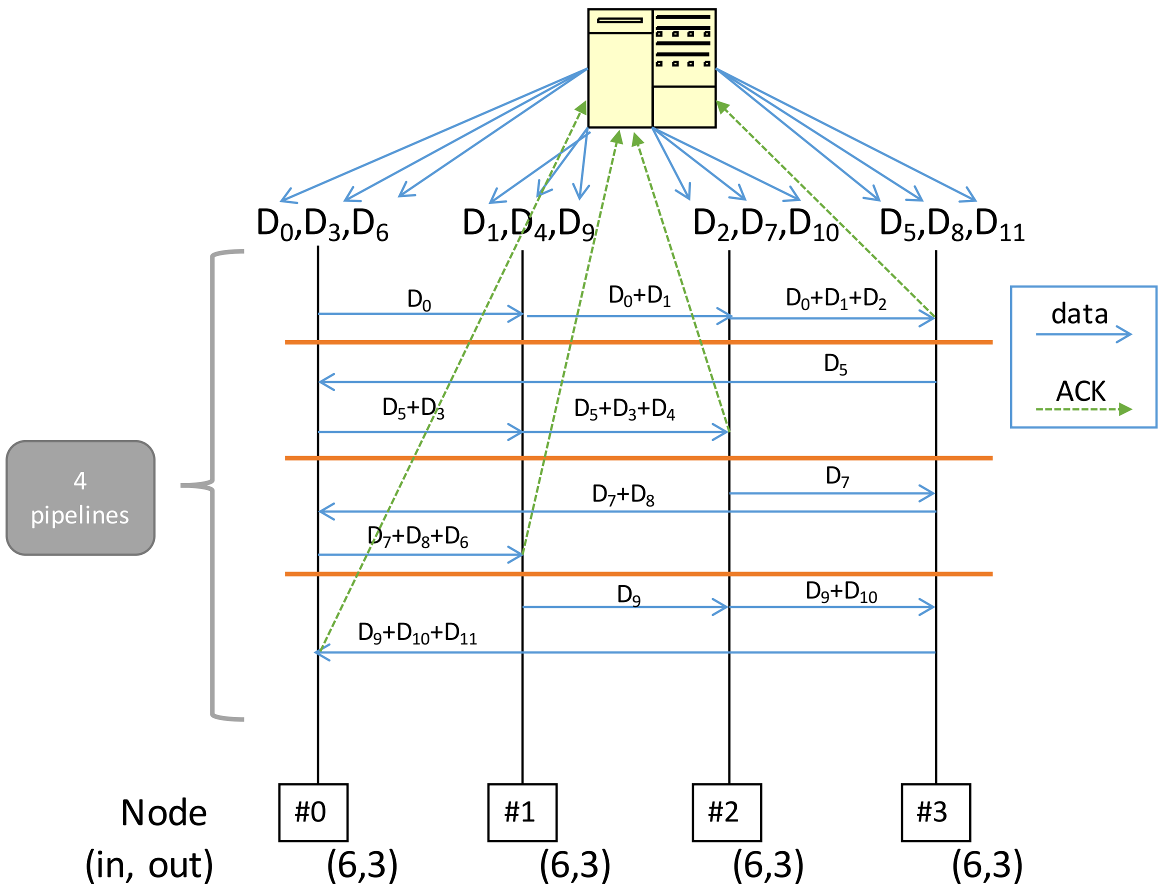

Figure 2.4: Network data movement of optimized cluster-wide RAID-5. Number(s) of (in, out) correspond to number of network path(s). In this case, 12 is the same bandwidth as a single network path.

sponds to the number of storage node and col corresponds to time step. Each storage node can send and receive (implemented with RDMA write operations) simultaneously because it has a full-duplex network. In addition, the system has a dedicated network (explained in the next subsection). Every pipeline finishes in the twelve time steps, which is the same duration as the client data write. It means the cluster-wide RAID mechanism never creates a bottleneck. There is no hazard in the pipelines so that the cluster-wide RAID-5 can utilize the full bandwidth of the network and has better scalability.

$$\begin{pmatrix} 0 & D_0 & 0 & 0 \\ 0 & 0 & D_9 & 0 \\ 0 & 0 & 0 & D_7 \\ D_5 & 0 & 0 & 0 \end{pmatrix} \tag{2.1}$$

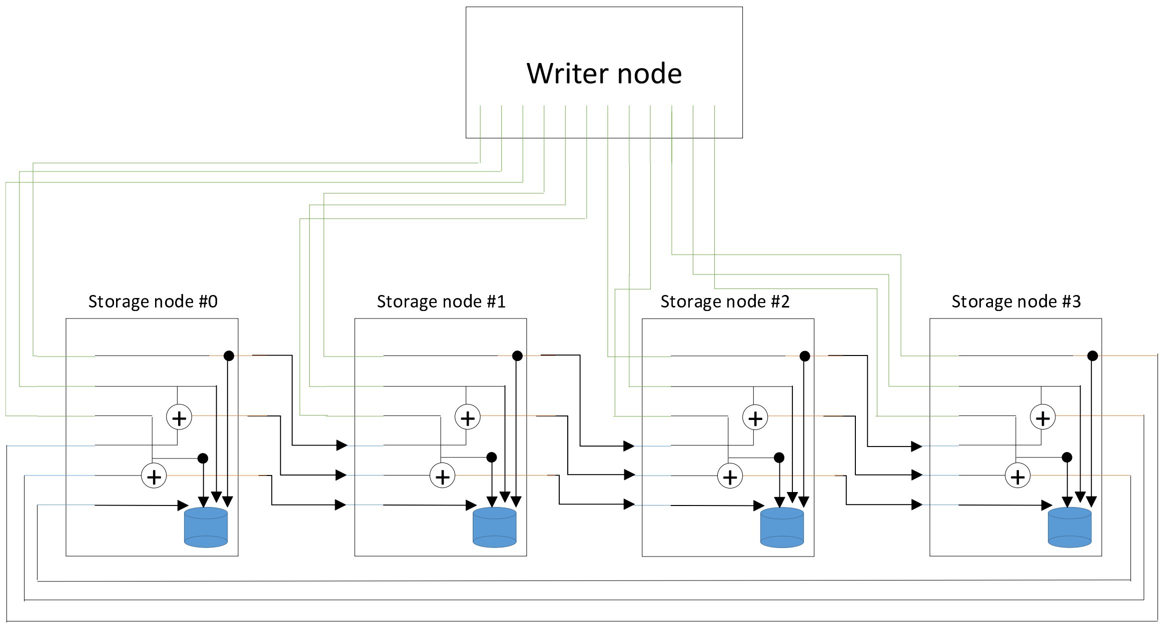

Figure 2.5: Pipeline connection of Cluster-wide RAID-5 (3D-1P). The writer node sends the striped data blocks to storage nodes. Storage nodes exchange the data blocks each other and calculate parity blocks. The data block movement is shown in Fig. 2.8.

$$\begin{pmatrix} 0 & D_5 + D_3 & 0 & 0 \\ 0 & 0 & D_0 + D_1 & 0 \\ 0 & 0 & 0 & D_9 + D_{10} \\ D_7 + D_8 & 0 & 0 & 0 \end{pmatrix} \tag{2.2}$$

$$\begin{pmatrix} 0 & D_6+D_7+D_8 & 0 & 0 \\ 0 & 0 & D_5+D_3+D_4 & 0 \\ 0 & 0 & 0 & D_0+D_1+D_2 \\ D_9+D_{10}+D_{11} & 0 & 0 & 0 \end{pmatrix} \tag{2.3}$$

2.5.4 Partial Write

This subsection describes a method for partial write. When the writer updates stripes partially, parity blocks also have to be updated. Fig. 2.6 depicts the data processing flow when the writer updates data blocks of storage node 1 and 2. The storage node 1 receives the new data blocks and calculates xor value with old data blocks. Then, the storage node 1 sends them to the storage node 2. The storage node 2 does the same thing as the storage node 1 and finally, the storage node p receives the updated parity blocks. The current prototype implementation does not support the partial write; however, it can be implemented with the same mechanism as the full-stripe write.

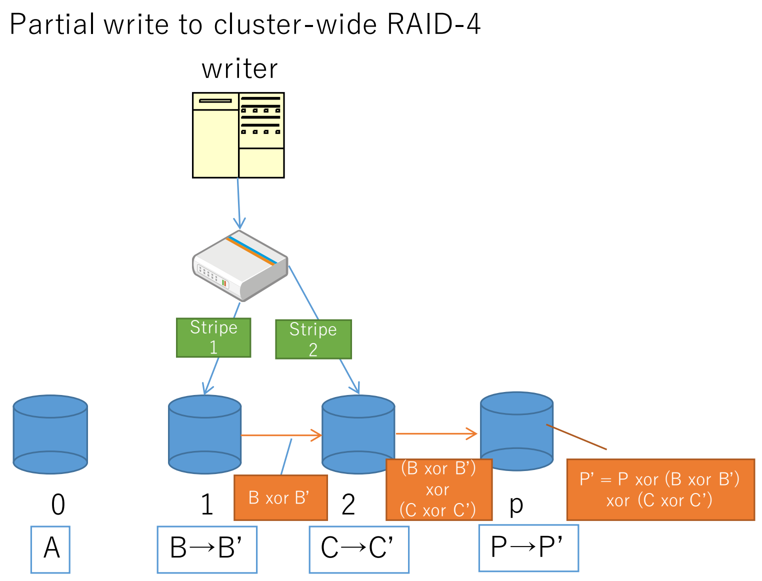
Partial write to cluster-wide RAID-4
writer
Stripe 1
Stripe 2
0
1
2
p
A
B→B'
C→C'
P→P'
B xor B'
(B xor B') xor (C xor C')
P' = P xor (B xor B') xor (C xor C')

Figure 2.6: Partial write to a cluster-wide RAID-4 system.

The estimated performance gain of the partial write (m blocks to update) with the proposed method is the same as the case of $N - m$D1P cluster-wide RAID.

2.5.5 Dedicated Network for Parity Generation

The proposed pipelined architecture can split the data flow into source data transfer and inter-storage node data exchange. This characteristic enables us to add another network hardware to storage nodes and provides additional network paths to data processing pipelines. In this paper, it is referred as dedicated network.

Parity generation process using dedicated network will disperse the traffic on the network. It will increase the scalability of the cluster-wide RAID system. Fig. 2.7 describes the dedicated network.

The dedicated network can be separated from the main network where the source data blocks come from. Thus, the dedicated network does not affect to the number of occupied ports of switches that belongs to the main network segment.

2.5.6 Scalability validation and latency of cluster-wide RAID-5 (3D-1P) system

This subsection describes the scalability and the latency issue of the cluster-wide RAID mechanism.

The amount of traffic of storage nodes of the cluster-wide RAID-5 (3D-1P) are shown in the bottom of Fig. 2.4. In this case, 12 means 100% usage of the

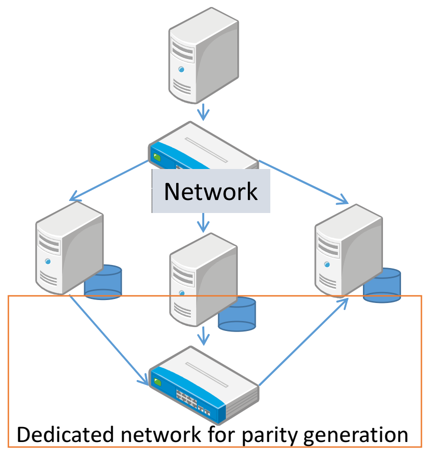

Figure 2.7: Dedicated network for parity generation

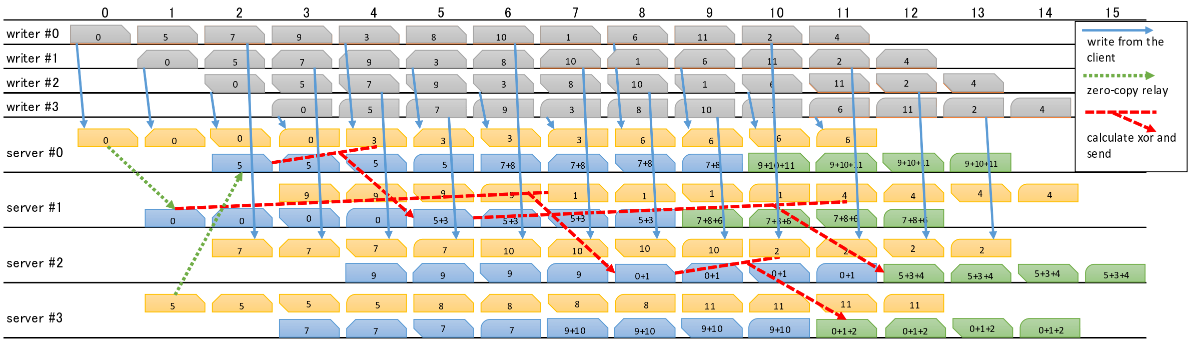

Figure 2.8: Pipeline stage of cluster-wide RAID-5 (3D-1P) with single writer node. Each number corresponds to the data stripe number in Fig. 3.3. The writer node sends striped blocks (0-11) to the server nodes. Then, server nodes exchange the data blocks each other and generate parity blocks. Server nodes store the blue blocks and green blocks to the storage device.

network because the writer node has 12 data paths in total.

Each storage node uses 50% of inbound network bandwidth and 25% of outbound network bandwidth when a client writes data to these storage nodes. Network traffic of the parity generation process can be off-loaded to the dedicated network. In this case, 50% of the inbound traffic and 100% of the outbound traffic are off-loaded. The write throughput scales up in proportion to the number of storage nodes because the traffic is well distributed among storage nodes and there is no traffic concentration.

Fig. 2.4 also depicts the latency of the system. As long as the writer sends the data blocks in the order in 2.5.3, the parity generation process completes two data transfer period after the last block is received by the server. For example, the parity block "5+3+4" is generated soon after the server #1 received the block

"4" from the writer #0. The generalized latency model will be discussed in the next subsection.

2.5.7 Quantitative modeling of parity calculation cost, network traffic, latency, and performance

This subsection shows a generalized model of the parity calculation cost and network traffic / latency of cluster-wide RAID-5 system with an arbitrary number of storage nodes. The amount of the data to write is 1 in the following formulas. t is the one-way trip time between storage nodes, s is the bandwidth of network, and x is the parity calculation throughput. The definition of the term "latency" in this section is the time to write one block to storage nodes.

Parity calculation cost

The prototype system uses simple XOR parity. As of 2016, XOR calculation throughput of CPUs is faster than the network throughput. However, future network devices will provide much faster interface so that this subsection discusses the parity calculation cost of the proposed method.

The writer node generates parity blocks with the naive method. In this case, the total amount of data to calculate parity blocks on the writer node is:

$$\frac{2(n-2)}{n-1}$$

With the active-storage mechanism, parity generation processes are distributed to multiple storage nodes. Each storage node processes the following amount of data:

$$\frac{2(n-2)}{(n-1)n}$$

The total amount of the data to calculate parity blocks is the same. However, the proposed method can reduce the calculation cost of the writer node.

Naive method

A writer node generates parity blocks and sends them with striped blocks with a naive method. Fig. 2.9 describes the amount of traffic of each network path of a cluster-wide RAID-4 with a naive method. Each network path has the same amount of traffic because the writer sends all data blocks and servers only receive them. Therefore, each node receives $1/(n-1)$ traffic of the outbound traffic of

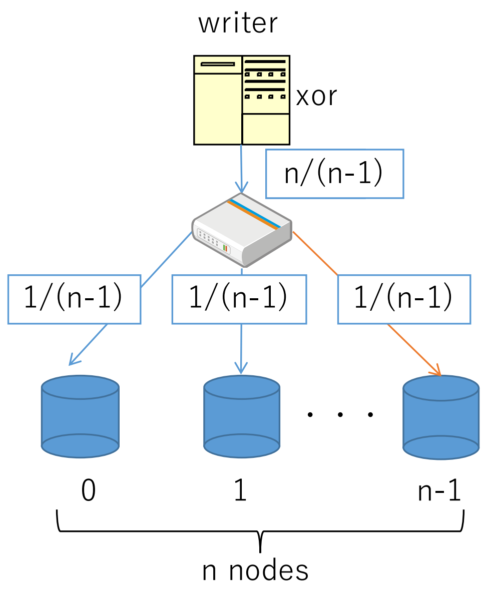

Figure 2.9: The amount of traffic of each network path with a naive cluster-wide RAID-4 implementation.

the size of the original data. In the case of cluster-wide RAID-5, the values are the same as the case of RAID-4 because the ratio of the parity blocks and original striped blocks are the same.

The latency of a naive RAID-4 and RAID-5 implementation is
XOR calculation time + latency + transfer time =

$$\frac{2(n-2)}{(n-1)x} + t + \frac{1}{s(n-1)}$$

With the active-storage mechanism

The cluster-wide RAID-5 network topology can be split into multiple cluster-wide RAID-4 systems. In the case of RAID-5 (3D-1P) (shown in Fig. 3.3), there are four RAID-4 pipelines in the system. For instance, the first one is the pipeline of D_0, D_1,D_2, and $D_0+D_1+D_2$. With regard to this data pipeline, the amount of traffic of each storage node is as follows.

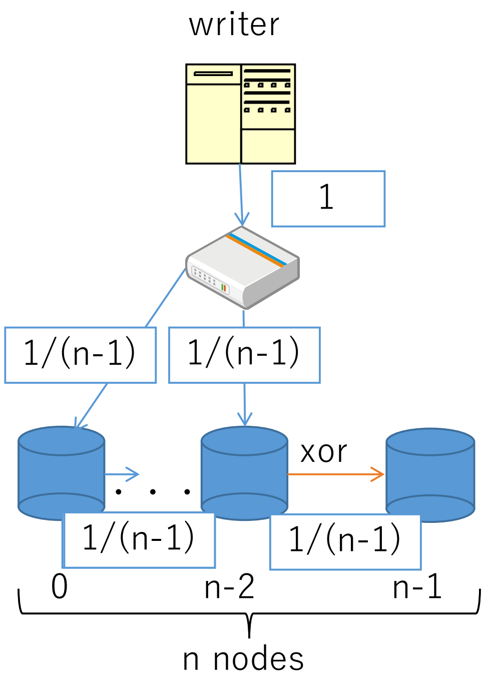

Figure 2.10: The amount of traffic of each network path with an active-storage mechanism.

$$\begin{aligned} S_{0\,in} &= \frac{1}{n-1} & S_{0\,out} &= \frac{1}{n-1} \\ S_{1\ldots n-2\,in} &= \frac{2}{n-1} & S_{1\ldots n-2\,out} &= \frac{1}{n-1} \\ S_{n-1\,in} &= \frac{1}{n-1} & S_{n-1\,out} &= 0 \end{aligned}$$

When the system has a dedicated network (2.5.5), the traffic for parity generation processes is off-loaded and the amount of traffic is as follows.

$$
\begin{aligned}
S_{0\,in} &= \frac{1}{n-1} & S_{0\,out} &= 0 \\
S_{0\,in\,dedicated} &= 0 & S_{0\,out\,dedicated} &= \frac{1}{n-1} \\
S_{1\ldots n-2\,in} &= \frac{1}{n-1} & S_{1\ldots n-2\,out} &= \frac{1}{n-1} \\
S_{1\ldots n-2\,in\,dedicated} &= \frac{1}{n-1} & S_{1\ldots n-2\,out\,dedicated} &= \frac{1}{n-1} \\
S_{n-1\,in} &= \frac{1}{n-1} & S_{n-1\,out} &= 0 \\
S_{n-1\,in\,dedicated} &= \frac{1}{n-1} & S_{n-1\,out\,dedicated} &= 0
\end{aligned}
$$

In the cluster-wide RAID-5 system, the amount of the traffic of each storage node is an average traffic of all storage nodes.

$$
\begin{aligned}
S_{in} &= \frac{2}{n} \\
S_{out} &= \frac{1}{n}
\end{aligned}
$$

Finally, with a dedicated network, the traffic is as follows.

$$
\begin{aligned}
S_{in} &= \frac{1}{n} & S_{in\,dedicated} &= \frac{1}{n} \\
S_{out} &= 0 & S_{out\,dedicated} &= \frac{1}{n}
\end{aligned}
$$

The latency of the active-storage RAID-5 (3D-1P) was discussed in the previous subsection. If the transfer block size is larger than $(n-2)\frac{t}{\frac{1}{s}+\frac{1}{x}}$ (the product of the number of the depth of pipelines and the sum of the inverted number of the network throughput and xor throughput), the effect of the pipeline length can be covered. The typical example values with InfiniBand FDR and RAID-5 (3D-1P) system are n = 4, s = 6,000 MB/s and , t = 1us. Therefore, if the block size is smaller than 12 KB in this condition, the length of the pipelines does not affect to the overall latency.

As explained in the previous section, the last parity block is generated the period of two data transfers after the time when the writer sent the last block. Therefore, the generalized latency of the cluster-wide RAID-5 system is as follows.

(Time to transfer the data blocks from the writer) +

(XOR calculation time of a block) + (time to send a block) + latency=

$$\frac{1}{s} + t + \frac{1}{xn(n-1)} + \frac{1}{sn(n-1)} + t$$

Theoretical performance

This subsection discusses the theoretical write throughput per writer node of Cluster-wide RAID systems. The theoretical performance can be calculated from the amount of traffic. w is the number of concurrent writer nodes in following formulas. All results are relative performance and 1 corresponds to the bandwidth of a single writer node.

The performance with the active-storage mechanism without a dedicated network (per writer nodes):

$$\min(1, \frac{n}{2w})$$

The performance with the active-storage mechanism with a dedicated network (per writer nodes):

$$\min(1, \frac{n}{w})$$

The performance with the naive method (per writer nodes):

$$\min(\frac{n-1}{n}, \frac{n-1}{w})$$

For instance, Fig. 2.11 describes the comparison of the theoretical relative performance per writer node of the Cluster-wide RAID-5 (3D-1P) system. The performance of the active-storage is superior to the case of the naive method when there are $n/2$ or less writer nodes. The performance of the active-storage with a dedicated network is always superior to the naive method. Dedicated networks require additional resources; however, the naive method cannot utilize them. "Naive method with an additional server-side network" is the performance of the case where the servers have extra networks, which are connected to the same network of writer nodes. In this case, the performance is the same as the naive method and the scalability is better than the active-storage with a dedicated network. "Naive method with an additional client-side network" describes the performance of the case where the all clients (writers) have equipped additional network device(s) with the naive method. The performance is better than all cases up to 2 writer nodes. The performance degrades afterwards because the total amount of bandwidth of the servers is not changed. Therefore, the advantage of

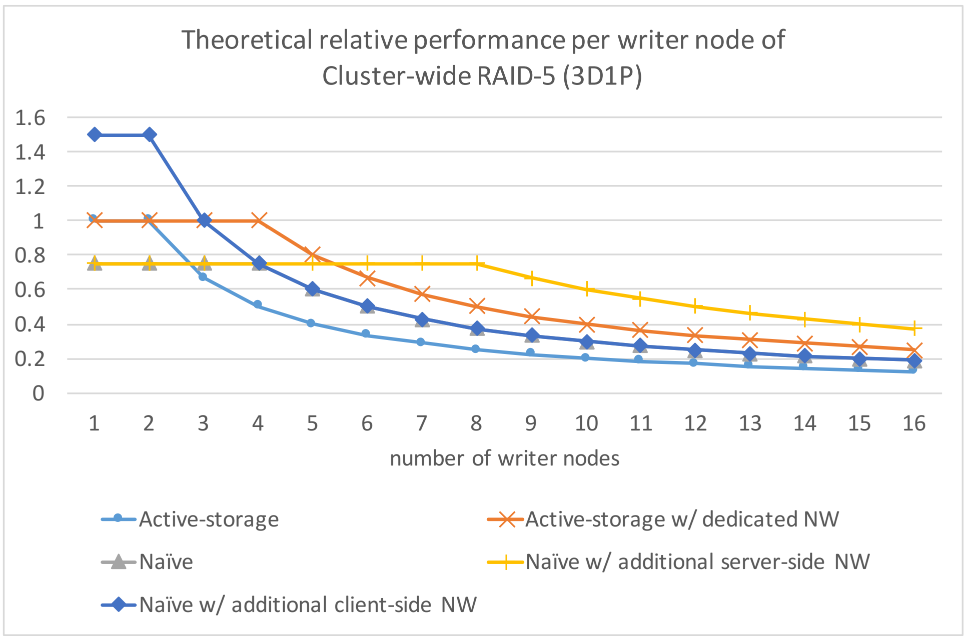

Figure 2.11: The comparison of the theoretical performance of the active-storage mechanism (w/, w/o dedicated network), the naive method, and the naive method with an additional network.

this configuration is very limited. Additionally, it requires the tremendous amount of the additional resources because all computing nodes have to be equipped with an additional network device.

However, there is a fundamental problem that the naive method increases the total amount of the traffic of the network between writer nodes and server nodes, which is the valuable bandwidth in the entire system. It depends on the design of the interconnection network; however, typically, the traffic between storage nodes and computing nodes goes through the long network path, which affects other network communications. Although the performance of the active-storage with a dedicated network is inferior to the naive method with an additional network in several cases, there is the advantages that the proposed method has less side effects to the entire system.

2.5.8 Bandwidth and latency trade-off

The proposed method builds multiple data pipelines over multiple storage nodes. This subsection discusses the trade off of the proposed method because long data pipelines increase the latency. Current major storage devices are hard disk drives, which take several ms to read and write data randomly. In contrast, the latency of the latest network device (InfiniBand) is less than few microseconds. In addition, the effect of the long data pipelines is minimized as described in 2.5.6 and 2.5.7. The situation is almost the same even with latest flash memory storage devices because their latency is longer than the latency of the latest network devices.

2.6 Implementation of Zero-copy transfer for XOR calculation and data store

2.6.1 Utilization of RDMA in storage network

The prototype implementation used RDMA to implement all part of network data access within storage / client nodes. RDMA helped us to reduce the number of copy and memory consumption. Modern network interface has RDMA function to reduce overheads of the network communication. InfiniBand is a widely used high performance network and supports RDMA function. RDMA enables programs to transfer their data on the memory to/from the memory in a remote node directly with small overhead compared with a socket-based communication that requires system calls to send and receive data. All data transfers are CPU-bypass. Therefore, all that a CPU needs to do is to issue a transfer operation to InfiniBand interface(s). In addition, RDMA can perform zero-copy transfer that eliminates overhead resulted from memory copy in system layers.

2.6.2 Data processing pipeline with RDMA

To implement the Active-Storage mechanism for a cluster-wide RAID, this study built an RDMA data processing pipeline mechanism. It will reduce the overhead and memory footprint of data processing pipelines on multiple nodes.

Usage of RDMA is very important in this study because the prototype implementation of a cluster-wide RAID maximize the advantage of RDMA communication to minimize the number of memory copies. As described in the previous section, storage nodes receive data and calculate the XOR value with their own data. Then send the result to another storage node. In this process, there are three memory copies (three times copies between user space and kernel space: twice for receiving and once for sending) in the storage node in an ordinary network. However, zero-copy is achieved with RDMA communication by using DMA hardware directly in the user space. Excessive memory bandwidth consumption degrades the performance of the entire system. Therefore, to minimize the side effect and gain the maximum I/O performance in a cluster-wide RAID, a specialized network data processing architecture is implemented. A detailed explanation of this mechanism is discussed in section 2.6.6.

2.6.3 Choice of RDMA implementation

RDMA is a game changing technology for cluster computing because it enables direct memory access in the user space for remote communication; however, utilizing RDMA requires additional programming issues. This section shows how RDMA communication operations are used.

Fig. 2.12 shows how RDMA function works to transfer data on the memory to another node's remote memory. As this figure shows, there is no middle layer between the client's memory and server's memory. InfiniBand Host Channel Adapter (HCA) is responsible for everything with regard to network issues such as congestion control, error correction and retransmission. There are several ways to utilize RDMA function. These involve Verbs APIs, IPoIB (IP over IB) and SDP (Sockets direct protocol).

The prototype implementation uses InfiniBand network with Verbs APIs in order to maximize the performance. Verbs APIs are group of primitive functions, which are provided by Open Fabrics Software [36]. They run in the user space and there is no context switching; however, they correspond to hardware functions. This is an advantage to implement efficient programs because we can control everything with regard to data transfer. It is also required to do initialization procedure such as memory registration, connection establishment, building queue structures, exchanging memory information, and so on. It takes a number of steps; however, it is is an advantage of the prototype implementation.

Besides Verbs APIs, there are several interfaces to use InfiniBand. There are several studies [37, 38] about different ways to use InfiniBand. The IPoIB (IP over InfiniBand) is the easiest way to use InfiniBand because it provides an interface like ordinary Ethernet adapter. The IPoIB still has the same problem as Ethernet because it uses network layer(s) of the operating system. In addition, InfiniBand has hardware flow control and error correction; therefore, the protocol level (i.e., TCP) control is a duplicate function and becomes bottleneck. Moreover, SDP replaces the TCP control functions with InfiniBand adapter's hardware. It eliminates duplicate functions of TCP/IP and InfiniBand. Therefore, the performance of SDP is better than IPoIB [37]. The last method is using Verbs API, which this study selected. SDP still has some bottleneck with network communication because it uses socket interface to send and receive data. In general, socket interface is not suitable for RDMA communication because it is impossible to specify whether the hardware sends the designated data using the given buffer area directly or by coping to an internal buffer. Utilization of Verbs API requires many proceadures; however, this is the best way to leverage the advantage of RDMA communication.

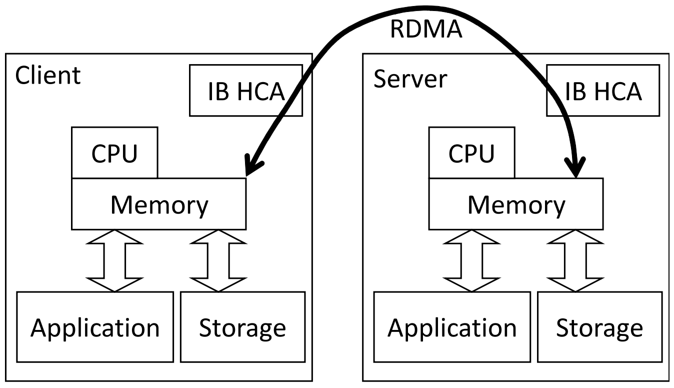

Figure 2.12: RDMA CPU by-pass data transfer

2.6.4 Types of RDMA operations

InfiniBand provides several communication methods [39]:

- Send
- Receive
- RDMA write (w/immediate)
- RDMA read

Send and receive are mainly used for message exchange. In the prototype implementation, send and receive are used to exchange control information. This includes the memory address and flow control information.

RDMA write and read are truly zero-copy operations. They provide interface for direct access to a remote memory. The prototype implementation uses RDMA write operation to send data to remote nodes. RDMA write supports the "with immediate" mode, that includes a small message to notify a receiver for data to be transferred. To implement a data processing pipeline, an efficient notification of data transfer is essential. That is why the prototype implementation uses RDMA write "with immediate" mode for data transfer in the data pipeline process.

2.6.5 Components for zero-copy pipelined data processing

The following subsections explain four components (listed below) for a zero-copy data processing pipeline.

- Ring buffer
- Peer (connection)
- Pipe
- Pipeline

2.6.6 Ring buffer

All memories used for RDMA communications must be registered to InfiniBand hardware. However, memory registration takes a long time compared with the transfer operations. In order to avoid frequent memory registration, the prototype implementation uses ring buffer. Once the entire memory of the ring buffer has been registered, memory registration processes are not necessary afterward.

Pipelined data processing often requires barriers to wait for other data. However, RDMA write operation is an one-sided communication; therefore, there is a possibility of unexpected overwriting. Fig. 2.13 shows the mechanism to prevent this problem. Each ring buffer has "Tail list" to mark the necessary and unused data. The length of this list is equal to the number of external references. For example, the length will increase when a certain ring buffer has to apply XOR to another buffer. The tail of the "Tail list" is the most important thing in this mechanism. Peer (to be described next) tells this value to the sender. As long as sender(s) does not write any data beyond the tail of the "Tail list," the necessary data is perfectly protected. More precisely, each ring buffer has a **current pointer**, which points the position of the current sending buffer, and this pointer cannot overtake any tail of receiver's ring buffer. After the data in a certain buffer is processed, the tail proceeds to the next position. After all tails are moved to the next position, the buffer is ready for its next receive.

2.6.7 Peer (connection) and Pipe

All nodes are connected with reliable connection (RC) of InfiniBand. "Peer" manages the connection and issues transfer operations to InfiniBand HCA. In Fig. 2.14, black arrows correspond to peers. Each peer is closely tied with the ring buffer. Peer interacts with the ring buffer, sends data, and receives information from a remote node. The information includes the position of the tail of next ring buffer.

Peer is just a connection between two nodes. To implement a cluster-wide RAID, peer(s) have to be connected properly to build a storage network topology (Pipe). Fig.2.14 describes the zero-copy relay pipe. In this figure, node #1 sends data to node #2. Then, node #2 re-sends the data to node #3 from **the same buffer**. This is how zero-copy relay works. It is impossible when using ordinarily socket based network programming.

Fig. 2.15 is an explanation of the XOR pipe. In the cluster-wide RAID system, XOR calculation is an important function. Nodes #s1 and #s2 send data to node #2. Node #2 receives data in each ring buffer, and calculates XOR value of the two data blocks, and stores it in one of the ring buffers. Next, node #2 sends the

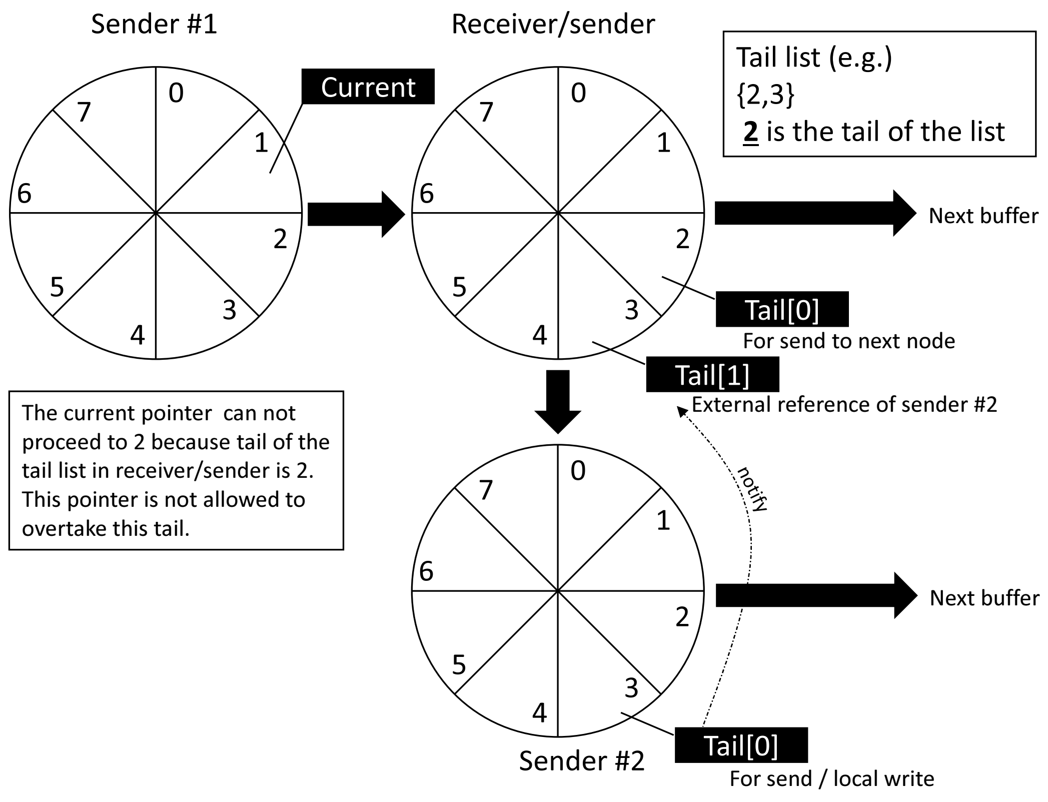
Sender #1
Receiver/sender
Tail list (e.g.)
{2,3}
2 is the tail of the list
Current
Next buffer
Tail[0]
For send to next node
Tail[1]
External reference of sender #2
notify
The current pointer can not proceed to 2 because tail of the tail list in receiver/sender is 2. This pointer is not allowed to overtake this tail.
Next buffer
Sender #2
Tail[0]
For send / local write

Figure 2.13: An example of ring buffers interactions

result to node #3 from the ring buffer directly. There is no additional memory copy; thus, this is the most efficient method to receive, calculate XOR and send the result.

2.6.8 Pipeline

Pipelines for a cluster-wide RAID system can be built by connecting the ring buffer, peer, and pipe.

Fig. 2.16 shows an optimized RAID-4 pipeline design that is one of RAID-5 data pipeline processes. This topology is the same as the right picture in Fig. 3.2. Node #s is a source node and splits data into two stripes and sends them to storage nodes #0 and #1. Node #0's ring buffer is connected to Node #s's buffer (0) and node #1's buffer using a zero-copy relay pipe. This pipe saves the data and transfers it to the next node. An XOR pipe connects buffer #1 of node #s, a buffer #1 of node #1, and the end point buffer of node #p. This pipe refers the data in buffer #0 of node#1. Thus, this XOR pipe will add a reference tail to the tail list of this ring buffer. The cluster-wide RAID-5 can be built using a combination of multiple RAID-4 pipelines.

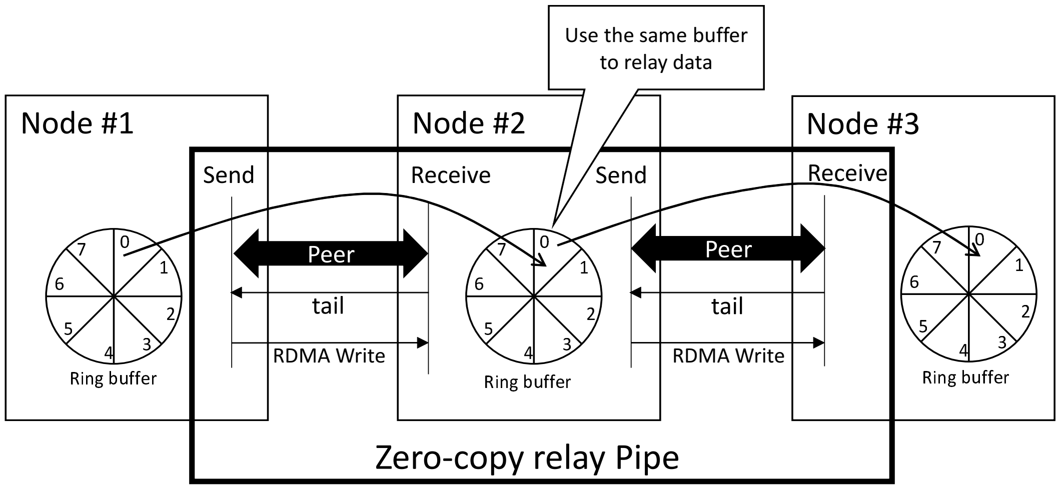

Figure 2.14: Explanation of zero-copy relay pipe

2.6.9 Implementation of RAID-5 with zero-copy pipeline

Three important elements (ring buffer, pipe, and pipeline) are used to build a cluster-wide RAID system. Those components interfaces to connect with each other. Initially, the network data processing topology for a specific RAID have to be determined. Then, they should be connected properly according to the determined topology. The actual topology of RAID-5 (3D-1P) is shown in Fig. 2.5.

2.7 Performance Evaluation

2.7.1 Evaluation condition

This section shows the write throughput evaluation results of the cluster-wide RAID-5. The target of the evaluation is sequential write because the increased network traffic mainly affects the throughput.

This evaluation focuses on the network and transfer mechanism without saving to the disks because the network speed is much faster than the disks we have today. The main target of the proposal is the data transfer mechanism. In future, we can implement the disk write phase using high-bandwidth storage devices. Flash memory based storage devices are getting more parallelism and bandwidth. In addition, overlapping the data transfer phase and the data write phase is a common thing in HPC applications. Moreover, if we try to build redundant burst buffer with the proposed method, the physical storage throughput has less impact on the performance. Because of the above reasons, this study evaluated the performance without a disk write phase. In addition, current the prototype

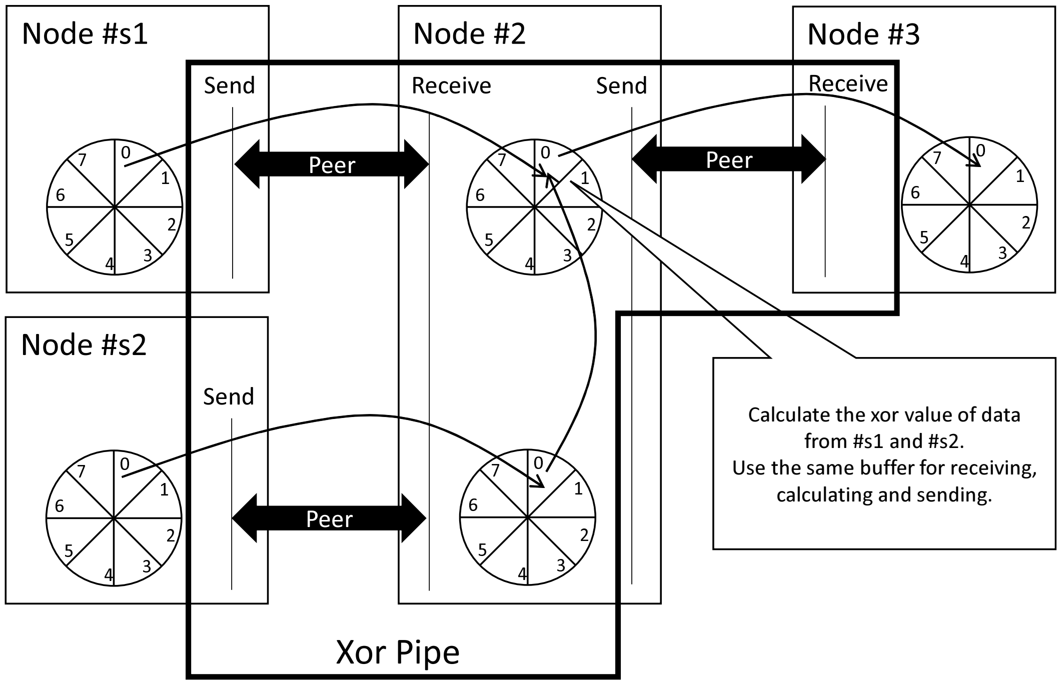
Node #s1
Node #2
Node #3
Node #s2
Send
Receive
Send
Receive
Send
Peer
Peer
Peer
Calculate the xor value of data from #s1 and #s2.
Use the same buffer for receiving, calculating and sending.
Xor Pipe

Figure 2.15: XOR pipe

implementation does not have a mechanism to handle ACK messages, which does not affect to sustained performance.

All evaluation results are average of three measurements. The test case for RAID-5 is 3D-1P, which has four storage nodes. Three of four blocks are striped data and one block is parity data. In addition, this study did scalability evaluation of the cluster-wide RAID with a dedicated network.

2.7.2 Preliminary Evaluation

InfiniBand FDRx4 throughput is shown in the Fig.2.17. These throughput results are measured with ib_write_bw of OFED perftest tools.

Fig. 2.18 shows the performance evaluation results of Pipe and XOR Pipe. The pipe throughput means that the transfer throughput between a server and a client. XOR Pipe receives data blocks from different two clients and calculates the xor value of received blocks, then sends the xor value to the receiver node. The transfer performance and the XOR Pipe throughput are almost same as the throughput of InfiniBand FDR so that the data processing pipeline mechanism will not became a bottleneck.

In addition, this subsection describes the performance results of the simplest case of cluster-wide RAID systems. Fig. 2.19 depitcts a result of performance evaluation of a cluster-wide RAID-4 system, which has 3 data nodes and 1 parity

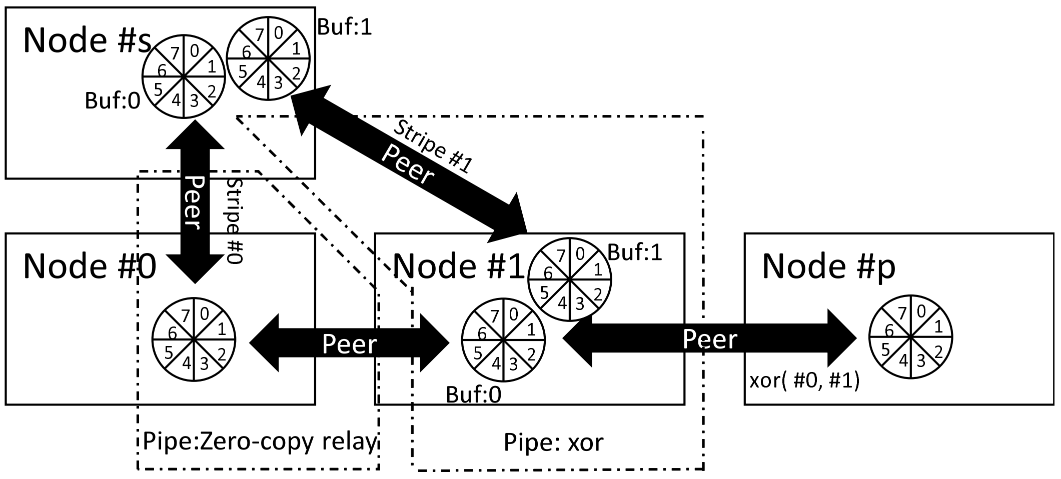

Figure 2.16: Example of optimized cluster-wide RAID-4 pipeline

Table 2.3: Node specification

CPU	Intel(R) Xeon(R) CPU E5-2665 8 cores x2
Memory	DDR3 1600MHz 64GB
Network	InfiniBand FDRx4 (Mellanox MT4099)

node. X-axis the block size, which is the size of striped block and parity block. We evaluated the performance with two block size: 64KB and 128KB. In the both case, results are almost same so that we can say the overhead comes from the pipelined data transfer architecture does not affect to the performance. In any case, the evaluation results of optimized RAID-4 are 32-33% better than the results of the naive method.

2.7.3 Write throughput evaluation of the cluster-wide RAID-5

Fig. 2.20 shows the performance evaluation result of the cluster-wide RAID-5. This study evaluated the optimized implementation and compared it with the theoretical performance of the naive implementation. Naive implementation means that the writer generates parity blocks and distributes them to the storage nodes. The theoretical naive performance was computed using these parameters: Maximum bandwidth of network is 6,000 MB/s and performance slow down is 25% (As shown in Fig. 3.3, 16 of 20 blocks are the original data blocks). This is the best estimation (without considering the overhead) of the naive method; therefore it is not a problem to compare it with the theoretical performance.

In the case of one client node, the performance gain (compared to the theoretical performance of the naive method) is 32.6% with 64KB block size and

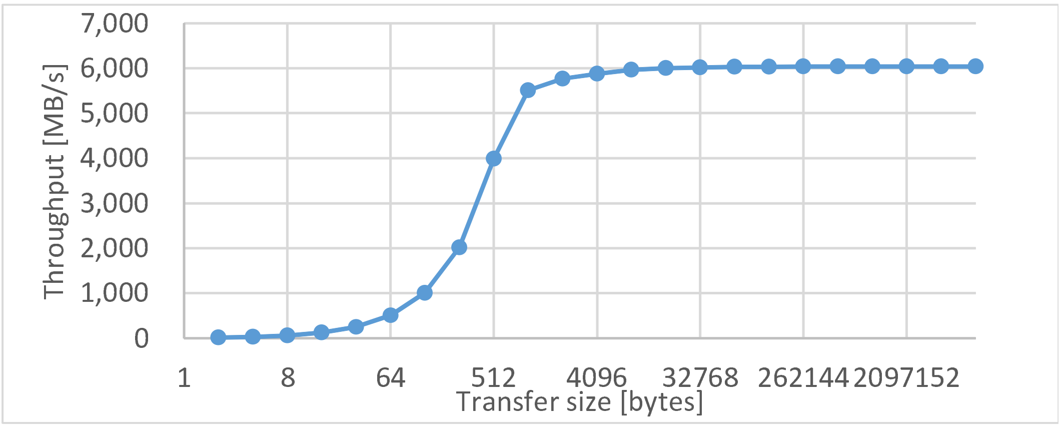

Figure 2.17: RDMA Write Throughput of InfiniBand FDRx4 with ib_write_bw (perftest tools)

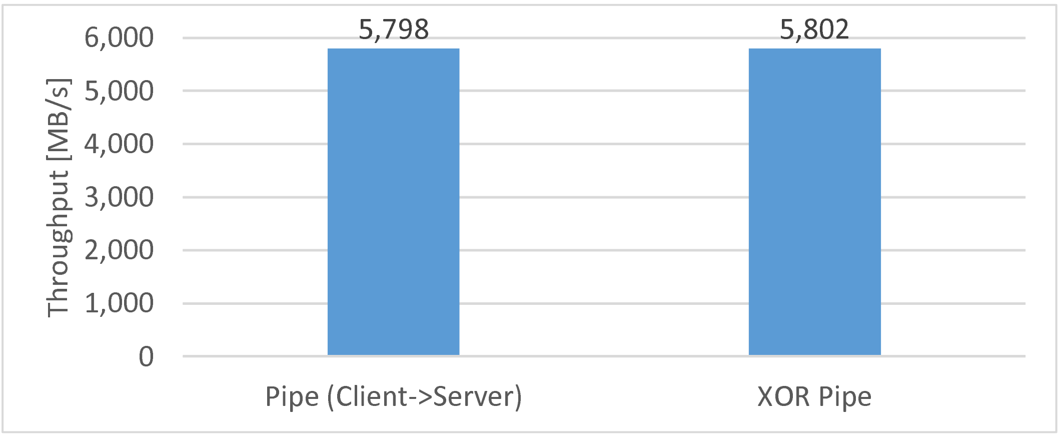

Figure 2.18: Throughput of Pipe and XOR Pipe

30.0% with 128KB block size. 64KB block size is chosen for following evaluations because the performance is almost the same and using small block size has advantages when the prototype system has an implementation of non-sequential write in future.

2.7.4 Scalability evaluation of Cluster-wide RAID-5

Fig. 2.21 describes the result of the scalability evaluation of Cluster-wide RAID-5. In this evaluation, a dedicated network (described in section 2.5.5) is used to enhance the scalability of the cluster-wide RAID.

The X-axis corresponds to the number of writer nodes and the Y-axis is the total write throughput for all clients (MB/s). The performance of the proposed method (optimized) scales up to 20GB/s while the theoretical performance of naive method scales up to 18GB/s. The performance gain is 11.9% in the case

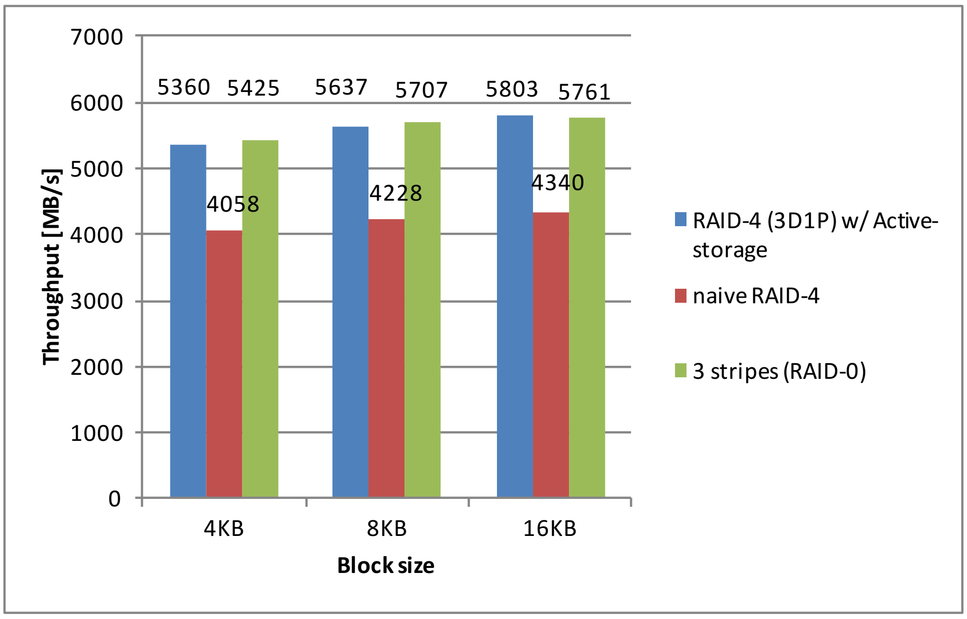

Figure 2.19: Write throughput comparison of RAID-4 (3D-1P), naive RAID-4, and 3 stripes (RAID-0)

of eight writer nodes. Moreover, the performance of cluster-wide RAID-5 with optimized method is almost same as the performance of RAID-0 (three stripes) with one to three clients nodes. With four or more clients, the performance of optimized method exceeds the RAID-0 because cluster-wide RAID-5 increased the scalability by using the network of parity nodes and the dedicated network.

2.8 Conclusion of this chapter

A cluster-wide RAID has an advantage to save storage capacity compared to the replication, but the additional computation and data transfer cost are problems to improve the write I/O performance. This paper proposed an active storage mechanism for cluster-wide RAID system to remove these problems from a client. The active storage mechanism offloads the parity computation and parity block transfer to storage nodes. This chapter showed the topology of data pipelines among storage nodes to calculate RAID-5 parity blocks, which enables a zero-overhead RAID-5 active storage. This chapter also showed the zero-copy RDMA implementation of zero-overhead data pipelines that comprise ring buffer, peer and pipe.

As a result of the proposed design, the evaluated performance was much higher than that of the naive method in all cases. The performance gain was 32.6% with RAID-5 (single client). In particular, the performance of RAID-5 and RAID-

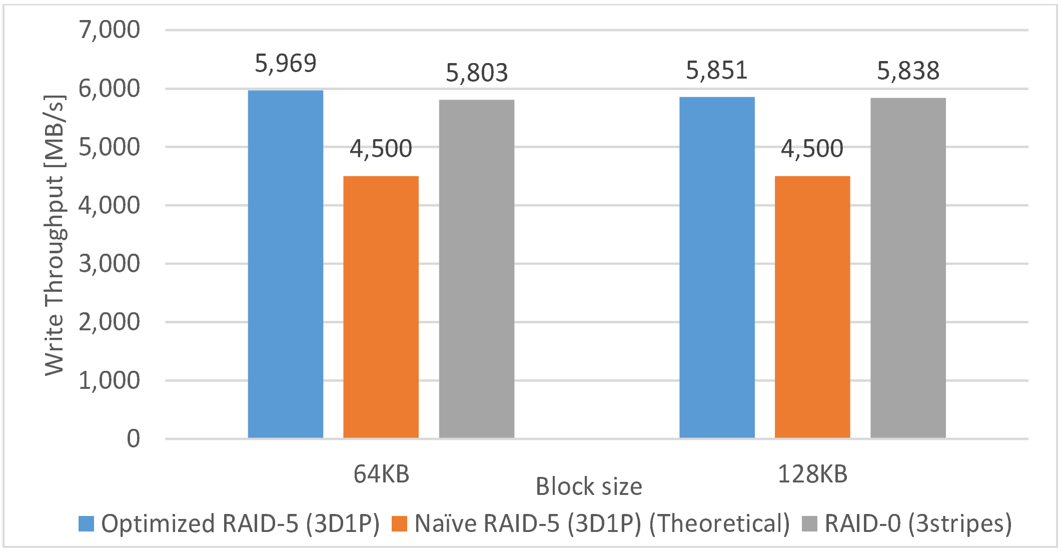

Figure 2.20: Write throughput comparison of optimized RAID-5 (w/ active-storage), naive RAID-5, and stripe (RAID-0).

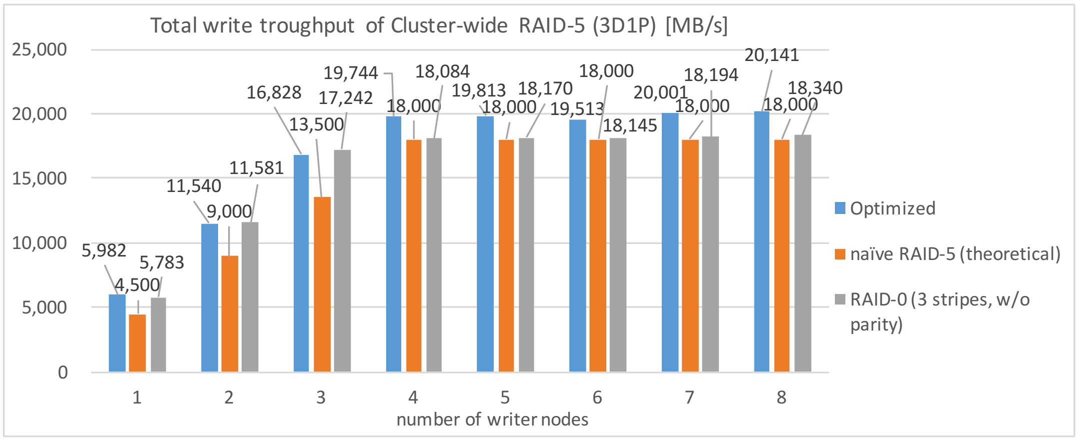

Figure 2.21: Scalability evaluation of Cluster-wide RAID-5 (3D-1P) X-axis is the number of writer nodes and Y-axis is the total write throughput (MB/s).

0 (without parity) was the same, even though RAID-5 had additional parity. Moreover, the scalability evaluation showed that the cluster-wide RAID had better scalability because of the parity generation off-loading strategy. It means that the proposed method can build a scalable, high-performance, and reliable cluster-wide RAID system.

In future work, we will apply the proposed method to an existing system. Here, we implemented and evaluated the concept system. The current prototype implementation does not support read operations, however, the proposed mechanism is sufficiently flexible to implement them. The evaluation results showed the significant advantage of the proposed method over existing systems; however, we can define more characteristics of the proposed method by running a real workload on the system.

Chapter 3

Network-based Data Processing Architecture for Reliable and High-performance Distributed Storage System

3.1 Abstract

In the era of post-peta scale computing, high-performance and reliable storage systems have become much more important. Close cooperation between network and storage is an emerging issue. This paper proposes a network-based data processing architecture to build reliable and high-performance distributed storage system using future programmable network devices. Distributed storage systems use replication or erasure coding for ensuring reliability. However, they require additional data transfer and computing resources. Satisfying both reliability and performance is an important issue for storage systems. Recent studies related to Software Defined Networking (SDN) imply that programmable network switch will become more functional. Currently, SDN intends to provide a flexible routing mechanism. Network switches are starting to have intelligent mechanisms and are expected to have a capability for data processing. In the proposed architecture, storage controller functionality is offloaded to a programmable network switch to eliminate additional data transfer. This study describes the results of experiments to show the advantage of the proposed network-based data processing mechanisms for erasure coding and show an optimized design for distributed storage systems. With the proposed method, the performance gain of a reliable data storage system is 44% compared with a client compute case.

3.2 Introduction

3.2.1 Background

Next generation distributed storage systems have to meet the demands of exa-scale computing systems. In particular, high-performance and reliable data handling mechanisms are critical problems. In order to add reliability to network storage systems, replication [8] and erasure coding are commonly used. However, when a writer node stores data to a storage system, amount of traffic from the writer node increases by the additional data. This additional data degrades the performance because of the bandwidth limitation of a client node. Fig. 3.1 describes how the parity blocks increase the traffic and cause the performance degradation. In this case, a parity block is added to striped blocks. It causes 50% traffic increase and 33% degradation of write throughput. In order to avoid this performance degradation, we need additional mechanisms to eliminate performance bottlenecks.

The target of this chapter is optimization of network storage systems which use erasure coding. The optimization utilizes the functions of a programmable network switch.

Existing systems provide reliability through use of the computational power of storage nodes or client nodes. However, as typified by Software Defined Networking (SDN), network hardware has started to shift toward programmable devices. This movement suggests the possibility of implementing a data processing mechanism on the storage network. This study assumes future network devices with programmable functions and proposes a method for utilizing them.

3.2.2 Contributions

This study proposes an architecture design for utilizing data processing mechanism on the network in order to improve the performance of reliable distributed storage systems. The architecture offloads parity generation processes to a network switch to eliminate the overhead of reliable storage systems. The proposal includes a design of network storage system and a methodology for utilizing programmable network switches. The proposal also implies the way to utilize those functions in real systems.

The proposed method achieves zero-overhead with erasure-codes generation, whereas existing systems degrade performance because of additional data transfer and computing. In preparation for next-generation programmable network devices, this chapter shows an efficient way of using them.

Write performance reaches to 5,548 MB/s with redundant data, which is almost same as the network throughput. The Performance gain is 14% to 44%.

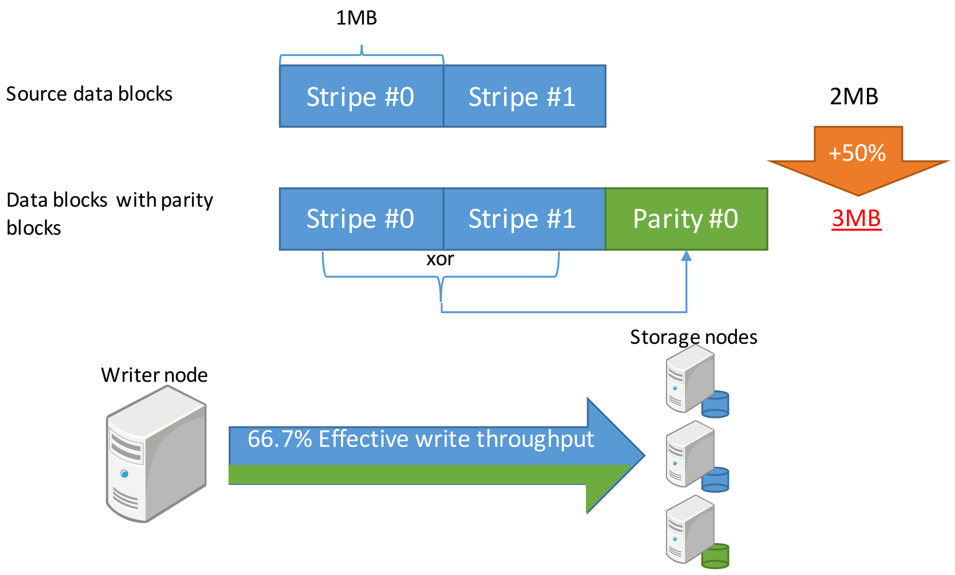

Figure 3.1: Parity blocks increase traffic and degrade write throughput.

Although there are additional data blocks for reliability, performance does not change. This means that the proposed method realizes the "zero-overhead" reliable network storage system.

3.3 Related work

The main focus of the study is a methodology for utilizing programmable network switches to eliminate bottlenecks from reliable network storage systems.

There are existing storage/file systems which use replication and erasure coding in order to improve reliability. RAID [4, 5] is a well-known example of reliable storage systems. However, ordinary RAID systems cover only disk failures and cannot recover from node-level failures. In contrast, the proposal intends to provide node-level redundancy for network storage systems. GlusterFS [17], Gfarm [18], Ceph [9], and HDFS [19, 10] are network storage/file systems and have a replication mechanism. However, as mentioned before, replication uses twice or more storage space and should be avoided with exa-scale systems. Ceph and HDFS also support Reed-Solomon [11] based erasure coding. HDFS supports erasure coding using the HDFS-RAID [20] module. The most important thing is that all of them does not support "on-the-fly" replication/encoding because of performance issues. The proposed method generates parity block (erasure code) on an "on-the-fly" basis, hence they are different from this work.

In order to build a zero-overhead reliable storage system, this study proposes a new architecture which utilizes programmable abilities of a network switch.

There are several studies related to this issue that are not only limited to the optimization of storage systems. [40] is a study to optimize MPI collective operations using network switches equipped with FPGA, which utilized NetFPGA [41] and OpenFlow switches and improved the performance of MPI operations. The optimization target is not the same as this work; however, the idea of improving network communication performance using specialized hardware functions is common.

From the perspective of existing network devices, Mellanox provides the function [42] for optimizing MPI communication operations. The target of this hardware is to optimize MPI operations; however, this is only an example of an HPC communication layer accelerated by hardware. Hardware functions that optimize the network of storage systems are in the extension of this type of idea. This is the reason why this study intends to utilize those hardware functions to optimize erasure codes in network storage systems.

In addition, the prototype implementation uses Remote Direct Memory Access (RDMA) to minimize the overhead of network communication by eliminating unnecessary memory copies. Advantages of applying RDMA communciation to network storage systems are shown in existing studies. NFS over RDMA [31] is an example of adding RDMA support to NFS [30]. [29] shows performance gain by adding RDMA support to PVFS [28].

3.4 System design

3.4.1 Network-based data processing architecture

Network-based data processing architecture moves parity generation processes from storage servers to programmable switches. This chapter describes a design for data processing architecture for parity generation processes and shows a prototype implementation.

The target of this study is not a dedicated hardware based large-scale block storage device but a system which consists of multiple storage servers. Conventional network storage systems use computing resources of servers to provide mechanisms for reliability. In that type of system, network only transfers the data between storage servers.

As discussed in 3.2.1, bottlenecks come from the reliability issues are owing to the increased amount of data and the limitation of network bandwidth. Utilizing programmable abilities of network switches is a good solution to solve these problems because network switches have enough bandwidth to spread the increased data. At this time, there is no network switch (in production and not an FPGA

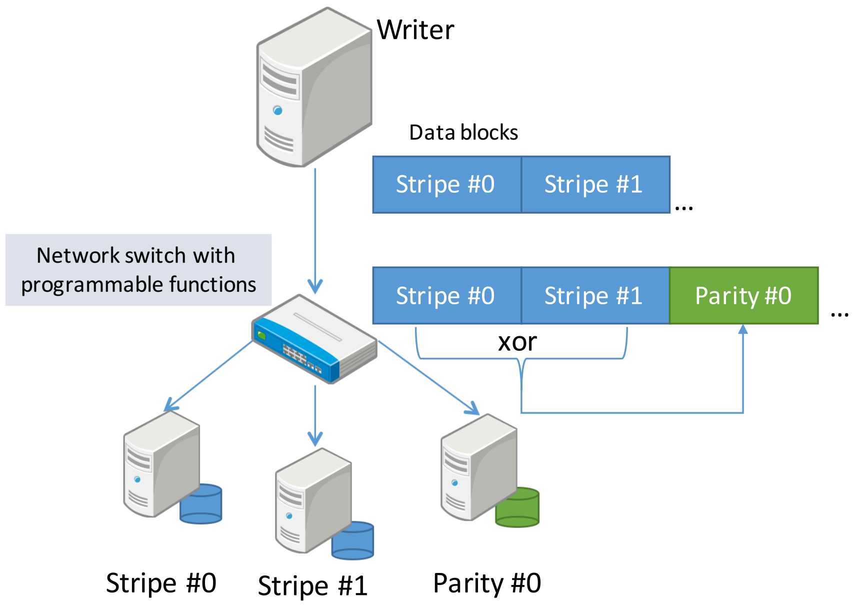

Figure 3.2: Target architecture of network storage systems. The proposal of this study is a method for utilizing a programmable network switch for erasure-coded network storage systems. A writer node sends source data blocks to a programmable switch. The switch generates parity blocks and sends them to storae nodes.

based devices) that has programmable function to implement a mechanism for erasure coding. However, it is possible to propose a method for utilizing the ability of future network switches for reliable storage systems and provide evaluation results with a proof of concept system. The proof of concept system consists of computing nodes with multiple network devices. Following sections describe the the proof of concept system of the network-based data processing architecture and the method for reliable storage systems.

3.4.2 Overview of the system

This study targets network storage systems with parity (erasure coding) data. The aim of this chapter is to propose a method to utilize network data processing functions and to show evaluation results of the proof of concept system.

Fig. 3.2 describes the architecture of the target system. A writer node sends the data bocks (Stripes #0 and Stripes #1) to a network switch. This switch has programmable functions and calculates a parity block from Stripe #0 and Stripe #1. The switch sends stripe and parity blocks to storage nodes.

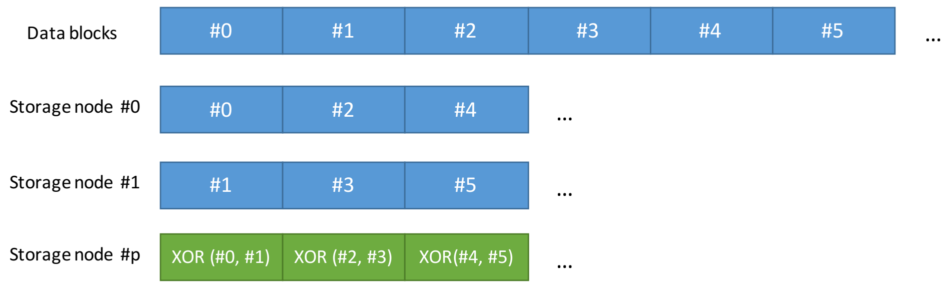

Figure 3.3: Data layout of striped blocks and parity block(s). Data blocks are split into striped blocks. Parity blocks are xor value of striped blocks. Missing data blocks can be recovered from rest blocks by calculating xor value of another block.

3.4.3 Data layout

Fig. 3.3 describes the data layout of the target system. In this figure, the original data blocks are #0 to #5. Two storage nodes store those data blocks and another stores the exclusive OR (XOR) value (parity) of each original data block.

3.4.4 Switch architeture

Fig. 3.4 shows how network-based data processing architecture works. The writer node sends data blocks to the switch, which then splits data blocks into striped blocks and calculates their XOR value. Each parity block is sent to storage nodes for striped blocks and a parity node stores the parity blocks.

3.4.5 Fallback mode

The current experiment environment is only for prototype purpose. However, when the method is applied to a large-scale environment, switch failures become a major trouble.

The system should have a fallback mode in preparation for switch failures. Fig. 3.5 describes the fallback mode of a storage system. If the switch loses programmable functions (left in the figure), the writer node can split the data blocks into striped blocks and calculate the parity blocks. Next, the writer node sends all blocks to storage nodes. In the case of complete network failure (right in the figure), another network mechanism is required. If there is an available network path, the writer can use it to apply the fallback mode.

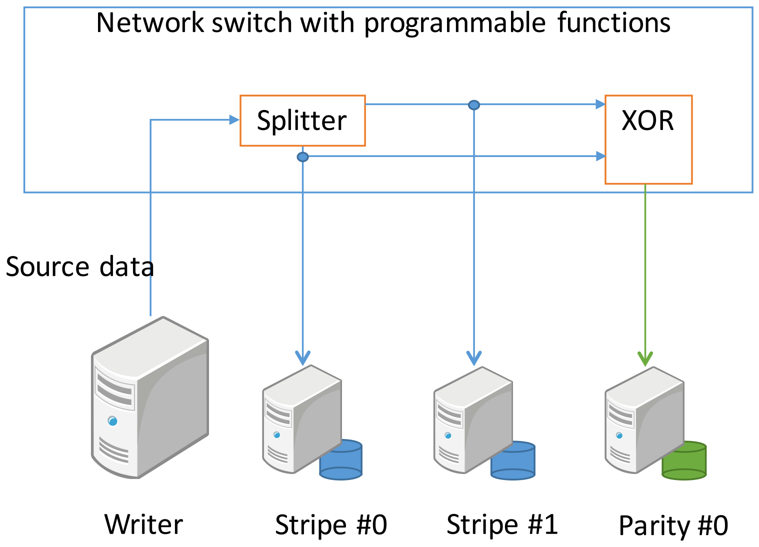

Figure 3.4: Architecture of a programmable network switch. A writer sends source data blocks. A switch has a splitter and an XOR calculation module. The splitter splits tha source data blocks into striped blocks. The XOR calculation module calculates the XOR value of the striped blocks. Then, the switch sends all data blocks to storage servers.

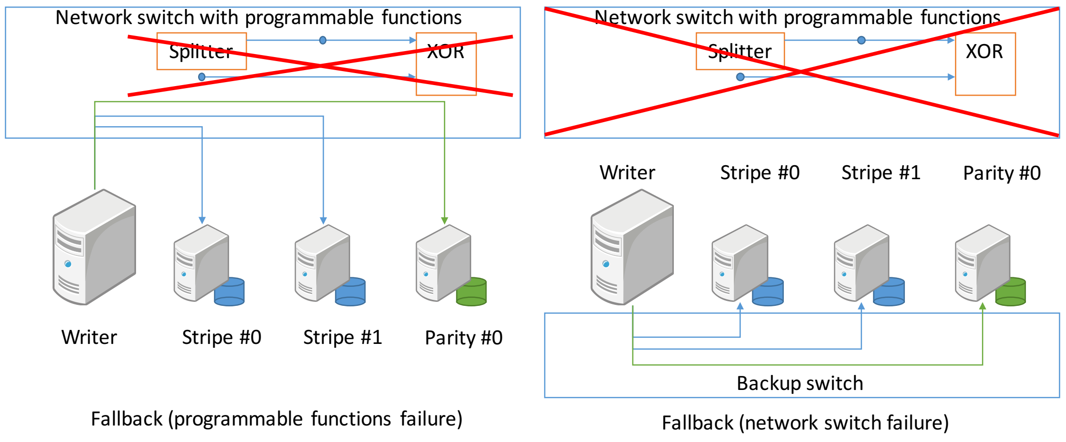
Network switch with programmable functions
Splitter
XOR
Writer
Stripe #0
Stripe #1
Parity #0
Fallback (programmable functions failure)
Network switch with programmable functions
Splitter
XOR
Writer
Stripe #0
Stripe #1
Parity #0
Backup switch
Fallback (network switch failure)

Figure 3.5: Two cases of failure of a network switch.

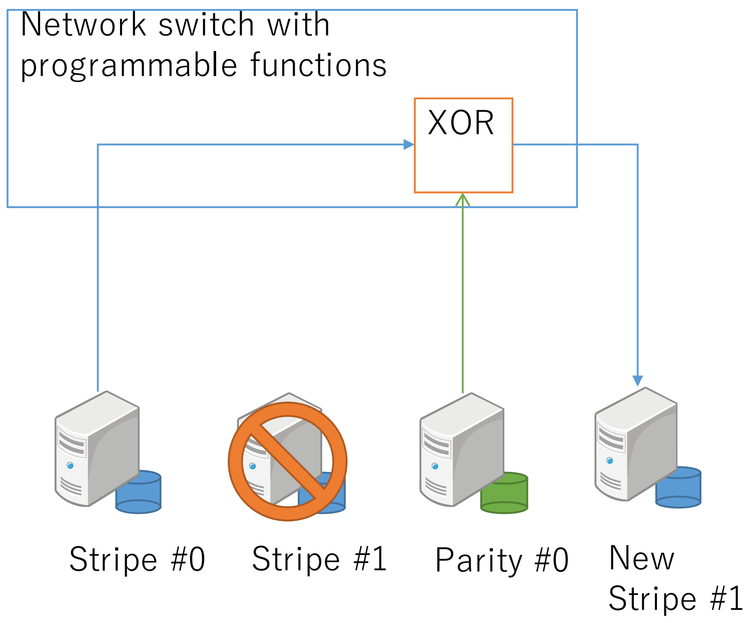

Figure 3.6: Rebuilding data blocks of stripe #1 from stripe #0 and parity #0

3.4.6 Rebuild

Rebuild feature is an important component to build a complete storage system. However the current prototype implementation does not have a rebuild feature, this subsection discusses a design to implement a rebuild feature on the proposed system. Fig. 3.6 depicts the topology when the system rebuilds missing data which is stored on node #1. Since the switch has sufficient bandwidth to receive data blocks from multiple node, the number of storage node of the system does not affect to the rebuild throughput.

3.4.7 Prototype implementation overview

Currently, there is no actual hardware for the programmable network switch; therefore this study implemented it with a computer and multiple network cards. Fig. 3.7 describes the connection of each component.

The node has multiple network cards (in this figure, network cards are InfiniBand HCAs). Because of the limitation of the number of PCI Express lanes, three InfiniBand HCAs are installed to the node. Fig. 3.8 describes the data transfer mechanism and parity generation process. The prototype implementation utilized the RDMA function of the InfiniBand HCAs to optimize the data transfer processes and save memory space. The data structure and data processing mechanism are described in the next subsection.

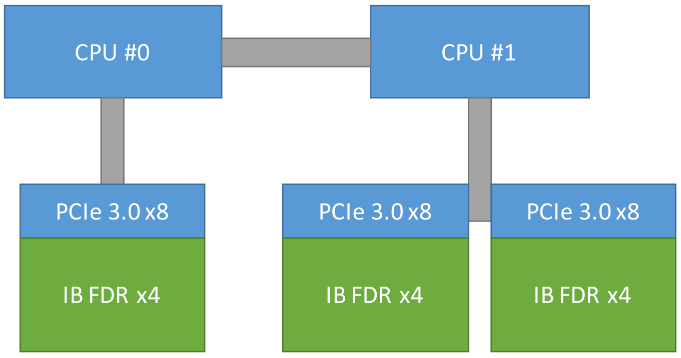

Figure 3.7: Components of a node with multiple network devices

3.4.8 Optimized data transfer and processing with RDMA

Fig. 3.8 describes the zero-copy data structure of the data processing mechanism. Each node has ring buffer(s) to transfer and process the data blocks. In order to use the RDMA data transfer functions, a memory registration process is required in advance to the actual transfer process. If different buffer blocks are used for multiple transfer processes, a memory registration process occurres each time a block is transferred. However, it requires considerable time and causes performance degradation [43]. Therefore, the prototype implementation uses ring buffer(s) for RDMA communication. Once the ring buffer is registered, further data transfer processes never require memory registration processes. In Fig. 3.8 the writer nodes send data blocks to the switch. The switch splits the data into two striped blocks and stores them to ring buffers. Then, the switch sends the striped blocks to storage nodes and calculates the XOR value of these two blocks. Finally, the XOR value is sent to the storage node (p).

3.4.9 Interaction between applications and switches

This study proposes a method itself and shows the evaluation results of the proposal. However, this subsection explains the assumed interface between applications and the proposed mechanism.

The proposal itself is a proposal of a mechanism so that it does not require a specific protocol. The implementation of the proposed method can have generalized operations to orchestrate designated storage layouts. They can be extensions of existing storage network protocol or remote file access protocols.

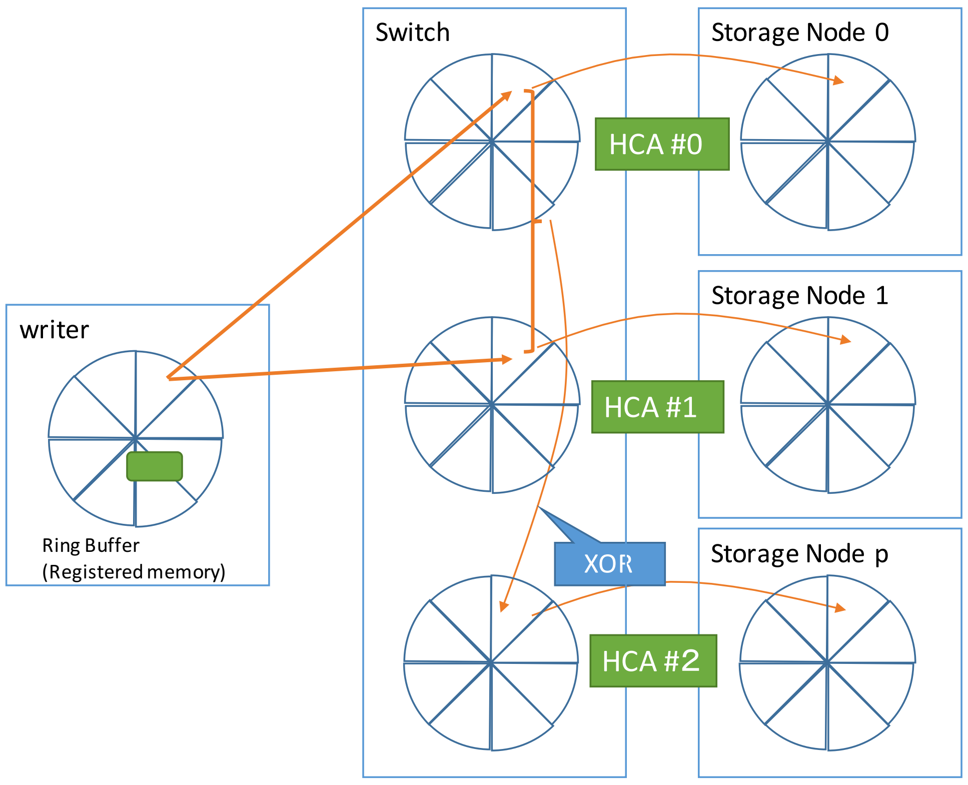

Figure 3.8: RDMA data transfer and parity generation. A writer send source data blocks to a programmable switch. The switch splits source data blocks into striped blocks (not described in the figure) and calculates parity blocks. Afterward, the switch sends them to storage nodes using different InfiniBand HCAs. All data blocks are stored in ring buffers and there is no memory copy.

3.5 Evaluation

3.5.1 Evaluation target and conditions

The proposed method intends to optimize the performance of data write to reliable storage systems.

This section describes the result of an evaluation on the cluster nodes connected with InfiniBand FDR 4x. To implement network-based data processing architecture, a node equipped with multiple InfiniBand HCAs is used. In this evaluation, three InfiniBand HCAs was installed to a computing node. First, the throughput of the multiple InfiniBand HCAs was evaluated to confirm that the test environment did not have any performance bottlenecks resulting from hardware specifications. Fig. 3.9 depicts the results of the total bidirectional bandwidth evaluation. The perftest tools (ib_write_bw) is used to evaluate network performance. As can be seen from the graph, the results are in proportion to the number of installed HCAs. There is no performance degradation caused by limitation of the PCI Express bus or other interconnect issues.

In addition, the evaluation skipped writing to the actual disk because of the limitation of existing storage hardware and the purpose of the evaluation. The aim of the proposed method is optimization of the data transfer mechanism. It does not mean that the network bandwidth is not a bottleneck in the system because as of 2016, the throughput of write and read of the latest product reaches 5,000 MB/s and 3,000 MB/s respectively [1].

All results are average of three measurements.

3.5.2 Evaluation results

Fig. 3.10 shows the results of the sequential write throughput evaluation. The X-axis corresponds to stripe size and the Y-axis corresponds to the write throughput from a client node. The blue graph is the result of optimization (using data processing architecture of a network switch) and the red graph is the result of naive implementation (the client sends both striped and parity blocks). Performance gain by the optimization was 14% (8KB stripe case) to 44% (64KB stripe case). With the best case of the optimized implementation, the throughput almost reaches network performance. This means that the proposed method successfully eliminated the bottleneck of the reliable data write. The performance of the fallback mode 3.4.5 will be the same as the results of the naive method, provided that the system has the same back up network.

The evaluation results show that the proposed method improves the performance of reliable data write by eliminating the bottleneck. In this evaluation,

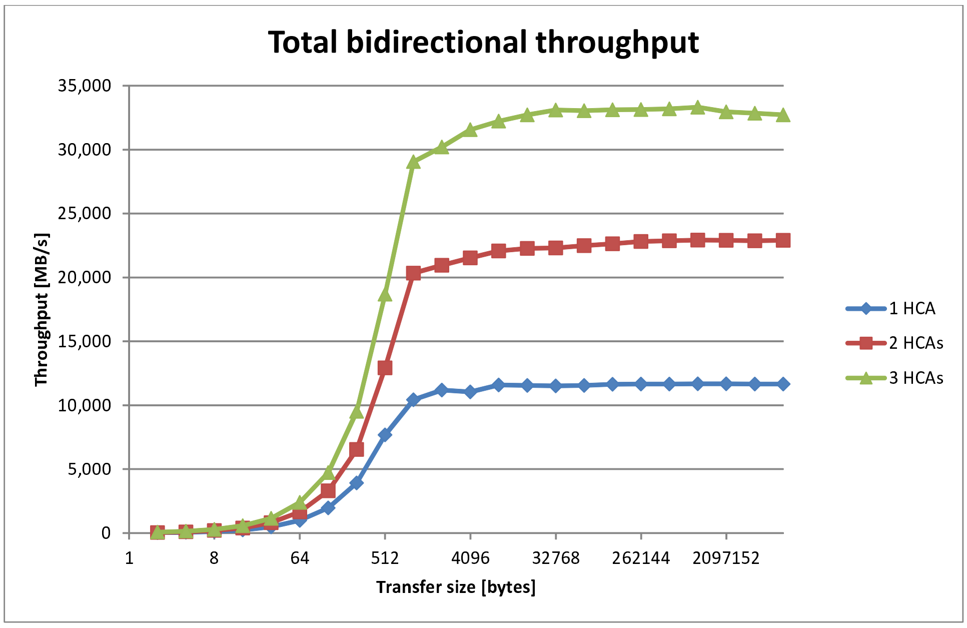

Figure 3.9: Bidirectional network (InfiniBand FDR 4x) throughput evaluation with ib_write_bw (perftest tools).

the data processing architecture using the node equipped with multiple network devices is used. However, the method consists of data processing (XOR) pipelines and can be implemented as hardware. This means that immediately after obtaining a network switch with programmable functions, the proposed method can be implemented by hardware without a major modification of the design.

3.6 Conclusion of this chapter

This study proposed a methodology for utilizing programmable network switches to eliminate the overhead of reliable network storage systems. The root cause of the performance degradation of reliable network storage system was the increased amount of traffic caused by additional parity data blocks. The proposed design moves the parity generation processes to a programmable network switch in order to avoid congestion in a writer node's network. A prototype system using RDMA transfer operations and conducted evaluations was implemented.

The evaluation results showed that the performance gain by the proposed method was 14% to 44%.

Currently, this study implemented the proposed method using a computer equipped with multiple network devices. However, the method can be implemented as a hardware and can be applied to future programmable network switches.

Applying the method to existing systems and adding support of random/stride

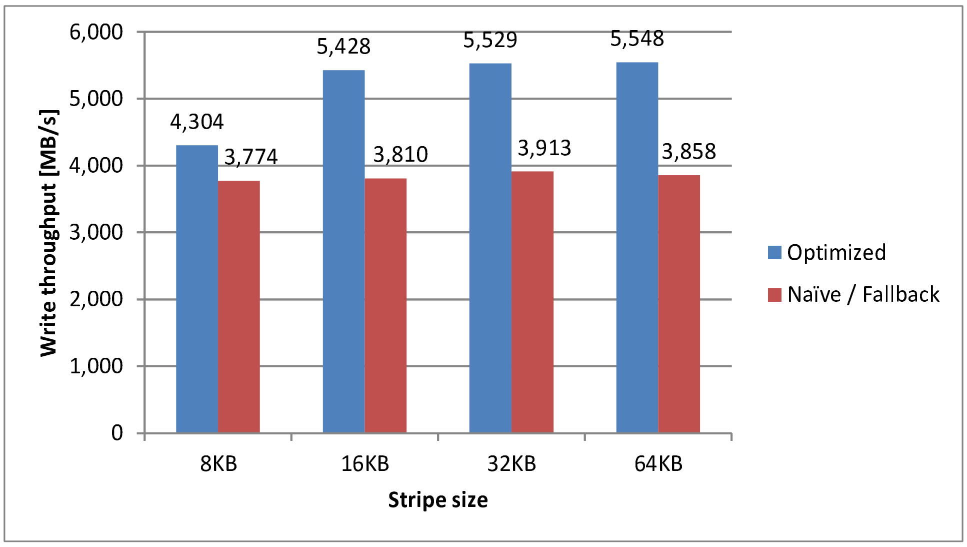

Figure 3.10: Write throughput comparison between optimized and naive method

writing is an important issue. In addition, a scalability issue is an important future work when the method is applied to huge systems because this study used a preliminary evaluation environment, which had a single network switch. Another future work is that the full-support of file system's functions. The prototype implementation did not support partial write, rebuilding, and read operations since this work focused on the performance degradation caused by the increased traffic.

The target for the evaluation in this chapter was the sequential write to storage systems; however, the results showed good performance in case of 16KB to 64KB stripe block size. It implies that the proposed method can achieve better performance with real workloads.

Chapter 4

A Method to Optimize Remote File Access Adapting to Access Pattern and Network Delay

4.1 Introduction

Access pattern and network latency have major impacts on performance of remote file access. Especially, non-sequential access under high network latency environment degrades the performance of remote file access. To achieve high performance, it is necessary to use an adaptive configuration method. This chapter investigates the performance of remote file access with synchronous RPC, in terms of access patterns and network latency, and proposes an adaptive RPC buffer management strategy to improve remote file access performance. This method considers network-delay, bandwidth, and data access patterns. The performance of remote file access under varied conditions is significantly improved by the method. Especially, performance of random access under high-latency environment with the proposed method is much higher than a naive implementation, which uses fixed parameters. The evaluation of the proposed method revealed that the proposed method can find optimal parameters and the performance with the parameters is close to the optimal case.

4.1.1 Background

Sharing data in wide area distributed environment becomes more and more important. Particularly, growth of e-science and data intensive computing boost demand of data sharing within super computers. Wide area distributed file system is suitable for data sharing. File duplication[8], which is commonly used to

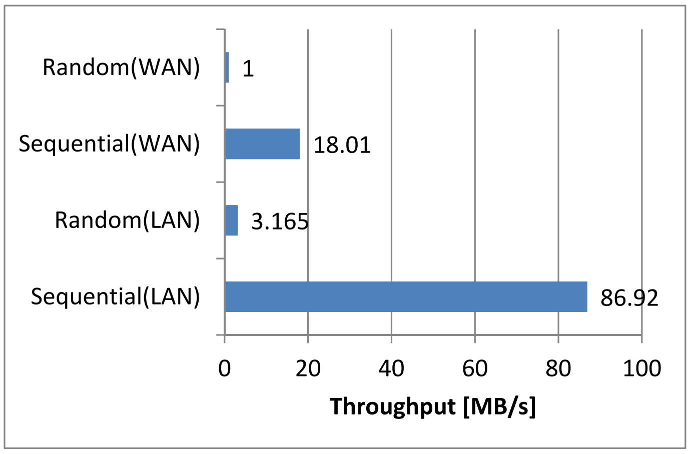

Figure 4.1: Performance evaluation result of remote file access using synchronous RPC. Network delay of LAN is 50us and WAN is 25ms.

speed up the data sharing is tolerant to network delay since it transfers the whole file. However, remote file access is used when a client computer does not have enough storage capacity or needs only a fraction of the file. Remote file access is sensitive to network delay. For example, the performance of the random data access with a high latency network is much lower than the performance with a low latency network. Fig. 4.1 is a result of performance evaluation of an existing method, which uses synchronous RPC. This figure shows that performance of random access with high latency network is low, although the performance of sequential access with low latency network is high. To improve the performance under the high latency environment, a system administrator must optimize configurations manually. However, it takes much time and there are many factors should be concerned. Therefore, a method to optimize configurations automatically, which is the main topic of the chapter is essential.

4.2 Related work

There are two types of remote file access method. The first method transfers a portion of a file with RPCs (Remote Procedure Call) which contain a file offset and size. The other method transfers the whole file as the web protocol. Gfarm [18] and NFS [44] are examples of the former method. The specific size of data will be transferred between the client and the server by this type of remote file access. The size is 1MB with Gfarm and several dozen kilobytes with NFS. AFS [45] and Coda [46] are examples of the latter type. This chapter focuses on the former

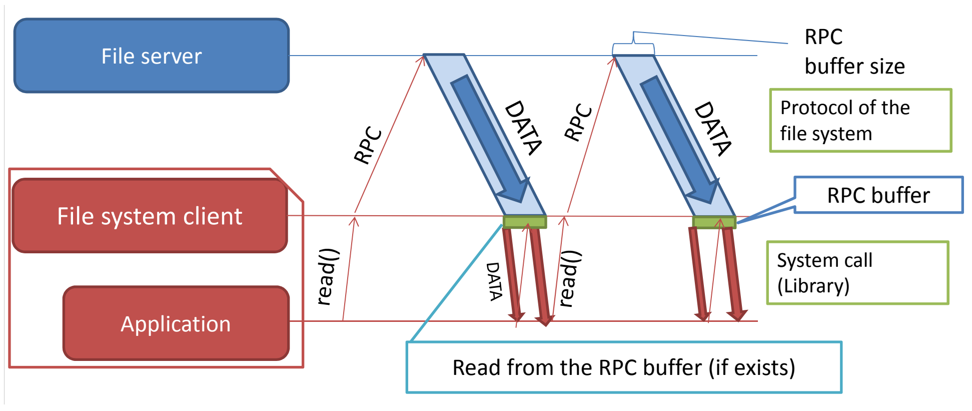

Figure 4.2: Sequence of remote file access using synchronous RPC

type of remote file access.

There are several studies to improve the performance of remote file access. PVFS [28] divides a file into stripes and stores them on several servers in order to permit striping access, which offers higher performance. Lustre [47] enables both striping access and collective I/O. In addition, lustre can issue many RPCs concurrently in order to tolerate network delay. However, the data transfer size is fixed, so the system administrator have to optimize the parameters manually. [48] is an example of an experiment of Lustre with a wide-area network and discussed the relation between Lustre's parameters and performance. This chapter also focuses on wide-area distributed environment so that [48] is a helpful research to know the performance characteristics with high latency network.

Other remote file access implementations have different methods to improve the performance of remote file access. Grid FTP, which is an extended version of FTP [49], adapted to grid environments. Grid FTP has a concurrent TCP connections feature, which enables high-throughput transferring with a high latency network. [50] is a research to optimize a number of concurrent TCP connections by monitoring throughput. This method demonstrates high performance in the real environment. [51] describes the method to determine the nature of data access and to select the file system policy in order to improve the performance. NFS has a performance tuning tool "nfso", however it is not a dynamic tuning method and system administrators have to set the parameters manually. The fixed parameters cannot handle different environments.

This chapter describes the method of automatic optimization of RPC-based remote file access, which considers the usage situation and the network environment.

4.3 A method of remote file access and performance evaluation results

As already mentioned in the previous section, there are two types of remote file access on a distributed file system. This study aims at remote file access using RPC. RPC can be divided into two types: synchronous RPC and asynchronous RPC. This chapter focuses on synchronous RPC, with which a client sends requests to a server one by one to and waits completions of previous RPCs.

4.3.1 Remote file access using synchronous RPC

Fig. 4.2 depicts a sequence of remote file access using synchronous RPC. This figure shows what information is send to the server by the client. The application issues requests and they will be sent to the file server by the file system client. The server receives the request which contains the file offset and size, and returns the corresponding data to the client. In case of remote file access with synchronous RPCs, network delay lags the execution time of the RPC. Thus the number of RPC execution per unit time will be decreased. This behavior degrades the performance.

4.3.2 RPC buffer size and performance

As shown in 4.3.1, the client requests the fixed size of data to the server by a RPC. The RPC has a “size” parameter which means the amount of data transferred by a single RPC. In this paper, this parameter is called “RPC buffer size”. Existing methods of remote file access fix the RPC buffer size.

The performance of remote file access will be degraded if the RPC buffer size is fixed, because the number of the issued RPC will be decreased. This problem is prominent in the case of sequential access.

On the other hand, the server returns the fixed size of data even if the client requests one byte. All the returned data except for one byte is useless for the client. This inefficient behavior degrades the performance of random-access and stride-access.

In order to evaluate the effect of the RPC buffer size on remote file access, this study measured the performance in case of various RPC buffer sizes and access patterns. A prototype implementation was used for the performance measurement. The implementation transfers data between a client and a server Network delay was made by the tc (netem) command of Linux and each computer was connected with a gigabit ethernet network. The server has RAID-0 7200rpm hard

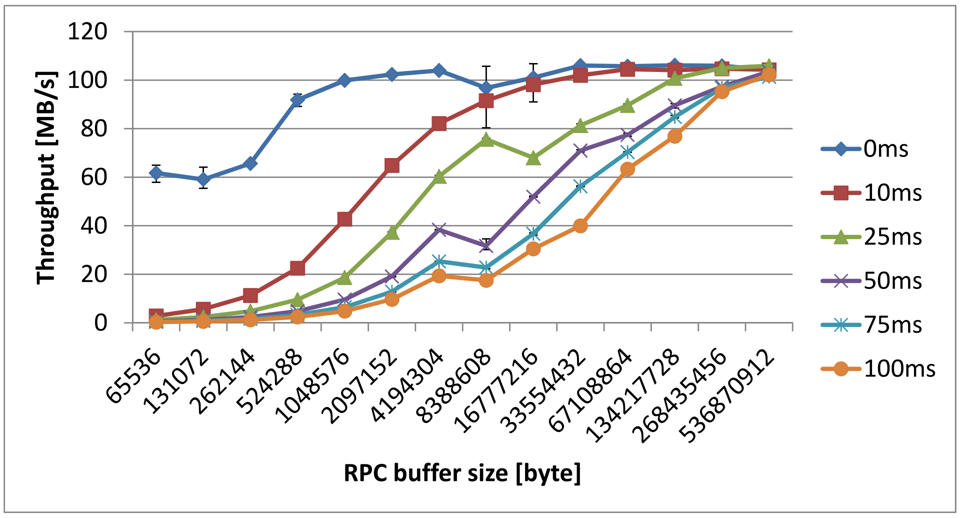

Figure 4.3: Performance of sequential access with each network delay and buffer size

disk drives to avoid that the server's local storage becomes a bottleneck. TCP stack is the cubic, which is the default on Linux 2.6.x. The tested range of the RPC buffer size is from 64KB to 512MB because the performance is saturated if the size is bigger than 512MB. The size of the transferred files was 6GB.

Fig. [4.3 - 4.5] are performance evaluation results of sequential access. The experiment reads 512KB and seeks 3.5MB stride access and the last is 3MB read and 6MB seek stride access. The X-axis is the RPC buffer size and the Y-axis is the measured throughput[MB/s]. Each line in the graph corresponds to the given network delay.

Fig. 4.3 describes that larger RPC buffer size brings better performance. With the less than 1MB of RPC buffer size, the effect of the overhead degrades the performance. If the network delay is higher, the RPC buffer size must be set to a large value in order to achieve high performance. For example, in the case of 100ms network delay, the RPC buffer size must be larger than 512MB to achieve more than 100MB/s throughput. Two factors characterize the performance of sequential access. The one is the time to complete RPC execution, which is affected by the network delay. The long network latency brings the long waiting time. The second is the behavior of the TCP implementation. Small size burst transfers can not expand the congestion window of TCP. The TCP window size should be large to achieve higher throughput transfer with the high latency network. For these reasons, larger RPC buffer size improves the performance of sequential access.

On the other hand, Fig. [4.4, 4.5] show that larger buffer size is not always optimal. In these cases, the small RPC buffer size improves the performance

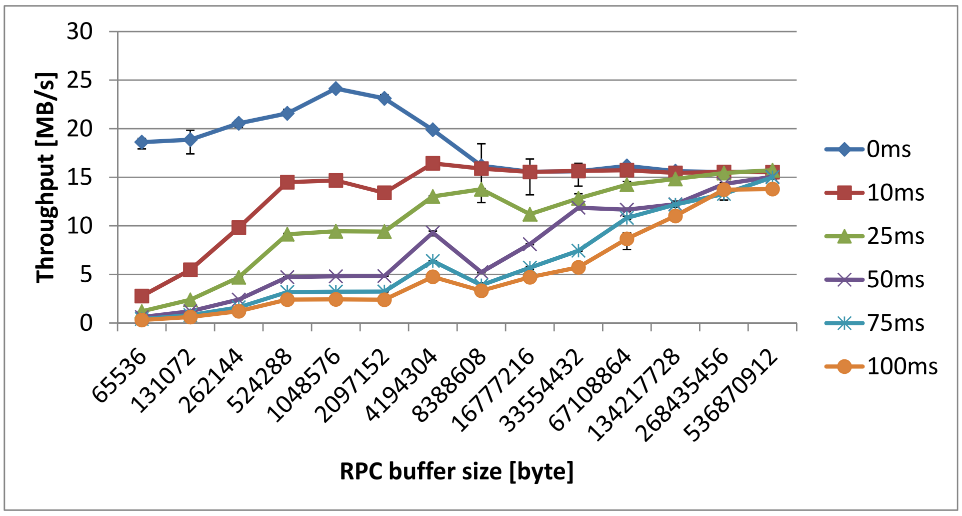

Figure 4.4: Performance of stride access(read 512KB and seek 3.5MB) with each network delay and buffer size

because of the low latency network. It means that the optimal value of the RPC buffer size depends on not only an access pattern but also the network delay. There is unnecessary data transfer(s) if the RPC buffer is too large than the amount of the data the client requested. This point is clear from the 0 ms delay case and 10 ms delay case of Fig. 4.5.

In contrast, the small RPC buffer size degrades the performance of random-access in the high network latency environment because each RPC execution takes the time longer than the network delay. Therefore, larger RPC buffer size can reduce the count of total RPCs that has a better effect than the small buffer size which reduces unnecessary data transfers.

It can be concluded that the larger RPC buffer size is optimal with sequential access. The best RPC buffer size depends on both the network delay and the access pattern with other data access patterns.

4.4 Access pattern recognition and dynamic RPC buffer size adjustment

The previous section showed effects of the network delay and the access pattern regarding the remote file access performance. It is necessary to use a method to recognize the access patterns in order to optimize the RPC buffer size adaptively. This section describes the method to recognize the access pattern.

As described in Section4.3.1, an RPC execution transfer the same amount of data as the parameter of RPC buffer size. The definition of "RPC buffer

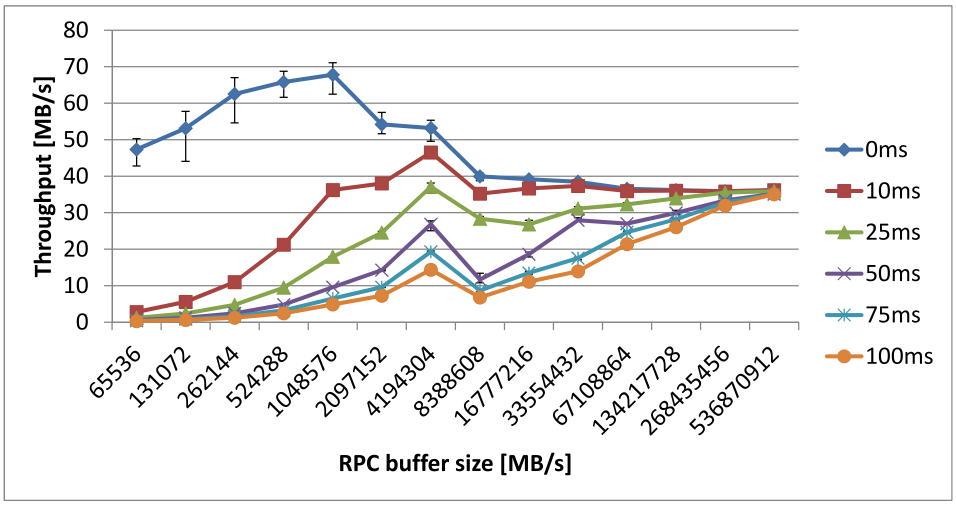

Figure 4.5: Performance of stride access(read 3MB and seek 6MB) with each network delay and buffer size

utilization ratio" is a percentage of the data that is used by the client application. A sequential access raises this percentage to 100% since the client accesses the RPC buffer continuously. On the contrary, the client uses the partial data of the RPC buffer in case of the random or stride access. Then, the RPC buffer utilization ratio will be less than 100%. For example, the RPC buffer size is 8 MB and the client application reads 1 MB data from the buffer, then the RPC buffer utilization ratio is 12.5%. Therefore, if the RPC buffer utilization ratio is high, the access pattern is sequential. If the percentage is low, it is a random or stride access.

There are several ways to measure RPC buffer utilization ratio. The simplest way to detect the access pattern is to collect access logs and to analyze them. However, this is not a suitable approach because the frequent file access also increases the log size and consumes a long time to analyze. In addition, the order of file access(es) is not essential information.

To avoid this problem, this study proposes the new method that keeps information with regard to the accessed area in the RPC buffer. The RPC buffer utilization ratio can be calculated by number of used and unused bytes in the buffer. However, there is still a problem. As the RPC buffer size increases, the amount of memory to keep the used or unused bits will also increase. Then, this study divided the RPC buffer into n blocks. When the client accesses the data which is corresponding to a certain block, the block is recorded as a used block. To calculate the RPC buffer utilization ratio, all the system have to do is just to count the number of used blocks.

Fig. 4.6 depicts the relationship between the actual data access and the RPC

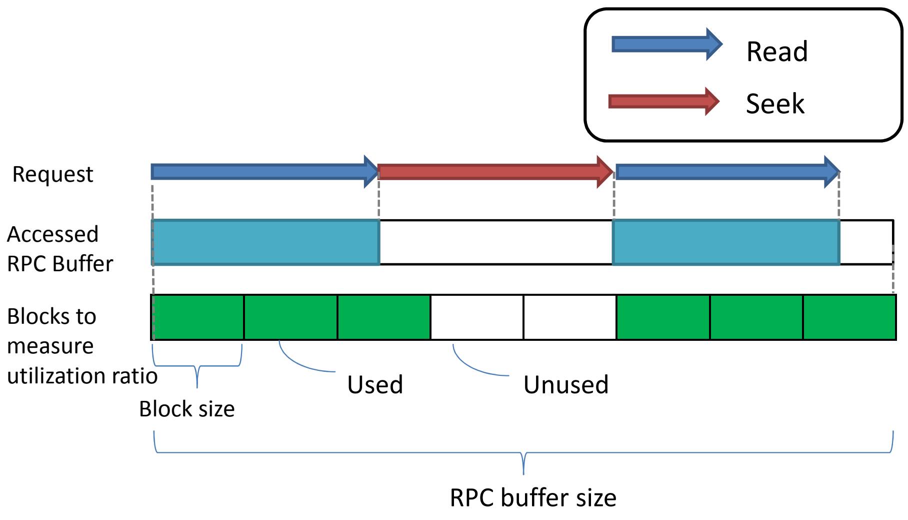

Figure 4.6: Procedure to measure the RPC buffer utilization ratio

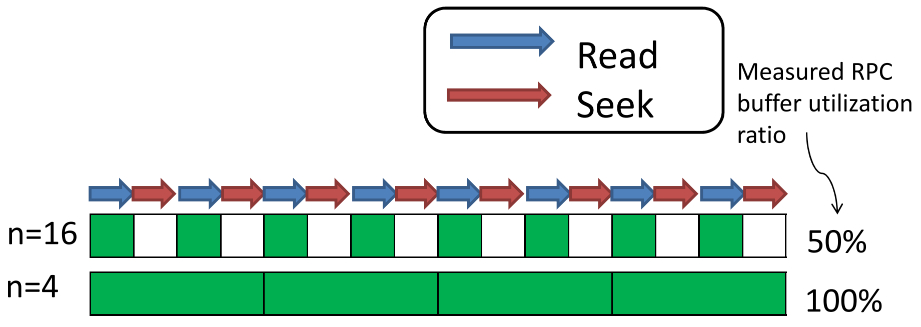

Figure 4.7: Difference of the measured RPC utilization ratio by n

buffer utilization ratio. The first row corresponds to the I/O requests issued by the client application. The second row is the RPC buffer, that saves the data from the server, The bottom row describes the blocks to measure the RPC utilization ratio.

The RPC buffer utilization can be calculated from the number of used blocks as shown in the bottom row and the formula is:

$$\text{RPC buffer utilization ratio} = \frac{\text{number of used blocks}}{n} \tag{4.1}$$

n is a number of partitions and the method to determine n is as follows. Larger n makes the result of the measured utilization ratio more precise.

It is not true that the precise utilization ratio brings the best performance. This point is discussed in 4.3.2. The optimal RPC buffer size depends on the network delay.

Fig. 4.7 shows the difference of the measured RPC buffer utilization ratio with the different n values: 4 and 16. First row indicates requests from the client application. If n is 16 then the result is 50%. On the other hand, if the n is 4, then the result is 100%. In this case, the access pattern is determined as sequential. Smaller n does not decrease the RPC buffer utilization ratio because a block will be marked as used if at least one byte is used in a certain block. Ultimately, every access pattern is recognized as sequential if the n is 1. This study proposes the method to optimize the RPC buffer size by using this behavior.

As described in the previous section, the main purpose to decrease the RPC buffer size is to reduce the useless data transfer. However, the small buffer size increases the number of RPCs. One RPC execution at least takes the time of the network delay. Therefore, the decreased RPC buffer size has advantages only if the is larger than the amount of data that can be transferred during the time of network delay. If smaller RPC buffer size cannot improve the performance, the access pattern should be treated as a sequential access and the data should be transferred in bulk. This strategy is called "data sieving"[52].

n can change the behavior of data sieving. The amount of data that the network can transfer in the time of network delay can be calculated by the product of the network bandwidth and the network delay. This value is called "Bandwidth-delay product". If the size of skipped area of the RPC buffer is bigger than the bandwidth-delay product, the area should be ignored and regarded as used. To realize this behavior, set the n to meet the following statements in order that the block size of the RPC buffer becomes the same value as the bandwidth-delay product.

$$\begin{aligned} \text{Bandwidth-delay product} &= \text{Block size} \qquad (4.2)\\ &= \frac{\text{RPC buffer size}}{n} \qquad (4.3)\end{aligned}$$

The RPC buffer utilization ratio can be calculated from the n and the measured network bandwidth and delay.

4.5 Dynamic optimization of the RPC buffer size

The access pattern can be determined by the method shown in the previous section. Sequential access patterns raise the RPC buffer utilization ratio and the discontinuous access patterns degrade it. These results imply that lower RPC buffer utilization ratio means the access pattern is random or stride, and higher

RPC buffer utilization ratio means the access pattern is sequential.

This method changes the RPC buffer size dynamically based on this rule. The RPC buffer size is changed by the average of the m RPC buffer utilization ratio values in order to avoid frequent changing of the RPC buffer size. The m should be large if the continuity of the access pattern is important. Moveover, larger RPC buffer size takes a long time to complete RPC execution, set $m$to M_{small} if the RPC buffer size is larger than $LARGE\_BUFSIZE$ and set to M_{large} if the RPC buffer size is smaller than $LARGE\_BUFSIZE$ to avoid continuing of the inefficient state. The optimal values of these parameters depend on the frequency of access pattern changing and the characteristic of applications.

The RPC buffer utilization ratio is equal to the percentage of the data used in the transferred buffer. Therefore, the RPC buffer size should be shrunken if the RPC buffer utilization ratio is low, or the RPC buffer size should be expanded if the ratio is high. The size of the RPC buffer utilization ratio is judged by by two threshold values: U_{high} and U_{low}. The procedure to change the RPC buffer size is as follows:

```
if $FLAG = 1$ then
   $FLAG = 0$
   if $U_t < U_{t-1}$ then
      $WAIT = p$
      $B_{size} = B_{size} / \sigma$
   end if
end if
if $WAIT > 0$ then
   $WAIT = WAIT$ -1
   CONTINUE
end if
if $U_t > U_{high}$ then
   if $B_{size} < B_{max}$ then
      $B_{size} = B_{size} * \sigma$
      $FLAG = 1$
   end if
else if $U_t <= U_{low}$ then
   $B_{size} = B_{size} * U_t$
end if
```

U_t is the current RPC buffer utilization ratio and U_{t-1} is the previous ratio. B_{size} is the RPC buffer size and B_{max} is the maximum size of the RPC buffer. σ is an increase rate of the RPC buffer size. Note that the U_t is an average value of M of the utilization ratio values (this is not described in the algorithm).

Table 4.1: Parameters

Parameter name	value
LARGE_BUFSIZE	128MB
M_{small}	1
M_{normal}	4
U_{high}	95%
U_{low}	50%
B_{max}	512MB
Growth rate(σ)	2
p	16

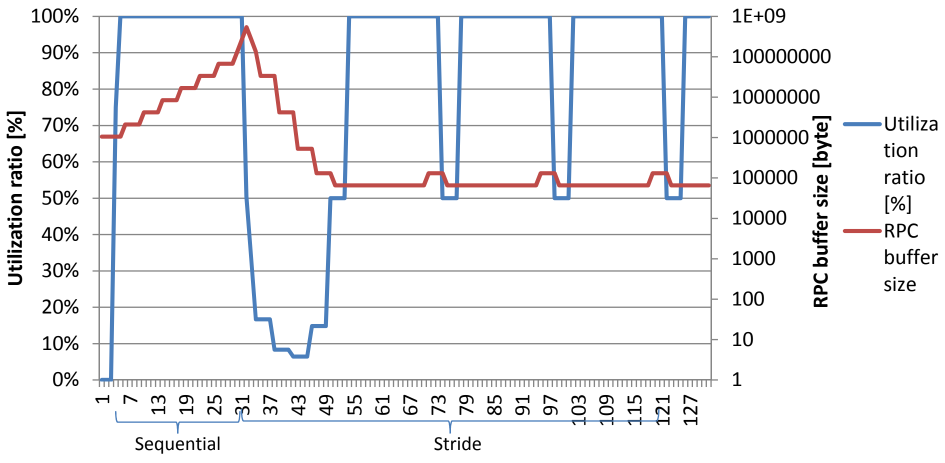

Figure 4.8: Behavior of the proposed method

The RPC buffer size increases if the RPC buffer utilization ratio is high for a long time (Line 14).In contrast, the random access pattern decrease the RPC buffer utilization ratio, and the RPC buffer size will be decreased (Line 18). Then, the RPC buffer utilization ratio raises when the RPC buffer size becomes an appropriate value; however, this behavior does not mean that the access pattern is sequential. The RPC buffer size will be increased by the procedure above, although it is an unexpected behavior when the access pattern is still not sequential. However, sequential access patterns can not be detected until the RPC buffer size is expanded. To combine the two behaviors, the method turns back the RPC buffer size to the previous value and fixes the RPC buffer size for next p RPC executions (Line 1 to 11). Smaller p enables the method to react quickly to sequential access patterns.

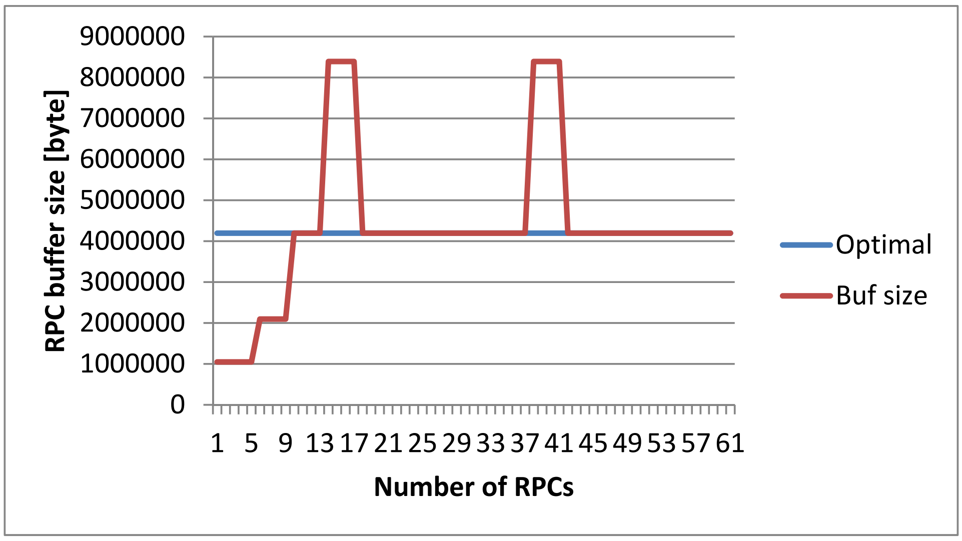

Figure 4.9: Behavior of the proposed method (stride: read 3MB and seek 6MB)

4.5.1 Behavior of the proposed method

This subsection describes how the proposed method changes the RPC buffer size dynamically. Table 4.1 shows those parameters. Fig. 4.8 shows the transition of the RPC buffer size and the RPC buffer utilization ratio in the case of a sequential access pattern (1 GB) and a stride access pattern. The X-axis is the count of RPCs issued, the left Y-axis is the RPC buffer utilization ratio[%], and the right Y-axis is the RPC buffer size[KB]. Until the 30th RPC execution, the RPC buffer utilization ratio is raised by the sequential access pattern and the RPC buffer size is also expanded. After that, the stride access pattern decreases the RPC buffer utilization ratio and the RPC buffer size. The RPC buffer utilization ratio will be 100% when the RPC buffer size is the optimal value. However, even if the RPC buffer utilization ratio is 100%, expanding the RPC buffer size decrease the RPC buffer utilization ratio because it is not a sequential access pattern. Then, the method fixes the RPC buffer size next p times of RPC execution in order to keep the optimal state as long as possible.

Fig. 4.9 shows the optimal value and the RPC buffer size which is changed by the algorithm. The tested access pattern is the repetition of reading 3MB and seeking 6MB. The optimal value means the best RPC buffer size, this is taken from Fig. 4.5. The initial size of the RPC buffer size is 1 MB and it increases to 4 MB, which is the optimal value in this case.

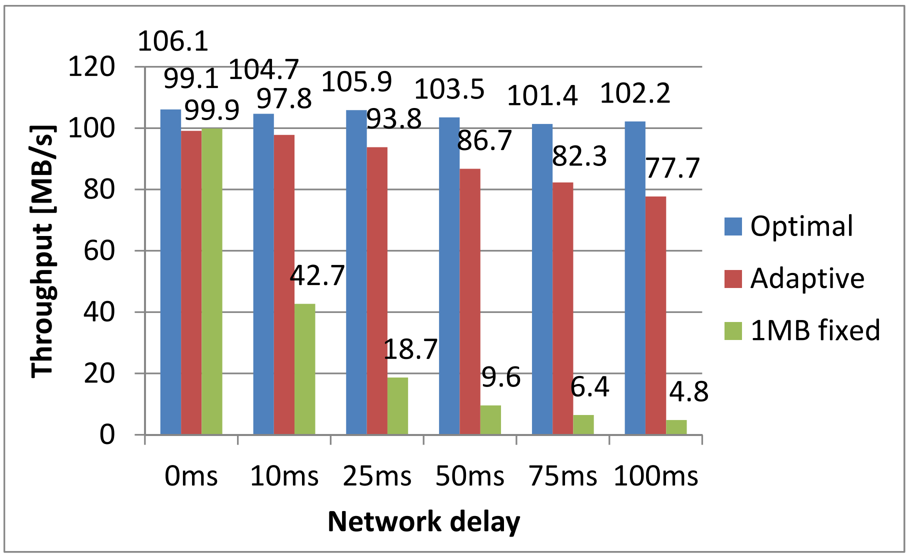

Figure 4.10: Performance of sequential access

Table 4.2: Selected RPC buffer size (sequential access)

Network delay	Adaptive	Optimal
0ms	512MB	128MB
10ms	512MB	256MB
25ms	512MB	512MB
50ms	512MB	512MB
75ms	512MB	512MB
100ms	512MB	512MB

4.6 Performance evaluation

This section describes the performance evaluation results of the proposed method. The evaluation environment is the same as described in section 4.3.2.

4.6.1 Sequential access

Fig. 4.10 depicts a performance comparison of sequential access patterns. The X-axis is the network delay and the Y-axis is a measured throughput of a 6 GB file transfer. The Table 4.2 shows the optimal RPC buffer sizes and the selected RPC buffer sizes by the proposed method under each network delay condition. These "Optimal" results were collected from the measured performance results with given the RPC buffer size values from 64 KB to 512 MB. "Optimal" in the graph is the performance result of when the RPC buffer size is set to the corresponding value in the Table 4.2. "Adaptive" is the performance with the

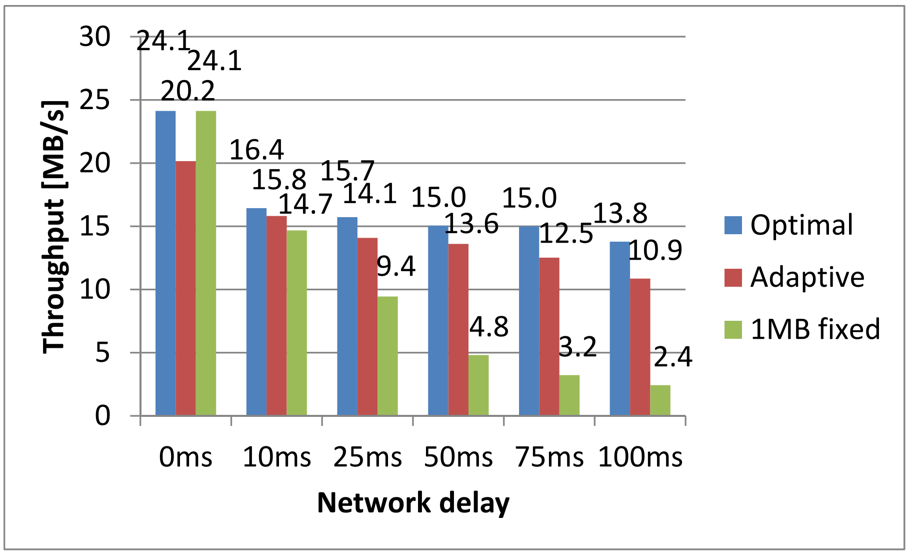

Figure 4.11: Performance of stride access(read 512KB and seek 3.5MB)

Table 4.3: Selected RPC buffer size in stride access(read 512KB and seek 3.5MB)

Network delay	Adaptive	Optimal
0ms	512KB	1MB
10ms	4MB	4MB
25ms	512MB	512MB
50ms	512MB	512MB
75ms	512MB	512MB
100ms	512MB	512MB

proposed method and "1MB fixed" is the result of the case where the RPC buffer size is fixed to 1MB, which is an example of conventional systems. In the case of "1MB fixed", the throughput is degraded by the network delay while "Adaptive" keeps the high throughput.

As shown in the Table 4.2, the proposed method selected the RPC buffer size, which is equal to the size of "Optimal" except with a low latency environment. For example, in the cases of 0 ms and 10 ms, the measured performance is maximized when the RPC buffer size is 128MB or 256MB;however, the optimal value is 512MB. The reason for this behavior is that the proposed method only monitors the RPC buffer utilization ratio. The difference in the performance between "Optimal" and "Adaptive" is caused by the amount of time to increase the RPC buffer size. Therefore, this difference is not persistent with large files.

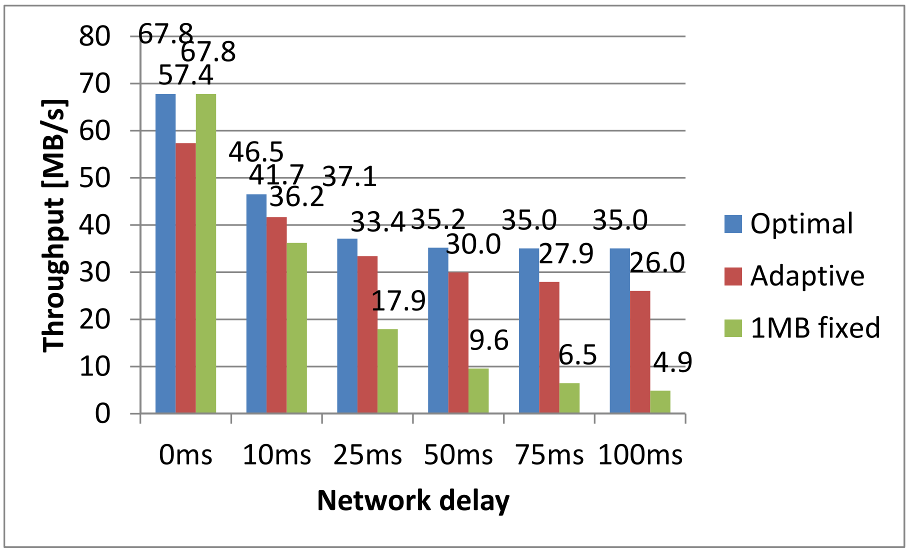

Figure 4.12: Performance of stride access(read 3MB and seek 6MB)

Table 4.4: Selected RPC buffer size in stride access(read 3MB and seek 6MB)

Network delay	Adaptive	Optimal
0ms	1MB	1MB
10ms	4MB	4MB
25ms	4MB	4MB
50ms	512MB	512MB
75ms	512MB	512MB
100ms	512MB	512MB

4.6.2 Stride access

Fig. [4.11, 4.12] are the results of the performance evaluation with stride access patterns. The X-axis and the Y-axis are the same to those of the sequential access evaluation. The performance is better than that of "1MB fixed" under almost all conditions with the proposed method.

Table 4.3 shows that the proposed method selected the same value as "Optimal" except for the case of the stride access (512KB reading) with the 0 ms network delay.

4.6.3 Overhead of optimization

If the number of blocks (n) is 4096, the proposed method requires 512 (4096/8) bytes of memory to save which block was used. In addition, it requires 64 bytes of memory to save the 8 histories to calculate the average utilization ratio.

The actual measured time to determine the RPC buffer utilization ratio and

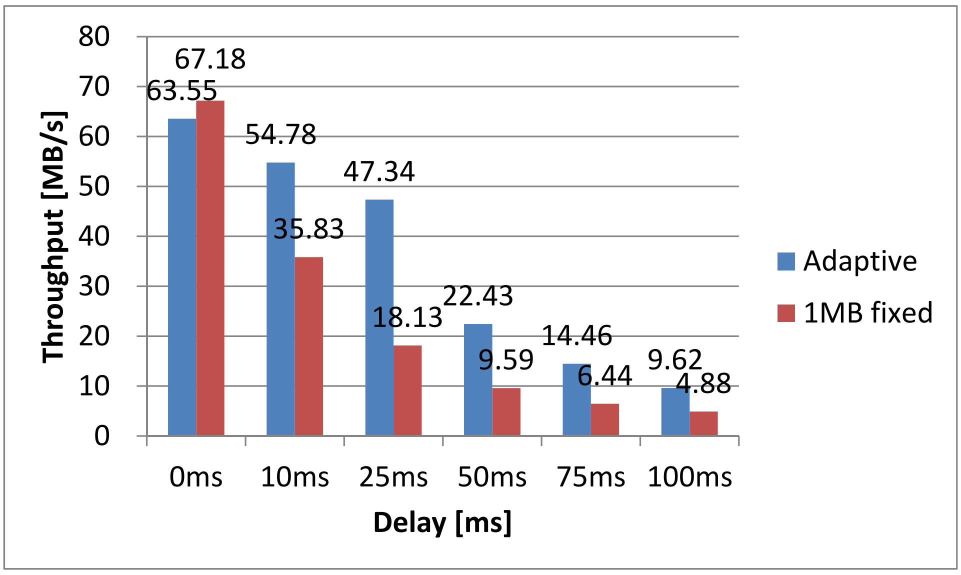

Figure 4.13: Performance comparison of mixture of sequential access and stride access

size was 160 microseconds per RPC execution. The overhead is very small compared to data transfer operations themselves. Thus, the proposed method can apply to remote file access without minimal disadvantage.

4.6.4 Mixed access pattern

Fig 4.13 describes the throughput with the mixed access pattern of sequential accesses (50 MB) and 25 iterations of the combination of read (2 MB) and seek (4 MB). The proposed method can achieve higher performance with the mixed access pattern. This performance improvement shows that the proposed method can adapt to not only a single access pattern but also mixed multiple data access patterns.

4.7 Conclusion of this chapter

The evaluation results showed that the proposed method can improve the performance of remote file access under various conditions. For example, the proposed method can handle the stride access 16 times faster than the implementation with the fixed RPC buffer size on the 100 ms delay network. The proposed method used only one parameter: the RPC buffer utilization ratio, which can consider the network bandwidth, the network latency and the access pattern. The advantage of this method is the ability to consider many factors with a simple way. Additionally, the overhead is small to apply to the real system. Therefore, this method can improve the performance of the common remote file access systems which use synchronous RPCs. Future work is the benchmark with real workloads, which is helpful to know the real value of the proposed method.

Chapter 5

Conclusion

5.1 Summary

This study defined an important problem of the existing distributed storage systems as the trade-off between performance and reliability. Existing methods for storage redundancy bring about performance degradation. This study revealed that the performance degradation is caused by the additional data blocks used for redundancy. These additional data blocks increase the network traffic of the writer node, whose network constitutes a critical path in dominant cases. This study proposed two different data-processing pipeline architectures for reliable and high-performance distributed storage systems. These methods offload the parity-generation processes to storage servers or network switches, to prevent the storage systems from degrading their performances. Both methods utilized the data-processing abilities of the storage nodes and the programmable functions of the future network switches. Existing studies never utilized these computing resources for reliable storage systems, as done in this study. This study also proposed an additional optimization mechanism for wide-area environments, which considered three important factors at the same time.

The first method, the "Active-storage mechanism" successfully utilized the CPUs and the network devices of the storage nodes to generate parity blocks by building data-processing pipelines across multiple storage nodes. This mechanism can off-load the parity-generation processes to the storage servers. Typically, the storage servers only receive data blocks from the writer nodes passively, whereas; this method utilized the storage servers actively, to generate parity blocks. The implementation of the prototype system utilized RDMA functions to minimize the overhead, and successfully proved the advantages of the proposed method. None of the existing studies achieved a reliable distributed storage system, without performance degradation; however, the evaluation results of this study revealed that this method could reduce overheads.

The second method “Network-based Data Processing Architecture for Reliable and High-performance Distributed Storage System” utilized the programmable functions of the future network switches. This method off-loaded the parity-generation processes to network switches to reduce the performance degradation caused by the increased traffic. The future programmable network switch is not available as on January 2016; however, the evaluation was conducted with the software implementation, and the results showed that the method could improve the performance of reliable storage systems. In addition, the proposed architecture used only simple data-processing pipelines, which could be implemented as hardware mechanisms.

The third method “A Method to Optimize Remote File Access Adapting to Access Pattern and Network Delay” was an adaptive optimization method for remote file access in wide-area environments. The method could consider the network bandwidth, the access patterns, and the network latency, simultaneously. This study examined the effects of these three factors with detailed preliminary evaluations, and the proposed method could figure out the optimal parameters for the given environments.

These three methods are not for a specific system layer and can be applied to any system layer. For example, they can be applied to the block-device layer or the file-system layer. The combination of these proposed methods has the possibility to change the design of distributed storage systems, since it was provided to possess the advantage of breaking the trade-off between performance and reliability.

5.2 Future work

The study proposed two methods for reliable and high-performance storage systems, and one additional mechanism; however, they have not been implemented as a complete file system. Moreover, the proposal to utilize the programmable network switches can be evaluated again when the hardware implementations of the switches are available. Although the performance evaluation results show the advantages of proposed methods, it is necessary to evaluate them with real workloads or well-known benchmark programs in order to know the real impact on the existing systems. In addition, these proposed methods can work well together because they have different advantages. A method of building a hybrid architecture using these proposed methods will also be of importance in the future.

The existing systems have started to replace the replication mechanisms with efficient codes. However. they might have performance-degradation problems related to code-generation processes. The proposals of this study can act as key ideas for future efficient and reliable storage systems of the future.

Acknowledgements

First, I would like to express my deepest gratitude to my advisor, Professor Osamu Tatebe, for the support in research activities. His continuous support made my research fruitful. In addition, he gave me numerous opportunities to join international conferences and workshops, which helped to broaden my outlook and to learn how to cooperate with international research communities.

Besides my advisor, I would like to thank Professor Taisuke Boku, Professor Kenjiro Taura, Associate Professor Shuichi Oikawa, and Dr. Yusuke Tanimura for their insightful comments and feedback regarding this thesis.

My gratitude extends to Professor Mitsuhisa Sato, Professor Daisuke Takahashi, Associate Professor Yoshiki Yamaguchi, Lecturer Hideyuki Kawashima, Assistant Professor Hiroto Tadano, Dr. Hiroaki Umeda, Dr. Kazuya Matsumoto, Dr. Masahiro Tanaka, and Dr. Mohamed Jabri for their support and inspiration.

I also thank all students in the HPCS laboratory, especially the students in the Big Data team, for their hard work in maintaining the experiment cluster nodes, and for the instructive discussions during the weekly meetings.

Finally, I would like to thank my parents for their continuous support and encouragement.

Bibliography

[1] Intel Solid-State Drive Data Center P3608 Series. http://www.intel.com/content/www/us/en/solid-state-drives/solid-state-drives-dc-p3608-series.html.

[2] Tony Hey, Stewart Tansley, and Kristin Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, Washington, 2009.

[3] A. Gharaibeh, S. Al-Kiswany, and M. Ripeanu. ThriftStore: Finessing Reliability Trade-Offs in Replicated Storage Systems. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):910–923, June 2011.

[4] David A. Patterson, Garth Gibson, and Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *SIGMOD Rec.*, volume 17, pages 109–116. ACM, June 1988.

[5] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: High-performance, Reliable Secondary Storage. In *ACM Computing Surveys*, volume 26, pages 145–185, 1994.

[6] A. Nisar, Wei keng Liao, and A. Choudhary. Delegation-Based I/O Mechanism for High Performance Computing Systems. In *IEEE Transactions on Parallel and Distributed Systems*, volume 23, pages 271–279, 2012.

[7] DDN®IME14K$^{\mathrm{TM}}$. http://www.ddn.com/products/infinite-memory-engine-ime14k/.

[8] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. In *Journal of network and computer applications*, volume 23, pages 187–200, 1999.

[9] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A Scalable, High-performance Distributed File Sys-

tem. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 307–320. USENIX Association, 2006.

[10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE Computer Society, 2010.

[11] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. In *Journal of the Society for Industrial & Applied Mathematics*, volume 8, pages 300–304. SIAM, 1960.

[12] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design Tradeoffs for SSD Performance. In *USENIX 2008 Annual Technical Conference*, ATC'08, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[13] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[14] Hakim Weatherspoon and John Kubiatowicz. Erasure Coding Vs. Replication: A Quantitative Comparison. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 328–338. Springer-Verlag, 2002.

[15] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *SIGPLAN Notices*, volume 35, pages 190–201, November 2000.

[16] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total Recall: System Support for Automated Availability Management. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, pages 25–25, Berkeley, CA, USA, 2004. USENIX Association.

[17] RedHat. Gluster FS, http://www.gluster.org/.

[18] Osamu Tatebe, Kohei Hiraga, and Noriyuki Soda. Gfarm Grid File System. In *New Generation Computing, Ohmsha, Ltd. and Springer*, volume 28, pages 257–275, 2010.

[19] Hadoop. http://hadoop.apache.org/.

[20] Bin Fan, Wittawat Tantisiriroj, Lin Xiao, and Garth Gibson. DiskReduce: RAID for Data-intensive Scalable Computing. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, PDSW '09, pages 6–10. ACM, 2009.

[21] Facebook's Realtime Distributed FS based on Apache Hadoop. https://github.com/facebookarchive/hadoop-20.

[22] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. XORing elephants: novel erasure codes for big data. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 325–336. VLDB Endowment, 2013.

[23] Yuchong Hu, Henry C. H. Chen, Patrick P. C. Lee, and Yang Tang. NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-clouds. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 21–21. USENIX Association, 2012.

[24] Chentao Wu, Xubin He, Jizhong Han, Huailiang Tan, and Changsheng Xie. SDM: A Stripe-Based Data Migration Scheme to Improve the Scalability of RAID-6. In *Proceedings of 2012 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 284–292, Sept 2012.

[25] Guangyan Zhang, Keqin Li, Jingzhe Wang, and Weimin Zheng. Accelerate RDP RAID-6 Scaling by Reducing Disk I/Os and XOR Operations. In *IEEE Transactions on Computers*, volume 64, pages 32–44, 2015.

[26] Pei Cao, Swee Boon Lim, Shivakumar Venkataraman, and John Wilkes. The TickerTAIP Parallel RAID Architecture. In *SIGARCH Comput. Archit. News*, volume 21, pages 52–63, New York, NY, USA, May 1993. ACM.

[27] Y. Birk and E. Zilber. TPT-RAID: a High Performance Box-Fault Tolerant Storage System. In *24th IEEE Conference on Mass Storage Systems and Technologies(MSST)*, pages 215–220, 2007.

[28] Philip H. Carns, Walter B. Ligon, III, Robert B. Ross, and Rajeev Thakur. PVFS: A Parallel File System for Linux Clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327. USENIX Association, 2000.

[29] Jiesheng Wu, Pete Wyckoff, and Dhabaleswar K. Panda. PVFS over InfiniBand: Design and Performance Evaluation. In *The 2003 International Conference on Parallel Processing (ICPP 03)*, pages 125–132, 2003.

[30] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Network File System (NFS) version 4 Protocol. RFC 3530 (Proposed Standard), April 2003.

[31] Brent Callaghan, Theresa Lingutla-Raj, Alex Chiu, Peter Staubach, and Omer Asad. NFS over RDMA. In *Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: experience, lessons, implications*, NICELI '03, pages 196–208. ACM, 2003.

[32] Anurag Acharya, Mustafa Uysal, and Joel Saltz. Active Disks: Programming Model, Algorithms and Evaluation. *SIGPLAN Not.*, 33(11):81–91, October 1998.

[33] Patrick Donnelly and Douglas Thain. Design of an Active Storage Cluster File System for DAG Workflows. In *Proceedings of the 2013 International Workshop on Data-Intensive Scalable Computing Systems*, DISCS-2013, pages 37–42, New York, NY, USA, 2013. ACM.

[34] Philipp Reisner. DRBD v8 Replicated Storage with Shared Disk Semantics. In *Proceedings of the 12th International Linux System Technology Conference*, pages 1–11, 2005.

[35] P.T. A. Marin Lopez, Arturo Garcia Ares. The Network Block Device. In *Linux Journal*, volume 2000. Belltown Media, 2000.

[36] OpenFabrics. http://beany.openfabrics.org/.

[37] P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Sockets Direct Protocol over InfiniBand in Clusters: Is It Beneficial? In *Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS '04, pages 28–35. IEEE Computer Society, 2004.

[38] N. S. Islam, M. W. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda. High Performance RDMA-based Design of HDFS over InfiniBand. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 35:1–35:35. IEEE Computer Society Press, 2012.

[39] Mellanox Technologies. RDMA aware programming user manual, http://www.mellanox.com/related-docs/ prod_software/RDMA_Aware_Programming_user_manual.pdf.

[40] Omer Arap, Geoffrey Brown, Bryce Himebaugh, and Martin Swany. Software Defined Multicasting for MPI Collective Operation Offloading with the NetFPGA. In *Euro-Par 2014 Parallel Processing*, volume 8632 of *Lecture Notes in Computer Science*, pages 632–643. Springer International Publishing, 2014.

[41] J.W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and Jianying Luo. NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing. In *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on*, pages 160–161, June 2007.

[42] Mellanox. CORE-Direct The Most Advanced Technology for MPI/SHMEM Collectives Offloads. *http://www.mellanox.com/related-docs/whitepapers/TB_CORE-Direct.pdf.*

[43] D. Dalessandro and P. Wyckoff. Memory Management Strategies for Data Serving with RDMA. In *High-Performance Interconnects, 2007. HOTI 2007. 15th Annual IEEE Symposium on*, pages 135–142, Aug 2007.

[44] B. Callaghan, B. Pawlowski, and P. Staubach. NFS Version 3 Protocol Specification. In *RFC 1813*, 1995.

[45] John H. Howard. Scale and performance in a distributed file system. In *ACM Trans. Computer Systems*, volume 6, pages 51–81, 1988.

[46] M. Satyanarayanan. Coda: A Highly Available File System for a Distributed Workstation Environment. In *IEEE Trans. Computers*, volume 39, pages 447–459, 1990.

[47] P. J. Braam. Lustre, http://www.lustre.org/.

[48] Stephen C. Simms, Gregory G. Pike, and Doug Balog. Wide Area Filesystem Performance using Lustre on the TeraGrid. In *Proceedings of the TeraGrid 2007 Conference*, 2007.

[49] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959, October 1985.

[50] Takeshi Ito, Hiroyuki Ohsaki, and Makoto Imase. GridFTP-APT: Automatic Parallelism Tuning Mechanism for Data Transfer Protocol GridFTP. *IEEE*

International Symposium on Cluster Computing and the Grid, 0:454–461, 2006.

[51] Tara M. Madhyastha, Christopher L. Elford, and Daniel A. Reed. Optimizing Input/Output Using Adaptive File System Policies. In *Proceedings of the Fifth Goddard Conference on Mass Storage Systems and Technologies*, pages 493–514, 1996.

[52] Rajeev Thakur, William Gropp, and Ewing Lusk. Data Sieving and Collective I/O in ROMIO. In *Proceedings of the Seventh Symposium on the Frontiers of Massively Parallel Computation*, pages 182–189. IEEE Computer Society Press, 1988.

Appendix A

List of Publications

Journal Papers (in Japanese)

1. 大辻弘貴，建部修見,「アクセスパターンと回線遅延を考慮した遠隔ファイルアクセスの最適化」，論文誌コンピューティングシステム (ACS)，情報処理学会，Vol. 4，No 4，pp.122-134，2011 年 10 月

Conference Papers (Refereed)

1. Hiroki Ohtsuji and Osamu Tatebe, "Network-based Data Processing Architecture for Reliable and High-performance Distributed Storage System", Euro-Par 2015: Parallel Processing Workshops, Lecture Notes in Computer Science, Vol. 9523, pp.16-26, 2015 (DOI: 10.1007/978-3-319-27308-2_2)

2. Hiroki Ohtsuji and Osamu Tatebe, "Active-Storage Mechanism for Cluster-wide RAID System", Proceedings of IEEE International Conference on Data Science and Data Intensive Systems (DSDIS), pp.25-32, 2015 (DOI: 10.1109/DSDIS.2015.101)

Conference Papers (Refereed short papers)

1. Hiroki Ohtsuji and Osamu Tatebe, "Server-side Efficient Parity Generation for Cluster-wide RAID System", Proceedings of 7th IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp.444-447, 2015 (DOI: 10.1109/CloudCom.2015.25)

Posters (Refereed)

1. Hiroki Ohtsuji and Osamu Tatebe, “ POSTER: Preliminary evaluation of optimized transfer method for cluster-wide RAID-4 ”, Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), pp.284-285, 2014 (DOI: 10.1109/CLUSTER.2014.6968775)

Symposium Papers (in Japanese, Refereed)

1. 大辻弘貴，建部修見,「アクセスパターンと回線遅延を考慮した遠隔ファイルアクセスの最適化」，先進的計算基盤システムシンポジウム (SACSIS2011) 論文集，2011 年 5 月

List of Other Publications (in Japanese)

1. 大辻弘貴，建部修見,「Cluster-wide RAID 向けの集中型コントローラ」，情報処理学会ハイパフォーマンスコンピューティング研究会 (HPC147, HOKKE22), 2014 年 12 月

2. 大辻弘貴，建部修見,「Cluster-wide RAID の実装と評価」，情報処理学会ハイパフォーマンスコンピューティング研究会 (HPC145, SWoPP2014)，2014 年 8 月

3. 大辻弘貴，建部修見,「分散ストレージシステムに対する低オーバヘッド冗長化書き込み手法の提案と評価」，情報処理学会ハイパフォーマンスコンピューティング研究会 (HPC142, HOKKE21)，2013 年 12 月

4. 大辻弘貴，建部修見,「RDMA による低オーバヘッドファイルアクセスと冗長記録」，情報処理学会ハイパフォーマンスコンピューティング研究会 (HPC137, HOKKE20)，2012 年 12 月

5. 大辻弘貴，建部修見,「Infiniband を用いたファイルアクセスの高速化」，情報処理学会ハイパフォーマンスコンピューティング研究会 (HPC135, SWoPP2012), 2012 年 8 月

6. 大辻弘貴，建部修見,「Non-blocking RPC を用いた遠隔ファイルアクセスの実装と性能評価」，情報処理学会ハイパフォーマンスコンピューティング研究会 (HPC132, HOKKE19)，2011 年 11 月

7. 大辻弘貴，建部修見,「Non-blocking RPC を用いた遠隔ファイルアクセスの最適化」，情報処理学会ハイパフォーマンスコンピューティング研究会 (HPC130, SWoPP2011)，2011 年 8 月

8. 大辻弘貴，建部修見,「遠隔ファイルアクセスにおけるアクセスパターンと遅延の影響」，情報処理学会ハイパフォーマンスコンピューティング研究会 (HPC128, HOKKE18)，2010 年 12 月

List of Presentations and Posters

1. Hiroki Ohtsuji and Osamu Tatebe, "Breaking the Trade-off between Performance and Reliability of Network Storage System ", PRAGMA29 (Poster), Depok, Indonesia, Oct. 2015
2. Hiroki Ohtsuji and Osamu Tatebe, "Pipelined Data Processing Architecture for Network Storage Systems", PRAGMA28 (Poster), Nara, Apr. 2015
3. Hiroki Ohtsuji and Osamu Tatebe, "Network-based Storage Architecture for Exa-scale Computing Systems", PRAGMA27 (Poster), Bloomington(IN), Oct. 2014
4. Hiroki Ohtsuji and Osamu Tatebe, "Towards 100% Solar-powered Exa-scale Computing", PRAGMA26 (Poster), Tainan, Apr. 2014
5. Hiroki Ohtsuji, "Building a Single-sided Communication Layer for Exa-scale Storage Systems", Argonne National Laboratory Seminar Series, Argonne, Aug. 2013
6. Hiroki Ohtsuji and Osamu Tatebe, "High-throughput Remote File Access for Exa-scale Storage Systems", IWCST2013, Hangzhou, Oct, 2013
7. Hiroki Ohtsuji and Osamu Tatebe, "High Throughput, Low latency and Reliable Remote File Access", PRAGMA24 (Poster), Bangkok, Mar. 2013
8. Hiroki Ohtsuji and Osamu Tatebe, "Remote File Access with Infiniband RDMA", PRAGMA23 (Poster), Seoul, Oct. 2012
9. Hiroki Ohtsuji and Osamu Tatebe, "High Throughput Remote File Access with Infiniband", PRAGMA22 (Poster), Melbourne, Apr. 2012
10. Hiroki Ohtsuji and Osamu Tatebe, "Optimization of Remote File Access Considering Access Pattern and Network Delay", PRAGMA20, HongKong, Mar. 2011

Awards and Projects

1. 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻 専攻長表彰, 2013 年 3 月
2. 第二回 HPC アイデア・コンテスト 最優秀賞, 2014 年 3 月
3. 日本学術振興会特別研究員 (DC2), 「次世代高性能計算機に向けたネットワーク指向ストレージシステムの研究」, 2014 年 4 月-2016 年 3 月
4. PRAGMA29 Workshop, Best Technical Talk, 2015 年 10 月