A Study on Reliability and Performance Improvement of Distributed System for Large-Scale Data Processing

July 2012

Yasuhiro KIRIHATA

A Study on Reliability and Performance Improvement of Distributed System for Large-Scale Data Processing

Submitted to
Graduate School of Information Science and Technology
Osaka University

July 2012

Yasuhiro KIRIHATA

List of Publications

A. Journal Papers

1. Yasuhiro Kirihata, Jason Leigh, Chaoyue Xiong, and Tadao Murata: “A Sort-Last Rendering System over an Optical Backplane,” *Journal of Systemics, Cybernetics and Informatics*, Vol.3, No.3, pp.63-69 (2005).

2. Yasuhiro Kirihata, Yoshiki Sameshima, Takashi Onoyama, and Norihisa Komoda: “Data Loss Prevention for Confidential Web Contents and Security Evaluation with BAN Logic,” *International Journal of Computers*, Vol.5, No.3, pp.414-422 (2011).

3. Yasuhiro Kirihata, Yoshiki Sameshima, Takashi Onoyama, and Norihisa Komoda: “WriteShield: A Pseudo Thin-Client for Prevention of Information Leakage,” *IEEJ Transactions on Electronics, Information and Systems*, Vol.132, No.2, pp.253-259 (2012).

4. Yasuhiro Kirihata, Takashi Onoyama, and Norihisa Komoda: “A Slot-based Virtual Node Method of Consistent Hashing for Optimization of Index Reconfiguration on Distributed Search,” *IEEJ Transactions on Electronics, Information and Systems*, Vol.132, No.10 (2012) (to be published).

B. International Conference Papers

1. Yasuhiro Kirihata and Yoshiki Sameshima: “A Web-based System for Prevention of Information Leakage,” The 11th International World Wide Web Conference (WWW2002), poster-127 (2002).

2. Yasuhiro Kirihata, Jason Leigh, Chaoyue Xiong, and Tadao Murata: “A Sort-Last Rendering System over an Optical Backplane,” in *Proceedings of the 10th International Conference on Information Systems Analysis and Synthesis (ISAS 2004) and the International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA 2004)*, Vol.1, pp.42-47 (2004).

3. Yasuhiro Kirihata, Yoshiki Samashima, and Takashi Onoyama: “WriteShield: A Pseudo Thin-Client System for Prevention of Information Leakage,” in *Proceedings of the 8th IEEE International Conference on Industrial Informatics (INDIN2010)*, pp.478-483 (2010).

4. Yasuhiro Kirihata, Yoshiki Sameshima, Takashi Onoyama, and Norihisa Komoda: “A Novel Approach for Protection of Confidential Web Contents,” in *Proceedings of the European Computing Conference (ECC’11)*, pp.110-115 (2011).

C. Domestic Conference Papers

1. Yasuhiro Kirihata and Yoshiki Sameshima: “A Web-based System for Prevention of Information Leakage,” in *Proceedings of the 61st Annual Convention IPS Japan – Distributed Systems and Security* (2000) (in Japanese).

2. Koji Nakayama, Yasuhiro Kirihata, and Yoshiki Samashima: “A Process-based Write Control for Software Update on the Pseudo Thin Client PC,” in *IPSJ SIG Notes on Computer Security Group*, Vol.2010, No.5, pp.1-6 (2010) (in Japanese).

Summary

This dissertation is a study on reliability and performance improvement of distributed system for large-scale data processing researched through 1999 to 2012 by the author who is enrolled at Hitachi Solutions, Ltd. and Graduate School of Information Science and Technology, Osaka University.

Recently, as the growth rate of data is getting larger, technologies of storing and processing data at low cost are essential in enterprises and organizations. Because the prices of high-performance commodity servers are going down, it is more popular to construct a cluster or a distributed system of those machines for parallel and distributed data processing of the large-scale data set. In those distributed systems, security, performance, scalability, and availability are principal measures of the system.

In terms of the security perspective, the thin client system is one of the effective security methods for the distributed systems dealing with a great number of files in the central file servers. However, it does not diffuse because of its expensive introduction and operational cost. For this reason, the methodology for construction of low-cost thin client system is required. In the high-performance perspective, technologies for batch and real-time processing on the cluster of commodity machines are significant. As for the batch processing, one of the important applications is the high-performance parallel generation of segmented indexes for the distributed search system. The resource saving and performance improvement of the index generation and reconfiguration are strongly required. Furthermore, as a representative example of the real-time processing, the parallel rendering on the PC cluster is a significant application. As the image data has high resolution and fine-grained quality with a great number of polygons, it is essential to realize the high performance rendering of the large-scale three dimensional images.

Based on this background, we address the following three issues in this dissertation.

The first one is realization of the low-cost thin client for data protection on the large number of PC terminals. The second one is the resource saving and performance improvement of the segmented index generation and reconfiguration. The third one is the realization of the high-performance rendering of large-scale three-dimensional image data. This dissertation is composed of the following five chapters.

Chapter 1 describes the trend of technologies related to the distributed system that consists of commodity servers for large-scale data and clarified its significant issues. Additionally, we explain the related works in terms of technologies relevant to the individual issues and our research strategies.

Chapter 2 proposes a reliable and secure pseudo thin client method based on the write protection of local disks with virtual cache mechanism. We explain our proposed pseudo thin client system called "WriteShield" that solves the issues of conventional methods. The fundamental system architecture and virtual cache mechanism of the write-protection driver are described. Next the three paging algorithms are evaluated using actual disk I/O logs and the prototyped simulation code to select the optimal one. Finally, we evaluate the disk I/O performances during the running of write-protection mechanism and paging process of virtual cache mechanism respectively.

Chapter 3 proposes the slot-based virtual node method of consistent hashing for optimal index generation and reconfiguration. We explain the conventional consistent hashing method and its issues in terms of the memory resources and the index reconfiguration time. Then, our proposed method is described. We define two data processing models of index reconfiguration for both conventional and our new methods. We evaluate the memory usage of our method with the prototype and estimate the lapse times of reconfiguration processes according to the defined data processing models of both conventional and new methods.

In Chapter 4, the sort-last rendering method over the cluster based on the photonic switch is discussed to realize the high-performance rendering of large-scale three-dimensional image data. We describe the issue of switching delay for application of the photonic switch based cluster to the parallel rendering and design the architecture of the sort-last rendering cluster based on the photonic switch to solve the issue. We evaluate the rendering performance of single image and stream process performance, and discuss the feedback mechanism to control queuing and sending messages to achieve the optimal smooth data flow inside the cluster.

Finally, Chapter 5 concludes this study and presents our future directions.

Contents

Chapter 1 Introduction

1.1 Background

Nowadays, as the number of terminal devices and network throughputs are getting greater, the amount of data is increasing exponentially. The recent investigation of International Data Corporation (IDC) indicates that unstructured data accounts for 80% of total data in enterprises and organizations. The growth rate of those data will reach over 60% per year until 2014. According to the "IDC Digital Universe Study" [1] published in May 2010, the total size of digital data existing in the world is 0.8 zettabytes (0.8×2^{70} bytes) in 2009. It is also said that its increasing rate will be 40% per year to reach 35.2 zettabytes (35.2×2^{70} bytes) in 2020.

Popularization of smart phones such as iPhone and Android also accelerates the explosive growth of data. Cisco Systems predicts that the data traffic of mobile environment increases 26 times larger in the next 5 years and it will reach to 6.3 exabytes (6.3×2^{60} bytes) per month in 2015 [2]. Previously, the digital data is mainly generated by humans through computers. In the future, there will be a remarkable increase in the digital data generated by various devices and sensors such as equipments in plants, heavy machineries, car navigation systems, vending machines, monitoring cameras, air conditioning management systems in the buildings, POS (Point of Sale) terminals, and so on.

As the growth rate of data is getting larger, the technologies of storing and archiving data at low cost are becoming essential. The effective reutilization of the large-scale data is getting significant to generate new valuable information. Price down and performance improvement of commodity machines promote the research and development activities in terms of storing and processing the large-scale data set on the PC (Personal Computer) cluster.

The representative researches on the massive data processing on the PC cluster are Google File System [3] and MapReduce [4]-[8] which are developed by Google Inc. to realize the high performance index generation and data archiving for the Web search system. Hadoop [9][10], which is a top-level Apache project and implements Google's MapReduce framework, is also a popular project applied to various fields to execute batch processes and analyze large-scale data set such as web access logs, corpus data, remote sensing data, DNA (Deoxyribo Nucleic Acid) sequence data, POS data, and so on [11]-[13]. The tight coupling parallel processing system such as MPICH [14]-[17] and LAM [18]-[21] still has needs. In addition, the stream data processing such as CEP (Complex Event Processing) [22]-[25] is also focused on recently as the on-memory data computing for massive data set instead of batch processing objective like Hadoop.

The distributed file systems such as GlusterFS [26]-[27] and GPFS [28]-[30] to construct the scale-out storage by commodity machines are also getting focused on recently. Key value stores such as Cassandra [31][32], HBase [33]-[36], CouchDB [37][38], MongoDB [39]-[41] are also getting popular because of their scalability. As the data is increasing enormously, the needs for those technologies on the cluster of commodity machines for storing and parallel processing of large-scale data will be greater in the future.

Fig. 1.1 shows the overview of the distributed systems dealing with large-scale data set in enterprises and organizations. In the data center, the front-end systems have several services such as Web applications, business applications, and file storages. The back-end system includes DB (Database) servers, normal PC clusters, and so on. The normal PC cluster is the cluster system constructed by multiple low-cost commodity servers connected over the high-speed networks. Parallel and distributed applications such as batch jobs, real-time processing applications, and distributed storage software are running on the cluster. Not only the cluster system of commodity machines but the C/S (Client/Server) system in which many client PCs accessing the large-scale data set stored in the storage through the front-end services is also regarded as the distributed system for large-scale data.

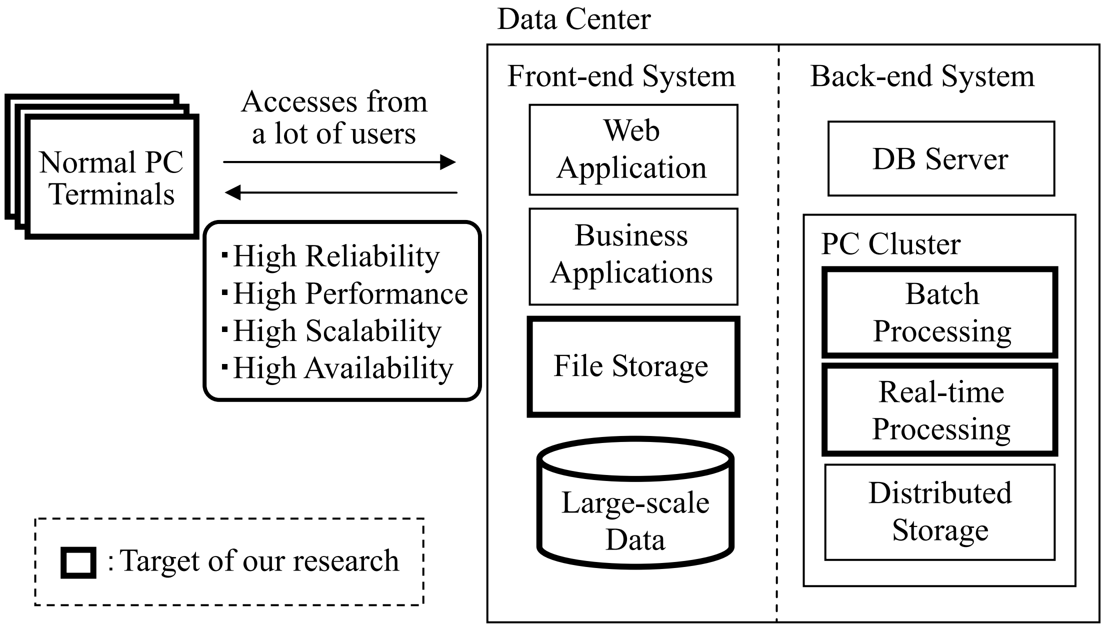


Fig. 1.1 Distributed systems for large-scale data

In this distributed system, security, performance, scalability, and availability are principal measures of the system. If the treated data includes confidential information such as personal information, customer information, and account information, it is necessary to introduce the appropriate backup system and data protection mechanism with access control and encryption method during the lifecycle of data from generation to retention, utilization, and deletion. The data protection and security are critical requirements of the system. In addition, as the target data is bigger, execution times of batch processing and real-time processing are clearly important measures for the system evaluation. Scalability of the distributed system means the possibility of the "linear" performance improvement by adding extra servers. The scalability of the system is also important to achieve the required performance easily at lower cost. Availability of the system is related to its reliability. If the provided services are critical, the system has to be working completely or partially even if some part of the system goes down. It is also important for the distributed system which deals with large-scale data. According to these points of view, we focus on the following three significant issues on the distributed system for large-scale data.

In terms of the security perspective on the distributed system, there are various data protection methods depending on target data type and applications. For instance, to protect the confidential web contents, one of the effective methods is to distribute special viewers to clients which prohibits functions of text copy, print, and snapshot of displayed data as to web applications [42]-[45]. As for the confidential files stored in the central file server, one of the effective security methods is the thin client system. Since the enforcement of the Personal Information Protection Law [46] of Japan in 2005, leakage of confidential information is one of the significant security issues in enterprises and organizations. The thin-client computing comes under the spotlight as one of the effective solutions for this issue. Thin client prevents from saving data in the local disks and restricts to save it in the central servers located in the data center. If the client terminal is lost or stolen, the confidential data itself is stored in the thin-client server or the file server and is neither lost nor leaked out. Thus, it is more secure than the conventional data encryption method on the local disks. However, it does not diffuse in spite of its efficient security features because it is expensive to introduce the special client terminals and back-end thin-client servers. To improve this situation, the thin client method using normal PC is required. In the viewpoint of the reliability and security on the large-scale distributed systems, we target the realization of the low-cost thin client for data protection on the large number of PC terminals as the first issue of this research.

In the high-performance perspective, the batch processing on the cluster of commodity machines is also an important technology. Because the unstructured data such as documents, images, and movies is increasing rapidly, the enterprise search technology is getting significant to reuse those data efficiently in enterprises and organizations. In order to increase the frequency of the update for the large number of data, the high-performance indexing method is essential in a parallel fashion. Additionally, the high-speed index reconfiguration is also very important as the amount of treated data is greater. The high performance batch processing technologies with the parallel distributed cluster system constructed by the commodity machines is required so much. Accordingly, we target the resource saving and performance improvement of the segmented index generation and reconfiguration derived from the needs of the search and reutilization of the large number of unstructured files as the second issue of this research. In this issue, we set the measure of evaluation to the memory usage and lapse time of index reconfiguration process.

Furthermore, as an example of the real-time processing on the commodity PC based cluster, there is a parallel rendering system. As the computer graphics is getting higher quality and resolution in CAD (Computer Aided Design) data and entertainment contents, it is required to realize the high performance rendering of the large-scale three dimensional images consisting of a lot of polygons. Corresponding to these needs, the real-time parallel rendering technology for the large-scale data set is required based on the low-cost commodity machines. Therefore, we target the realization of the high-performance rendering of large-scale three-dimensional image data on the PC cluster as the third issue of this research.

To summarize the above points, we address the following three issues in this dissertation. The first issue is the realization of the low-cost thin client for data protection on the large number of PC terminals. The second issue is the resource saving and performance improvement of the segmented index generation and reconfiguration. The third issue is the realization of the high-performance rendering of large-scale three-dimensional image data.

1.2 Related Works

1.2.1 Low-cost Thin Client for Data Protection on the Large Number of PC Terminals

There are venders which develop and release the products providing the solutions of the data loss prevention on client terminals. In those products, the thin client technology is one of the effective products to prevent from losing and leaking confidential data because of the lost or stolen of the client PCs. The following describes the conventional methods of thin clients.

Firstly, the server-based computing is one of the methods of thin clients. In this method, several users share the desktop environment of the single Windows Server, and they access the server with remote operation tools over RDP (Remote Desktop Protocol) or ICA (Independent Computing Architecture) protocols. In this protocol, the keyboard and mouse events are transferred from the client computer to the server, relaying the graphical screen images from the server to the client over the network. Representative systems of this type are Windows Terminal Service [47][48], XenDesktop [49][50], Sun Ray [51], X window system [52], and VNC [53]. However, the network delay seriously affects the transfer times of keyboard and mouse events to degrade the usability of GUI (Graphical User Interface).

The second one of the methods is the virtual PC method. In this method, the virtual server is adopted as the thin-client server. Compared with the server-based computing method in which several users share the single OS (Operating System) environment, this method provides each OS environment to each user. Every user can possess all OS resources on the assigned virtual server. If some users' applications freeze their OSes on the virtual servers, other users are not influenced by those errors. VMware vSphere [54] and XenDesktop are the representative systems adopting this method. However, there is a problem in terms of the usability of GUI if the network has a non-negligible delay, because it uses RDP as similar to the server-based computing method. Similarly, it is necessary to enhance the hardware on the server side. There is an issue on the introduction cost of the system. Furthermore, it is required to use the special client terminals instead of the normal PCs. It is also a problem that it is difficult to take a countermeasure against the computer viruses and apply OS patches to the client terminals.

Finally, the third one is the blade PC method [55]. In this method, we adopt the blade server instead of the virtual one in the previous virtual PC method. The advantage of this method is that each user can possess the hardware resources in the assigned blade PC. However, there is also a usability problem if network delay exists between the client terminal and blade PC as similar to the above two methods. Because every user has to be assigned the blade PC, it takes much cost of installation, setting-up location, and electricity for running the system.

On the above related works, they all need to set up the thin client servers in the data center and enhance the networks to hold the bandwidth for network accesses of consolidated servers. The introduction and operational costs are problems. In addition, usability of GUI over RDP and ICA protocols is affected by the network delay seriously. Our proposed approach solves these problems by using the commodity hardware without changing the existing system configuration.

1.2.2 Resource Saving and Performance Improvement of the Segmented Index Generation and Reconfiguration

In order to realize the high-performance information retrieval system for the large number of unstructured data, the distributed search is essential for multiple segmented indexes on the search cluster nodes. Crawling and indexing on the parallel computing environment are getting popular to improve the speed of the indexing process. It is necessary to have a mapping method among target files and segmented indexes for the distributed search system. The conventional mapping methods are as follows.

Firstly, the range partitioning method [97][98] is one of the applicable methods of mapping among files and segmented indexes. This method is the simplest one to divide the set of partitioning keys with the ranges of key values corresponding to the partitions. If this method is applied to the distributed search system, the data migrations among all nodes are executed when the cluster nodes are added or removed. In order to keep the equal amount of data distribution among nodes, all definitions of key value ranges should be changed.

Secondly, the hash partitioning method [97]-[99] is another candidate of mapping method. This method takes hash values of keys such as file paths and maps them to the segmented indexes to realize the mapping among target files and segmented indexes. As similar to the first method, this method has the problem of data migration cost because of the redundant movement of index data among the original cluster nodes in the index reconfiguration process.

Thirdly, there is the consistent hashing method [81]-[83] to realize the efficient mapping between keys and nodes. In this method, the multiple representative points for each node, called the virtual node, are plotted on the hash ring space at random positions. Given a key, the hash value of the key is calculated and plotted on the hash ring space. Then, the plotted point is moved along with the ring space in the anti-clockwise direction. The representative point which the moving point of the key reaches for the first time is that for the corresponding node. In order to uniform the distribution of key entries among the nodes, a lot of virtual nodes should be plotted on the hash ring space. This causes the huge memory utilization to load mapping data on the memory. Furthermore, many splitting processes are executed to be a bottleneck of the index reconfiguration process when the multiple nodes are added or removed.

Additionally, the distributed hash table such as Chord [56]-[59] can be applicable to the mapping process among files and segmented indexes. Chord algorithm actually provides the correspondence between the key and the node based on the consistent hashing and is one of its applications. In Chord, each node has the routing table called "finger table" to resolve the next node to transfer the query for the calculation of corresponding node to the given key. The maximum number of the skipping is at most *log(N)* where *N* is the number of nodes. In order to uniform the key entry distribution for nodes, virtual nodes are necessary as similar to the conventional consistent hashing. Thus, the memory resources are required if all nodes store the whole mapping information on the memory. Additionally, Chord does not provide any mapping reconfiguration methods to realize the efficient data migration among nodes.

On the above related works, they all do not provide the optimal solution for the administration of segmented indexes of distributed search system which requires memory resource saving and fast index generation and reconfiguration on the node adding or removing processes.

1.2.3 High-performance Rendering of Large-scale Three-Dimensional Image Data

In order to realize the high-performance rendering of large-scale three-dimensional image data effectively, there are three parallel rendering algorithms running on the cluster; Sort-fast, Sort-middle, and Sort-last methods [60]-[63].

Firstly, the sort-fast rendering method is a parallel rendering method in which the output display region is divided into several parts and they are allocated to cluster nodes for rendering. This method is well applicable to the tiled display system consisting of multiple display devices to construct a huge display device for high resolution images. However, it is difficult to implement this for the single display system, because it is required to take the special hardware configuration to aggregate the multiple display images rendered by the allocated cluster nodes. Furthermore, it is difficult to create an effect of the parallel computing if the target imagery consists of a large number of polygons. Actually, some display parts include a lot of polygons to other nodes to break the load balancing among the cluster nodes.

Secondly, the sort-middle method is a parallel rendering method, which divides display into multiple parts and allocates them to multiple nodes. The set of polygons is also divided into equal number of subsets to be allocated to the nodes. This method is not well applicable to the single display system as same as the sort-fast rendering. It could be expensive because the number of nodes is getting larger.

Thirdly, the sort-last rendering is the parallel rendering method in which the set of polygons is equally allocated to the cluster nodes to calculate the geometry each polygon and compose the output images with Z-buffer composition method [64]. As for the single display system, this method is best compared with other two methods. However, if the resolution of the final image is large or the contents are movies, the backbone network could be the serious bottleneck of the rendering cluster.

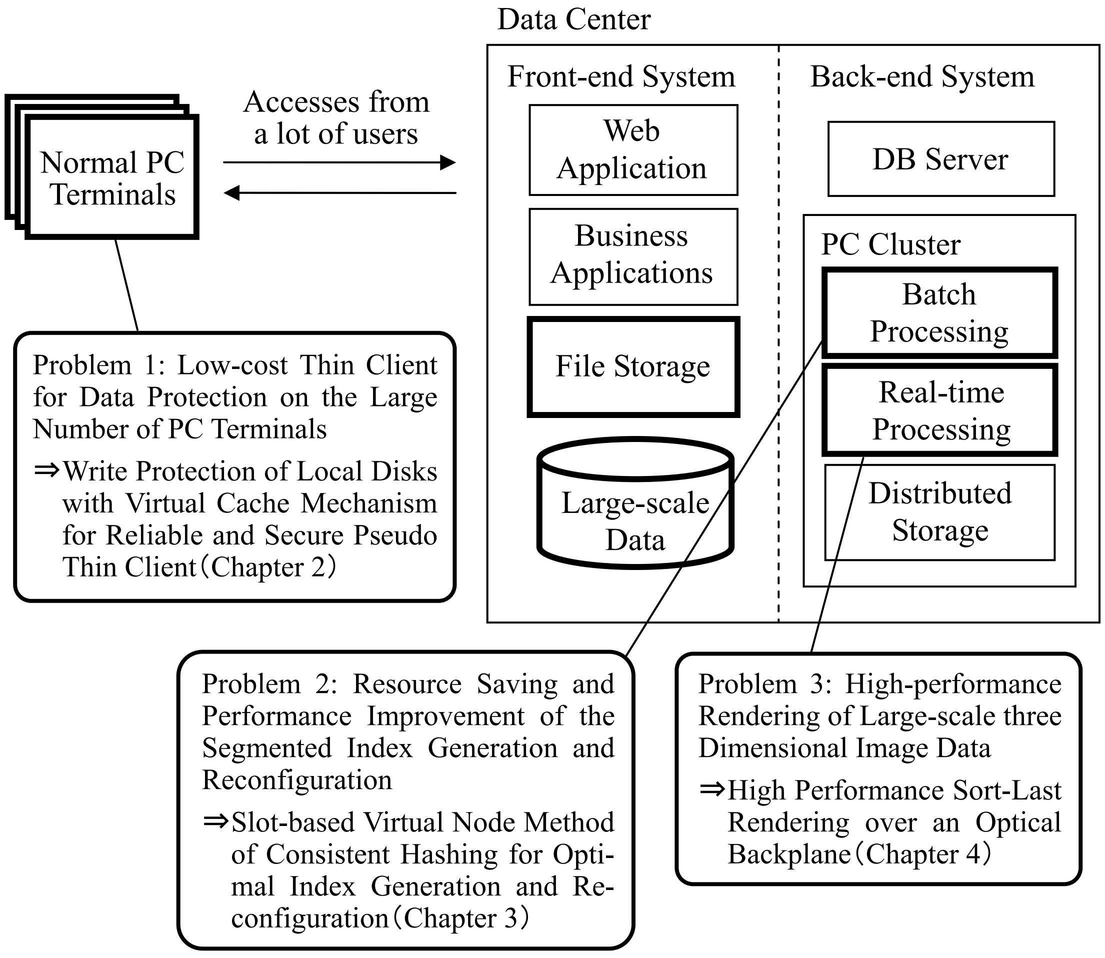
On the above related works, they all do not give any solutions for the problem of massive data transfer among the cluster nodes. They do not solve the data flow control inside the cluster to maximize the performance of rendering processing neither.

1.3 Research Strategies

In this section, we explain the research strategies in our research. Fig 1.2 illustrates the problems and approaches in this research. Research strategies of the following chapters are described as follows.

(1) Low-cost thin client for data protection on the large number of PC terminals

In order to achieve the low-cost thin client system, we propose the pseudo thin client method based on the normal PC terminals and file servers. Actually, the pseudo



Data Center
Front-end System
Back-end System
Normal PC Terminals
Accesses from a lot of users
Web Application
Business Applications
File Storage
Large-scale Data
DB Server
PC Cluster
Batch Processing
Real-time Processing
Distributed Storage
Problem 1: Low-cost Thin Client for Data Protection on the Large Number of PC Terminals
⇒Write Protection of Local Disks with Virtual Cache Mechanism for Reliable and Secure Pseudo Thin Client（Chapter 2）
Problem 2: Resource Saving and Performance Improvement of the Segmented Index Generation and Reconfiguration
⇒Slot-based Virtual Node Method of Consistent Hashing for Optimal Index Generation and Reconfiguration（Chapter 3）
Problem 3: High-performance Rendering of Large-scale three Dimensional Image Data
⇒High Performance Sort-Last Rendering over an Optical Backplane（Chapter 4）

Fig. 1.2 Distributed systems for large-scale data

thin client indicates a sort of thin client in which the normal PC is adopted as the client terminal and saving local disks and copying data to external storage devices such as USB (Universal Serial Bus) flash drives are prohibited by the software. Restricting the saving location to the central file servers, it has same effect of data protection as the conventional thin client system. Because the OS and applications run on the local PC, its usability of GUI does not affect the network delay unlike the conventional thin client. The main issue is the realization of reliable and secure write protection mechanism to the local disks. This is realized by the write protection mechanism of local disks with virtual memory cache. In this proposal, we adopt the volume filter driver method to realize the write protection of local disks to avoid the security vulnerabilities for the file filer driver method. Furthermore, it has the virtual cache mechanism to extend the cache size of the volume filter driver and improve the reliability of the system.

(2) Resource saving and performance improvement of the segmented index generation and reconfiguration

First of all, we describe two requirements for the mapping methods between files and segmented indexes in the distributed search system, and discuss the problems of typical mapping methods and conventional consistent hashing method if they are adopted. Because segmented indexes are regarded as a sort of segmented databases, typical partitioning methods of the databases, that is, range partitioning method and hash partitioning method, can be applied to the distributed search system. However, these typical mapping methods have the problem on data migration cost because of the redundant movement of index data among original cluster nodes in node adding or removing process. Because the consistent hash method can minimize the network traffic and redundant index data processing in the index reconfiguration process, we focus it as the basis of our proposed method. If the conventional consistent hashing method is simply applied to the partitioning of indexes, the memory usage is so large because the number of plotting virtual nodes is so large to uniform the distribution of key entries on the nodes if the number of nodes is greater. The overhead of splitting indexes from the original nodes is also the problem in the index reconfiguration process when multiple nodes are added to the cluster in the conventional method. In order to overcome these issues, we propose the slot-based virtual node method of consistent hashing to realize the resource saving and performance improvement of the segmented index generation and reconfiguration. The proposed method divides the hash ring space of consistent hashing into multiple slots with equal size and manages to allocate the almost same number of slots to all nodes in order to reduce the variance of entry distribution on each node. It can save the number of virtual nodes plotted on the hash ring space to save the memory resources. It also realizes “a bunch of” slot migration from original nodes to reallocated nodes to reduce the splitting times of partial indexes and the total lapse time of index reconfiguration process.

(3) High-performance rendering of large-scale three-dimensional image data

To realize the high-performance rendering of large-scale three-dimensional image data, we propose the pure optical sort-last rendering method over the cluster based on the photonic switch. The proposed method adopts the sort-last method because it is suitable for the parallel rendering of image data consisting of massive polygons. In the proposed method, the photonic switch based cluster is constructed and the sort-last rendering process is executed on the cluster to avoid the network bottleneck among cluster nodes. Because the cluster network is pure optical, it has broad network bandwidth. The image generation time and frame rate on the client side are the evaluation measures of the system. It is necessary to check the response time and data processing speed inside the cluster to verify the feasibility of this method to be able to use in the practical use cases. The huge network traffic among cluster nodes and the degradation of the frame rate derived from deterioration of the data flow inside the cluster are problems to be solved.

1.4 Outline of the Dissertation

The reminder of this dissertation is organized as follows.

Chapter 2 proposes a reliable and secure pseudo thin client method based on the write protection of local disks with virtual cache mechanism [65]-[67]. First of all, we describe two methods of write protection of local disks for the conventional pseudo thin-client system and explain their issues individually. One issue is the security vulnerability on the write permission of write protection mechanism for some system processes. Another issue is the monotonous growth of cache data which makes a pressure of the physical memory region. Next we explain our proposed pseudo thin client system called “WriteShield” that solves the issues of conventional methods. The fundamental system architecture and virtual cache mechanism of the write-protection driver are described. Next the three paging algorithms are evaluated using actual disk I/O logs and the prototyped simulation code to select the optimal one. Finally, we evaluate the disk I/O performances during the running of write-protection mechanism and paging process of virtual cache mechanism respectively.

Chapter 3 proposes the slot-based virtual node method of consistent hashing for optimal index generation and reconfiguration in order to realize the resource saving and performance improvement of the segmented index generation and reconfiguration in the distributed search system [68]. Firstly, we explain the conventional consistent hashing method and its issues in terms of the memory resources and the index reconfiguration time if it is applied to the mapping function between file paths and segmented indexes in the distributed search system. In the next, our proposed method called slot-based virtual node method of consistent hashing is described. Then, we define two data processing models of index reconfiguration for both conventional and our new methods. Finally, we evaluate the memory usage of our method with the prototype and estimate the lapse times of reconfiguration processes according to the defined data processing models of both conventional and new methods.

In Chapter 4, sort-last rendering method over the cluster based on the photonic switch is discussed to realize the high-performance rendering of large-scale three-dimensional image data [69][70]. At first, the summary of sort-last rendering and the issue of switching delay for application of the photonic switch based cluster to the parallel rendering. Next, the architecture of the sort-last rendering cluster based on the photonic switch is explained. After that, we evaluate the rendering performance of single image and stream process performance. We also discuss the feedback mechanism to control queuing and sending messages to achieve the optimal smooth data flow inside the cluster.

Finally, Chapter 5 concludes the results from this research and shows directions for the future research.

Chapter 2

Write Protection of Local Disks with Virtual Cache Mechanism for Reliable and Secure Pseudo Thin Client

2.1 Introduction

The realization of the low-cost thin client for data protection on the large number of PC terminals is significant issue in the aspect of the reliability and security on the large-scale distributed systems. In this chapter, we discuss the pseudo thin client method based on the normal PC terminals and file servers to achieve the low-cost thin client system.

Conventional thin-client systems have problems on usability and cost. For example, network latency causes slow response of GUI. Generally, system response for simple I/O tasks such as typing and mouse selection should be less than 50-150 ms threshold of human perception to keep users from noticing any delay [71]. Because network latency depends on the congestion and network distance, the GUI response cannot reach enough performance to use if client and server are located thousands of miles apart such as from overseas [72]-[76]. Performance of playing movies could also be worse because of the poor frame rate in delayed network environment [77]. Introduction cost of the thin client system is also a problem. It is required to prepare the special diskless machines or terminal devices on the client side. On the server side, thin-client servers, storages, and high-bandwidth backbone are necessary. Because the thin-client servers are consolidated in the data center, network enhancement is required to overcome the critical issue of the burst of the central network traffic especially for the printing data. The enhancement of power supply and air conditioning is also necessary. The cost of hardware enhancement and location space is a greater burden. This cost issue could be a significant threshold to introduce the system on a large scale.

To overcome these issues, there is another approach called pseudo thin-client system [78]-[80]. “Pseudo” means its architecture is not strictly categorized as the thin client. In this system, local disks on the client are write-protected and users save data on the remote file server if necessary. Introducing this system, it is not necessary to set up special diskless thin-client terminals and thin-client servers. This approach is more cost-saving than other traditional thin-client systems. Furthermore, it is simple to scale out since the system architecture is similar to the conventional distributed system. Administrators only have to concern about the scalability of central file servers. There are two types of methods to protect the local disks to realize the pseudo thin client; file system filter method and volume filter method. In the first method, the write operations are canceled at the file system filter layer by rejecting request packets for the write operations. In the second method, the write data is cached at the volume filter driver layer and not transferred to the lower device stacks. Write operations of OS and applications succeed without saving data on the local hard disk devices.

However, the conventional pseudo thin-client systems with these two methods have following issues. Firstly, it is impossible to realize the flexible file copy operation to external device control. Some pseudo thin-client product realizes copy prohibition to USB flash memories with Windows Group Policy. But it is impossible to apply the copy prohibition to the privileged users who can change the Group Policies and does not support flexible copy control. Secondly, it could be possible to leak the data via the network. Conventional pseudo thin-client terminal can access any open networks and has no restriction of access destination as the normal PC does. Thus, it is possible for malicious users to download data from the internal file servers and transfer it to the outside open sites. Thirdly, in the file system filter method, there could be a security hole since it is necessary to permit the write operation of the special processes such as system processes to run regularly. Finally, it could be possible to exhaust the physical memory because the written data cannot be released even if the corresponding files are removed in the case of volume filter method.

In this chapter, we propose a new pseudo thin-client system named WriteShield which overcomes the above issues on the conventional systems. We adopt the volume filter method as the write protection mechanism for the data written by OS and applications to be just cached on the physical memory but not written on the local disks. Because physical memory consumption monotonically increase in this write protection mechanism, the cache extension mechanism is necessary to avoid running short of the available memory. We realize the virtual cache mechanism to solve this issue. Additionally, it provides network traffic control and external device access control mechanism in order to realize the prohibition of data leakage via network and flexible copy control to the external media respectively.

The rest of this chapter is organized as follows. Section 2.2 provides the fundamental design of WriteShield and Section 2.3 describes implementation details of write protection mechanism. Section 2.4 discusses virtual cache mechanism and paging algorithms to select the optimal one. Section 2.5 provides security analysis, evaluation of usability and scalability, and evaluation of the disk I/O performance. Finally we conclude in Section 2.6.

2.2 Design of WriteShield

2.2.1 Concept and System Requirements

The basic concept of WriteShield is to prevent from saving confidential data on the local disks in each client machine and consolidate user data to the file server. Actually, write protection function to local disks and external media works on each client to prohibit saving data in and out of the client machines. This architecture restricts save location to the central file servers. In order for WriteShield to solve the issues on the conventional pseudo thin-client systems, there are following four system requirements.

(1) Write protection without interfering with OS and application

One of the key technologies of the pseudo thin client is write prohibition to the local disks on the normal PC. However, the write prohibition mechanism has to let write operations of OS and some applications succeed.

(2) Write control to the external media

In some cases, it is required to allow some privileged users to copy the external media to bring it outside offices. Thus, it is necessary to realize the control mechanism to encrypt and copy the data to the external media according to the users' privileges.

(3) Prohibition of data leakage via the network

Conventional pseudo thin-client terminal can access any open networks and has no restriction of access destination as the normal PC does. Thus, it is possible for malicious users to download data from the internal file servers and transfer it to the outside open sites. WriteShield client should have access destination control to restrict the connection to the protected network regions.

(4) OS and application update method by users

Conventional pseudo thin clients allow for only the system administrator to update the system such as OS patches and configuration settings. However, the machine update and maintenance should be operated by individual users to reduce the burden of the system administrator.

2.2.2 System Architecture

The system architecture of WriteShield is shown in Fig. 2.1. In the client, Disk Protection Driver, External Device Control Driver and Network Traffic Controller are installed as the client components of WriteShield. The Disk Protection Driver implements the write protection feature to the local disks. This is a filter driver over the volume devices. At this filter layer, the read/write packets are flied through. The read/write packets can be fetched and modified with this filter. The Disk Protection Driver provides the write-protection of local disks without prohibition of write operations. OS and applications can write any data on the local disks. However the written data is cached in the memory and is not recorded on the physical disks.

External Device Control Driver provides copy control mechanism to external media such as USB flash drives. This is implemented as a file system filter driver. In the file system filter layer, it fetches read/write requests to the external media and cancels the write requests to prohibit the copy operation to it. It can also write copy data to the external media in encrypted form when the copy operation is executed by permitted users.

Network Traffic Controller implements network filter function to monitor the network traffic and prohibits data transformation to the illegal sites to leak the confidential data downloaded from the file server. Initially, it checks the connection to the Access Control Server whether the connected network is trustworthy LAN (Local Area Network) or WAN (Wide Area Network). If the client terminal is connected to WAN, it monitors outbound packets and allows only the packets transferred to the VPN (Virtual Private Network) gateway in the office site.

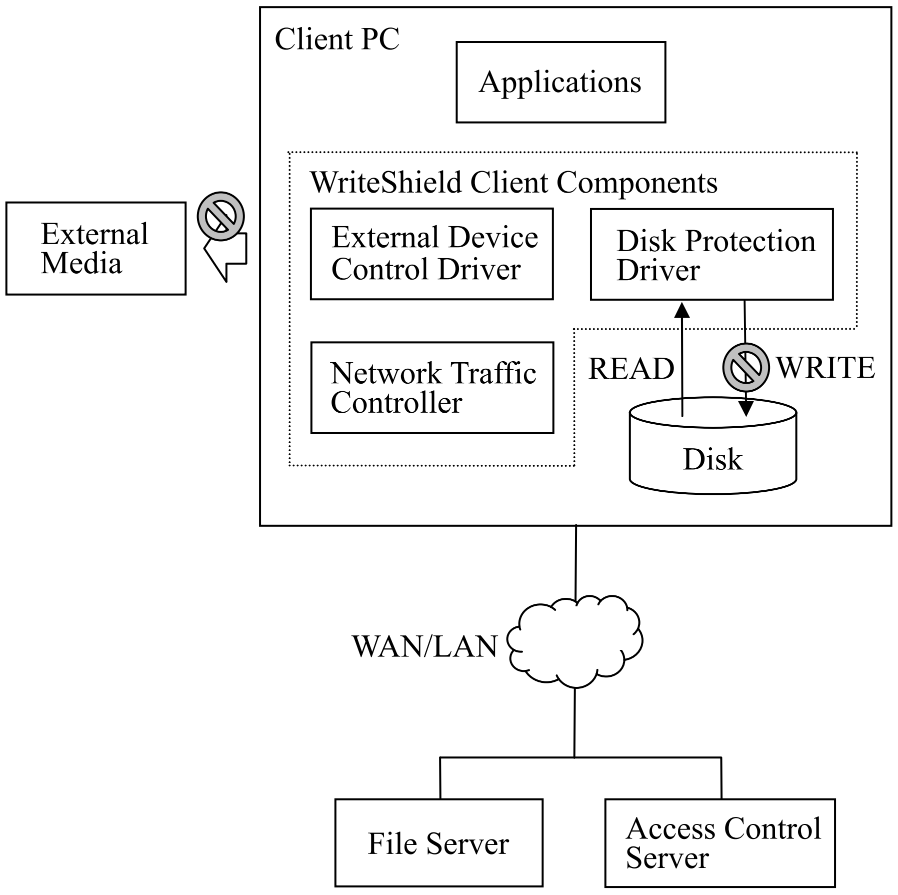


Fig. 2.1 System Architecture of WriteShield

Access Control Server is on the network which distributes security policies to each client. This policy is about copy prevention to the external media. For example, privileged users can copy data to the external media in encrypted form; some other users can execute neither read nor write operations to the external media. The security policy configured on the Access Control Server is distributed on the fly to every client. External Device Control Driver interprets the security policy and realizes the external device control according to it.

In order to realize the convenience of operation and maintenance, WriteShield has the following three operational modes; write-protected mode, user maintenance mode and administrator mode.

Write-protected mode is the default one. In this mode, write protection function of the Disk Protection Driver is worked and users cannot save data to the local disks.

Usually, users make use of clients in this mode. Because the write-protected mode does not allow users to store data on the local machine, it is impossible for users to update OS patches and set configurations in this mode. To solve this problem, the user maintenance mode and administrator mode are implemented.

In the user maintenance mode, a specialized desktop is launched to restrict user operations. On the desktop, a menu window is appeared and other applications such as Windows Explorer and Web browser are not available. Since Disk Protection Driver does not work in this mode, data write operation is permitted. User can only update OS patches and change configurations with permitted control panels. Additionally, configurations of applications can change in configuration-change menu mode. While this menu mode is on, Windows normal desktop appears and network access is disabled. User cannot access and copy confidential data from the file server in this menu mode.

In the administrator mode, the protection of Disk Protection Driver is simply off. In order to be off the protection, Disk Protection Driver writes down the all cache data to the local disk and quite the read/write request processing. This process is called "commit." After the commit process, the write-protected mode transfers to the administrator mode. In this mode, client becomes normal PC. System administrator can update, install applications, and change configurations. This mode requires user authentication to restrict its usage to the specified system administrators in order to keep up the security.

Disk Protection Driver has issue of memory restriction. Written data cache cannot be flushed until the PC is shutdown. Therefore, as the written data increases, physical memory is exhausted to freeze the OS. To avoid this problem, Disk Protection Driver evacuates the cache data to some files on local disks or network servers. This function is called the virtual cache mechanism. The detail of this mechanism is presented in the later section.

2.3 Write Protection Mechanism

Write protection to the local storages is a main function of WriteShield. Usually, OS and applications are writing data on local storages. If write operations are simply prohibited, OS and applications cannot run normally. For example, log database has to be updated while OS is running. If write operation of the database cannot succeed, the OS cannot run continuously.

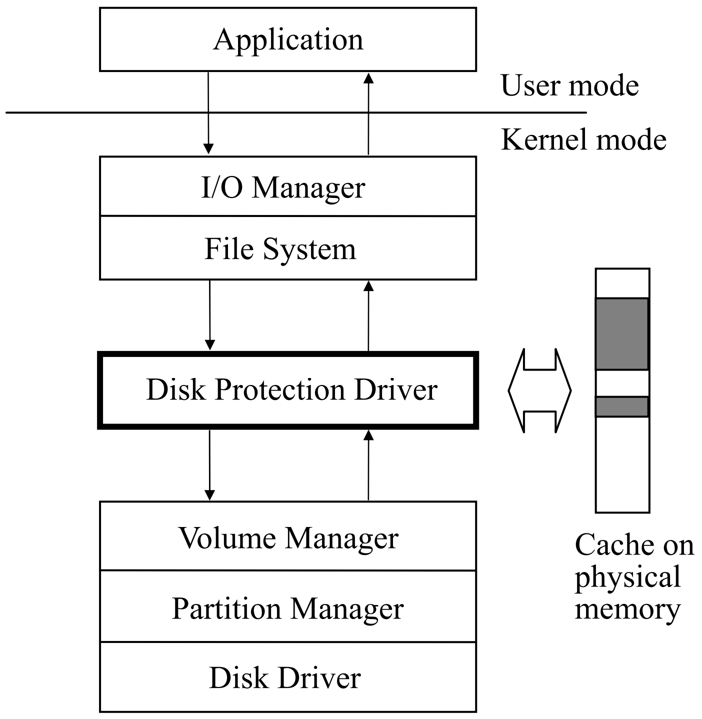


Fig. 2.2 Stack of kernel modules and Disk Protection Driver

Our write protection mechanism has to solve this issue. We implement this mechanism as the filter driver which caches write data on the physical memory and cancels write operations to the local disks. It generates the differential volume images to original ones and shows them to OS and applications as the original ones. Fig. 2.2 illustrates the stack of Windows kernel modules and position of Disk Protection Driver.

When applications access files on the local disk, file I/O requests are sent to the file system via I/O Manager. The requests are analyzed and reorganized to send to the destination volume. Disk Protection Driver is located between the file system and Volume Manager to fetch the requests targeted to the local volume. In this process, the driver manipulates read/write buffers to construct or cancel the I/O request packets.

The generation of a new cache buffer to be registered in the write request process is illustrated in Fig. 2.3. When the file system sends a write request packet to the Disk Protection Driver, it checks the sector address of write target region on the local volume specified by the packet and finds the registered cache buffers partially or fully intersected to the write target region; cache A, B and C. Then, it generates a new buffer for register and copies the cache A, B and C on it. After that, it overwrites the register buffer with the write buffer.

Read process follows similar way to this. After reading data from the local volume, Disk Protection Driver checks the sector address and size of read data, and searches all cache buffers overlapped on the read data. Then, it overwrites the cached buffers to read buffer and return it to the upper file system. This read/write request process of Disk Protection Driver provides virtual write operation and disk write protection efficiently.

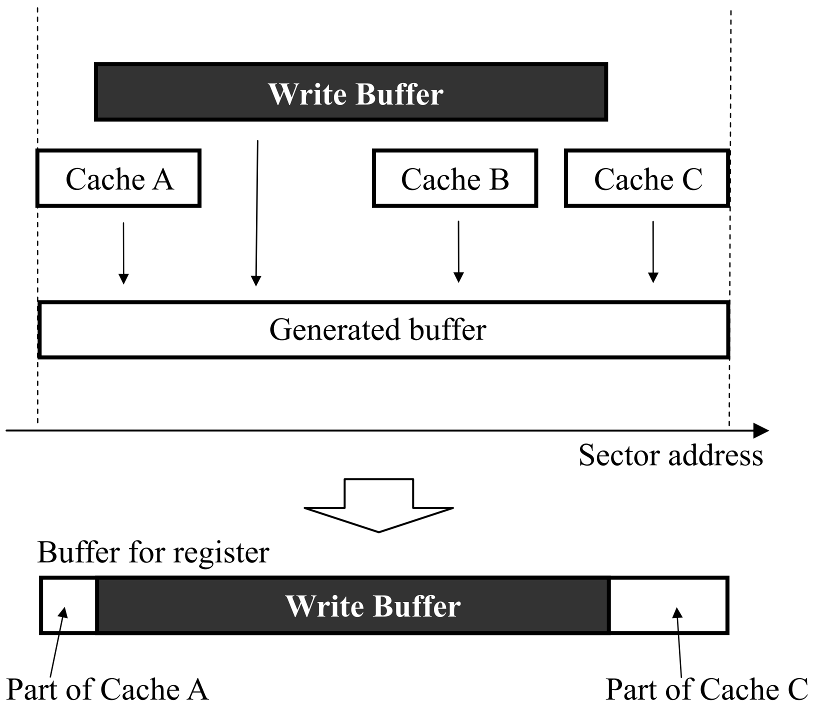


Fig. 2.3 Generation of new cache data in write request process

2.4 Virtual Cache Mechanism

2.4.1 Overview of Virtual Cache Mechanism

One of the technical issues on WriteShield is that the write data cached on the physical memory is not cleared even if the original written files are removed. This is because the file system does not free the all data of written files when they are removed. It just updates the file table entries and other written region is remained. Therefore, the trash of write data increases monotonously in the physical memory as OS and applications run. Finally the amount of free memory space will be exhausted and OS will get freeze. In order to solve this issue, we implemented a virtual cache mechanism to extend the physical memory. In general, OSes have virtual memory mechanism to extend the physical memory. However, it is not available because all write requests to the virtual memory files are cancelled by Disk Protection Driver as with other write requests.

The virtual cache mechanism provides the similar function to the virtual memory mechanism to extend the cache size of Disk Protection Driver. The cache data is administrated in a B-tree structure for each volume, which is called the cache list. Each leaf node has 4KB chunk of buffer called page which stores the part of cache data. If the total cache size exceeds to the specified threshold, Disk Protection Driver evacuates the cache pages to the page files on the local disk or the file server over the network. If read accesses to the paged-out cache region are occurred, Disk Protection Driver reads the corresponding pages from the page files and completes the read buffer. The paging algorithm affects the disk I/O performance during the page-out processing as the memory usage is over the configured threshold. In order to keep the system security, Disk Protection Driver saves the pages to the page files in encrypted form if it is on the local volume. The encryption keys are randomly generated and the page files are deleted and newly created in every boot process. In the next section, we provide a discussion of the paging algorithms to select the optimal one for the virtual cache mechanism.

2.4.2 Selection of Paging Algorithms

One of the questions is what the optimal paging algorithm is for the virtual cache mechanism. We discuss a comparison among several paging algorithms to select the best one which optimizes the I/O performance for the cache of Disk Protection Driver. In order to evaluate the following paging algorithms, LRU (Least Recently Used), FIFO (First In, First Out), and RANDOM, we implemented a simulation code of them and applied the following two test cases. Here these cases cover major I/O access patterns such as read/write a lot of small files and a single large file.

Case 1: A user boots OS and logs in to the desktop environment. Then it executes the Windows Explorer to download and save the 2750 files (total size 232MB) on the local disk from a remote file server.

Case 2: A user boots OS and logs in to the desktop environment. Then it executes the Windows Explorer to download a zip-compressed file with size of 200MB and decompress it on the local disk.

Actually, we took cache access logs in the Disk Protection Driver with respect to the above two test cases as the input data of the simulation code and executed the code to see the relation between cache hit ratio and cache size. Here, we define the cache hit ratio as follows;

$$\text{Cache hit ratio} = \{\#(\text{Reference}) - \#(\text{Page fault})\}/\#(\text{Reference}) \qquad (2.1)$$

where #(Reference) and #(Page fault) are the total number of referenced pages and the total number of page fault events in the test case respectively. Page fault occurs when the cache is full and writing page is not cached or reading page in not on the cache. This indicates the performance of paging algorithms. Fig. 2.4 and Fig. 2.5 illustrate the graphs of the cache hit ratio for cache size of the Disk Protection Driver for three paging algorithms in case 1 and case 2.

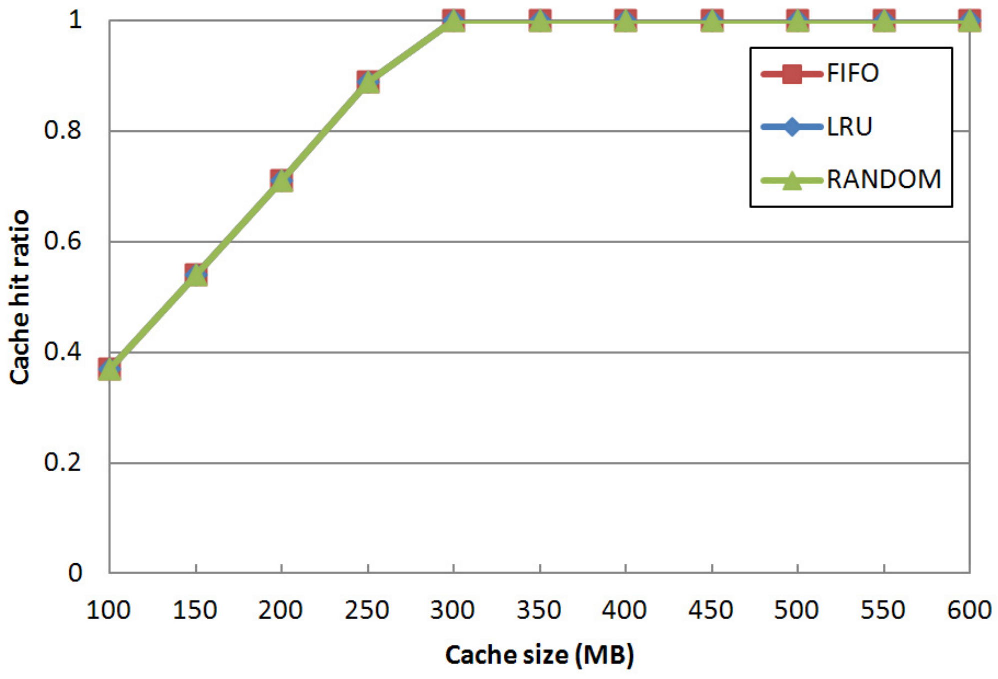


Fig. 2.4 Cache hit ratio in case 1

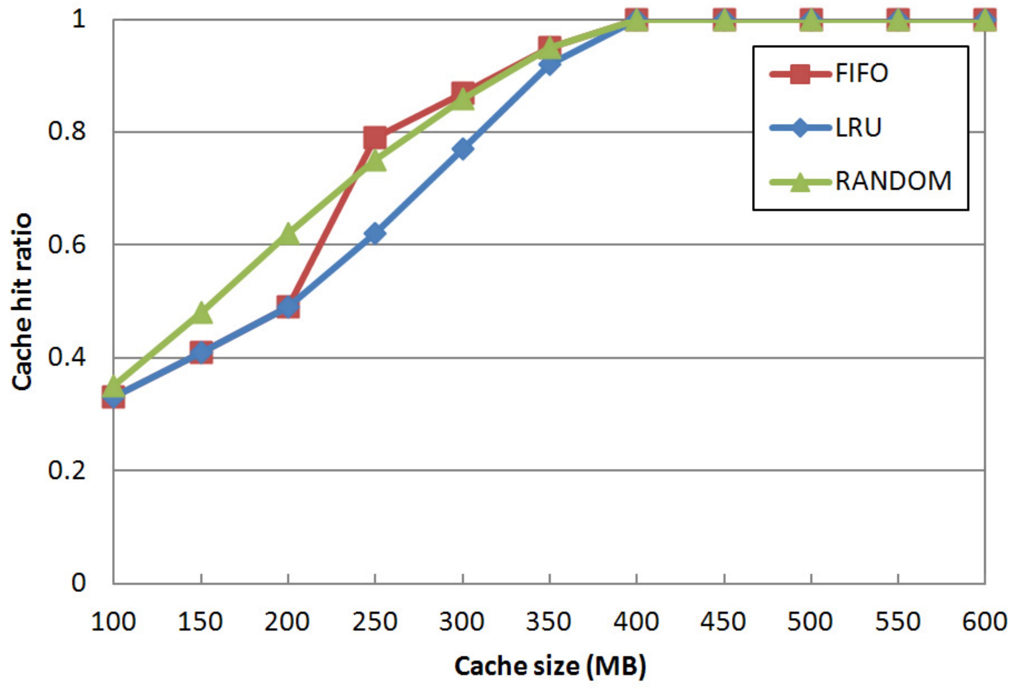


Fig. 2.5 Cache hit ratio in case 2

The graph of Fig. 2.4 indicates that the performances of the three paging algorithms are almost same. This is because the cache hit number is relatively small to the total number of page references in the disk access pattern in case 1. In this case, choice of the paging algorithm does not affect the performance of the cache access in the Disk Protection Driver. On the other hand, the graph of Fig. 2.5 shows that three algorithms have different performance curves in terms of the cache size. In this case, the cache hit number is relatively large to the total number of page references. Thus the choice of paging algorithm is significant to the cache access performance. From the graph, we can see that the RANDOM algorithm is best of them.

In order to give further consideration to the above result, we analyze the cache access logs in detail. Fig. 2.6 and Fig. 2.7 show the graphs of the cache references of read/write accesses in case 1 and 2 respectively. The horizontal axis is the log event number and the vertical axis is the logical address of the read/write references of the cache for the protected volume.

Log event number means the sequence number for the each I/O request in the log. From Fig. 2.6, we can notice that the read access for the cache did not occur in case 1. This means the page fault for read access does not occur and selection of paging algorithms makes no difference in this scenario. From Fig. 2.7, we can see that 200MB zip data is sequentially written to the cache from 2800MB of logical address on the volume and sequential read access to the same region occurs later in case 2. Comparing to Fig. 2.6, the number of read access is large. This indicates the page fault for read access could happen more likely in case 2 instead of in the case 1. This creates the performance difference between the three paging algorithms in Fig 2.5.

According to these two graphs, we can notice the following things on the I/O characteristics at the volume filter layer. At first, OS and application files are installed and accessed in read-only mode but those files are not cached on the Disk Protection Driver layer. Additionally, because written small files are cached in the file system and not purged unless the cache memory is full, the read accesses to those files cached in the file system are not reached to the Disk Protection Driver layer. Actually, several system processes and Windows Explorer are executed and they read and write the files on the local disks during the log capturing. However, there is no read access to the cache in the Disk Protection Driver layer in Fig. 2.6. The cache mechanism of file system above the Disk Protection Driver affects cache access pattern on the Disk Protection Driver. Furthermore, relatively large size of data access affects the cache of the Disk Protection Driver and they are usually sequential accesses and the number of read accesses for small files is smaller than that for large files according to Fig. 2.7. So, we should focus on the read access pattern of the large files to consider the performance difference of the three paging algorithms. Finally, when a file is copied or overwritten by copying another file with the same name, the file system writes the data to the new region on the disk. There is no overwriting process to the sectors of overwritten file, even if it is the overwrite operation of files with the Windows Explorer. Therefore, ratio of the cache size for the overwritten sectors to that for new written sectors is relatively small.

Based on these characteristics, we can indicate the reason of the best performance of RAMDOM as follows. Because of the small ratio of the cache size for the overwritten sectors to that for new written sectors, the paging process of LRU is close to that of FIFO. Additionally, the relatively large files, which are likely to account for large part of the memory consumption, are usually read or written sequentially in terms of the ordinary file operations and application accesses. This indicates that the probability of the page fault for FIFO is higher than that for RANDOM because the probability of page fault for the next sector access is higher in the sequential I/O if we adopt FIFO paging algorithm. At the result, RANDOM works more effectively than FIFO and LRU.

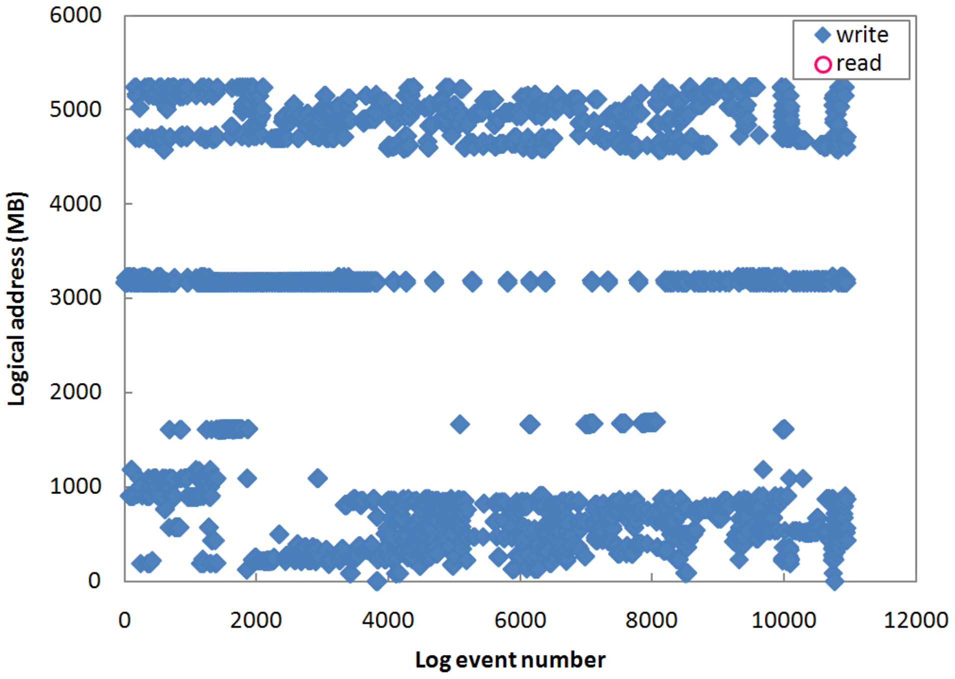


Fig. 2.6 Distribution of references to the cache in case 1

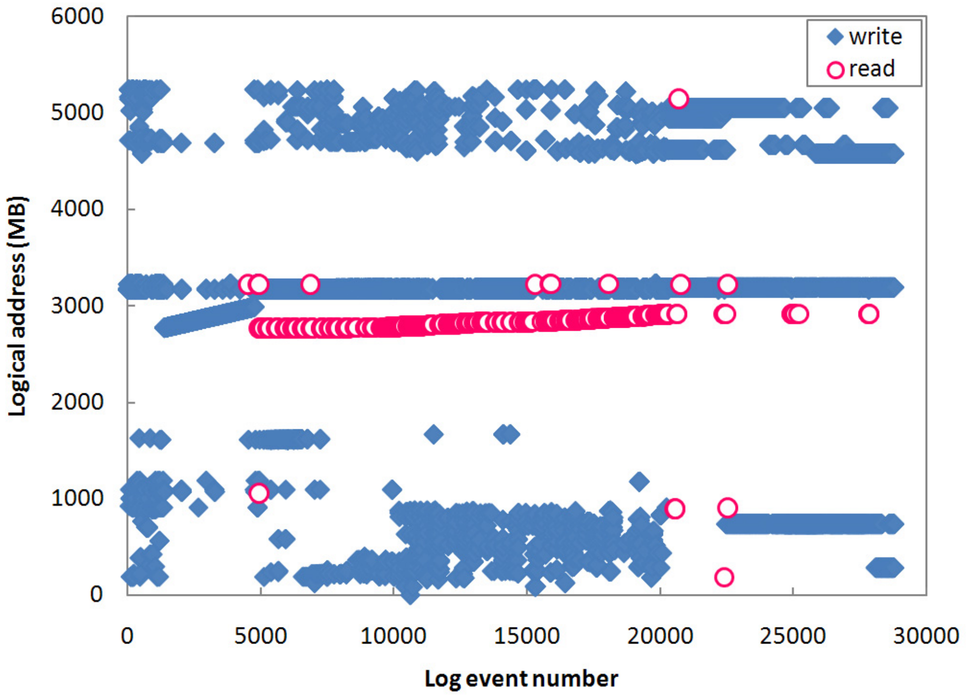


Fig. 2.7 Distribution of references to the cache in case 2

2.5 Evaluation

2.5.1 Security Analysis of the System

We describe the system analysis on the security perspective in this section. Firstly, while the conventional pseudo thin-client systems do not have flexible copy control mechanism to the external media, WriteShield realizes that with External Device Control Driver and Access Control Server. Security administrators can allow some specific users to copy data in encrypted form from the client to the USB flash drive and take a copy log in the Access Control Server. Thus, they can adopt flexible and appropriate security policy for the actual demands of users in individual offices. If the attackers would like to copy the data to the USB flash drive illegally, they have to invalidate the External Device Control Driver. In order for them to uninstall the External Device Control Driver, it is necessary to change registry data, remove files and reboot the machine. However, those operations are cleared in the reboot process because of the write protection by Disk Protection Driver. Therefore, External Device Control Driver cannot be invalidated without changing to the administrator mode. This copy control mechanism to the external media is strongly protected by the write protection feature of the Disk Protection Driver in contrast to the protection based on the standard access right scheme on Windows.

Secondly, the following vulnerabilities are considered that malicious users can download and send confidential data from the file server to the outside unauthorized region over the network by Web uploading, email attachments, telnet, etc. In the LAN, we can take countermeasures to set up the conventional network filtering applications such as firewall, web and email filtering products. For the WAN environment, the Network Transfer Controller works to prohibit the illegal data transfer while the WriteShield client is not connected to the internal LAN via VPN. If it is connected to the LAN, the internal network filtering controls described above work appropriately.

Thirdly, if the pseudo thin client adopts file system filter method to prohibit write data on the local disks, some processes have to be permitted to write data on them because OS and some applications can run normally. However, the write-permitted processes could be the security holes if they are hijacked by attackers. In order to avoid this vulnerability, WriteShield adopts the volume filter method to realize the write prohibition on the local disks for any processes.

“Administrator mode” is the special operation mode for system administrators. The user authentication is required to use this mode. This authentication restricts the users with full access control to hold the security. In the user maintenance mode, Windows shell is not running and custom menus are displayed only for changing configuration and updating OS and applications. It is impossible for users to execute any applications via command prompt or Windows shell in this mode. In addition, Network Traffic Controller controls the network traffic to restrict access to the servers for system configuration and update. Especially, it prohibits the client from accessing the file servers storing confidential files while it is in the user maintenance mode. Thus, it is impossible for users to download the confidential files to the local disks with some background process in the user maintenance mode.

Besides, there are following security benefits compared to conventional thin-client systems. Firstly, virus protection can be applied to the thin-client terminal. Conventional thin-client terminal adopts the embedded OS such as Windows XP Embedded, Linux, FreeBSD, and so on. Usually, OS is written on the flash ROM drive. To protect the thin-client terminal from viruses, anti-virus software products have to be installed to each terminal. However, they usually do not support embedded OS. Actually, virus patch files cannot be updated because OS and applications are stored on the read-only media.

Secondly, WriteShield can take OS patch update to the thin-client terminal. When the security holes of OS are revealed, it is necessary to update OS patches to fix them. Thin-client OS is usually an embedded OS and recorded on read-only media. It is impossible for users to update the OS patches on demand. WriteShield overcomes this problem by adopting generic OS and providing the user maintenance mode. In this mode, every user can apply patches to OS. During the update, GUI operation is restricted. Only the update data is written to physical disks. Anti-virus pattern update is done in the similar way.

2.5.2 Evaluation of Usability and Scalability

In the conventional thin-client system with the remote control protocol, GUI response can be too slow to use if it is long distance access. In the WriteShield, OS and applications run on local CPU and memory. Thus, GUI response is not affected by network delay. Applications requiring high frame rate can run in equal performance of the normal PC. Actually, we subjectively evaluated and confirmed equality of the performances to normal PC as for GUI response and 3D model rendering through the network connection with about 100ms RTT (Round-Trip Time). WriteShield can overcome the usability issue of conventional thin client.

One of the critical issues of the conventional thin client system is the burst of printing data on the network of the consolidated thin client servers in the data center. Because issued printing data are distributed in our proposed thin-client system, they do not interfere with the other data transfer over the LAN. It is not necessary to enhance the network bandwidth for printing data in the data center. Besides, the enhancement of the file servers is same as that in the conventional distributed system. Thus, system enhancement is simple compared to the conventional thin-client systems. Actually we have a use case in which over four thousand users utilize WriteShield clients as mobile PC in their daily work in order to avoid the data loss and leakage outside the office. Hardware specification of the client machine is Intel Core2 Duo 2.4GHz CPU, 3GB RAM and 1500GB SATA HDD. Users can connect to the LAN from outside offices to access file servers, e-mail servers, groupware servers, etc. They can update the OS patches and virus pattern files in the user maintenance mode. This use case verified its usability, scalability and validity of the proposed system operation mechanisms.

2.5.3 Disk Performance Evaluation

Disk Protection Driver provides a caching mechanism for disk I/O. Once new data is written, it is cached in the cache list and the following access speed to the same data gets faster. Thus, the disk I/O in write-protected mode can be faster than raw physical disk I/O. On the other hand, disk access can take overhead in paging process. Local disk I/O can drop if the used memory size exceeds the configured threshold of the memory.

According to the simulation evaluation in the previous section, we implemented the virtual cache mechanism with RANDOM paging algorithm on the Disk Protection Driver and evaluated disk I/O performance in write-protected mode and paging process. As for the paging process, we measure the disk I/O performance in two cases; page file location is on the local storage or the iSCSI volume. To measure the performance in paging process, we copy several files on the client to exceed the specified threshold of memory usage and benchmark the disk performance while paging process is running. Table 2.1 illustrates the experimental environment.

Table 2.1 Experimental environment

Client PC	CPU: Intel Core 2 Duo 1.86GHz Memory: RAM 2GB HDD: SATA 80GB NIC: Broadcom Gigabit Ethernet Card OS: Windows XP SP3
Network	100 Base-T Ethernet
Benchmark tool	CrystalDiskMark 2.2

Table 2.2 Experimental results of disk I/O measurement

Measurement item	Disk performance (MB/s)			
	Raw phys. disk	Write-protected	Page-out (local disk)	Page-out (iSCSI vol.)
Serial Read	23.0	46.1	24.4	11.4
Serial Write	28.1	45.7	23.8	10.9
Random Read(512KB)	16.1	46.2	20.9	9.5
Random Write(512KB)	23.8	45.8	19.0	9.6
Random Read(4KB)	0.3	37.6	15.4	7.5
Random Write(4KB)	0.7	31.8	11.8	6.3

Test read/write buffer size of the benchmark tool is 50MB. We measured 3 times and took average values for each measurement item. Table 2.2 illustrates the results of disk I/O performance measure for raw physical disks, write-protected mode, and paging process.

The disk I/O performance of write protected mode is about 1.63 times faster than physical disks. This is because the memory cache works as RAM disks. Especially, random read/write operations of 4KB buffers are about 125.3 times faster than actual physical disks. The disk performance in the paging process is worse than that in write-protected mode. Location of the page file affects to the disk performance simply because read/write speeds to the local disk and the iSCSI volume are different.

Comparing to the write-protected mode, performances in the paging process take up to 62.9% (local disk) and 79.4% (iSCSI volume) degradation. However, comparing to the raw physical disk I/O, the disk I/O in paging process on the local disk takes up to 20.2% degradation. Even the disk I/O in paging process on the iSCSI volume is not so worse (up to 61.2% degradation). It indicates that the virtual cache mechanism achieves to a practical level. Actually, the paging process does not start unless the cache data in the Disk Protection Driver does not exhaust the physical memory region. Thus, the disk performance of the WriteShield client is usually equal to that of write-protected mode.

2.6 Conclusion

In this chapter, we proposed WriteShield, a pseudo thin-client system for prevention of confidential information leakage. One of the key issues on this pseudo thin client is how to realize the write protection mechanism of local disks. Write operations of OS and applications have to succeed without saving data on the local hard disk devices. We implemented the write protection mechanism as a volume filter driver for the data written by OS and applications to be just cached on the physical memory but not written on the local disks. Cached data on the client is cleared when the power is off and any data cannot be saved on the local disks. Since physical memory consumption monotonically increase in this write protection mechanism, the cache extension mechanism is necessary to avoid running short of the available memory. We realize the virtual cache mechanism to solve this issue. According to the simulation evaluation, RANDOM takes the best hit ratio as the paging algorithm of virtual cache mechanism. Disk I/O evaluation indicated that the I/O performance is faster than the raw disk I/O of physical disks in write-protected mode and it is feasible in the actual use cases. In addition, we subjectively confirmed equality of the performance to normal PC as for GUI response and 3D model rendering through the network connection with about 100ms RTT. It can overcome the usability issue of conventional thin-client system. We also had a use case with over four thousands of WriteShield clients and verified its usability, scalability and validity of the proposed system operation mechanisms such as user maintenance mode and administrator mode. It can apply to the large scale system effectively.

Finally, there are following remaining problems on this research. In the current implementation of the write protection driver, it is required to reboot the system in order to change the mode from administrator mode to write-protection mode. In order to keep the data consistency for open files through the mode changing, it is necessary to close all files on the volume. However, it is difficult to close all files on the system volume because some system processes running on it have to keep opening log files and configuration files. The flexible mode changing feature is one of the challenges to improve the operations of client terminals.

Next, the seamless saving configuration in the write-protected modes is another issue. Actually, any processes cannot save the data on the local disks without any exceptions in that mode. In order to modify and fix the configurations, it is required to change the mode to user maintenance mode or administrator mode. This is a troublesome operation because of the reboot process. Write permission function for the specified files and processes in the write-protected mode can improves such kind of inconvenience.

Chapter 3

Slot-based Virtual Node Method of Consistent Hashing for Optimal Index Generation and Reconfiguration

3.1 Introduction

In enterprises and organizations, archive, management, and reuse of unstructured data are significant issues and key factors for their evolution and streamlining of daily knowledge work. Search technology is one of the key technologies for data management and reutilization. In this chapter, we describe a new method for resource saving and performance improvement of the segmented index generation and reconfiguration derived from the needs of the search and reutilization of the large number of unstructured files.

In order to realize efficient search for large data set, distributed search is essential to achieve quick responses for search queries. The target documents are registered into multiple segmented indexes which are located on separate search nodes. The load of search process is divided into multiple nodes. Listing up hit documents and calculating scores are executed in a parallel manner on them. It is necessary to have a mapping among document IDs such as file paths and segmented index IDs to construct the segmented indexes. Actually the distributed search system for large-scale data has following two requirements; (1) high-speed indexing, and (2) elapse time reduction for reconfiguration of segmented indexes in the node adding or removing processes. The optimal mapping method should satisfy these two requirements.

Because segmented indexes are regarded as a sort of segmented databases, typical partitioning methods of the databases, that is, range partitioning and hash partitioning, can be applied to the distributed search system. However, these methods cannot satisfy the requirement (2). They have the problem on data migration cost because of the redundant movement of index data among original cluster nodes in the node adding or removing processes. On the other hand, the consistent hash method is an advanced mapping method with the hash function which minimizes the network traffic and redundant index data processing in the index reconfiguration process for adding or removing cluster nodes. Actually, it can avoid data transfer among original cluster nodes to realize efficient data migration in the index reconfiguration process. It can overcome the problem of the typical methods on the requirement (2). Thus, we focus on it as the basis of our approach.

However, if we simply adopt the conventional consistent hashing as the mapping method, it still cannot satisfy the above requirements. Actually, it requires huge memory to hold mapping information because it consists of millions of virtual nodes. It actually requires several gigabytes of memory resources if the search system consists of several thousands of nodes. This disables on-memory processing of mapping calculation and causes dissatisfaction of the requirement (1). Furthermore, it takes much time to execute the process of index reconfiguration. When multiple search nodes are added to the search cluster, it is necessary to split and retrieve several partial indexes from each cluster node and send them to the added nodes. This multiple-splitting process could be a great burden for index reconfiguration if the number of added nodes is greater. So, it still does not satisfy the requirement (2) completely.

In this chapter, we propose a new improved method named the slot-based virtual node method of consistent hashing to overcome issues and satisfy the requirements mentioned above. In our approach, the hash ring space of consistent hashing is divided into multiple regions with equal size. The virtual nodes are plotted on the slot boundaries on the hash ring space where the sizes of allocated range spaces are almost equal for every node to uniformize the probability of entry allocation to each node. When the multiple nodes are added or removed, our slot-based virtual node reallocation algorithm works to plot or reallocate the virtual nodes on the hash ring space to realize "a bunch of" data migration as far as possible to optimize the index reconfiguration. In order to evaluate our approach, we measure the memory consumptions and estimate the lapse times of index reconfiguration processes to check advantages of our approach compared with the conventional method.

The rest of this chapter is organized as follows. Section 3.2 provides overview of the distributed search system, conventional consistent hashing and issues on its application to the search system. Section 3.3 describes our new method of consistent hashing for distributed search. Section 3.4 describes data processing models as for both methods. Section 3.5 provides evaluation of our approach according to the resource consumption and estimation of index reconfiguration processing time based on the data processing models defined in Section 3.4. Finally, we conclude in Section 3.6.

3.2 Consistent Hashing and Issues

3.2.1 Overview of Distributed Search

Distributed search is an effective method if the number of search target files is so large. In this method, target files are registered on multiple segmented indexes which are stored on multiple search nodes to reduce load of the search process on each node. Fig. 3.1 shows the system overview of the distributed search system.

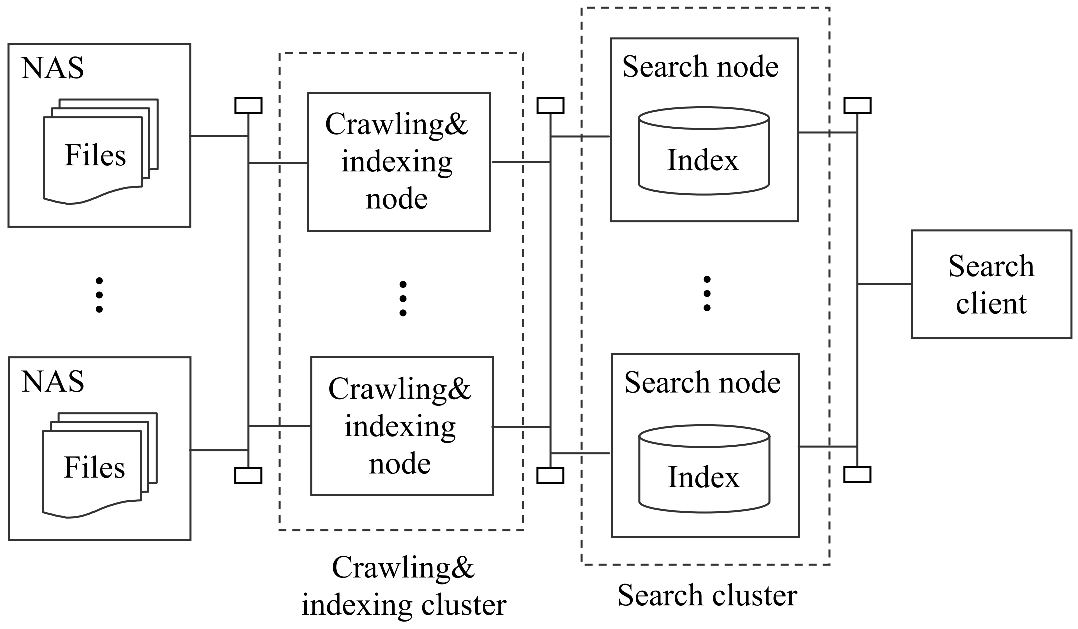


Fig. 3.1 Overview of distributed search system

Here, the target files are stored on NASes. Crawling and indexing nodes connect to NASes to fetch the target files and generate indexes. Search cluster consists of several search nodes which store segmented indexes separately. The search client accesses one of the search nodes to send queries to the cluster. The queries are dispatched to each search node and search results are aggregated to output a query result for overall target files. Though the system configuration in Fig. 3.1 has crawling and indexing cluster and search cluster separately, it is also possible to have one single cluster which has all crawling, indexing and search features.

The adopted search engine software is Lucene [84] which is a top project of Apache software foundation. It is the most popular search engine software with Java-based implementation. Using the Lucene API (Application Program Interface), it is possible to construct indexes and search files from a set of target files. It also provides functions of index splitting, merging, optimization, and so on.

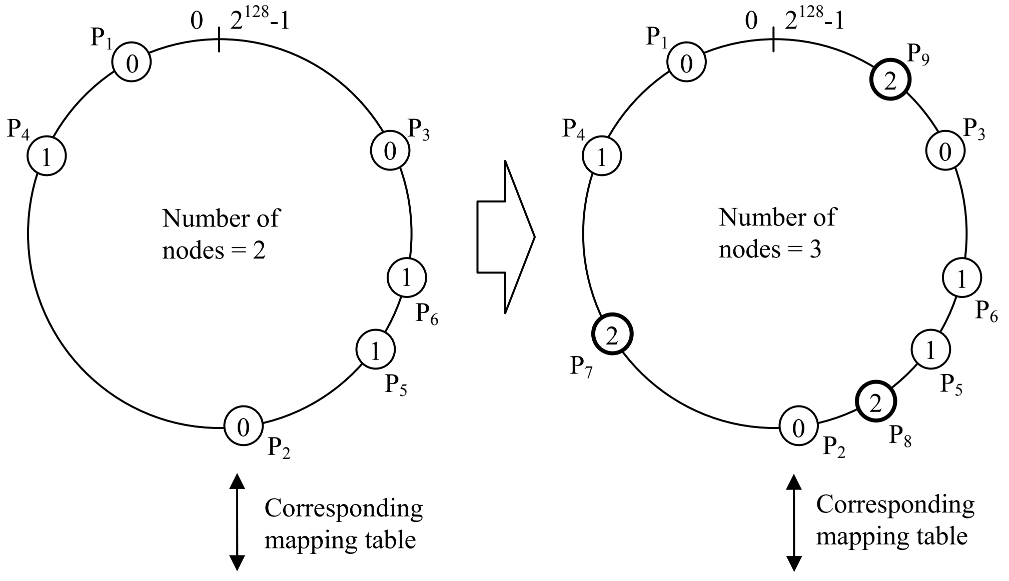
In order to create segmented indexes stored in multiple search nodes, it is required to determine a mapping function that defines the relation between file paths and segmented indexes. Consistent hashing is one of the great options as the mapping function. We describe the consistent hashing and its issues if it is applied to the distributed search system in the following sections.

3.2.2 Conventional Consistent Hashing and Memory Consumption Issue

Consistent hashing is a special sort of hashing which provides hash table functionality in a way that addition and removal of one node does not significantly change the mapping of keys to nodes. Originally, this is invented for management of distributed data caching among several nodes. It is used in many applications such as memcached [85][86], Cassandra, ketama and ROMA [87]. Only K/n keys are required to be remapped on average where K is the number of keys and n is the number of nodes. In order to balance the key entry numbers, virtual-node method of consistent hashing is utilized popularly. Fig. 3.2 shows the basic idea of virtual-node method of consistent hashing.

Here, we assume that the base hash function is MD5 [88] whose range of value is from 0 to 2^{128}-1. The hash ring space is the hash value space in which start and end points, that is 0 and 2^{128}-1, are identified. Three virtual nodes are allocated to each node. Those are candidate points of nodes plotted on the hash ring space. The virtual points are registered on the mapping table with addresses on the hash ring space and corresponding node IDs.

Given a file path of indexing target, the indexer calculates the MD5 hash value of the path and plots it on the hash ring space. Then, the point moves along with the hash ring space to anti-clockwise direction until its hitting to the first virtual node. This mapping process can determine that the corresponding node to the virtual node is that to register the file.



V-node address	Node ID
P_1	0
P_2	0
P_3	0
P_4	1
P_5	1
P_6	1

V-node address	Node ID
P_1	0
P_2	0
P_3	0
P_4	1
P_5	1
P_6	1
P_7	2
P_8	2
P_9	2

Insert entries for new virtual nodes

Fig. 3.2 Virtual-node method of consistent hashing

When the new node of number 2 is added, the corresponding three virtual nodes are plotted on the hash ring space at random position and new three entries are inserted into the mapping table. Since the plotting position is at random, the cover range of each node on the hash ring space takes various sizes to make the variance of entry distribution larger if the number of virtual nodes is small. Table 3.1 shows the relation among the number of virtual nodes, standard deviation of entry distribution on the segmented indexes, its ratio to average, and memory usage where the total number of documents is 10 million. Here, the memory usage is for storing the mapping table data. We implemented a prototype of the conventional consistent hash method to measure the actual memory usage and the distribution of the numbers of entries registered to each node for 10 million URIs generated randomly to get values in Table 3.1.

As the number of virtual nodes getting larger, the standard deviation of key entry distribution is getting smaller. For instance, the number of virtual nodes should be greater than 3000 for 5000 nodes in order to limit the ratio of 3 times SD (Standard Deviation) to average under 10%. In this case, the memory consumption is greater than 1894.3MB. The greater is the number of nodes such as several thousands of segmented indexes, the larger are the required memory resources more than several gigabytes to hold the large mapping table in the memory. This is one of significant issues if we apply the conventional consistent hashing to the generation of segmented indexes on the distributed search system.

Table 3.1 Relation among the number of virtual nodes, standard deviation of registered entries, and memory usage

Node #	Virtual node #	SD	3*SD/Average (%)	Memory Usage (MB)
100	100	9236.0	27.7	10.5
	500	4673.6	14.0	36.7
	1000	3026.0	9.1	73.4
	3000	1853.7	5.6	104.3
	5000	1460.0	4.4	135.0
1000	100	997.2	29.9	70.8
	500	437.7	13.1	135.0
	1000	340.2	10.2	270.0
	3000	213.5	6.4	589.2
	5000	178.1	5.3	752.9
5000	100	203.8	30.6	126.4
	500	101.5	15.2	674.4
	1000	77.9	11.7	753.2
	3000	57.6	8.6	1894.3

SD: Standard Deviation

3.2.3 Issue of Index Reconfiguration for Conventional Consistent Hashing

In the distributed search system, index reconfiguration process is executed corresponding to the changed mapping table among file paths and segmented indexes when search nodes are added or removed in the search cluster. Fig. 3.3 illustrates the overview of the index reconfiguration in the node adding process with conventional consistent hashing. This figure shows the case in which N search nodes are added to the cluster with M search nodes. The reconfiguration process consists of the following sub processes.

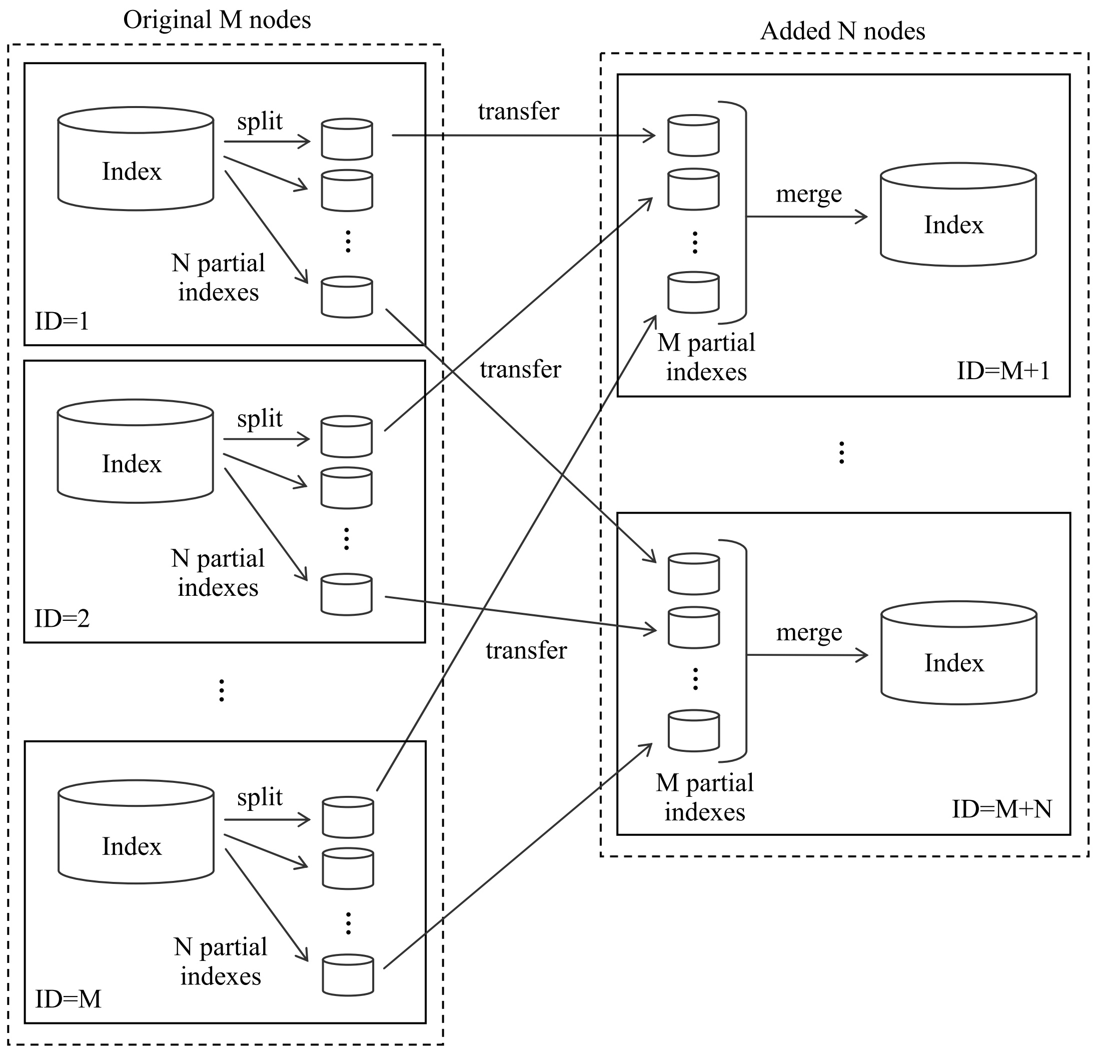


Fig. 3.3 Index reconfiguration in the node adding process for the conventional consistent hashing

(1) Splitting partial indexes

After the calculation of new mapping table corresponding to the addition of N nodes to the cluster, the documents which should be migrated to the new nodes are retrieved and added to the partial indexes with same node ID based on it. For instance, if document entries with ID = M+1 and M+2 are in the index with ID = 1, those document entries are fetched and two indexes with ID = M+1 and M+2 are constructed in the node. This process is called splitting indexes. Fig. 3.4 illustrates an example of the hash ring space reallocation in the node adding process for conventional consistent hashing. Because relatively a large number of virtual nodes for added nodes are plotted on the hash ring space at random position, each original node allocates parts of the assigned hash space region to all added N nodes. Thus, every original node has N times of index splitting processes for added N nodes. The following figure illustrates an example of the hash ring space reallocation in the node adding process for conventional consistent hashing.

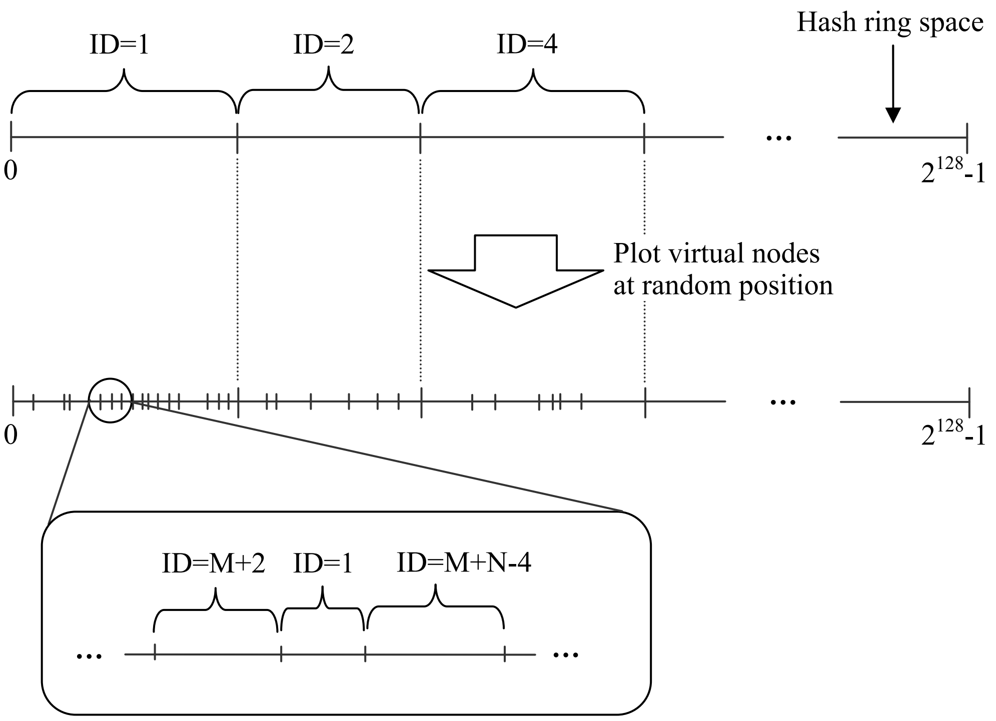


Fig. 3.4 Hash ring space reallocation in the node adding process for conventional consistent hashing

(2) Transferring partial indexes

After splitting and retrieving the partial indexes on each original node, the split data is transferred to the target nodes. The target nodes are determined by the new mapping and IDs of the partial indexes.

(3) Merging partial indexes and optimizing merged indexes

When the transferred data is stored on the added node, it is added to the new index sequentially. The order of merging indexes is free. Finally, the merged indexes are optimized to be one segmented index in each node to improve the search performance.

(4) Deleting extracted entries and optimization in original nodes

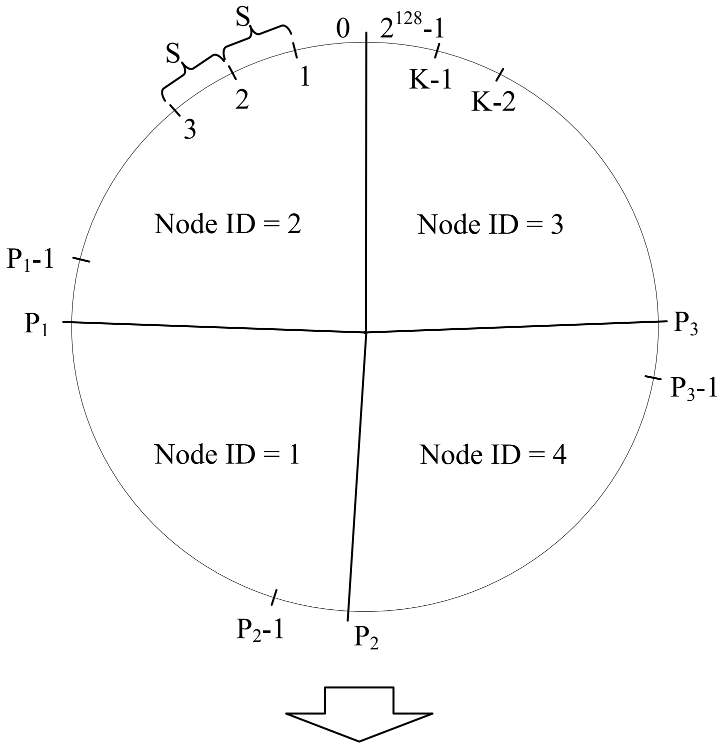
After splitting all partial indexes, the original index still includes data of documents extracted into partial indexes. Thus, deletion of extracted entries and index optimization are executed. Because document deletion API of Lucene just makes the document invisible but does not delete document data itself, the index optimization process is required to remove those invalid data from the original index actually.

If we apply the conventional consistent hashing to generation of segmented indexes on the distributed search system, it takes so much time of index reconfiguration when multiple nodes are added or removed. For instance, if N nodes are added to the cluster, N times of splitting process should be executed on each original node because the virtual nodes of added nodes are plotted at random position on the hash ring space. This could be a burden for the index reconfiguration process. Additionally, the process of merging and optimizing a large number of indexes take relatively longer time than that for small number of large indexes in the Lucene framework. This is the second issue for application of conventional consistent hashing to the segmented index management in the distributed search system.

3.3 Slot-Based Virtual Node Method of Consistent Hashing

In this section, we describe the fundamental concept of our new consistent hashing method to solve the issues of conventional method described in the previous section. Fig. 3.5 illustrates the fundamental concept of the hash ring space management in our approach.

Let H be the size of hash ring space. For example, H = 2^{128} if adopted hash function is MD5. Basically, the hash ring space is divided into K regions called slots with the same size S (=H/K) and dividing points are numbered to indicate the addresses of virtual nodes on the hash ring space. Given a node set, some virtual nodes are assigned to each node to define the allocated regions on the hash ring space. The information of virtual node address is recorded in the mapping table which defines the relation among nodes and allocated regions on the hash ring space. When the new node is added or removed, the mapping table is changed under the condition of almost equal size of allocated region to every node. Keeping this condition realizes the nearly equal probabilities of entry allocations to the nodes.



V-node address	Node ID
P1	2
P2	1
P3	4
K(=P4)	3

Fig. 3.5 Hash ring space of the slot-based virtual node method

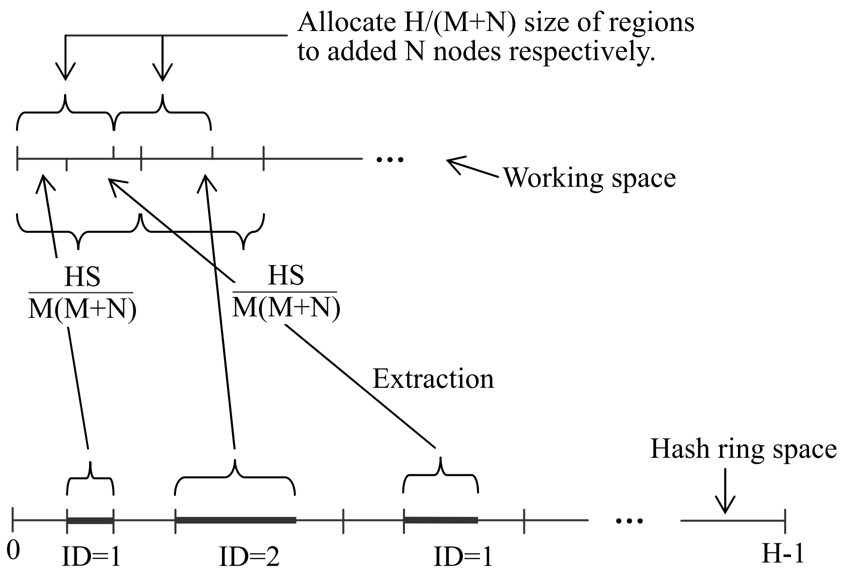


Fig. 3.6 Hash ring space reallocation in the node adding process

We describe the hash ring space reallocation algorithm on adding or removing multiple nodes. Fig. 3.6 shows the reallocation method of hash ring space in the multiple-node adding process.

Let M and N be the numbers of original nodes and added nodes respectively. Assume that there are M nodes mapped on the hash ring space. The region in the size of H/M is allocated to each node. If N nodes are added to the cluster, then the allocated regions are reconfigured in the following way.

1. Regions in the size of HN/{M(M+N)} are extracted from each original node and concatenated on the working space in order of the node ID. The total size of concatenated regions is HN/(M+N).
2. Regions in the size of H/(M+N) are fetched from the head of the working space and allocated to added N nodes sequentially. The dividing points on the working space are mapped on the hash ring space to calculate actual addresses of virtual nodes on it.
3. The pairs of actual virtual node addresses and corresponding added node IDs are recorded on the mapping table.

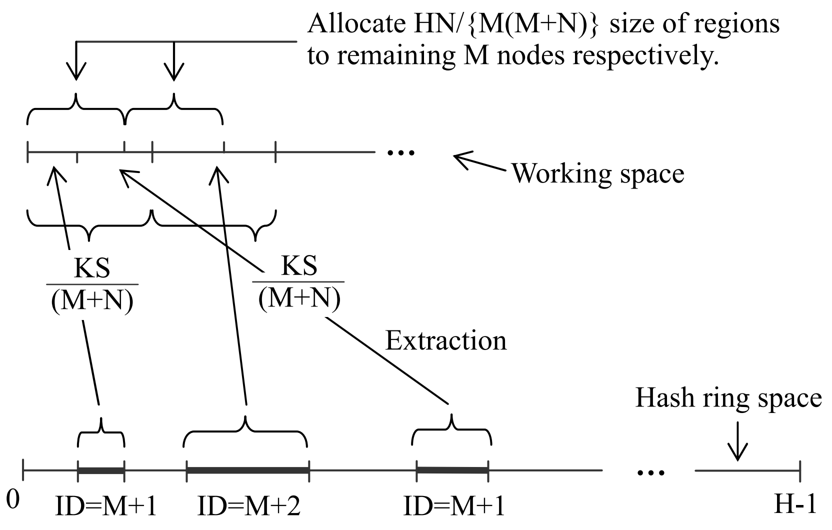


Fig. 3.7. Hash ring space reallocation in the node removing process

In the next, we explain reallocation of regions on the hash ring space when some nodes are removed from the cluster. In this case, let M+N and N be the number of original nodes and removed nodes respectively. Fig. 3.7 illustrates the reallocation of hash ring space in the multiple-node removing process. This process is similar to the reallocation process in the node adding process described above.

1. Allocated regions to every removed node are extracted and concatenated on the working space. The total size of extracted regions from each node is H/(M+N). The total size of concatenated regions on the working space is HN/(M+N).
2. Regions in the size of HN/{M(M+N)} are fetched from the head of the working space and allocated to the remaining M nodes sequentially. The dividing points on the working space are mapped on the hash ring space to calculate actual addresses of virtual nodes on it.
3. The pairs of actual virtual node addresses and corresponding node IDs are recorded on the mapping table and entries of the removed virtual nodes are removed from the table.

After the reallocation of hash ring space, the index reconfiguration process is executed based on the changed mapping table in the similar way to the case of conventional consistent hashing described in Section 3.2.3. Because this reallocation achieves “a bunch of” data migrations as far as possible, the number of index splitting times in this process satisfies the following formula.

$$\textit{spliting times} \leqq \begin{cases} \lceil N/M+1 \rceil & \textit{in the node adding process} \\ \lceil M/N+1 \rceil & \textit{in the node removing process} \end{cases} \tag{3.1}$$

In the node adding process, the splitting times are smaller than N that is equal to those for the conventional consistent hashing method. Especially, if M≧N, the splitting times are one or two. On the other hand, the splitting times are smaller than M that is equal to those for the conventional consistent hashing method in the node removing process. This reduction of splitting times for the conventional method realizes the reduction of the total lapse time of the index reconfiguration process.

If the index reconfiguration processes are executed a lot of times, the number of virtual nodes is growing and hash ring space is fragmented like disks. However, the maximum number of virtual nodes is limited to K because of the restriction in which the virtual nodes can only be plotted on the K dividing points on the hash ring space. Therefore, the size of mapping table is limited up to K*(the size of one virtual node entry). We can estimate the worst case of memory consumption for our new approach with this formula.

Compared to the conventional random plotting method of virtual nodes, the above method manages to allocate the almost same number of slots to every node. Allocating slots with the same size prepared preliminarily, the number of virtual nodes that determine the slot allocation is saved efficiently than the conventional random plotting method to save the memory consumption.

3.4 Data Process Model of Index Reconfiguration

In order to consider the validity of our approach, we define the data processing models of the index reconfiguration processes as for the conventional virtual node method of consistent hashing and our proposed method to estimate their lapse times. The detail of estimation is described in Section 3.5.3. In this section, we explain both models.

3.4.1 Data Processing Model for Conventional Consistent Hashing

Let M and N be the numbers of original nodes and added nodes respectively. In the index reconfiguration process, the N partial indexes are split from the original index if we adopt the conventional consistent hashing method. Fig. 3.8 shows the ideal data processing model of the index reconfiguration process for the conventional consistent hashing.

In this case, we assume that M≧N. The splitting processes are executed N times for each node as previously described in Section 3.2.3. If the original index size is larger and M+N is not smaller, the splitting time is considerably longer than split data transfer and merging process because the size of split index is not bigger. Thus, the pipeline

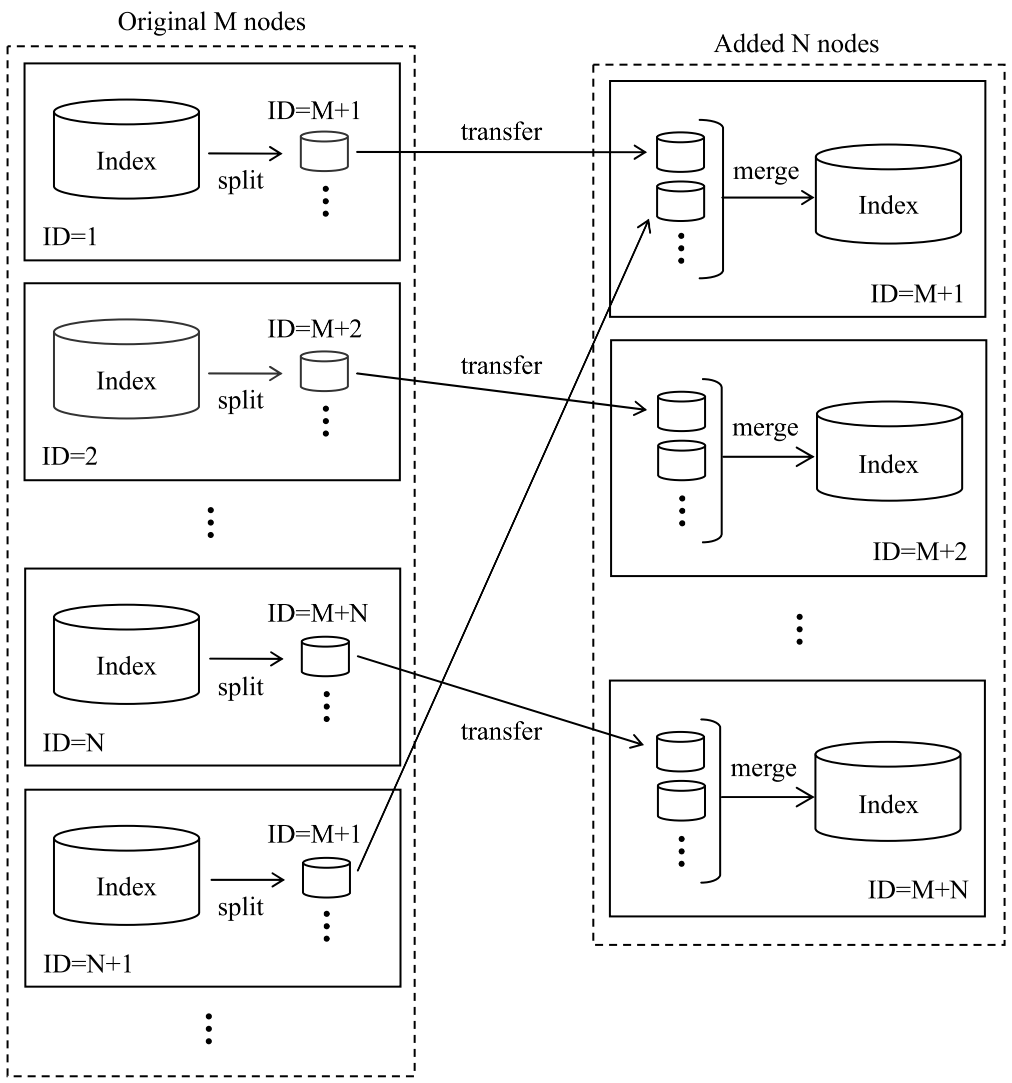


Fig. 3.8 Ideal data processing of index reconfiguration for the conventional consistent hashing

processing of splitting, transferring and merging processes are executed in parallel to reduce the total lapse time optimally. That is, the process of data transfer and merge is executed while the following process of splitting a partial index from the original one is running. The actual pipeline processing timeline is described in Fig. 3.9. According to this timeline, we can estimate that the total lapse time of index reconfiguration process as follows.

$$\begin{aligned} lapse\ time \geqq N \cdot \min_{\substack{i=1,\dots,M\\ j=1,\dots,N}} \{T_{split}(i, M+j)\} + \min_{\substack{i=1,\dots,M\\ j=1,\dots,N}} \{T_{trans}(i, M+j) \\ + T_{merge}(i, M+j)\} + T_{opt}(M+j)\} \end{aligned} \tag{3.2}$$

Here, $T_{split}(i, j)$ indicates the lapse time of splitting the partial index with ID = j from the original index with ID = i. $T_{trans}(i, j)$ and $T_{merge}(i, j)$ mean the data transfer time and merging time as for the partial index with ID = j extracted from the original index with ID = i respectively. $T_{opt}(i)$ denotes the optimization time of the index with ID = i. If the adding node number N is greater, the total time is also greater to be a bottleneck of the process in conventional method.

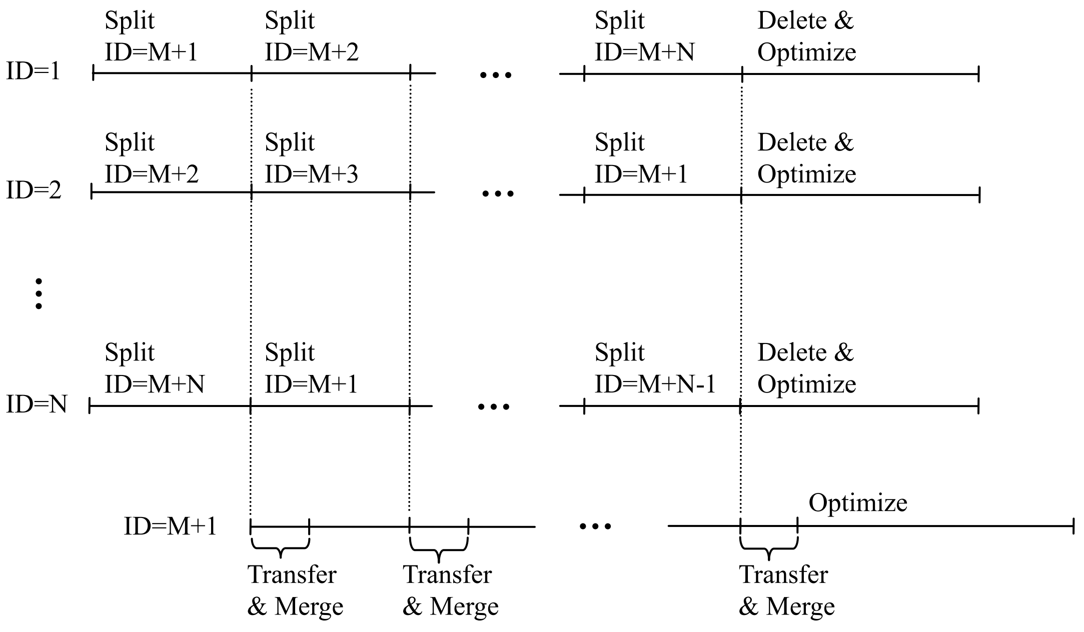


Fig. 3.9 Timeline of data processing for the conventional consistent hashing method

3.4.2 Data Processing Model for Slot-based Virtual Node Method

Assume that M ≧ N where M and N is the numbers of original nodes and added nodes respectively. Then, the number of split indexes is one or two for each node in our approach as described in Section 3.3. Fig. 3.10 shows the data processing model of index reconfiguration with our new approach.

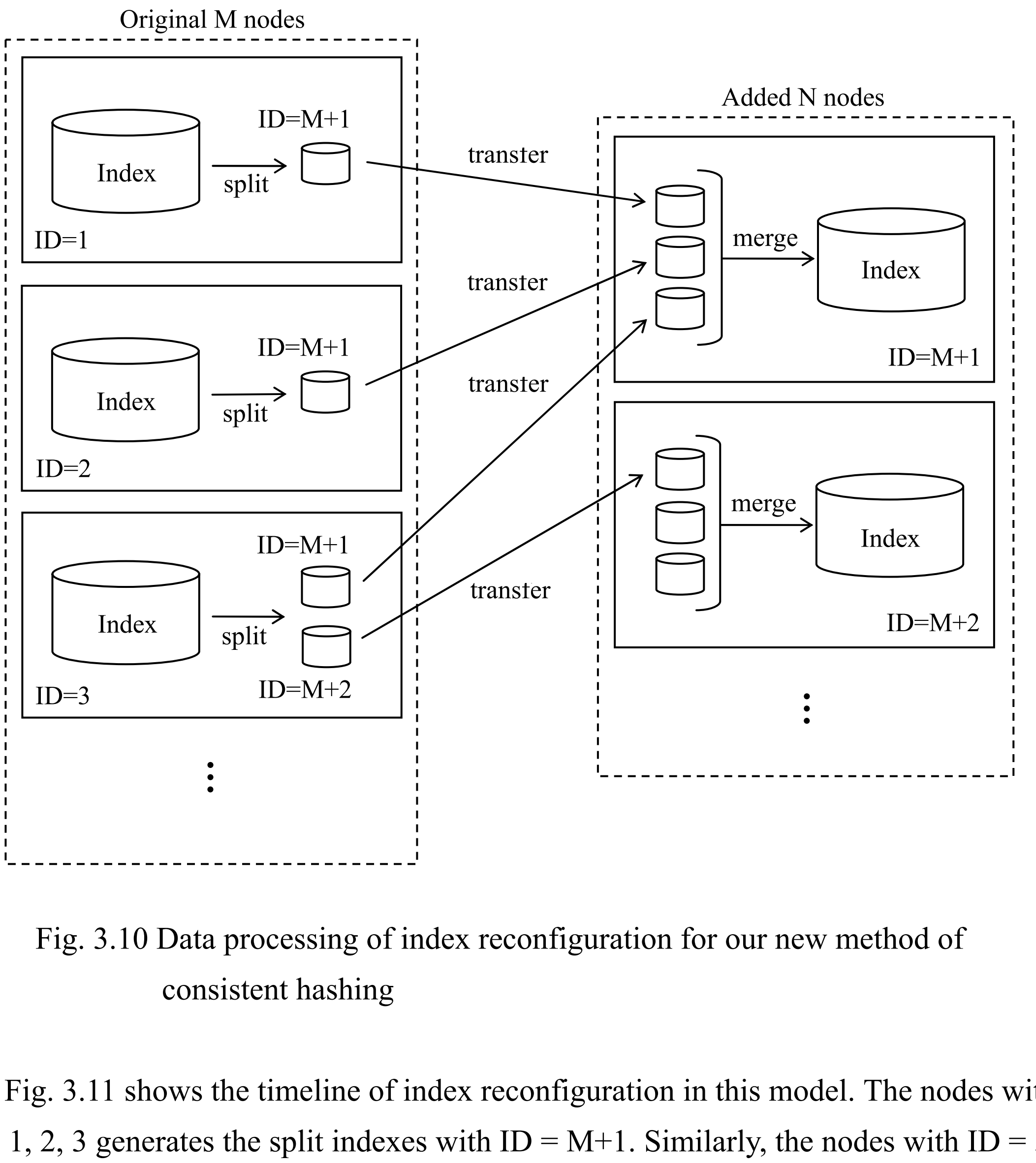


Fig. 3.10 Data processing of index reconfiguration for our new method of consistent hashing

Fig. 3.11 shows the timeline of index reconfiguration in this model. The nodes with ID = 1, 2, 3 generates the split indexes with ID = M+1. Similarly, the nodes with ID = 3, 4, 5 generates the split indexes with ID = M+2. They are transferred to the node with ID = M+1 and ID = M+2 respectively.

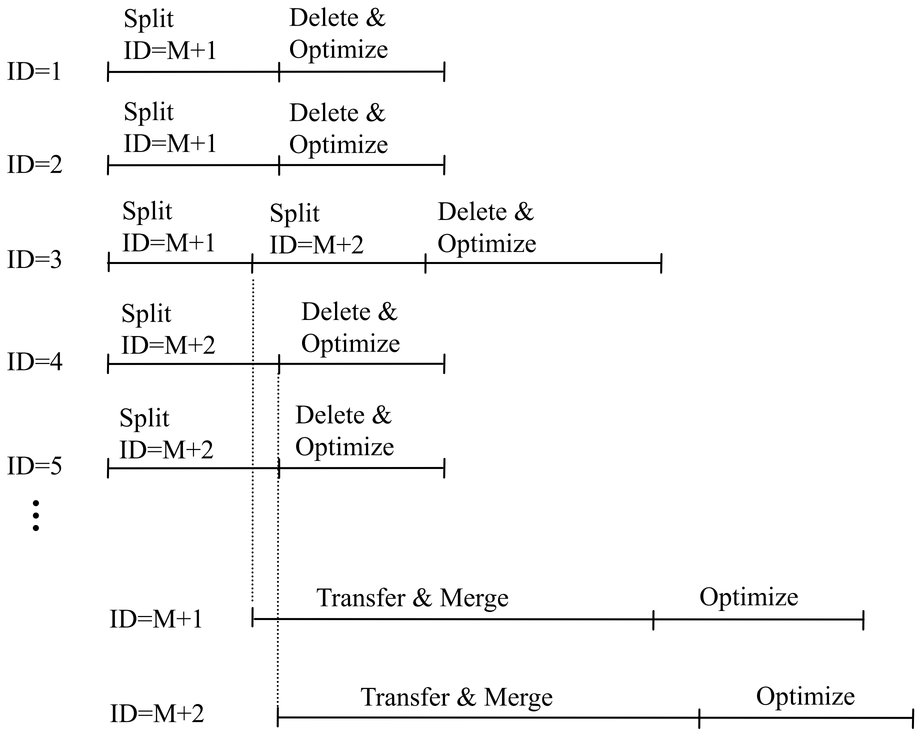


Fig. 3.11 Timeline of data processing for our new method of consistent hashing

In this timeline, after the end of splitting the index data with ID = M+2 on the node with ID = 4, the node with ID = M+2 can start to transfer and merge processes. However, it cannot start transfer the split index data from the node with ID = 3 until the end of its split process on the node. On the other hand, the split indexes with ID = M+1 can be transferred around the time of first splitting on the nodes with ID = 1, 2, 3. Therefore, the finish time of transfer and merge process on the node with ID = M+2 could be later than that of the node with ID = M+1 even if the total sizes of transferred data to both nodes are same. According to this timeline, it is possible to estimate that the maximum lapse time of total data processing as follows.

$$lapse\ time \leqq \max_{k=1,\ldots,N}\{T(k)\} \tag{3.3}$$

$$\begin{aligned} T(k) = {} & \max_{i=1,\ldots,M}\{T_{split}(i, M+k) + T_{split}(i, M+k+1)\} \\ & + \sum_{i=1}^{M}(T_{trans}(i, M+k) + T_{merge}(i, M+k)) + T_{opt}(M+k) \end{aligned} \tag{3.4}$$

Here, $T_{split}(i, j)$ denotes the lapse time of splitting the partial index with ID = j from the original one with ID = i. If there is no splitting of partial one with ID = j from the original one with ID = i, $T_{split}(i, j)$=0. Similarly, $T_{trans}(i, j)$ and $T_{merge}(i, j)$ mean the data transfer time and merging time as for the partial index with ID = j extracted from the original one with ID = i. $T_{opt}(i)$ denotes the optimization time of the index with ID = i.

3.5 Evaluation

3.5.1 Experimental Environment

In this section, we describe the experimental environment to evaluate our new consistent hashing approach to confirm the advantage in terms of resource consumption and index reconfiguration time. As for the evaluation of resource consumption, we measure the actual memory usage for our prototyping. As for the lapse time of index reconfiguration, we estimate them based on the data processing models defined in Section 3.4 and actual measurements of lapse times for index splitting, transferring, merging, and optimization processes under the following condition in Table 3.2. Our experimental prototype is implemented based on Lucene. It uses the Lucene API for index splitting, merging, optimization, and so on. The adopted base hash function is MD5. The indexing target data is a set of 10 million plain text files with a size of 16KB. These text files are generated by picking up words automatically from a corpus data based on the distribution of word occurrence. The corpus data has 12620 English words and their occurrence information extracted from the actual English documents on a web site. The reason to adopt the above file generation method is following. If they are generated by simply picking up words at random from the corpus data without the distribution of word occurrence, every word has almost uniform size of position data in the index which has different composition from the index of actual document set.

Table 3.2 Specification of the system

Node specification	- CPU: Intel Core i5 2.4GHz - Memory: RAM 4GB - HDD: SATA 350GB - OS: Linux 2.6.35 64bit
Network	- 1000 Base-T Ethernet

3.5.2 Memory Consumption

One of the issues in the utilization of the conventional consistent hashing is memory usage. As we mentioned in the previous section, if the number of nodes is over several thousands, the memory consumption reaches over several gigabytes. It is hard to store the mapping table on the memory to realize the high performance access to it. The reduction of the memory usage is the important issue as for the consistent hashing. Table 3.3 shows the memory consumption of our approach.

Table 3.3 Memory consumption of slot-based virtual node method

K	Average	SD	3*SD/Average (%)	Memory Usage (MB)
2^{10}	10000	1498.84	44.97	2.6
2^{11}	10000	1044.45	31.33	2.6
2^{12}	10000	723.95	21.72	2.6
2^{13}	10000	490.71	14.72	2.6
2^{14}	10000	314.31	9.43	5.2
2^{15}	10000	160.69	4.82	7.9
2^{16}	10000	125.86	3.78	13.1
2^{17}	10000	99.10	2.97	28.8
2^{18}	10000	98.33	2.95	57.0
2^{19}	10000	100.04	3.00	113.8
2^{20}	10000	101.75	3.05	107.4
2^{21}	10000	100.75	3.02	200.3
2^{22}	10000	99.81	2.99	147.0

SD: Standard Deviation

In this test case, the number of document entries is 10 million and number of nodes is 1000. As mentioned in section 3.1, this table data is the worst case of memory consumption when all dividing points have virtual node. In order to realize that 3*SD/Average (%) is under 10%, the number of dividing points on the hash ring space should be larger than 2^{14} (=16384). In that case, the utilized memory size is 5.2MB. This is really smaller than 589.2MB, that of conventional consistent hashing where node number = 1000.

Table 3.4 Relation between node number and standard deviation

K	Node #	Average	SD	3*SD/Average (%)	Memory Usage (MB)
2^{12}	100	100000	595.44	1.79	2.6
	500	20000	966.74	14.50	2.6
	1000	10000	723.50	21.70	2.6
	5000	2000	940.60	141.09	2.6
2^{13}	100	100000	493.47	1.48	2.6
	500	20000	612.39	9.19	2.6
	1000	10000	494.74	14.84	2.6
	5000	2000	588.32	88.25	2.6
2^{14}	100	100000	354.63	1.06	5.3
	500	20000	300.55	4.51	5.2
	1000	10000	317.13	9.51	5.2
	5000	2000	278.28	41.74	5.2
2^{15}	100	100000	342.41	1.03	5.2
	500	20000	201.17	3.02	7.9
	1000	10000	161.96	4.86	7.9
	5000	2000	157.98	23.70	7.9
2^{16}	100	100000	312.29	0.94	10.5
	500	20000	143.33	2.15	13.1
	1000	10000	123.00	3.69	13.1
	5000	2000	64.56	9.68	13.1
2^{17}	100	100000	340.55	1.02	21.0
	500	20000	143.44	2.15	26.2
	1000	10000	98.96	2.97	28.8
	5000	2000	54.58	8.19	28.8

SD: Standard Deviation

Table 3.4 illustrates the relation between the node number and standard deviation while changing the dividing number (=K) as a parameter. If the node number is 5000, K should be greater than 2^{16} in order to limit 3*SD/Average (%) up to 10%. In this case, the necessary amount of memory is 13.1MB while the conventional method requires 1894.3MB of memory resources.

Appropriate value of K is determined by the maximum number of nodes in the system to be supported. Then, the worst case of memory usage can be estimated by K. Actually, the target data properties such as average size of files and the maximum number of files in the system affects K and required memory resources. Similarly, the memory usage in the conventional method also depends on the maximum number of nodes the system can support. Thus, the memory saving efficiency of our approach against the conventional one is achieved similarly if the target data set is different.

3.5.3 Lapse Time of Index Reconfiguration Process

Finally, we evaluate the lapse time of the index reconfiguration process between the conventional method and our new approach. Assume that the number of original nodes M = 1000 and that of adding nodes N = 100 to increase 10% nodes for load reduction of search process in each node. Then, based on the discussion in Section 3.4, we can estimate lapse times of index reconfiguration processes as for the conventional method and the new method as follows. In the following estimation, we assume that individual processing time such as splitting time and merging time takes same value if the treated data size is same. Thus, the maximum or minimum values in the estimation formulas described in Section 4.1 and 4.2 can be calculated simply based on this assumption.

(1) Conventional method

(a) Splitting time: It takes 365 (sec) to retrieve the split index data with 9090 documents from the original index with 10 million documents (51.7GB).

(b) Transfer time: It takes 4.33 (sec) to transfer the single split data (45MB). Because this time is shorter compared with the splitting time, it is possible to finish the data transfer while the splitting process is executed as we mentioned in section 3.4.1.

(c) Merging time: Lapse time to merge 1000 indexes with 9090 documents is 1033 (sec). This is also executed during the splitting process similar to the data transfer.

(d) Optimization time: This is 14778 (sec). Optimization process starts only after the finish of merge processes of all partial indexes.

(2) Our new method

(a) Splitting time: It takes 575 (sec) to retrieve the split index data with 0.909 million documents (4.7GB) from the original index.

(b) Transfer time: It takes 4327 (sec) to transfer the total index data (45GB).

(c) Merging time: Lapse time to merge 10 indexes with 0.909 million documents is 1651 (sec).

(d) Optimization time: It takes 3021 (sec) to optimize the index generated by previous merge process of 10 indexes.

Table 3.5 Lapse time estimation of index reconfiguration process

M	N	Lapse time (sec) (conventional)	Lapse time (sec) (new)	Lapse time ratio (new/conventional)
500	100	50822	9364	0.184
	200	86165	8478	0.098
	300	106610	7926	0.074
1000	100	51283	9575	0.187
	200	76603	8845	0.115
	300	123992	9600	0.077

According to the above lapse times of individual processes, we can estimate that the total lapse time for conventional method = 365 (sec) * 100 + 4.33 (sec) + 1.033 (sec) + 14778 (sec) = 51283 (sec). Similarly, that for our new method = 575 (sec) + 4327 (sec) + 1652 (sec) + 3021 (sec) = 9575 (sec). Thus, the lapse time of new method is about 11.6 hours shorter than that of conventional method. In this case, our new method reduces the lapse time to 18.4% of that of conventional one, that is, our approach is about 5.43 times faster than conventional one. Furthermore, we estimate lapse times in other cases which are illustrated in Table 3.5. According to this result, we can see that our new method can effectively reduce the lapse time as the number of splitting times is greater. Multiple splitting processes could be a bottleneck for the conventional method and this is the reason why it can improve the lapse time of index reconfiguration effectively.

If the treated target data properties such as the number of files and average size of files are changed, the lapse times for both methods also changes. However, the key point of the reduction of lapse time of index reconfiguration process is still the number of splitting times because each process depends on the data size of indexes and it is proportional to the number of files and average size of files. Therefore, the saving effect of the lapse time of index reconfiguration process with our approach works efficiently if those data properties are changed. Finally, drawbacks of our approach are complex implementation and necessity to share the mapping table information among cluster nodes. Data protection of the mapping table is essential to realize robustness of the system.

3.6 Conclusion

In this chapter, we proposed a new method called slot-based virtual node method of consistent hashing to realize the efficient index segmentation and reconfiguration in the distributed search system. This new approach overcomes the issue of large memory consumption and improves the index reconfiguration process for the conventional consistent hashing. Adopting our approach, the memory consumption can be reduced to 13.1MB against 1894.3MB in the conventional method if the number of nodes is 5000. As for the lapse time of index reconfiguration process, our approach can reduce it to avoid the multiple splitting processes on each node. In the case when the numbers of original nodes and added nodes are 1000 and 100 respectively, the total lapse time can be improved about 11.6 hours according to our estimation.

The remaining problems on this research are as follows. Firstly, the implementation and evaluation of the actual index reconfiguration function is a challenge to validate our approach. System evaluation over the several hundred node cluster is important to validate the actual effectiveness of our algorithm.

Another issue is the security enhancement for the indexing and searching on the cluster. If target files have access controls put by the system administrator, access to the search result including the parts of contents should be controlled based on the ACLs (Access Control Lists). This is really required in the enterprise search field.

Chapter 4

High Performance Sort-Last Rendering over an Optical Backplane

4.1 Introduction

Recently, the parallel processing with the commodity PC cluster is the significant technologies for the batch processing and real-time processing of the large-scale data set. In the previous chapter, we discussed about the batch processing technologies of the distributed system for large-scale data processing. In this chapter, we focus on the real-time processing technology on the PC cluster. Especially, we target on the parallel rendering system as an example of the real-time processing on the cluster. Increasing in the needs of graphics technologies for high quality and resolution of three dimensional images with large data size such as MRI (Magnetic Resonance Imaging), CT (Computer Tomography) scan and CAD, the real-time parallel rendering methods are required based on the low-cost commodity machines. In this chapter, we describe the high performance sort-last rendering method over the optical backplane to correspond to this nowadays requirement.

The continual drop in the cost of commodity computers has motivated aggressive research in the development of techniques for realizing and optimizing large scale computation on PC clusters. On a cluster, each node is connected to each other by a high-speed communication network whose bandwidth can reach anywhere between 1~10Gbps, depending on the technology used. The cluster system takes the MIMD (Multiple Instruction stream Multiple Data stream) architecture on which each node can deal with its own data set on its own memory and execute the programs in parallel. Compared with the SIMD (Single Instruction stream Multiple Data stream) architecture, it is cost effective and utilization of computing resource is more efficient.

When we consider the efficiency of a parallel algorithm over a cluster, we should take one major overhead into account, i.e., communication among processing elements. To minimize the communication cost, we need to (1) communicate in bulk, (2) minimize the size of transferred data, and (3) minimize the distance of data transfer. The first and second requirements are for minimizing the start up time and transmission time, respectively. The third point depends on the topology of the cluster system and the mapping of the parallel programs. Since the propagation time over the medium of the network is usually very small on the cluster, we can ignore the distance among the nodes. We need to take (1) and (2) to optimize the communication among the processing elements. If the communication payload in a system becomes larger, e.g. the multimedia application, it is not easy to realize (1) and (2) at the same time. Because if one would like to send data in bulk, the size of each message becomes large, and if the size of each message is small, one needs to send messages more frequently. Using a huge bandwidth network can greatly reduce transmission time, and thus is the most effective way to realize these two requirements. Therefore, constructing a cluster over a high bandwidth network such as an optical network is one of the most effective solutions to handle large data sets on clusters.

The OptIPuter [89], a project at the Electronic Visualization Laboratory (EVL) and the University of California San Diego, is a computing model which uses optical networking as a backplane to connect clusters of computers that are collectively regarded as large computer peripherals. For example, a cluster of computers with massive RAID disks are thought of as a single large disk drive; and a cluster of computers with advanced graphics cards are thought of as a single giant graphics card. These peripherals are then interconnected with optical networks to form a wide variety of virtual computers that can be specifically customized to meet application's requirements. The Gigabit Ethernet switch which supports the optical fiber connection converts the optical signal into the electrical signal to realize the packet switching internally in the conventional way. Although the achievable bandwidth of the optical fiber is over 50Tbps, the current limitation of the throughput is 100Gbps through the Gigabit Ethernet switch which is realized by parallel transmission channels such as 10Gbps$\times$10 or 25Gbps$\times$4. This is due to the response time of a photodiode. The limitation of the signal sampling causes the limitation of the traditional optical-electrical network switch. Meanwhile, the photonic switch adopts totally different architecture for switching network. It uses the all-optical MEMS devices to switch the connection inside. The optical signal incoming via the inbound fiber is routed to the outbound fiber with the micro-mirrors and lenses in the silicon. There are no signal changes from optical to electrical. This technology can avoid the bottleneck of the optical-electrical converting signal and make utilize of the optical fiber's bandwidth possible up to the upper limitation. The advantage of the photonic switch based cluster is that the bandwidth of the interconnection among cluster nodes could be over 1000 times larger than the traditional Gigabit Ethernet cluster.

For example, if the rendering cluster deals with high resolution images such as QUXGA wide resolution at 20 fps, the required network throughput for the peer-to-peer node connection through the network switch is about 2.9Gbps. Assume that the number of interconnections inside the switch is 21 in 13 node tree-structured cluster case, the backbone throughput is required over 60.9Gbps for one cluster. If we construct multiple clusters for the stereo view or tiled display, it is required multiple times of the throughput for a single cluster. So, the high bandwidth switches like pure photonic switches works effectively to avoid the network bottleneck for rendering the large-scale image data. Furthermore, the cost performance of the photonic switch is better than conventional electronic switch which has the similar bandwidth specification. It is also the energy-saving appliance with 50 watts of power consumption against the electronic switch whose power consumption is several thousand watts. The every connection inside the photonic switch can occupy the full bandwidth of the network without congestion is also the great advantage over the electronic one.

However, there is a serious drawback on the photonic switch based cluster. The switching delay takes about over 20 milliseconds. If the switch changes connections among the cluster nodes frequently, the performance of the parallel computation will be degraded. We can expect the performance improvement of the parallel computation when the connection among the cluster nodes does not change frequently compared with the processing time for the assigned task on each node. Especially, if we adopt the parallel algorithm in which the data flow is static among the nodes and is going to the single node like a tree-structured connection, the switching does not happen after the initial connection establishment. We can hide the drawback and get the benefit of the photonic switch based cluster.

In this chapter, we discuss the design and implementation of the sort-last rendering system on the photonic switch based cluster. Contents of this chapter are as follows; first of all, we explain the sort-last rendering in Section 4.2. Then, we discuss the design and implementation of our system in Section 4.3. In Section 4.4, we will provide experimental results and analysis of the system. Finally, we discuss a feedback control mechanism to realize a stable burst flow in the cluster in Section 4.5.

4.2 Sort-Last Rendering

There are three well-known parallel rendering algorithms, sort-first, sort-middle, and sort-last rendering. Their differences are characterized by the time when the primitives are distributed to several processors in the graphic pipeline. The following Fig. 4.1 illustrates the taxonomy of the parallel rendering architecture.

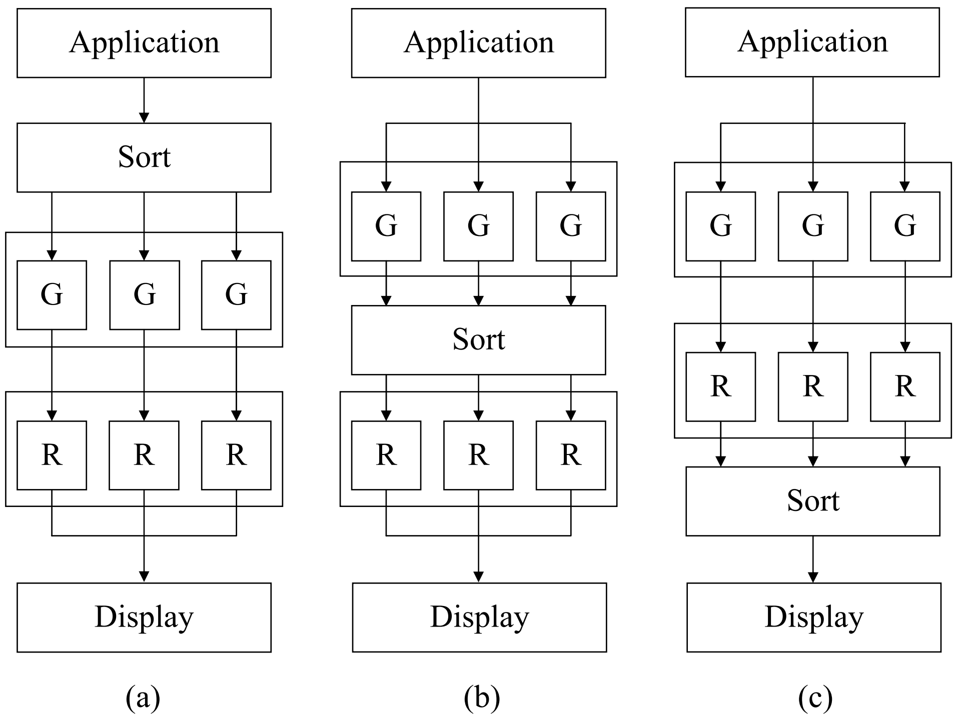


Fig. 4.1 Taxonomy of the parallel rendering architecture
(a) sort-first, (b) sort-middle, and (c) sort-last

The graphic pipeline has three stages, the application processing, the geometry processing, and the rasterization. At the geometry processing stage, each geometry unit

G processes the geometry to be rendered. At the rasterization stage, each rasterizer unit R handles the pixel calculations. In the sort-first rendering, the "raw" primitives are distributed early to each processor during the geometry processing stage. Each processor is assigned to a part of the entire display which is divided into disjoint regions, and it renders the assigned primitives individually.

In the sort-middle rendering, the distribution of the work is arbitrary and even among the geometry units. Each rasterizer unit is responsible for a screen space region. After the geometry processing, each primitive is allocated to the corresponding rasterizer unit that is responsible for the screen space location of the primitive.

The sort-last rendering, on the other hand, defers sorting primitives until the end of the rendering pipeline, i.e. after primitives have been rasterized into pixels. Each processor is assigned a subset of primitives and renders them no matter where they locate on the screen. After rendering, processors communicate with each other to composite those pixels to generate the final entire image. In order to handle the real-time high quality image rendering, the high data rates over the internetwork among the rendering processors is required. This is one of the reasons why we target on the parallel computing system over the high bandwidth optical network.

There are some techniques to optimize the data transfer in the sort-last rendering. One is the bounding rectangle method. It is also called SL-sparse. It minimizes the data transfer by only sending the pixels with actual data (active pixels). In order to encode the active pixels, (1) you find a smallest rectangle which contains all actual pixels in the rendered image, (2) take coordinates of upper left and lower right points, and (3) pack these coordinates and the image data inside the rectangle as the buffer to send. When the original image is sparse, the optimization is done efficiently.

At the composition stage of the sort-last rendering, because the composition of active pixel and non-active pixel is the active pixel, we should only compose the overlapping region of two rectangles. This composition technique reduces the time to compose two images.

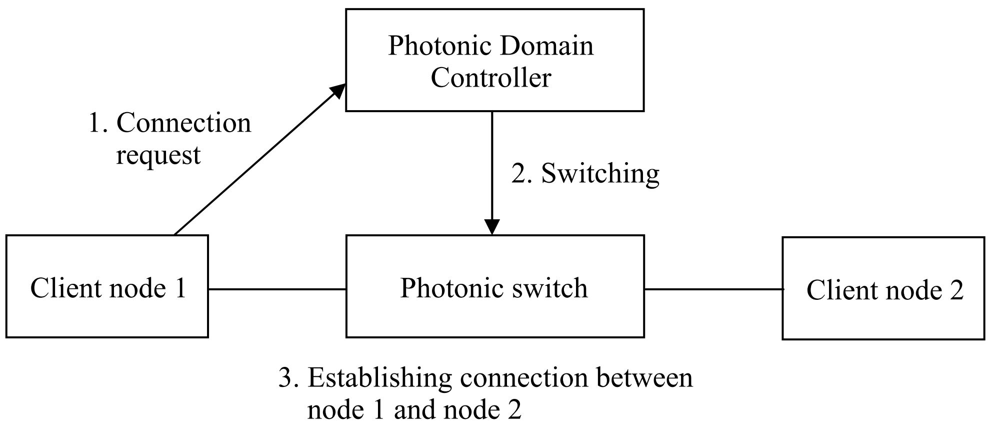
Another optimization technique is the run-length encoding method. In the method, each pixel is classified into two kinds of pixel, active pixel and non-active pixel. Counting the continuously locating non-active pixels and encode the count as the integer into the sending buffer, the total size of pixels shrinks. Combining these two methods, we can optimize the data transfer rate in the sort-last rendering and improve its performance.

4.3 System Design and Implementation

4.3.1 Photonic Computing Engine

The Photonic Computing Environment provides a high performance computing mechanism over the photonic switch based cluster system. It constructs the pipelines among the cluster nodes and manages the computation flow. In order to use the photonic switch to construct the rendering cluster, the cluster application needs to use the PDC (Photonic Domain Controller) [90] to generate the pipeline connection among the cluster nodes. Because the current existing library for parallel programming such as MPICH does not support the manipulation of the connection inside the photonic switch, it is necessary to implement the network application which generates the network pipeline among the cluster nodes over the photonic switch. Fig. 4.2 illustrates the system configuration of PDC, photonic switch and client nodes of the switch. The PDC provides the interface to create the link inside the photonic switch. The network application that uses the PDC at first invokes the PDC's interface to establish the connection between two nodes. After generating the connection, those nodes can communicate each other with any protocols such as TCP and UDP. The connection can occupy the whole bandwidth allocated at the initialization time. It is disconnected when communicating nodes explicitly invoke the disconnect function on the PDC.

Fig.4.3 shows the architecture of the Photonic Computing Environment. Network applications run on the cluster of PCs over the photonic switch in this computing



Photonic Domain Controller
1. Connection request
2. Switching
Client node 1
Photonic switch
Client node 2
3. Establishing connection between node 1 and node 2

Fig. 4.2 System configuration of PDC, photonic switch and client nodes

environment. The PCE (Photonic Computing Engine) handles the establishment of the connection among the cluster nodes and provides the functionality to synchronize messages to the add-in calculation module such as image rendering module and image composition module.

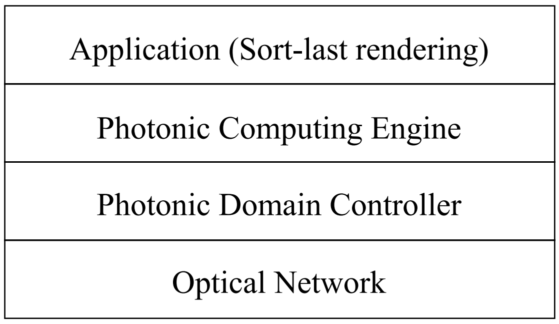

Application (Sort-last rendering)
Photonic Computing Engine
Photonic Domain Controller
Optical Network

Fig. 4.3 Architecture of Photonic Computing Environment

The PCE has the two types of data transfer mechanism, pull-up mode and push-out mode. In the pull-up mode, the client sends a request to the PCE and it returns the results as the C/S system. In the sort-last rendering case, the viewer on the client send rendering request to the PCE each time when it needs to change the view. On the other hand, the outputs of calculations on the PCE are generated as much as possible and sent to the client in the push-out mode. The push-out mode is useful if the computation results are automatically generated like animations and movies.

4.3.2 Architecture of the Sort-Last Rendering System over the PCE

The PCE is the application that provides the network pipeline among the nodes on the photonic switch based cluster and synchronization mechanism to realize the sort-last rendering. Fig. 4.4 shows the architecture of the PCE with 7 nodes for the sort-last rendering.

On the each node, the PCU (Photonic Computing Unit) is running and generates the pipeline. The client application accesses to the root PCU to get the computing result. In the sort-last volume rendering system, the PCU plays two types of roles, the composition proxy and the rendering server. Each rendering server fetches the allocated part of volume data and renders the image. After rendering the image, the rendering server sends to the composition proxy, which is a parent node of it. At the composition proxy, it synchronizes the output images and composes them.

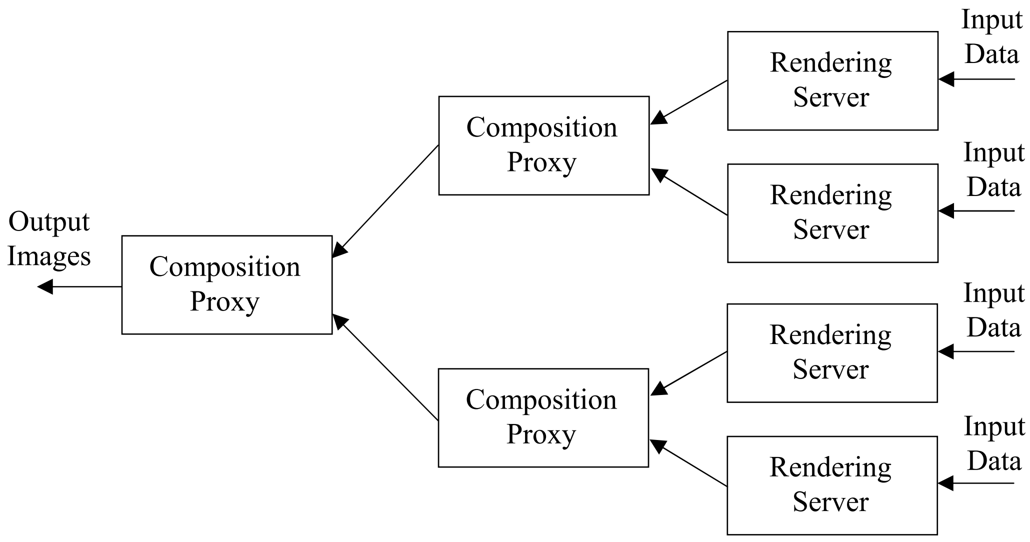


Fig. 4.4 Architecture of the 7-node sort-last rendering cluster

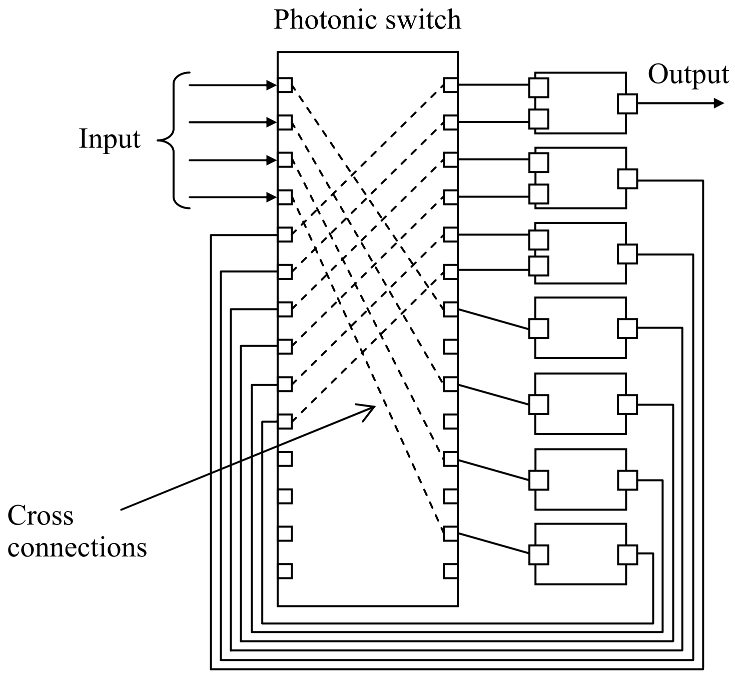


Fig. 4.5 Physical configuration of the sort-last rendering cluster

Fig. 4.5 shows the physical configuration of the sort-last rendering cluster. Seven nodes are connected to the photonic switch through optical fiber cables. Each node has several network cards to be assigned individual IP addresses. Because the network connections among nodes are fixed, they have to have individual network cards for inbound and outbound data connections. The cross connections are established inside the photonic switch at the initial stage of the data processing. Once they are established, they do not change during the parallel processing over this pipeline.

4.3.3 Implementation of PCE

In this section, we will describe the actual implementation of the PCE. PCE has basically the following functionalities, message transferring, message queuing, flow control, and module add-in.

(1) Message transferring

The PCE generates the cluster as a tree. When the client sends the request to the PCE, the root node has to propagate the request message to the computing nodes such as the rendering servers. Since switching the connection among the nodes in the photonic switch takes much cost, the PCE does not change the connection pattern. The message needs to be passed along the tree-structured connection. Therefore, the each PCU has the message transferring mechanism from the parent node to the child nodes.

(2) Message queuing

On the intermediate PCU, the synchronization mechanism is required because the intermediate PCU might use both results sent from two child PCUs. Each message sent from the child PCUs has a sequential number and it is used to synchronize the output results. Since the output messages from the child PCUs are sent to the intermediate PCU asynchronously, it needs to store the messages in a queue to synchronize them.

(3) Flow control

In the push-out mode, the rendering server sends output image to the composition proxy. If the output message rate of the rendering server is better than that of the other rendering servers or ability of message processing at the composition proxy, the queue could overflow for the message burst. Therefore, the flow control mechanism is required in the composition proxy. In order to control the flow, we use the socket buffer and TCP flow control mechanism. If the socket buffer is full, the sender process is blocked on a TCP connection. Thus, if the length of the queue becomes maximum, the composition proxy blocks the receiving process until a queue element is consumed by another process. The blocking of the receiving process on the proxy is propagated to the child node and stop sending data.

(4) Module add-in

Besides the message routing mechanism, the PCE provides the computation add-in mechanism, that is, you can replace the composition and rendering part of implementation to other one like the add-in module. You can easily change the computation algorithm on the PCE by overwriting the computation part of composition nodes and rendering nodes. For example, the proxy provides the callback function that is invoked when all output data from child nodes reach at the proxy. One can overwrite the callback function that handles the output buffers to implement other composition algorithms. Also, the rendering server provides the display function as the callback function. If one would like to implement the other rendering algorithms, one can modify the display function to realize it.

We explain the implementation of the rendering server, composition proxy, and client viewer. The rendering server renders the part of the volume data with the 3D texture mapping method. After rendering the assigned part of volume data, it fetches the image data from the frame buffer. The fetched image is cut into the smallest rectangle which includes the active part of the image, encoded by the Run Length Encoding algorithm, and sent to the composition proxy [91]. Fig. 4.6 indicates the rectangle extraction including active pixels from an image described above.

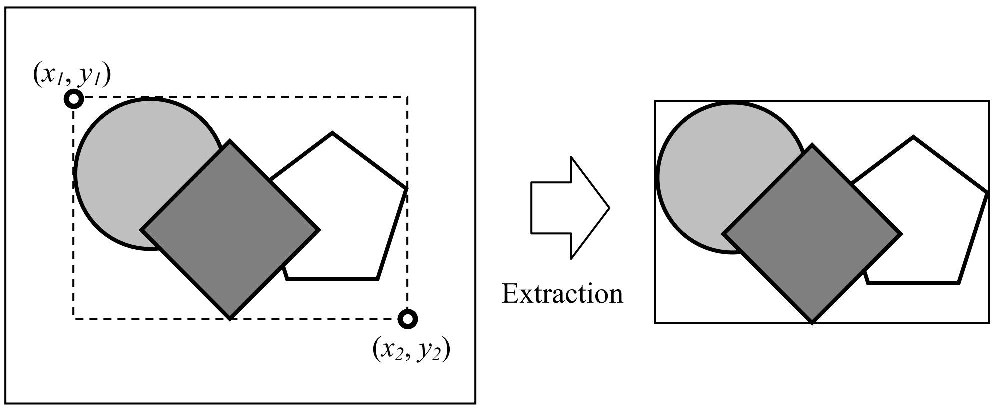


Fig. 4.6 Rectangle extraction including active pixels

The composition proxy receives the encoded images from the child nodes such as the rendering server and other composition proxy. The encoded images are decoded and checked whether the two image rectangles have overlapping part or not. If so, overlapping part of two image rectangles are fetched and composed. Then, the composed part is embedded into the image rectangle which includes two image rectangles inside. Fig. 4.7 shows the composition algorithm. Composing the overlapping part of the rectangles, we can omit the other redundant composition such as the composition with the blank part of pixels. After finishing the composition, it packs the data as the message with appropriate header and sends the packed message to the parent node.

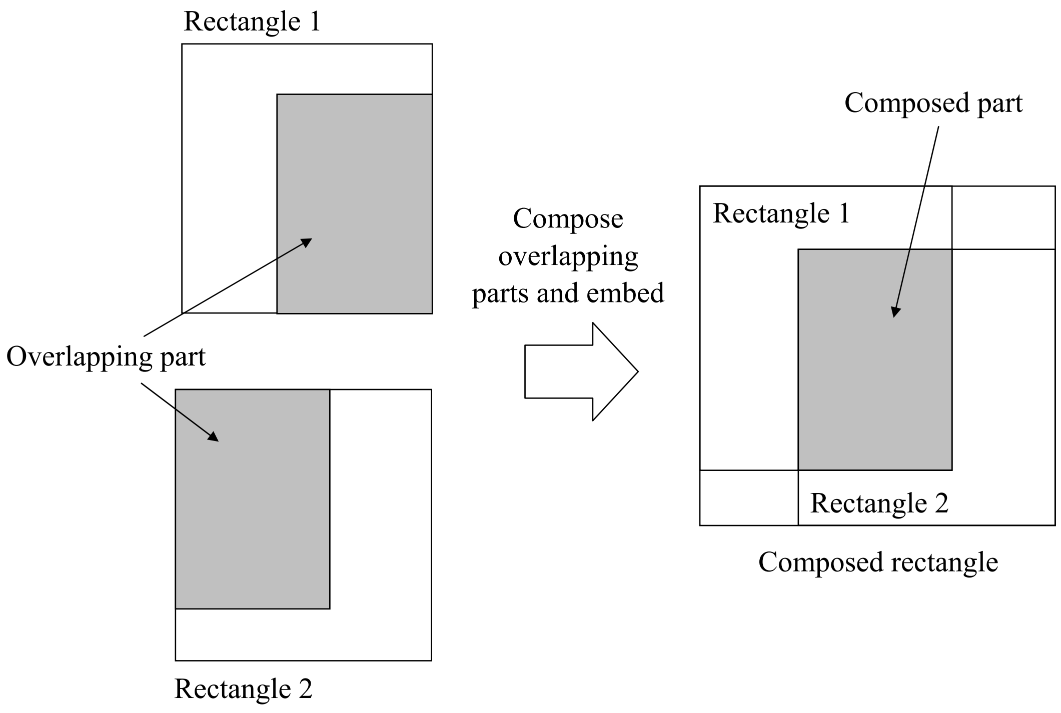


Fig. 4.7 Composition of the overlapping part of two rectangles

The client viewer also has a queue to store the image data sent from the composition proxy. It fetches the encoded image and pushes it into the queue. The display routine of the client viewer popes the image data from the queue, decoding data, embedding it into the original size of blank rectangle, and maps it onto the square polygon. It also has an interface to change the argument of the volume image. When you drag the mouse over the display window, the bounding box rotating on the window and send the request message to the composition proxy when you release the mouse button. We can switch pull-up mode or burst image mode with the client viewer.

Setting the configuration file, one can specify the tree structure of the cluster. In the configuration file, the information of the network connection can be described by the port numbers and network addresses of parent node, child nodes and message transfer service on each node. Each node has the ID specified by the command line argument at the beginning of the execution. In the configuration file, the set of parameters for each node is separated and identified with the node ID. Each node reads its configuration part from the configuration file according to its ID.

Since the cluster in EVL consists of the nodes on which Linux is running and Linux cannot recognize more than 2 optical network cards, the proxy cannot have 3 network cards to construct the data processing pipeline currently. Therefore, the current implementation does not have the function to construct the connections over the photonic switch with PDC. However, once the pipeline is constructed with PDC, the communication overhead in terms of the PDC does not happen during the computation of image rendering. Additionally, bandwidth of an optical network card and a regular Gigabit network card are similar to each other (both have around 1Gbps). We can simulate and evaluate the performance of PCE somehow in the current implementation.

4.4 Experimental Result

In order to evaluate the PCE, we implemented the sort-last volume rendering system over the PCE and took some experiments. The volume rendering is a method to visualize the volume data which is sampled by MRI or CT scanner. The sampled data has the scalar value for each point in the three-dimensional spatial data. Several methods are proposed to visualize the volume data. The representative methods are ray casting, splatting, shear-warp and hardware-assisted 3D texture mapping [92][93]. We implemented the 3D texture mapping method to render the volume in the system

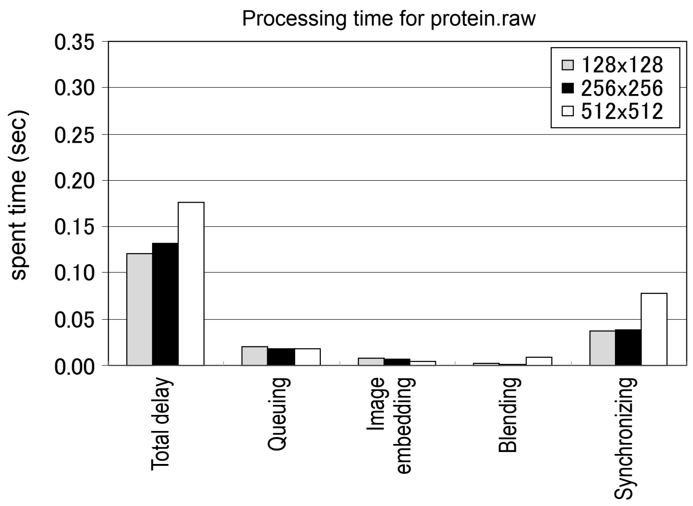
We used the cluster that has 16 nodes, 1 master and 15 slaves. Each node has dual Xeons 1.8GHz and 1.5GB memory. The graphics card is PNY Quadro FX3000 and the Gigabit Ethernet card is equipped on each node. All nodes are connected to the Gigabit Ethernet Switch to construct a cluster. In the experiment, we constructed the 7-node sort-last rendering system on the cluster and rendered the three sample volume data, protein.raw, hydrogen.raw and foot.raw, which have sizes of $64\times64\times64$, $128\times128\times128$, and $256 \times 256 \times 256$ respectively [94].

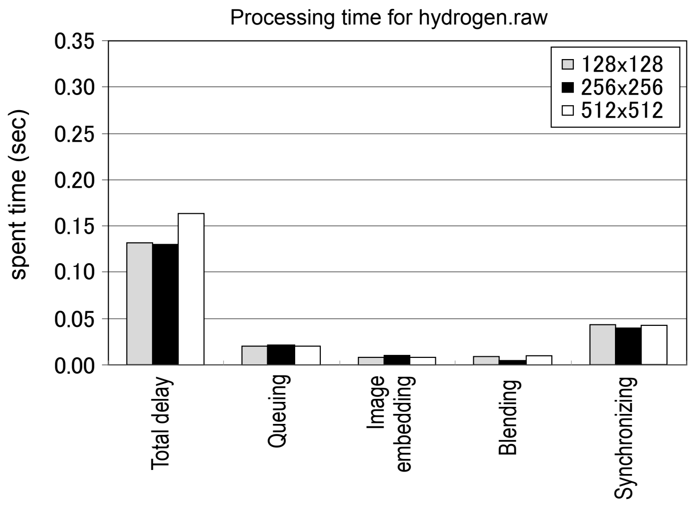
We took several trials for image resolutions 128×128, 256×256 and 512×512, and measured time intervals on the client viewer, the composition proxy, and the rendering server. The measured time intervals are the total delay, queuing time, blending time, bounding rectangle calculation time and so on.

The graphs in Fig. 4.8 show the total delay and spent time for each process in the system. Total delay means how long it takes from the start of sending request message to final image displaying on the client viewer. Image embedding time is the time to embed the partial rectangle image into the original size of blank image to generate the final one. Blending time is the composition processing time for two received images. Synchronization time is the time to take for synchronizing the received data, that is, the time interval from the arrival of the first image to the arrival of the final image. It is actually the time the data spent in the queue until it is popped out. Finally, queuing time is the time to push and pop the data in the queue respectively. Queue data is stored in the shared memory. Attaching, detaching, reading and writing data to the shared memory is the main processes of the queue handling.

From the experimental results, the total delay and synchronization time for a single image rendering are up to 0.288 seconds and 0.167 seconds respectively in the case when the resolution size is 512×512. It actually achieves enough performance for dialogical real-time rendering system. As it can be seen in these graphs, the total delay increases as the resolution size does. The number of polygons did not affect the performance explicitly, since the performance of the rendering server is so much better compared with the processing performance inside the proxies. From these results, we can see that the blending time increases as the size of the resolution becomes larger. The reason is like this. If the resolution is larger, the overlapping part of the two rectangles on the composition process becomes larger. The blending calculation spends more time as the size of the overlapping part of the two rectangles increases. Actually, blending time is proportional to the image size.

Other time intervals such as synchronization, queue push and queue pop do not change explicitly in this experiment. However, the synchronization time can increase if the transferred data size is getting larger, since it includes the data receiving time. Thus, we can say that the transmission time affects the synchronization time and total performance of the frame rate on the client viewer significantly.





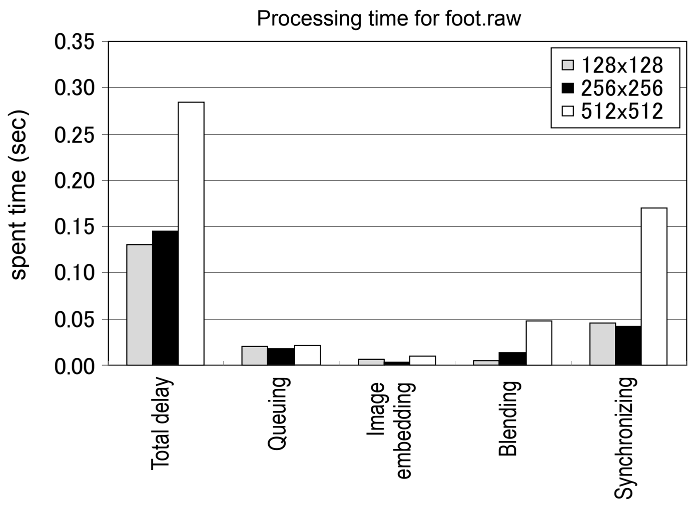


Fig. 4.8 Total delay and spent time of each processing in the 7-node volume rendering cluster system

Here, we give further considerations for the result in Fig. 4.8. In terms of the synchronization time, it exceeds largely over 0.05 seconds in 512×512 resolution comparing with other ones. This implies that the data transmission time affects the synchronization time strongly in this case. Thus, if the transmission speed goes to 10 times faster, the synchronization time is under 0.05 seconds. On the other hand, the blending time is proportional to the size of image resolution. For instance, changing the resolution from 256×256 to 512×512, the blending time becomes 4 times longer. In the case of foot.raw, the bottle neck is changing from synchronizing process to blending process if the transmission speed is 4 times faster because the synchronization time is nearly equal to that in 256×256, which is under 0.05 seconds.

Fig. 4.9 shows the frame rate on the client viewer when the system pushes out the output image as fast as possible or keeps the sending rate in a certain speed, such as 10 fps, 15 fps, and 20 fps. In order to keep the sending rate, the rendering server takes sleep for appropriate time in the redraw routine.

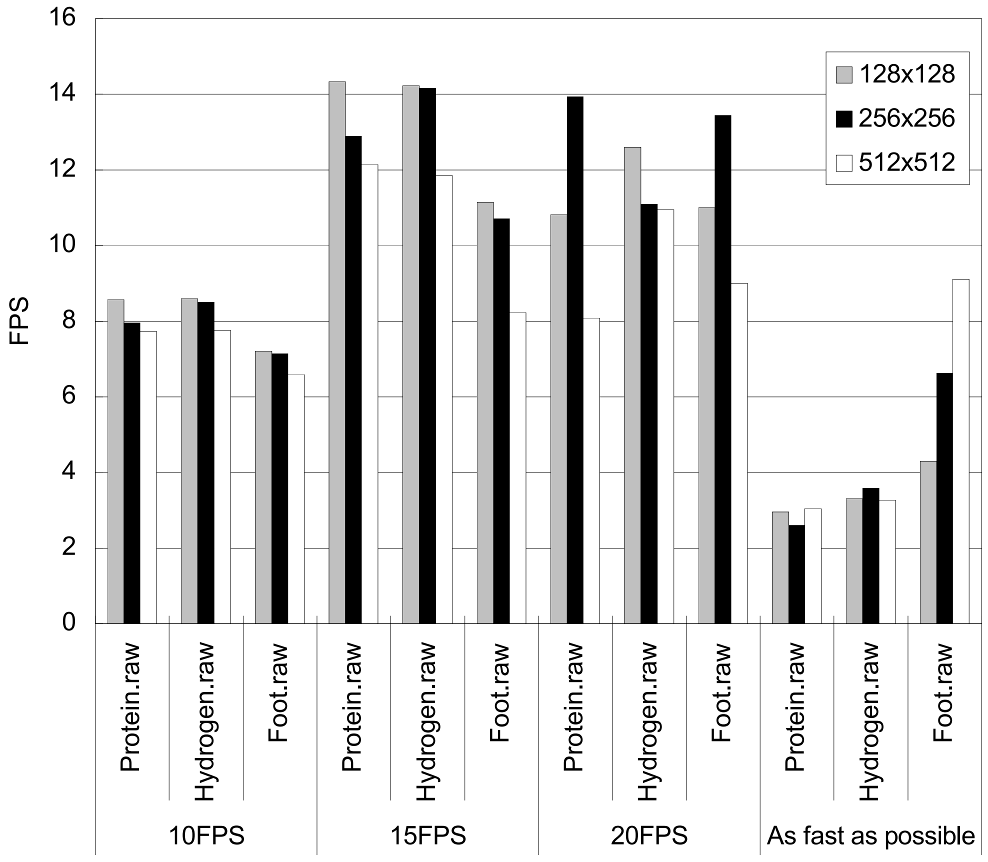


Fig. 4.9 Frame rate in push-out mode

From the experimental result of the push-out mode, our system can achieve the throughput from 9 to 14 fps on the image streaming process. If the rendering server sends the data as much as possible, the actual frame rate is not good, because the data flow in the cluster is not smooth. When the message-sending rate at the rendering servers is too high, the queues in the composition proxies can be full easily and frequently because once one rendering server's performance get worse, the other one send messages during the time and the many messages which cannot be synchronized arrive at the composition proxy. Controlling the output of the rendering server, data flow inside the cluster get smooth and the frame rate is improved as you can see in the other sending rate cases. What is the optimal message-sending rate on the rendering servers is the significant problem in order to maximize the performance.

4.5 Data Flow Control Mechanism in PCE

In this section, we discuss the flow control mechanism [95][96] for the push-out mode. The objective of this mechanism is to adaptively determine the optimal push-out rate for the rendering system to ensure maximum frame rate. Fig. 4.10 shows the data flow model of the 7-node cluster, where n_{ij} denotes the number of messages generated by a rendering process *j*, and n_{oj} denotes the number of messages handled by a composition process *j*.

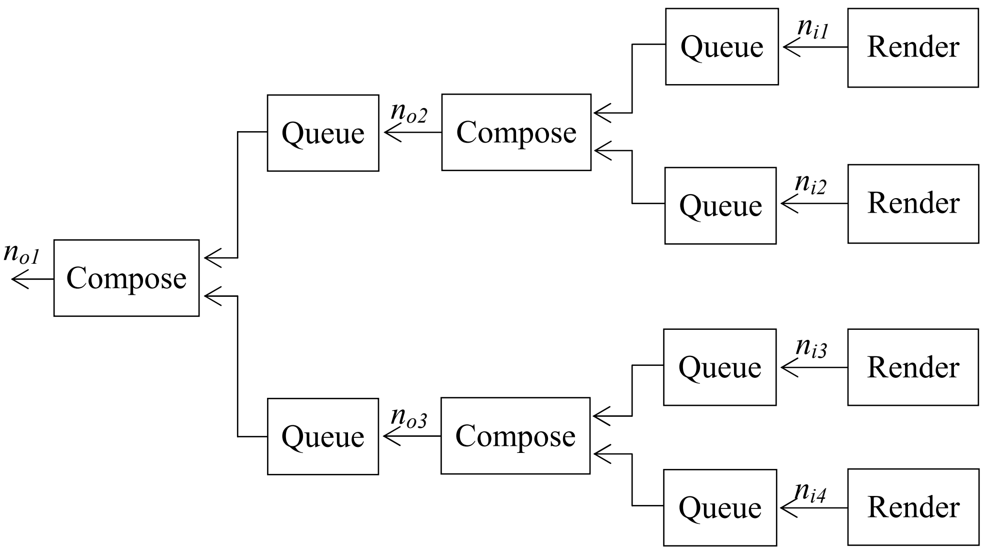


Fig. 4.10 The data flow model of the 7-node cluster

A composition process assembles two messages together each of which comes from a different queue, and creates one message. The whole system is composed of six small equivalent sub-systems as shown in Fig. 4.11, where n_i is the number of input message, and n_o is the number of output message.

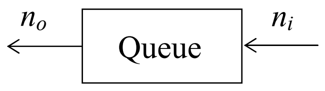


Fig. 4.11 The sub-system

A system achieves the maximum-speed response with a predictable operation if the system operates at a state close to stability boundary. A queue pair refers to two queues feeding the same composition process. If two queues in a queue pair are stable, i.e., always have some messages to feed a composition process and at the same time no queue has an overwhelming number of messages than another, the system can achieve the maximum message-sending rate. In other words, the system can achieve the maximum rate by maintaining a non-zero and small queue in a steady state and draining queues when the sources do not have messages to send. Therefore, in order to achieve the maximum rate, we should maintain a small number of messages in every queue. This is the problem of making the sub-system be stable.

A queue does not change the number of messages, but it imposes delay to message flow. Thus a queue can be modeled as a delay part, so is a composition process. Let *l=l(t)* to express the length of a queue, then we have:

$$\frac{dl}{dt} = \begin{cases} r_i - r_o & if \quad l > 0 \quad or \quad r_i > r_o \\ 0 & otherwise \end{cases} \tag{4.1}$$

Since queue length is greater than zero in a stable state, therefore, we have,

$$l(t) = \int_0^t (r_i(u) - r_o(u))du \tag{4.2}$$

whose corresponding Laplace transform is

$$L(s) = \frac{R_i(s) - R_o(s)}{s} \tag{4.3}$$

The message-sending rate is the derivative of the number of messages generated by a rendering process and output rate is the derivative of the number of messages processed by a composition process, i.e.,

$$r_o(t) = \frac{d}{dt} n_o(t) \tag{4.4}$$

$$r_i(t) = \frac{d}{dt} n_i(t) \tag{4.5}$$

Laplace transform for a derivative is

$$L\left[\frac{d}{dt} f(t)\right] = sF(s) - f(0\pm) \tag{4.6}$$

Since $r_o(0) = 0$ and $r_i(0)=0$, we have Laplace transforms for r_o(t) and r_i(t) as follows.

$$R_o(s) = L\left[\frac{d}{dt} n_o(t)\right] = sN_o(s) \tag{4.7}$$

$$R_i(s) = L\left[\frac{d}{dt} n_i(t)\right] = sN_i(s) \tag{4.8}$$

Therefore, the fluid-flow model for the open-loop sub-system can be modeled as shown in Fig. 4.12, where τ is the delay of the sub-system, R_i and R_o is the Laplace transform of the message-sending rate and message-output rate respectively.

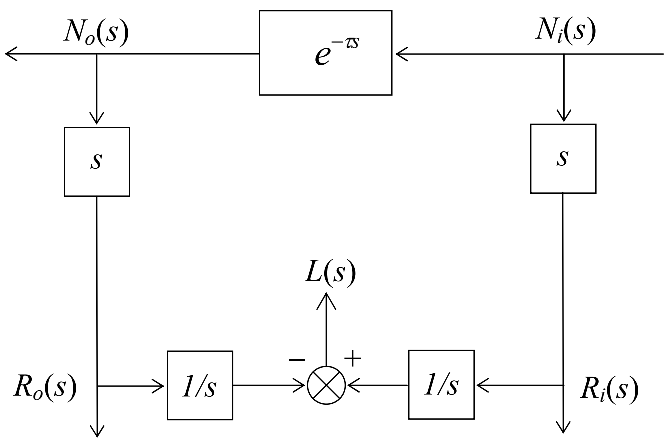


Fig. 4.12 Open-loop sub-system

From Fig. 4.12, we have the following equations.

$$R_o(s) = sN_o(s) = se^{-\tau s}N_i(s) = se^{-\tau s}\frac{R_i(s)}{s} = R_i(s)e^{-\tau s} \tag{4.9}$$

From Eq. (4.3) and Eq. (4.9), we know the open-loop transfer function of the system is as follows.

$$\frac{L(s)}{R_i(s)} = \frac{1-e^{-\tau s}}{s} \tag{4.10}$$

Here, τ is small, so the system can be considered as a linear system within a small range of time. Thus the system can be analyzed by the stability criterion of a linear system, which says that a system is stable if all roots of its characteristic equation lie to the left of imaginary axis in the s-plane.

The characteristic equation of this system has only one root, s = 0, which means that the system is boundary stable. However, usually a boundary-stable system is not stable in operation. To make this system stable, one option is to add a negative feedback to the system, as shown in Fig. 4.13, where λ is a feedback gain.

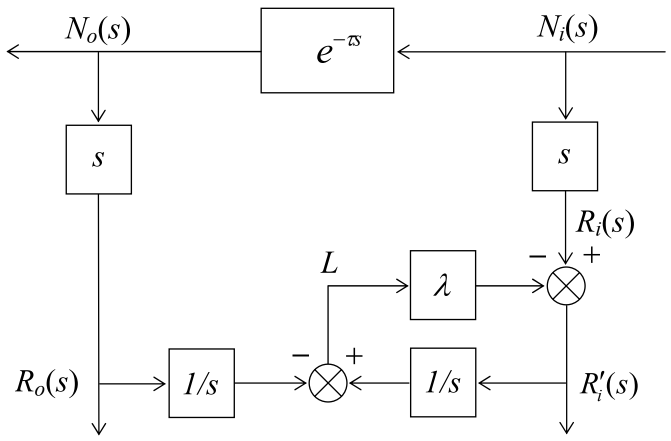


Fig. 4.13 Closed loop subsystem

From Fig. 4.13, we know the following equations.

$$L(s) = \frac{R_i'(s) - R_o(s)}{s} \tag{4.11}$$

$$R_i'(s) = R_i(s) - \lambda L(s) \tag{4.12}$$

$$R_o(s) = R_i e^{-\tau s} \tag{4.13}$$

Thus, the closed-loop transfer function is the following.

$$\frac{L(s)}{R_i(s)} = \frac{1 - e^{-\tau s}}{s + \lambda} \tag{4.14}$$

The system has a pole at -λ, therefore, the system is stable provided $\lambda > 0$.

We can apply this control mechanism to the 7-node cluster as follows. When the composition proxy receives the message, it returns the feedback λL to the child node where L is the current queue length. Then, the child node controls the message-sending rate to r_i-λL. If the child node is the rendering server, it can add some time interval in the display loop to control the sending rate. If the child node is the composition proxy, it adds some time interval in the composition routine to control the message-sending rate to realize the feedback control.

4.6 Conclusion

We designed the sort-last rendering cluster system with photonic switch over the optical fiber network and implemented the system to evaluate its performance. From the experimental results, the total delay and synchronization time for a single image rendering are up to 0.288 seconds and 0.167 seconds respectively in the case when the resolution size is 512×512. It actually achieves enough performance for dialogical real-time rendering system.

According to the analysis of the internal processing times for a single image rendering, performance of the sort-last parallel rendering system is mainly affected by the image blending time and synchronization time. Especially the synchronization time accounts for almost half percentage of the total lapse time. Other time intervals such as queuing time and image embedding time do not change explicitly in the different conditions. The synchronization time can increase if the transferred data size is getting larger, since it includes the data receiving time. Thus, we can say that the transmission time affects the synchronization time and total performance of the frame rate on the client viewer significantly. We can also see that the blending time increases as the size of the resolution becomes larger. If the resolution is larger, the overlapping part of the two rectangles on the composition process becomes larger. The blending calculation spends more time as the size of the overlapping part of the two rectangles increases.

If the transmission speed goes to 10 times faster, the synchronization time is under 0.05 seconds because it is possible to ignore the effect of the transmission time. On the other hand, the blending time is proportional to the size of image resolution. In the case of foot.raw, the bottle neck is changing from synchronizing process to blending process if the transmission speed is 4 times faster because the synchronization time is nearly equal to that in 256×256, which is under 0.05 seconds.

While the frame processing ability of the composition proxy is related to the resolution and the density of the active pixel, it is relatively independent on the number of polygons rendered on the rendering server. Thus, the system can keep the rendering performance until when the number of assigned polygons to the rendering server reaches to the limitation of graphics cards equipped on it, even if the number of polygons increases much more. On the other hand, the rendering performance will be getting worse if the image resolution size is greater because the transmission cost is larger. When the display sizes of contents on the image are not large, the performance could be kept better because the compression rate of the messages is higher. But if they are enlarged on the screen, the compression rate is worse to drop the rendering performance. Partitioning the image and increasing the clusters could be a remedy in this case.

As for the experimental results of the push-out mode, our system can achieve the throughput from 9 to 14 fps on the image streaming process. However, when rendering servers generate images as much as possible and the message-sending rate exceed the processing capability on the composition proxy, the frame rate on the client viewer gets worse. To realize the optimal data flow inside the cluster, we propose the flow control mechanism which calculates the optimal message- sending rate from the current queue length L. Actually the feedback system needs to set the message-sending rate r_i-λL for given L to optimize the data flow inside the cluster. Evaluating the efficacy of the adaptive flow mechanism is the future work, as well as testing in a fully realized optical network.

Chapter 5
Conclusions

5.1 Concluding Remarks

In this dissertation, we discussed reliability and performance improvement of the distributed system for large-scale data and proposed the reliable and secure pseudo thin client method, the slot-based virtual node method of consistent hashing, and the sort-last rendering method over the cluster based on the photonic switch. This dissertation reports these research results individually in the following four chapters.

In the chapter 1, we described the trend of technologies related to the distributed system consist of commodity servers for large-scale data set and clarified its significant issues. Additionally, we explained the related works in terms of technologies related to the individual issues and our research strategies.

In the chapter 2, the reliable and secure pseudo thin client based on the write protection of local disks with virtual cache mechanism was proposed. The proposed method realizes the write protection to the local disks to make the commodity PC as the pseudo thin client terminal. In order to solve the memory exhausting issue, we implemented the virtual cache mechanism on the write protection driver to evacuate the cached data from memory to local disk or iSCSI volumes. The simulation result of the paging algorithm indicates that RANDOM is the best one for the virtual cache mechanism. We also evaluated the disk I/O performances in the write-protected mode and paging processing to local disks and iSCSI volume to check its feasibility. According to the evaluation results, the disk I/O performance in the write-protected mode is much better than the raw disk access performance because of the RAM disk effectiveness. The disk I/O performance during the paging process also takes practical values to verify its feasibility.

In the chapter 3, we proposed the slot-based virtual node method of consistent hashing. The proposed method can save the memory resources and shrink the reconfiguration time of distributed indexes in the processes of adding and removing cluster nodes. In our approach, the hash ring space of consistent hashing is divided into multiple regions with equal size. The virtual nodes are plotted on the slot boundaries on the hash ring space where the sizes of allocated range spaces are almost equal for every node to uniformize the probability of entry allocation to each node. When the multiple nodes are added or removed, our slot-based virtual node reallocation algorithm works to plot or reallocate the virtual nodes on the hash ring space to realize "a bunch of" data migration as far as possible to optimize the index reconfiguration. In order to evaluate our approach, we measure the memory consumptions and estimate the lapse times of index reconfiguration processes to check advantages of our approach compared with the conventional method. Adopting our approach, the memory consumption can be reduced to 13.1MB against 1894.3MB in the conventional method if the number of nodes is 5000. As for the lapse time of index reconfiguration process, our approach can reduce it to avoid the multiple splitting processes on each node. In the case when the numbers of original nodes and added nodes are 1000 and 100 respectively, the total lapse time can be improved about 11.6 hours according to our estimation.

In the chapter 4, we proposed the sort-last rendering method over the cluster based on the photonic switch to realize the high-performance rendering of large-scale three-dimensional image data. In order to solve the drawback of switching delay in the photonic switch, our sort-last rendering system constructs the tree-structured pipelines to avoid the network switching during the parallel rendering process. From the experimental results, the total delay and synchronization time for a single image rendering are up to 0.288 seconds and 0.167 seconds respectively in the case when the resolution size is 512×512. It actually achieves enough performance for dialogical real-time rendering system. Performance of the rendering system is mainly affected by the image blending time and synchronization time. The total delay increases as the resolution size does. In addition, our system can achieve the throughput from 9 to 14 fps on the image streaming process. To realize the optimal data flow inside the cluster, we proposed the flow control mechanism which calculates the optimal message-sending rate from the current queue length L. Actually the feedback system needs to set the message-sending rate r_i-λL for given L to optimize the data flow inside the cluster.

5.2 Future Directions

Finally, we describe the future problems and directions on this research.

(1) High-speed achieve and search technologies for massive sensor data and log data

In the chapter 3 of this dissertation, the parallel batch processing is discussed to realize the high-speed index generation for the large number of target files. Hadoop technology is one of the powerful and effective methodologies for this kind of processing. Recently, these data archive and processing frameworks using commodity servers are getting popular in the big data market. Especially, the technologies for archiving and analyzing the machine generated data such as sensor data and log data are getting more important. The high-speed archive and search technologies for massive machine generated data is one of the significant research issues with respect to the distributed system for large scale data processing

(2) Secure batch processing framework with PC cluster for sensitive information

In the chapter 2 and 3 of this dissertation, we discussed on the security and high speed batch processing on the distributed system that consists of commodity machines. In the near future, the improvement of batch processes by using the Hadoop framework will be more popular in the world. In fact, the Hadoop-based solutions for improving the daily batch processes of business data stored in DB are provided by several IT vendors recently. However, there are not enough methodologies of data security handled in the Hadoop framework currently. The secure batch processing architecture for the sensitive information is the significant issue in this area.

Acknowledgements

I am cordially grateful to Professor Norihisa Komoda of the Graduate School of Information Science and Technology at Osaka University for his visionary directions and constructive suggestions on this research activity and on writing this dissertation.

I would like to thank to Professors Toru Fujiwara, Shinji Shimojo of the Graduate School of Information Science and Technology at Osaka University for their numerous suggestions for revising this dissertation.

I also so much appreciate to Professors Shojiro Nishio, Koh Hosoda, and Associate Professor Masanori Akiyoshi (Currently, Professor of Hiroshima Institute of Technology) of Graduate School of Information Science and Technology at Osaka University for their valuable comments and supports on this research.

I would also like to acknowledge Distinguished Professor Emeritus Tadao Murata and Professor Jason Leigh of Department of Computer Science, University of Illinois at Chicago for their supervision and valuable suggestions during my stay and study in the beautiful windy city Chicago.

My sincere appreciation also goes to ex-Vice President and Executive Officer Mr. Hiroyuki Maezawa of Hitachi Solutions, Ltd. for giving this opportunity and full accommodation to study in the Graduate School of Information Science and Technology at Osaka University.

I would like to express my appreciation to Mr. Tsutomu Shomura, Deputy General Manager, Engineering and Information Technical Management Group, Dr. Takashi Onoyama, Department Manager of Research and Development Department, and Dr. Yoshiki Sameshima, Chief Researcher, for their understanding, support, and a lot of useful advices ever since my joined Hitachi Solutions, Ltd. I would also like to give my thanks to my seniors, colleagues, and juniors of the company.

I must extend my deepest gratitude to my parents, brother, and sister for their sincere encouragement and support in my endeavors. Finally I would like to thank my wife Satomi, who has been encouraging and supportive, and my daughter Tamaki, who always brings a cheerful air and happiness to our family.

References

[1] J. Gantz and D. Reinsel: “The Digital Universe Decade – Are You Ready?,” EMC (online), available from <http://www.emc.com/collateral/analyst-reports/idc-digital-universe-are-you-ready.pdf> (accessed 2012-05-02).

[2] “Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2010-2015,” Cisco Systems, Inc. (online), available from <http://newsroom.cisco.com/dlls/ekits/Cisco_VNI_Global_Mobile_Data_Traffic_Forecast_2010_2015.pdf> (accessed 2012-05-02).

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung: “Google File System,” in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp.29-43 (2003).

[4] J. Dean, and S. Ghemawat: “MapReduce: Simplified Data Processing on Large Clusters,” in *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI 2004)*, pp.137-150 (2004).

[5] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis: “Evaluating MapReduce for Multi-core and Multiprocessor Systems,” in *Proceedings of the 13st International Conference on High-Performance Computer Architecture (HPCA 2007)*, pp.13-24 (2007).

[6] C. Chu, S. K. Kim, Y. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun: “MapReduce for Machine Learning on Multicore,” in *Proceedings of Advances in Neural Information Processing Systems 19*, pp.281-288 (2006).

[7] R. Chen, H. Chen, and B. Zang: “Tiled-MapReduce: Optimizing Resource Usages of Data-parallel Applications on Multicore with Tiling,” in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT 2010)*, pp.523-534 (2010).

[8] A. Alexandrov, S. Ewen, M. Heimel, F. Hueske, O. Kao, V. Markl, E. Nijkamp,

and D. Warneke: "MapReduce and PACT – Comparing Data Parallel Programming Models," in *Proceedings of the 14th International Conference on Database Systems for Business, Technology, and Web (BTW 2011)*, pp.25-44 (2011).

[9] "Welcome to Apache^{TM} Hadoop^{TM}!," The Apache Software Foundation (online), available from <http://hadoop.apache.org/> (accessed 2012-05-02).

[10] Tom White: "*Hadoop: The Definitive Guide*," O'Reilly (2009).

[11] J. Lin: "Scalable Language Processing Algorithms for Masses: A Case Study in Computing Word Co-occurrence Matrices with MapReduce," in *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pp.419-428 (2008).

[12] N. Golpayegani, and M. Halem: "Cloud Computing for Satellite Data Processing on High End Compute Clusters," in *Proceedings of 2009 IEEE International Conference on Cloud Computing*, pp.88-92 (2009).

[13] A. Matsunaga, M. Tsugawa, and J. Fortes: "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications," in *Proceedings of the 4th IEEE International Conference on eScience*, pp.222-229 (2008).

[14] "MPICH2: High-performance and Widely Portable MPI," Argonne National Laboratory (online), available from <http://www.mcs.anl.gov/research/projects/mpich2/> (accessed 2012-05-02).

[15] D. Buntinas, G. Mercier, and W. Gropp: "Implementation and Evaluation of Shared-Memory Communication and Synchronization Operations in MPICH2 using the Nemesis Communication Subsystem," *Parallel Computing*, Vol.33, No.9, pp.634-644 (2007).

[16] W. Gropp and R. Thakur: "Issues in Developing a Thread-Safe MPI Implementation," in *Proceedings of the 13th European PVM/MPI Users' Group Meeting (Euro PVM/MPI 2006)*, pp.12-21 (2006).

[17] S. Byna, W. Gropp, X. Sun, and R. Thakur: "Improving the Performance of MPI Derived Datatypes by Optimizing Memory-Access Cost," in *Proceedings of the IEEE International Conference on Cluster Computing (Cluster 2003)*, pp. 412-419 (2003).

[18] "LAM/MPI Parallel Computing," Indiana University (online), available from <http://www.lam-mpi.org> (accessed 2012-05-02).

[19] S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman: "The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing," *International Journal of High Performance Computing Applications*, Vol.19, No.4, pp.479-493 (2005).

[20] G. Burns, R. Daoud, and J. Vaigl: "LAM: An Open Cluster Environment for MPI," in *Proceedings of Supercomputing Symposium '94*, pp.379-386 (1994).

[21] J. M. Squyres and A. Lumsdaine: "A Component Architecture for LAM/MPI," in *Proceedings of the 10th European PVM/MPI Users' Group Meeting*, pp.379-387 (2003).

[22] D. Luckham: "*The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*," Addison-Wesley Professional (2002).

[23] D. Gyllstrom, E. Wu, H. Chae, Y. Diao, P. Stahlberg, and G. Anderson: "SASE: Complex Event Processing over Streams," in *Proceedings of 3rd Biennial Conference on Innovative Data Systems Research (CIDR 2007)*, pp.407-411 (2007).

[24] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman: "Efficient Pattern Matching over Event Streams," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2008)*, pp.147-160 (2008).

[25] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma: "Query Processing, Resource Management, and Approximation in a Data Stream Management System," in *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, pp.245-256 (2003).

[26] "Gluster.org Community Website | GlusterFS is a cluster file-system capable of scaling to several peta-bytes," Gluster, Inc. (online), available from <http://www.gluster.org> (accessed 2012-05-02).

[27] R. Noronha and D. K. Panda: "IMCa: A High Performance Caching Front-end for GlusterFS on InfiniBand," in *Proceedings of the 37th International Conference on Parallel Processing*, pp.462-469 (2008).

[28] "IBM General Parallel File System," IBM (online), available from <http://www-03.ibm.com/systems/software/gpfs> (accessed 2012-05-02).

[29] F. Schmuck and R. Haskin: "GPFS: A Shared-Disk File System for Large Computing Clusters," in *Proceedings of the FAST'02 Conference on File and Storage Technologies*, pp.231-244 (2002).

[30] P. F. Corbett and D. G. Feitelson: "The Vesta Parallel File System," *ACM Transactions on Computer Systems*, Vol.14, No.3, pp.225-264 (1996).

[31] "The Apache Cassandra Project," Apache Software Foundation (online), available from <http://cassandra.apache.org> (accessed 2012-05-02).

[32] A. Lakshman, and P. Malik: "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, Vol.44, No.2, pp.35-40 (2010).

[33] "HBase Home," Apache Software Foundation (online), available from <http://hbase.apache.org> (accessed 2012-05-02).

[34] L. George: "*HBase: The Definitive Guide*," O'Reilly (2011).

[35] C. Zhang and H. D. Sterck: "CloudBATCH: A Batch Job Queuing System on Clouds with Hadoop and HBase," in *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2010)*, pp.368-375 (2010).

[36] C. Zhang and H. D. Sterck: "Supporting Multi-row Distributed Transactions with Global Snapshot Isolation Using Bare-bones HBase," in *Proceedings of the 11th ACM/IEEE International Conference on Grid Computing (Grid 2010)*, pp.177-184 (2010)

[37] "Apache CouchDB," Apache Software Foundation (online), available from <http://couchdb.apache.org> (accessed 2012-05-02).

[38] "*CouchDB: The Definitive Guide*," O'Reilly (2009).

[39] "MongoDB," 10gen, Inc. (online), available from <http://www.mongodb.org> (accessed 2012-05-02).

[40] K. Banker: "*MongoDB in Action*," Manning Publications (2011).

[41] Z. Wei-ping: "Using MongoDB to implement textbook management system instead of MySQL," in *Proceedings of IEEE 3rd International Conference on Communication Software and Networks (ICCSN)*, pp.303-305 (2011).

[42] Y. Kirihata, Y. Sameshima, T. Onoyama, and N. Komoda: "Data Loss Prevention for Confidential Web Contents and Security Evaluation with BAN Logic," *International Journal of Computers*, Vol.5, No.3, pp.414-422 (2011).

[43] Y. Kirihata and Y. Sameshima: "A Web-based System for Prevention of Information Leakage," *The 11th International World Wide Web Conference (WWW2002)*, poster-127 (2002).

[44] Y. Kirihata, Y. Sameshima, T. Onoyama, and N. Komoda: "A Novel Approach for Protection of Confidential Web Contents," in *Proceedings of the European*

Computing Conference (ECC'11), pp.110-115 (2011).

[45] Y. Kirihata and Y. Sameshima: "A Web-based System for Prevention of Information Leakage," in *Proceedings of the 61st Annual Convention IPS Japan – Distributed Systems and Security* (2000) (in Japanese).

[46] "Act on the Protection of Personal Information," Government of Japan (online), available from <http://www.caa.go.jp/seikatsu/kojin/houritsu/houritsu.pdf> (accessed 2012-05-02).

[47] A. Y. Wong and M. Seltzer: "Evaluating Windows NT terminal server performance," in *Proceedings of the Third USENIX Windows NT Symposium (USENIX, Seattle)*, pp.145-154 (1999).

[48] A. Y. Wong and M. Seltzer: "Operating System Support for Multi-User, Remote, Graphical Interaction," in *Proceedings of the USENIX 2000 Annual Technical Conference*, pp.183-196 (2000).

[49] "Citrix XenDesktop: Designing an Enterprise Solution Guild," Citrix, Inc. (online), available from <http://www.citrix.com/site/resources/dynamic/salesdocs/XD_Enterprise_Design_White_Paper.pdf> (accessed 2012-05-02).

[50] T. W. Mathers and S. P. Genoway: "*Windows NT Thin Client Solutions: Implementing Terminal Server and Citrix MetaFrame*," Macmillan Technical Publishing (1998).

[51] B. K. Schmidt, M. S. Lam, and J. D. Northcutt: "The interactive performance of SLIM: a stateless, thin-client architecture," 17th ACM Symposium on Operating Systems Principles, Vol.33, No.5, pp.32-47 (1999).

[52] R. W. Scheifler and J. Gettys: "The X Window System," *ACM Transactions on Graphics*, Vol.5, No.2, pp 79-106 (1986).

[53] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper: "Virtual Network Computing," *IEEE Internet Computing*, Vol.2, No.1, pp.33-38 (1998).

[54] "VMware vSphere Basics Guide," VMware, Inc. (online), available from < http://pubs.vmware.com/vsphere-50/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-50-basics-guide.pdf> (accessed 2012-05-02).

[55] "HP Consolidated Client Infrastructure (CCI) Network Consideration Guide," Hewlett-Packard Company (online), available from <http://bizsupport1.austin.hp.com/bc/docs/support/SupportManual/c01163844/c01163844.pdf> (accessed 2012-05-02).

[56] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H.

Balakrishnan: "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," *IEEE/ACM Transactions of Networking*, Vol.11, No.1, pp.17-32 (2003).

[57] P. Ganesan and G. S. Manku: "Optimal Routing in Chord," in *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, pp.169-178 (2004).

[58] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan: "Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service," in *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pp.81-86 (2001).

[59] B. Leong, B. Liskov, and E. D. Demaine: "EpiChord: Parallelizing the Chord lookup algorithm with reactive routing state management," *Journal of Computer Communications*, Vol.29, No.9, pp.1243-1259 (2006).

[60] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs: "A Sorting Classification of Parallel Rendering," *IEEE Computer Graphics and Applications*, Vol.14, No.4, pp.23-32 (1994).

[61] K. Moreland, B. Wylie, and C. Pavlakos: "Sort-last Parallel Rendering for Viewing Extremely Large Data Sets on Tile Displays," in *Proceedings of IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics (PVG 2001)*, pp.85-92 (2001).

[62] S. Eilemann and R. Pajarola: "Direct Send Compositing for Parallel Sort-Last Rendering," in *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization (EGPGV 2007)*, pp.29-36 (2007).

[63] C. Mueller: "Sort-First Rendering Architecture for High-Performance Graphics," in *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pp.75-84 (1995).

[64] M. Woo, J. Neider, T. Davis, and D. Shreiner: "*OpenGL Programming Guide, Third Edition*," Addison-Wesley (1999).

[65] Y. Kirihata, Y. Sameshima, T. Onoyama, and N. Komoda: "WriteShield: A Pseudo Thin-Client for Prevention of Information Leakage," *IEEJ Transactions on Electronics, Information and Systems*, Vol.132, No.2, pp.253-259 (2012).

[66] Y. Kirihata, Y. Samashima, and T. Onoyama: "WriteShield: A Pseudo Thin-Client System for Prevention of Information Leakage," in *Proceedings of the 8th IEEE International Conference on Industrial Informatics (INDIN2010)*, pp.478-483

(2010).

[67] K. Nakayama, Y. Kirihata, and Y. Samashima: “A Process-based Write Control for Software Update on the Pseudo Thin Client PC,” in *IPSJ SIG Notes on Computer Security Group*, Vol.2010, No.5, pp.1-6 (2010) (in Japanese).

[68] Y. Kirihata, T. Onoyama, and N. Komoda: “A Slot-based Virtual Node Method of Consistent Hashing for Optimization of Index Reconfiguration on Distributed Search,” *IEEJ Transactions on Electronics, Information and Systems*, Vol.132, No.10 (2012) (to be published).

[69] Y. Kirihata, J. Leigh, C. Xiong, and T. Murata: “A Sort-Last Rendering System over an Optical Backplane,” *Journal of Systemics, Cybernetics and Informatics*, Vol.3, No.3, pp.63-69 (2005).

[70] Y. Kirihata, J. Leigh, C. Xiong, and T. Murata: “A Sort-Last Rendering System over an Optical Backplane,” in *Proceedings of the 10th International Conference on Information Systems Analysis and Synthesis (ISAS 2004) and the International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA 2004)*, Vol.1, pp.42-47 (2004)

[71] B. Shneiderman: “*Designing the User Interface: Strategies for Effective Human-Computer Interaction, 4th edition*,” Addison-Wesley (2004).

[72] J. Nieh, S. J. Yang, and N. Novik: “Measuring Thin-Client Performance Using Slow-Motion Benchmarking,” *ACM Transactions on Computer Systems*, Vol.21, No.1, pp.87-115 (2003).

[73] S. J. Yang, J. Nieh, and J. Novik: "Measuring Thin-Client Performance Using Slow-Motion Bench-marking,” in *Proceedings of USENIX 2001 Annual Technical Conference*, pp.35-49 (2001).

[74] S. J. Yang, J. Nieh, M. Selsky, and N. Tiwan: “The Performance of Remote Display Mechanism for Thin-Client Computing,” in *Proceedings of the 2002 USENIX Annual Technical Conference*, pp.131-146 (2002).

[75] A. Lai and J. Nieh: “On the Performance of Wide-Area Thin-Client Computing,” *ACM Transactions on Computer Systems*, Vol.24, No.2, pp. 175-209 (2006).

[76] A. Lai and J. Nieh: “Limits of Wide-Area Thin-Client Computing,” in *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Vol.30, No.1, pp.228-239 (2002).

[77] J. Nieh and S. J. Yang: “Measuring the Multimedia Performance of Server-Based Computing,” in *Proceedings of the 10th International Workshop on Network and*

Operation System Support for Digital Audio and Video, pp. 55-64 (2000).

[78] "Windows SteadyState Handbook," Microsoft Corporation (online), available from <http://download.microsoft.com/download/d/2/6/d261b347-2f03-4bcf-8240-8b7a6 6beef8a/Windows SteadyState Handbook.pdf> (accessed 2012-05-02).

[79] "Deep Freeze Enterprise User Guide," Faronics Corporation (online), available from <http://www.faronics.com/assets/DFE_Manual.pdf> (accessed 2012-05-02).

[80] "Eugrid SecureDesktop," Eugrid K. K. (online), available from <http:// www.eugrid.co.jp/products/ securedesktop/> (accessed 2012-05-02).

[81] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy: "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," in *Proceedings of the 29th ACM Symposium on Theory of Computing*, pp.654-663 (1997).

[82] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi: "Web Caching with Consistent Hashing," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, Vol.31, No.11-16, pp.1203-1213 (1999).

[83] X. Wang, and D. Loguinov: "Load-Balancing Performance of Consistent Hashing: Asymptotic Analysis of Random Node Join," *IEEE/ACM Transactions of Networking*, Vol.15, No.4, pp.892-905 (2007).

[84] M. McCandless, E. Hatcher, and O. Gospodnetic: "*Lucene in Action Second Edition*," Manning Publications Co. (2010).

[85] B. Fitzpatrick: "Distributed caching with memcached," *Linux Journal*, Vol.2004, No.124, pp.1-5 (2004).

[86] J. Latency, A. Bahety, S. Konda, and S. Mahindrakar: "*Low-Latency, High-Throughput Access to Static Global Resources within the Hadoop Framework*," Technical Report HCIL-2009-01, University of Maryland, College Park, pp.1-15 (2009).

[87] "roma-prj – ROMA: A Distributed Key-Value Store in Ruby," Rakuten, Inc. (online), available from <http://code.google.com/p/roma-prj/> (accessed 2012-05 -02).

[88] R. Rivest: "The MD5 Message-Digest Algorithm," The Internet Engineering Task Force (online), available from <http://www.ietf.org/rfc/rfc1321.txt> (accessed 2012-05-02).

[89] J. Leigh, L. Renambot, T. A. DeFanti, M. Brown, E. He, N. Krishnaprasad, J. M. A.

Meerasa, A. Nayak, K. Park, R. Singh, S. Venkataraman, C. Zhang, D. Livingston, and M. McLaughlin: "An Experimental OptIPuter Architecture for Data-Intensive Collaborative Visualization," in *Proceedings of the 3rd Workshop on Advanced Collaborative Environments (in conjunction with the High Performance Distributed Computing Conference)*, in CD-ROM (2003).

[90] E. He, J. Alimohideen, J. Eliason, N. K. Krishnaprasad, J. Leigh, O. Yu, and T. A. DeFanti: "QUANTA: A Toolkit for High Performance Data Delivery over Photonic Networks," *Future Generation Computer Systems*, Vol.19, No.6, pp.919-933 (2003).

[91] D. Yang, J. Yu, and Y. Chung: "Effecient Compositing Methods for the Sort-Last-Sparse Parallel Volume Rendering System on Distributed Memory Multicomputers," *The Journal of Supercomputing*, Vol.18, No.2, pp.201-220 (2001).

[92] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis: "A Practical Evaluation of Popular Volume Rendering Algorithms," in *Proceedings of the 2000 IEEE Symposium on Volume Visualization (Volviz 2000)*, pp.81-90 (2000).

[93] B. Cabral, N. Cam, and J. Foran: "Accelerated Volume Rendering and Tomographic Reconstruction using texture mapping hardware," in *Proceedings of the Symposium on Volume Visualization 1994*, pp.91-98 (1994).

[94] "http://www.volvis.org," D. Bartz (online), available from <http://www.volvis. org > (accessed 2012-05-02).

[95] D.C. Dorf and R.H. Bishop: "*Modern Control Systems*," Addison-Wesley (1998).

[96] T. Sugie and M. Fujita: "*Introduction to Feedback Control*," Corona Publishing Co., Ltd. (1999) (in Japanese).

[97] "Oracle Database, Data Warehousing Guide, 10g Release 2 (10.2)," Oracle Corporation (online), available from <http://da2i.univ-lille1.fr/doc/oracle/B19306 _01 /server.102/b14223.pdf> (accessed 2012-06-21).

[98] C. T. Yu and W. Meng: "*Principles of Database Query Processing for Advanced Applications*," Morgan Kaufmann Publishers, Inc. (1998).

[99] K. Sekiguchi, J. Otani, Y. Sanbe, K. Takeda, and T. Nakano: "*Introduction to Apache Solr —Open Source Full-Text Search Engine*," Gijutsu-Hyohron Co., Ltd. (2010) (in Japanese).