Auto-construction for Distributed Storage System Reusing Used Personal Computers

Toshiya Kawato*, Shin-ichi Motomura, Masayuki Higashino, Takao Kawamura
Tottori University, Japan.

* Corresponding author. Tel.: +81-859-38-6482; email: t.kawato@tottori-u.ac.jp
Manuscript submitted May 5, 2018; accepted August 20, 2018.
doi: 10.17706/jcp.13.10.1156-1163

Abstract: Storage for storing data is indispensable. If a storage capacity becomes insufficient, we can increase its capacity by adding new disks. It is, however, difficult to purchase a new disk when a budget is limited. On the other hand, there are many unused idle resources such as used personal computers despite those use value. In order to solve those problems, used personal computers can be reused as storage. Accordingly, it is necessary to discuss a storage system that considers characteristics of used personal computers. In addition, in order to reduce works required for an installation and operation, it is necessary to automate processing necessary for construction. In this paper, we organize the characteristics of used personal computers, and consider a storage system based on the characteristics. We have then implemented a system that automatically establishes an environment as a node of the distributed storage system on each used personal computer.

Key words: Distributed storage, used personal computer, idle resource.

1. Introduction

Recently, the amount of data handled by an information system continues to increase, and storage for saving data is indispensable. In order to cope with the increasing amount of data, capacity of HDD and SSD is growing, and online storage that can be used via a network has been widespread. Moreover, organizations such as companies and universities use not only built-in storage such as personal computers used by individuals but also shared storage and storage of servers such as business systems.

Now, if the amount of data to be stored increases and a storage capacity becomes insufficient, we can increase its capacity by adding new disks. It is, however, difficult to purchase a new disk when a budget is limited. On the other hand, in modern times when information equipment floods, there are many unused idle resources despite those use value, and resources are not effectively utilized. For example, there are used personal computers, *used-PCs*, which are left unused or being discarded due to renewals or other reasons.

In order to solve those problems, used-PCs which are idle resources can be reused as storage. Since used-PCs are existing devices, there is no initial cost in reusing themselves. We can, therefore, inexpensively introduce used-PCs to reuse compared with introducing new storage. Moreover, in an environment such as a university where the total number of PCs is large and PCs are frequently replaced, there are many used-PCs. These many unused-PCs can be collected to be reused. It is widely common to disassemble and recycle used-PCs. There are few attempts to reuse used-PCs for another use application

which is different from one when they were installed. It would be effective utilization of resources more and more to reuse used-PCs for different purpose while they can still be available for generic PC usage.

It has already been attempted to construct a storage system with a cluster configuration of personal computers. Google File System [1] is a distributed file system on the premise that a large scale storage system is constructed using servers based on PCs. Google File System is designed to allow built-in PCs to fail, and focuses on automatic recovery without losing data even in failure. In addition to Google File System, there are distributed file systems such as Ceph [2] and GlusterFS [3]. It is possible to construct a storage system using PCs by using these systems.

These existing methods are, however, based on the premise that servers have been dedicated for a storage system since they are installed. They are not supposed to reuse *used-PCs*, which are used for a different usage other than storage, and research that reuses used-PCs as storage has not been conducted. In order to use used-PCs as storage, it is necessary to consider characteristics of used-PCs. In addition, in order to reduce works of introduction and operation and make them easier, it is necessary to automate processing necessary for constructing a storage system with used-PCs.

In this paper, we organize the characteristics of used-PCs and design a storage system based on its characteristics. Moreover, we focus on distributed storage, and then implement an *auto-construction system* that automatically constructs a *distributed storage environment* in used-PCs which makes the used-PC join the distributed storage as a part of the storage.

2. Characteristics of Used-PCs

Before reusing used-PCs as storage, it is necessary to discuss practical methods that have advantages and can be used for appropriate use. In this section, we organize the characteristics of used-PCs. Moreover, we discuss a storage system considering its characteristics.

Small Capacity

Although recent PCs with TB class disks are common, capacity of a used-PC that can be collected at present may be about several hundred GB. For the reason, in order to have large capacity such as TB class, it is necessary to secure numerous used-PCs. It is, however, inefficient and not realistic in terms of installation locations and power consumption in this state.

For this reason, if an existing disk of a used-PC cannot satisfy a required capacity, we replace the existing disk with a new large capacity disk in order to secure a capacity. Although costs are incurred on a disk to replace, disks for PCs are less expensive than ones for storage products and servers, and storage can be constructed at low cost as a whole. Moreover, it is possible to reduce a risk of an occurrence of a failure by replacing a disk having a high failure rate with a brand-new one.

Low Performance

Compared with storage products and servers, PCs are inferior in performance of CPU and NIC. Since their disks are also supposed to be used in general PCs, it is difficult to request high-speed communication and high-load processing. Their disks are, thus, not suitable for a system in which reading and writing occur frequently. In addition, used-PCs demonstrate even lower performance than current PCs.

For this reason, a storage system needs to be a light-weight system that can demonstrate sufficient performance even for PCs use. An appropriate utilization is supposed to be a backup system for storing data with low frequency of utilization or update.

Different Lifetime

Depending on how hard a PC is used until the PC becomes unused, performance may deteriorate compared to fresh condition, and it is difficult to predict a lifetime such as how long it can be used after

collection and so on. Even if a PC can be used inexpensively, availability should not be reduced by its failure rate.

For this reason, it is necessary to sufficiently deal with failure prepared by monitoring a state of used-PCs, e.g., estimation of a lifetime, automatic detachment on failure and automatic incorporation of a spare machine.

High Diversity

Since various manufacturers sell various models, PCs are more diverse than storage products and servers. It would then take a lot of works to configure a used-PC as storage. The same model and performance are desirable when PCs are used in a cluster configuration. It is, however, difficult when used-PCs are reused because their performance and configurations vary among different models.

For this reason, for reducing construction works, a method that can automatically construct an environment for use as storage is necessary.

Available Numerous Units

In an environment such as a university where the total number of PCs is large and replace is frequent, we can collect numerous used-PCs. Moreover, since there is no initial cost for introduction to reuse used-PCs themselves, it is possible to use numerous used-PCs as long as there is a place for installation of used-PCs. Disks are distributed and arranged in case of using numerous disks, and high availability can be realized at low cost if data can be arranged distributedly or redundantly. In addition, there are few restrictions on a place to install used-PCs compared with rack-mounted servers. It is, therefore, possible to physically distribute used-PCs and place them easily. Moreover, it is possible to flexibly install and use used-PCs regardless of locations if there are power supply and network connectivity.

For this reason, we focus on distributed storage that is a configuration in which distributed disks on a network connecting to form one large storage. Data is distributed and stored in distributed storage because disks are distributed. Now, in addition to a distributed arrangement, availability in the case of a redundant arrangement of data will depend on the number of disks. Used-PCs are, hence, suitable since numerous used-PCs are available. In addition, it is desirable that addition and deletion of used-PCs in an operating system are easy, and distributed storage can flexibly deal with them.

We listed 5 points as characteristics of used-PCs. As a storage system that takes into consideration the characteristics of used-PCs, we, therefore, assume an inexpensive and highly available distributed storage system that uses numerous used-PCs after securing capacity by replacing an existing disk of used-PCs with a large-capacity disk as needed. In order to realize high availability, we distribute potential parts of failure, and we redundantly distribute data over numerous used-PCs. In addition, it is assumed that an installation location is physically distributed in various places taking advantage of the flexibly selectable characteristic.

As the main use, we assume to handle data that can be processed even with low performance of used-PCs, for example, data with low usage or update frequency. A storage system reusing used-PCs cannot meet all requirements in various situations. It is, therefore, necessary to clearly separate roles and to use a storage system reusing used-PCs in combination with other storage systems, e. g., we should leave data that is frequently utilized or updated to storage products and servers. In the next section, as a necessary measure for constructing the assumed distributed storage system, we discuss a system for automatically constructing a distributed storage environment.

3. Design of Auto-construction System

We design a system that automatically builds a distributed storage environment on used-PCs. In using used-PCs as storage, numerous used-PCs are used from their characteristics. Moreover, many works to add used-PCs to an existing distributed storage system are necessary. Such as replacement of used-PCs when

used-PCs in failure and adding newly collected used-PCs. It is, therefore, extremely inefficient to perform manually necessary works for each used-PC.

In this system, by automating additions required for using used-PCs as distributed storage, it reduces works required for adding. In order to facilitate construction and management, this system executes each process for the used-PCs via a network and the central server manages all at once. Moreover, manual works should be reduced as few as possible except for physical operation to connect used-PCs to a network such as LAN cable connection.

In this system, there are two processes required for used-PCs. One is processing for using used-PCs as nodes on a network that constitute a distributed storage system. Used-PCs can be used as nodes even right after they are collected. There are, however, problems that OS is different and it is necessary to delete data before collection. We, therefore, delete an environment before collection by overwriting it with open source OS and use used-PCs as nodes after unifying them in an easy-to-use environment for distributed storage.

The other is installing distributed storage software that constructing distributed storage in used-PCs. Open source software is also used in *distributed storage software* as well as OS. By using open source software that can be used for free, used-PCs can be reused as distributed storage with low cost.

4. Implementation of Auto-construction System

4.1. Automatic Installation of OS

We used Kickstart [4] and PXE boot [5] to install OS on used-PCs. Kickstart is an automatic installation tool for Red Hat related OS. Moreover, PXE Boot is a network boot mechanism using Preboot eXecution Environment (PXE) and PXE boot can install and start OS via a network.

Now, it is necessary to assign IP addresses to used-PCs in order to execute PXE boot. For administrative purposes, it is desirable to be able to identify each used-PC. We, therefore, created a script to automatically assign fixed IP addresses in this paper. A sent DHCP DISCOVER message is recorded in a log of a DHCP server when PXE boot performs. The script then always monitors the log and extracts a MAC address of a used-PC from a log of a DHCP DISCOVER message. If this MAC address is not found in the DHCP configuration file, the script adds new configuration and reloads the configuration.

We constructed a *PXE server* that CentOS with Kickstart and PXE boot can be installed. The PXE server is composed of DHCP, TFTP, and HTTP servers. Fig. 1 shows a flow of an automatic installation of OS. The DHCP server assigns a fixed IP address to a used-PC by the script when a used-PC sends a PXE boot request. Moreover, OS is automatically installed by downloading a boot image from the TFTP server, and an OS image and Kickstart related files from the HTTP server. It is, therefore, possible to automatically complete an installation of OS and network setting only by connecting used-PCs to the network and executing PXE boot.

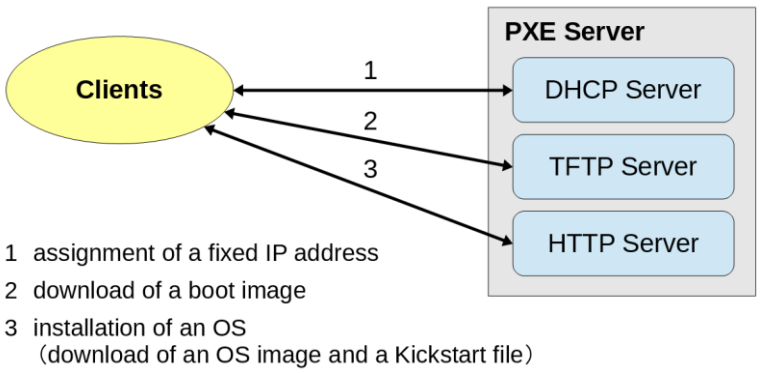


Fig. 1. Installation of OS.

4.2. Automatic Installation of Distributed Storage Software

We focused on object storage [6], [7] as a type of distributed storage. Object storage manages data in units of an object. Object storage is widely used in online storage such as Amazon S3 [8], and they are used in educational and research information systems at a university [9].

Object storage uses an HTTP protocol conforming to Representational State Transfer (REST) [10] to access objects. Object storage can, therefore, be used like a web application and hide underlying actual file systems such as the Extended File System of Linux. Software that tries to use object storage may, however, not support an HTTP access. We can solve this issue by introducing a gateway such as s3ql [11] that can access object storage and enable to mount object storage as if it were an ordinary physical disk.

Moreover, an object is stored in a flat space that is not a hierarchical structure after provided with a unique identifier indicating a storage location and metadata that can records various information. Since a unique identifier does not depend on a location of a disk where it is actually stored, there is a little restriction on arrangement of data. Object storage, therefore, is easy to move and distribute data, and easy to realize scaling out by storing copies to remote locations and adding disks.

OpenStack *Swift* [12] was used as software for constructing object storage. Swift is a component that realizes an object storage among OpenStack which is open source software for cloud infrastructure. Fig. 2 shows a basic configuration of Swift. In Swift, objects are managed by containers, and containers are managed by accounts. An *Object Server* stores objects, a *Container Server* manages containers, and an *Account Server* manages accounts. All these servers are called a *Storage Node*. A Storage Node is grouped by a unit of a zone, and multiple zones can be created. The same objects, containers, and accounts are stored in each zone. Even if failure occurs in one zone, redundancy and improvement in fault tolerance can be, therefore, realized by the presence of other accessible zones. Moreover, communication between clients and Storage Nodes is authenticated by an *Auth Server*, and is relayed by a *Proxy Server* called a *Proxy Node*.

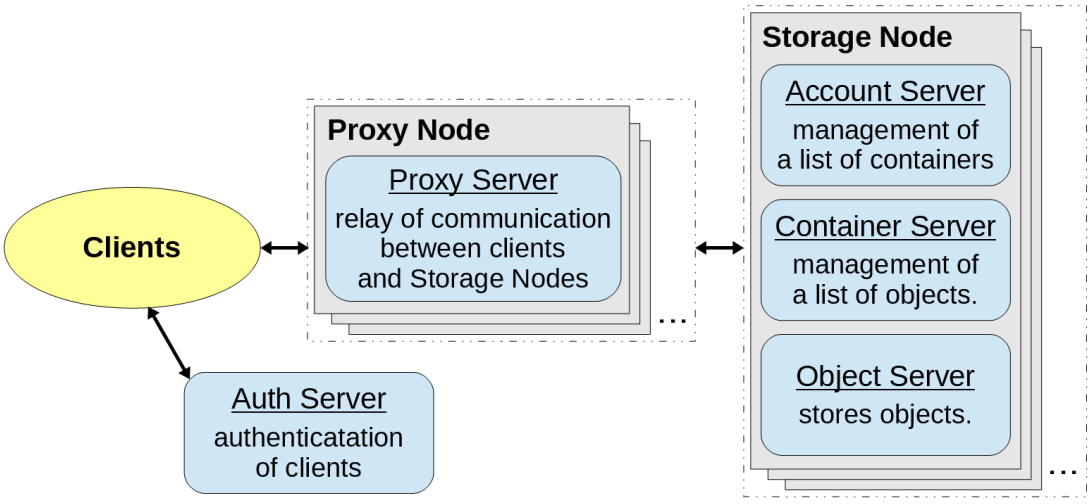


Fig. 2. Basic configuration of swift.

We used Ansible [13] for an automatic installation of Swift. Ansible can construct an environment for software to operate by writing codes like programming. Moreover, Ansible has idempotency that is the property that the same code always produces the same result. There are Chef [14] and Puppet [15] as similar software. Ansible, however, has the feature that it is unnecessary to install an agent on construction targets. If Python can operate on construction targets that can be connected by SSH, Ansible can be used. Ansible can be, therefore, easily introduced by simple configuration. Basic usage is to create and execute a file called *Playbook* that describes a configuration of an environment that you want to construct. Playbook

is described in a general YAML [16] format. Playbook can, therefore, be simply described and never requires to master an especial notation.

First, we constructed an *Ansible server* on the PXE server. The Ansible server is a server on which Ansible itself is installed. Next, we created Playbooks and related files that describe configurations necessary for installing Swift. Object storage environment by Swift, therefore, constructs automatically on used-PCs. Furthermore, since there is no need to create a proxy node and an auth server for each used-PC, and they were created manually and only storage nodes constructed in this paper. Moreover, since Python that is necessary for executing Ansible was included as standard, it was unnecessary to perform additional processing for used-PCs. For a SSH connection, we used an arbitrary script that can be executed after installing OS by Kickstart. By the script, we automatically established a SSH connection using public key authentication.

5. Experiment of Auto-construction System

We actually constructed a distributed storage environment for used-PCs by the auto-construction system. Fig. 3 shows the used-PC here. This was dc7900 by Hewlett Packard which became unnecessary by replacement at Tottori University. Its capacity of HDD was 160 GB and its speed of NIC was 1 Gbps. Moreover, its dimension was as follows: height 66 mm, wide 251 mm, long 254 mm. We used 15 used-PCs for this experiment, and we distributedly installed them on two remote campuses in Tottori University. Fig. 4 shows installation status on one campus. On this campus, we used an aluminum rack to effectively utilize space.

Fig. 3. The used-PC.

Fig. 4. Installation status on one campus.

Each used-PC was aggregated and connected to one switching hub for each campus. Moreover, their switching hubs were connected to the experimental network. Speed of its ports was all 1 Gbps. After installation of used-PCs, we executed PXE boot. Moreover, we confirmed that an installation of OS by PXE server and an installation of Swift by Ansible server were automatically completed. Total capacity was about 2,400 GB and by using 15 used-PCs. Moreover, capacity that can actually be used as distributed storage excluding an area for OS was about 2,100 GB. Actual available capacity for data storage then decreases by increasing the number of zones of Swift.

Through this experiment, we needed manual operation such as wiring and execution of PXE boot. We, however, confirmed that an auto-construction system could automate the most of a processing and utilized used-PCs as distributed storage. We measured the required time to construct a distributed storage environment for one used-PC as a reference. The result was about 5 minutes and 40 seconds from turning on power until installation of OS was completed. Moreover, the result was about 2 minutes from installation of OS was completed until installation of Swift was completed. Since time required for manual operation with keyboard input was 20 seconds or less, operation time required for construction was much shorter than in the case where all operation of construction was manual.

6. Conclusion

In this paper, we have organized characteristics of used-PCs and considered a storage system based on their characteristics. Results of considering, we have assumed an inexpensive and highly available distributed storage system using numerous used-PCs after replacing existing disks of used-PCs with large capacity as needed to secure capacity. Moreover, in order to reduce and facilitate labors required for construction and operation, we have implemented an auto-construction system of a distributed storage environment for used-PCs and experimented. Through this experiment, we have confirmed that the most of the processing of construction were successfully automated and we could reuse used-PCs as distributed storage.

In future work, an implementation automatically handles failure such as disconnecting used-PCs that experience failure. We, therefore, automate a series of tasks from an installation to failure handling which are necessary for operation and reduce further labors. Moreover, there is practical operation of a distributed storage system reusing used-PCs. We will, therefore, evaluate performance such as speed and a failure rate, and investigate concretely what kind of data is suitable for reading and writing. In addition, we will calculate the advantage in cost effectiveness for storage products and servers by verifying a running cost such as power consumption and maintenance costs. By implementing these, we will show effectiveness of reusing used-PCs as distributed storage.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 16K00477.

References

[1] Sanjay, G., Howard, G., & Shun-Tak L. (2003). The google file system. *Proceedings of the 19th ACM Symposium on Operating Systems Principles.*

[2] Ceph Homepage – Ceph. Retrieved January 5, 2018, from http://ceph.com/

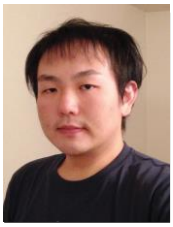
[3] Gluster. Retrieved January 5, 2018, from https://www.gluster.org/

[4] Chapter 26. Kickstart Installations. Retrieved January 30, 2018, from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/installation_guide/chap-kickstart-installations

[5] Preboot Execution Environment (PXE) Specification. Retrieved January 30, 2018, from http://www.pix.net/software/pxeboot/archive/pxespec.pdf
[6] Michael, F., Kalman, M., Dalit, N., Ohad R., & Julian S. (2005). Object storage: The future building block for storage systems. *Local to Global Data Interoperability - Challenges and Technologies* (pp. 119-123).
[7] Mike, M., Greg, G., & Erik, R. (2005). Object-based storage. *IEEE Communications Magazine, 41,* pp. 84-90.
[8] Amazon Simple Storage Service (S3) — Cloud Storage — AWS. Retrieved January 30, 2018, from https://aws.amazon.com/s3/
[9] Shin-ichi, M., Toshiya, K., & Masaya, K. (2015). Usecase of object storage for education and research computer systems. *Journal for Academic Computing and Networking, 19,* pp. 26-34.
[10] Roy Thomas, F. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Unpublished Ph.D thesis, University of California, Irvine.
[11] nikratio / S3QL – Bitbucket. Retrieved January 5, 2018, from https://bitbucket.org/nikratio/s3ql/
[12] Welcome to Swift's documentation!. Retrieved January 30, 2018, from https://docs.openstack.org/swift/latest/
[13] Ansible is Simple IT Automation. Retrieved January 30, 2018, from https://www.ansible.com/
[14] Chef - Automate IT Infrastructure | Chef. Retrieved January 30, 2018, from https://www.chef.io/chef/
[15] Puppet: Get on the shortest path to better software. Retrieved January 30, 2018, from https://puppet.com/
[16] The Official YAML Web Site. Retrieved January 5, 2018, from http://yaml.org/

Toshiya Kawato was born in 1989. He has been a graduate student of Tottori University for his doctoral degree. He is graduated from National Institute of Technology, Yonago College and received his B.Eng. degree from National Institution for Academic Degrees and University Evaluation in 2012. He has been a technical staff at Tottori University since 2012. His interest includes distributed storages. He is a student member of IPSJ.

Shin-ichi Motomura was born in 1973. He has been in Tottori University as associate professor in Center for Information Infrastructure & Multimedia since 2008. He obtained his B.Eng. and M.Eng. degrees in computer engineering from Toyohashi University of Technology, Japan, in 1995, 1997, respectively. His research interests include distributed systems. He is a member of IPSJ.



Masayuki Higashino was born in 1983. He received his B.Eng., M.Eng., and D.Eng. degree from Tottori University, Japan, in 2008, 2010, and 2013, respectively. He has been an assistant professor at Tottori University since 2015. His research interest includes mobile agent systems, distributed systems, and network architecture. He is a member of the IPSJ, JSSST, IEEE CS/ComSoc and ACM.

Takao Kawamura was born in 1965. He obtained his B.Eng., M.Eng. and Ph.D. degrees in Computer Engineering from Kobe University, Japan in 1988, 1990 and 2002, respectively. Since 1994 he had been in Tottori University as a research associate and has been in the same University as a professor in the Faculty of Engineering since 2009. His current research interests include mobile-agent systems and distributed systems.