OFF2F: A New Object File Format for Virtual Memory Systems to Support Volatile/non-Volatile Memory-Mixed Environment

Masaya Sato and Hideo Taniguchi

***Abstract*—Research on non-volatile memory has advanced in recent years. Non-volatile memory is not installed on practical computers; however, it will appear in the near future. This study proposes a new executable file format to enable a virtual memory system to support program execution on a volatile and non-volatile memory-mixed computer. To be more precise, this study proposes OFF2F: a new object file format consisting of 2 files, which leverages the characteristics of volatile and non-volatile memories. OFF2F focuses on access form when a program is loaded into memory at program execution. This paper presents evaluation results in terms of reduction of page fault processing time.**

***Index Terms*—Volatile memory, executable file format, virtual memory system, demand paging, page fault.**

I. Introduction

By assuming the change of data manipulation form or storage mechanisms for non-volatile memories, advanced research treats memories as non-volatile [1], [2]. Yamauchi *et al.* proposed an operating system structure for non-volatile main memory [1]. Koshiba *et al.* proposed a light-weight emulator considering asymmetric read/write latency for analyzing behavior of applications running on non-volatile memory environment [2]. As many researchers proposing new techniques for non-volatile memories, researches on software treating memories as non-volatile has accelerated.

In recent years, thanks to advances in hardware technology, types of non-volatile memory have been released. Software technologies that leverage non-volatile memory have also begun to be developed. One study proposed a method for leveraging non-volatile memory by using it for the storage of metadata on a file system [3][4]. Another study proposed a method that uses non-volatile memory as a write cache [5]. Automated tiered storage with fast memory and slow flash storage (ATSMF) is a tiered storage system for leveraging non-volatile memory to reduce its response time [6]. Xue *et al*. proposed a log-structured file system achieves high performance and strong consistency guarantees by exploiting characteristics of volatile/non-volatile mixed system [7]. Poremba *et al*. analyzed the effect of non-volatile and volatile memory instrument rate on performance and power consumption in high performance computing [8]. Yoon *et al.* conducted research where they treated non-volatile memory as one part of a memory hierarchy [9]. Guo *et al*. proposed a method of treating non-volatile memory as a language with various properties [10].

In this paper, we propose a new file format for executable programs that enables virtual memory systems to support program execution. This new file format leverages the characteristics of a memory mixed computer that employs both volatile and non-volatile memory. More precisely, we propose OFF2F, which is a new object file format consisting of 2 files; it focuses on access form when a program is loaded into memory when a program is executed. In this paper, we also present the results of our study on the reduction of the processing time of page faults.

The rest of the paper is organized as follows. Section II presents features of volatile/non-volatile memories and their mixed environments. Section III introduces existing file formats and Section IV presents a new executable file format: OFF2F. Section V describes a virtual memory system using OFF2F. Section VI shows evaluations with OFF2F. Finally, Section VII presents the conclusion and future challenges.

II. Volatile/non-Volatile Memory Mixed Environment

In conventional computers, all memory is volatile. However, recently, non-volatile memory, which can continuously hold data when a computer is switched off, has emerged. The performance and functions of non-volatile memory, including access speed, capacity, power consumption, durability, and pricing, are rapidly improving. Non-volatile memory is used taking advantage of its features (e.g. building a file system into non-volatile memory for personal digital assistants).

Volatile memory has a high-access speed. Its capacity is large and it is low-priced due to the current advances in technology. Compared to volatile memory, the access speed of non-volatile memory is low (especially for writes); this is because of the structure of hardware and its energy efficiency. In addition, its capacity is low and its price is high. For this reason, it is expected that in the future, not all memory will be replaced with non-volatile memory, but it will be partially replaced. As stated in [11], non-volatile memories are unable to meet various requirements for future systems. Therefore,

Manuscript received March 3, 2019; revised June 15, 2019. This work was partially supported by JSPS KAKENHI Grant Number JP18K11244 and FUJITSU LABORATORIES LTD.

The authors are with Okayama University, Okayama, 7008530 Japan (e-mail: sato@cs.okayama-u.ac.jp, tani@cs.okayama-u.ac.jp).

doi: 10.18178/ijmlc.2019.9.4.815

future mainstream technology is expected to leverage a volatile/non-volatile memory-mixed processor environment.

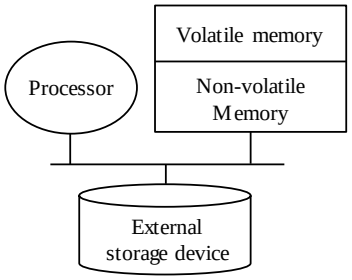


Fig. 1. An example of system construction.

The main difference between non-volatile external devices and non-volatile memory is the access unit. External devices are block-accessible, whereas non-volatile memories are byte-accessible. In addition, magnetic disk devices (DKs) and solid state drives (SSDs) have the properties of large capacity, low price, and high durability.

Fig. 1 shows the system configuration of future computers. Non-volatile and volatile memory each construct memories in a similar way. Address continuity between volatile and non-volatile memories is not required. External devices are connected by a bus and treated as input/output (I/O) devices. We consider the building of a file system in non-volatile memory as an example of how a system utilizes the construction shown in Fig. 1. Collaboration between non-volatile memory file systems (NVM-FS) and conventional external device file systems (nFS) is efficient.

III. Executable File Format

A. Format

The form of an executable program is regulated through the combination of a compiler and an operating system. Whether the compiler outputs based on the demand from operating system or the operating system supports outputs from the compiler, there is a deep relationship between the two. In any case, executable programs consist of the following content:

1) Program text region: enumeration of instructions

2) Program data region: area for data with initial values

3) Related information region: information about external variables

4) Header region: this field holds the position of the above information

To store these information as file format, 4) is required. Fig. 2 shows the format of an executable file.

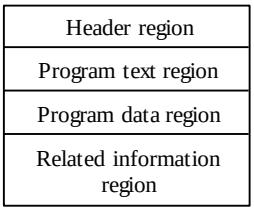


Fig. 2. An example of file format of executable program.

Examples of conventional executable file formats are the a.out format in UNIX, the Common Object File Format known as COFF, and Executable and Linkable Format (ELF). Both of these formats store the above-listed information in one file.

B. Problems with Program Execution

Because a conventional executable program is stored in one file, program execution using the on-demand paging of a virtual memory system incurs the following problems:

1) Page loading (page in) takes a long time if an executable program is stored in external storage devices (e.g. disk). Page out also takes a long time. An SSD can be used as an external device but the time for page out and page in is longer compared to memory copying.

2) The time that page in takes will be extremely low if an executable program is stored in a virtual memory system on non-volatile memory. This is because page in is made by a memory copy from non-volatile memory to volatile memory. However, this method requires a large amount of non-volatile memory to store all executable programs, and a large amount of non-volatile memory is too expensive.

IV. New Executable File Format: OFF2F

A. Basic Concept

Reading from and writing to volatile memory is fast. Reading from non-volatile memory is also fast and byte unit accessible, but writing to non-volatile memory is slow. Therefore, demand paging time can be reduced if read-only data are stored in non-volatile memory and mapped directly by a virtual memory system.

As mentioned earlier, an executable file consists of four regions: program text region, program data region, related information region, and header region. From the perspective of access patterns, these regions can be divided into two categories: read-only regions, which are program text region, related information region, and header region, and read and write regions, which is just the program data region.

Our aim was to create a new file format consisting of multiple files. Because an executable program consists of four regions, it can be divided into four files. However, we decided that the number of files should be kept minimal to prevent complications and to suppress the volume of the file system. Therefore, we propose OFF2F—a new object file format consisting of 2 files (OFF2F).

B. Format and Comparison

OFF2F can be constructed in various ways, but from the perspective of access patterns the construction can be divided into two cases. Fig. 3 demonstrates these two cases.

In Type A, header region, program data region, and related information region are stored in one file, and program text region is stored in the other file. Thus, the parts that are frequently read at program execution stored in a separate file. In Type B, header region, program text region, and related information region are stored in one file, and program data region is stored in the other file. Thus, the parts that are frequently read and written at program execution are stored in a separate file. Type A places the file XYZ on non-volatile memory and Type B places the file ABC on non-volatile memory. As a result of the following reasons, Type A is more suitable.

1) Conventional procedure is reusable because the name of the executable program is the name of the file ABC, which holds the header region, and it is stored in an external storage device. This mechanism is explained in detail in the next section.

2) The existence of non-volatile memory is not effective in Type B because the file XYZ is not always stored in non-volatile memory.

Compared to conventional executable file formats including a.out, COFF, and ELF, OFF2F can utilize non-volatile memory efficiently thanks to the isolation of program text region from other regions. File loading of conventional executable file formats can be fast by placing executables onto non-volatile memory. However, non-volatile memory would be more expensive than volatile memory. Thus, moving existing executables onto non-volatile memory wastes the expensive part. Contrary to this, only text region of OFF2F is stored onto non-volatile memory. Thereby, OFF2F can save non-volatile memory and speeding up program loading.

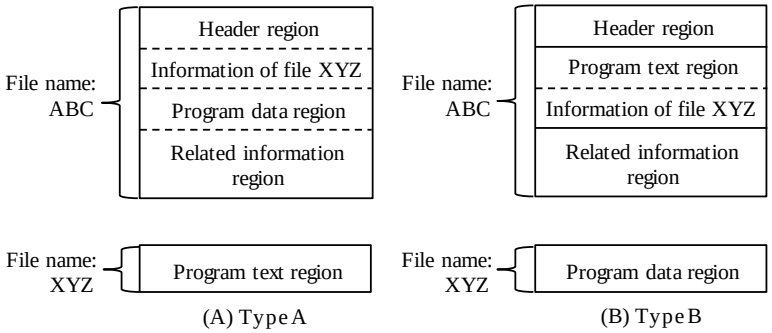


Fig. 1. OFF2F format.

V. Virtual Memory System Using OFF2F

A. Usage

This section describes the usage of OFF2F Type A on a virtual memory system. The file ABC is stored on the external storage device and the file XYZ is stored on the non-volatile memory. The process for the allocation of program text and program data to virtual memory space using the file ABC can be seen as a flow diagram in Fig. 4 and is described in full below.

1) Load the header region of the file ABC from the external storage device.

2) Recognize the program text region is stored in the non-volatile memory based on the information from the header region.

3) Register each page of the program text region on the non-volatile memory to the mapping table.

4) Recognize the program data region is stored in the external storage device based on the information from the header region.

5) Allocate memory and load the program data region from the external storage device.

6) Register each page of memory to the mapping table.

Consequently, I/O for the program text region is reduced by Step 3 compared to the case where the program text region is stored in the external storage device. Moreover, memory copy is reduced compared to the case where the program text region is stored in the non-volatile memory. Furthermore, OFF2F does not require a large amount of non-volatile memory.

When an operating system has a demand paging system, steps 3 and 6 manipulate a page fault flag, and in this case, Step 5 is not required.

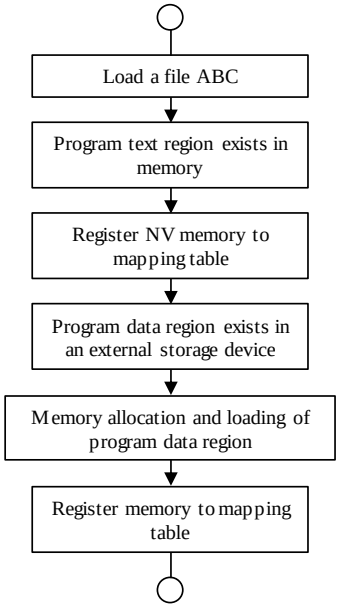


Fig. 4. A flow to create virtual memory space.

B. Processing of Page Faults

Fig. 5 shows the processing of a page fault with a demand paging system.

As shown in Fig. 5, if a page fault occurred at the text area on non-volatile memory, the paging system simply registers the page to a mapping table. Thus, I/O for page in and page out is not required. In addition, memory allocation and inter-memory copying can also be reduced.

C. Formulation of Time for Page Faults

Based on the flow for a page fault shown in Fig. 5, this section formulates the time for page faults.

(Conventional) All programs are stored in the external storage device.

(Proposal) Program text should be stored in non-volatile memory and program data should be stored in an external

device. Page faults are handled by OFF2F (a memory page in the non-volatile memory is registered to the mapping table).

Time for each procedure is denoted as follows:

t_1: allocation of memory

t_2: load of one page (4 KB) from the external storage device

t_3: registration of the memory to the mapping table

t_4: registration of the page of non-volatile memory to the mapping table

In addition, sizes and rates are assumed as follows:

P: the size of the program (text + data)

S: the rate of text for the program

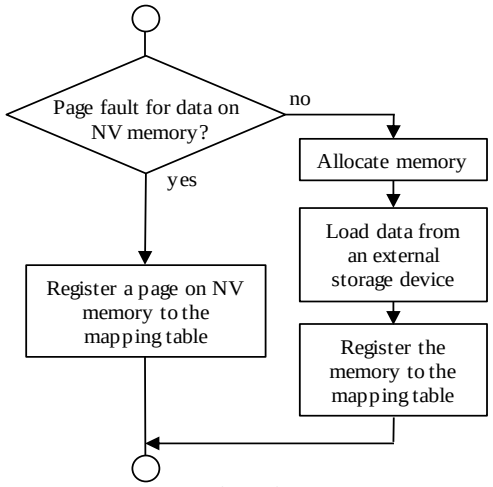


Fig 5. A flow of page in.

In each case, sum of the time for page faults of the whole program, which is the total time of page faults for all pages of text and data regions, is given by Equation (1) for the conventional case and by Equation (2) for our proposed case.

$$\text{(Conventional)}\ (t_1 + t_2 + t_3)\ P \quad (1)$$

$$\text{(Proposal)}\ (t_1 + t_2 + t_3)\ P\ (1 - S) + t_4\ PS \quad (2)$$

VI. Evaluation

A. Point of View

It can be expected that the processing time for page faults in an on-demand paging (ODP) function with OFF2F will be reduced. This section demonstrates the efficiency of OFF2F by predicting basic performance on the FreeBSD operating system focusing on the size of text and data area.

B. Basic Performance

We measured the time taken for a 4 KB random read of 1 GB of data in the following environment:

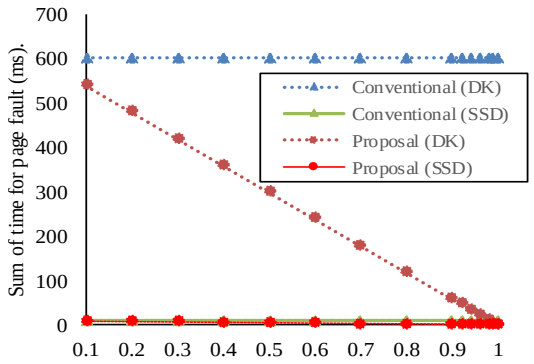
- Operating system: FreeBSD 6.3-R
- Processor: Intel Core i7-2600 (3.4 GHz)
- DK: Seagate ST400DM002 (7,200 rpm)
- SSD: Intel SSD 540s Series

The results showed that the mean time for read is approximately 6.22 milliseconds for the DK and 93.70 microseconds for the SSD. Accordingly, we assume the time of read (t2) from DK to be 6 milliseconds and from SSD to be 0.1 millisecond. Fig. 6 shows the sum of the time for page faults based on Equations (1) and (2). The horizontal axis shows the rate of text for the program. The size of the program (P) was set to 100 pages. The time of procedures other than t2 was set to 0.001 milliseconds.

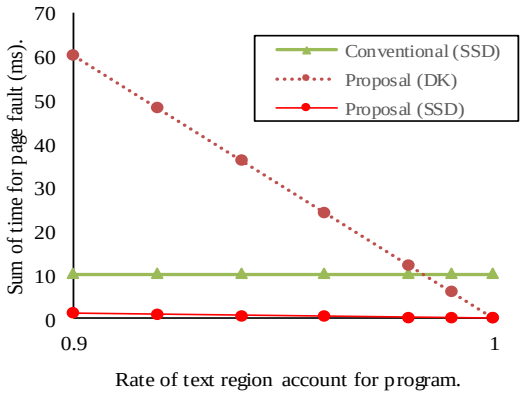
1) Fig. 6(A) shows that the time for page faults is constant for the conventional method (Conventional). This is because the procedure for page faults for the text and data regions is the same despite of the rate of the text region for the whole program. Incidentally, the gap of time between DK and SSD is due to the read performance. The time for read from DK is sixteen times longer than that from SSD. For this reason, even in the conventional method, the time for page faults can be greatly reduced by using an SSD instead of a DK.

2) Fig. 6(A) shows that the time for page faults is reduced for the proposed method (Proposal). This is because the number of read from the external device would be reduced as the rate of text for the whole program increases.

3) Fig. 6(B) shows that the time for page faults with the proposed method on the DK (Proposal (DK)) becomes shorter than that the conventional method on the SSD (Conventional (SSD)). Consequently, if the rate of text region for the whole program is high (over than 98.3%), Proposal can reduce the time for page faults more by replacing DK by SSD.



(A) Rate of text region (0.1 – 1.0).



(B) Rate of text region (0.9 – 1.0).

Fig. 6. Sum of time for page faults for all programs.
(t_2(DK): 6 ms, t_2(SSD): 0.1 ms, others are set to 0.001 ms)

C. Prediction of Effect on FreeBSD

1) Analysis of executable program

We analyzed executable programs in /bin and /sbin of FreeBSD 11.0-RELEASE. We found that files in /bin and

/sbin have similar properties; thus, the following describes files in /bin. Fig. 7 shows files sizes of executable programs in /bin. As shown in Fig. 7, the size of program text is approximately twenty times larger than that of program data for many files. For all programs, forty-four files are stored in /bin. In addition, the total size of program text is 1,518,784 bytes and the total size of program data is 80,136 bytes. Therefore, the size of program text is twenty times larger than that of program data.

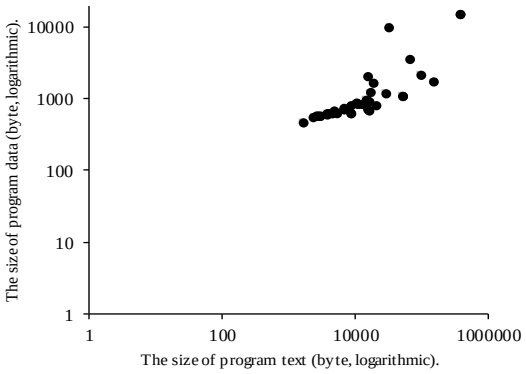


Fig. 7. The size of each program in /bin of FreeBSD 11.0-RELEASE.

2) Comparison

Here, we compare the time for page faults between the conventional method and the proposed method based on the total time for page faults of all programs under /bin assuming they all are started just once.

As shown in Fig. 8, we assume the access (read/write) form for the execution of each program. To be more precise, we define the following:

x: rate of program text certainly accessed

y: rate of program data certainly accessed

In addition, we assume the following for each program:

The parts of $(1 - x)$ of the program text are accessed at random rate α.

The parts of $(1 - y)$ of the program data are accessed at random rate β.

Consequently, when *program i* is executed and run as a process, the number of page faults is the sum of the following formulae:

The number of page faults of program text is

$$(T_i * x + T_i (1 - x) * \alpha) / (\text{page size}) + 1 \quad (3)$$

The number of page faults of program data is

$$(D_i * y + D_i (1 - y) * \beta) / (\text{page size}) + 1 \quad (4)$$

where T_i and D_i are the size of program text and program data of *program i, respectively*.

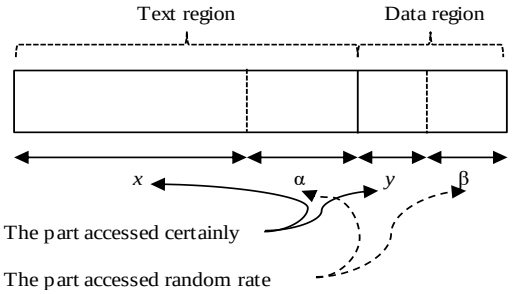
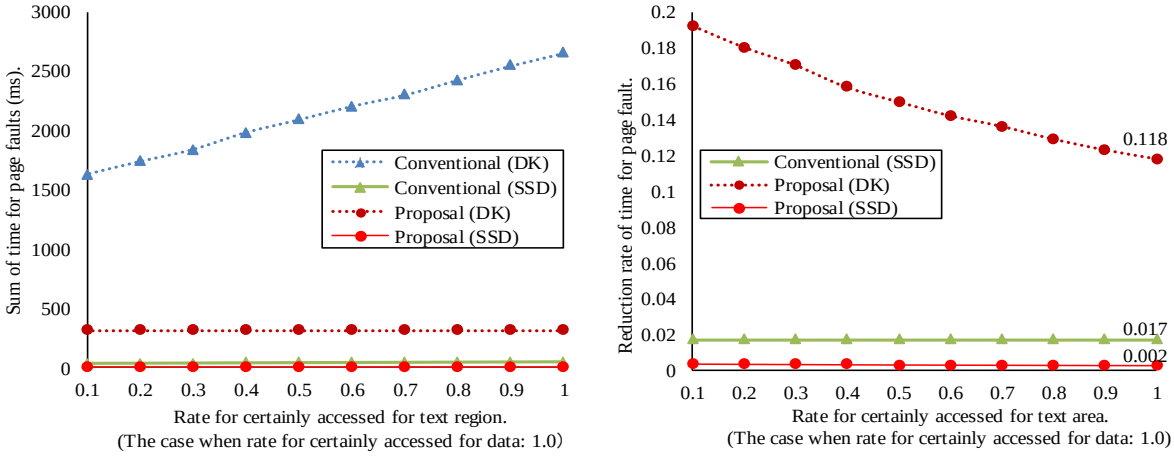


Fig. 8. Access form at program execution.



(A) Sum of time for page faults.

(B) Reduction of time for page faults.
(The case when conventional(DK) is set to 1.0)

Fig. 9. Sum of time for page faults of programs under /bin in FreeBSD 11.0-RELEASE.

The sum of time for page faults of the programs under /bin in the conventional method and the proposed method are calculated by applying Equations (3) and (4) to each program and using Equations (1) and (2). Fig. 9 shows the sum for programs under /bin. Fig. 9(A) shows the sum of the time for page faults and Fig. 9(B) shows the relative time of the proposed method compared to that of the conventional method. Relative time refers to the reduction rate of the conventional method. Here, $t1$, $t2$, $t3$, and $t4$ were set to the same values as mentioned earlier ($t2$(DK) was 6 ms, $t2$(SSD) was 0.1 ms, and all others were 0.001 ms). The rate accessed certainly in program data (y) was set to 1.0. Even when the rate was set to 0.5, the sum of the time for page faults is almost the same; this is due to two reasons: the size of program data is much smaller than that of program text and the size based on page granularity is one mostly because the size of program data is small. More precisely, the sum of the number of accessed pages for program data is 52 pages when y is between 0.6 and 1.0 and 51 pages when y is between 0.1 and 0.5.

Fig. 9 demonstrates the following results.

1) Fig. 9(A) shows that the sum of page time for page faults increases in most cases. This is because an increase in the access rate for program text is equivalent to an increase in

the number of page faults. Although the increase for the conventional method with a DK is remarkable, the increase in all other cases is small. Consequently, as confirmed in Fig. 9(B), as the access rate to the program text increases, reduction of time for page faults increases—especially on the proposed method with a DK.

2) Fig. 9(B) shows that the performance of the procedure for page faults in comparison to the conventional method with a DK can be improved 60 fold for the conventional method with an SSD, 8 fold for the proposed method with a DK, and 500 fold for the proposed method with an SSD.

VII. Conclusion

This paper proposed a new file format called OFF2F (Object File Format consisting of 2 Files) to accelerate program execution based on a virtual memory system on a volatile and non-volatile memory mixed computer. This paper also showed the processing flow of a page fault with OFF2F.

We formulated the processing time of page faults by focusing on the rate of the sizes of the text and data areas. Evaluations by prediction showed that the time for page fault with OFF2F is eight times faster than the time of the existing method. We also described that OFF2F is effective not only on a DK but also on an SSD.

In our future work, we plan to implement and evaluate OFF2F.

References

[1] T. Yamauchi, Y. Yamamoto, K. Nagai, T. Matono, S. Inamoto, M. Ichikawa, M. Goto, and H Taniguchi, "Plate: Persistent memory management for nonvolatile main memory," in *Proc. 31st ACM Symposium on Applied Computing*, Apr. 2016, pp. 1885–1892.

[2] A. Koshiba, T. Hirofuchi, S. Akiyama, R. Takano, and M. Namiki, "Towards write-back aware software emulator for non-volatile memory," in *Proc. 2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, Aug. 2017, pp. 1–6.

[3] Q. Wei, J. Chen, and C. Chen, "Accelerating file system metadata access with byte-addressable nonvolatile memory," *ACM Transactions on Storage*, vol. 11, issue 3, pp. 1–28, Jul. 2015.

[4] Q. Wei, C. Wang, C. Chen, Y. Yang, J. Yang, and M. Xue, "Transactional NVM cache with hight performance and crash recovery," in *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[5] E. Lee, J. Kim, H. Bahn, S. Lee, and S. H. Noh, "Reducing write amplification of flash storage through cooperative data management with NVM," *ACM Transactions on Storage (TOS) – Special Issue on MSST 2016 and Regular Papers*, vol. 13, issue 2, Jun. 2017.

[6] K. Oe, M. Sato, and T. Nanri, "Automated tiered storage system consisting of memory and flash storage to improve response time with input-output (IO) concentration workloads," in *Proc. 2017 Fifth International Symposium on Computing and Networking (CANDAR)*, Nov. 2017, pp. 311–317.

[7] J. Xue and S. Swanson, "NOVA: A log-structured file system for hybrid volatile/non-volatile main memories," in *Proc. the 14th USENIX Conference on File and Storage Technologies (FAST '16)*, Feb. 2016, pp. 323–338.

[8] M. Poremba, I. Akgun, J. Yin, O. Kayiran, Y. Xie, and G. H. Loh, "There and back again: Optimizing the interconnect in networks of memory cubes," in *Proc. the 44th Annual International Symposium on Computer Architecture*, Jun. 2017, pp. 678–690.

[9] D. H. Yoon, T. Gonzalez, P. Ranganathan, and R. S. Schreiber, "Exploring latency-power tradeoffs in deep nonvolatile memory hierarchies," in *Proc. the 9th conference on Computing Frontiers*, May 2012, pp. 95–102.

[10] X. Guo, A. Shrivastave, M. Spear, and G. Tan, "Languages must expose memory heterogeneity," in *Proc. the Second International Symposium on Memory Systems*, Oct. 2016, pp. 251–256.

[11] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1537–1550, May 2016.

Masaya Sato received his B.E., M.E. and Ph.D. degrees from Okayama University, Japan in 2010, 2012 and 2014, respectively. In 2013 and 2014 he was a research fellow of the Japan Society for the Promotion of Science. He has been an assistant professor of Graduate School of Natural Science and Technology at Okayama University. His research interests include computer security and virtualization technology.

Hideo Taniguchi received a B.E. degree in 1978, a M.E. degree in 1980 and a Ph.D. degree in 1991, all from Kyushu University, Fukuoka, Japan. In 1980, he joined NTT Electrical Communication Laboratories. In 1988, he moved to Research and Development Headquarters, NTT DATA Communications Systems Corporation. He has been an associate professor of computer science at Kyushu University since 1993, a professor of the Faculty of Engineering at Okayama University since 2003. He has been a dean of Faculty of Engineering from April 2010 to March 2014 and a vice president from April 2014 to March 2017 at Okayama University. His research interests include operating system, real-time processing and distributed processing..