International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 10, Number 2, pages 127–143, July 2020

Non-volatile memory driver to drastically reduce input-output response time and maintain Linux device-mapper framework

Kazuichi Oe
FUJITSU LABORATORIES LTD.
KAWASAKI, JAPAN
ooe.kazuichi@fujitsu.com

Takeshi Nanri
Kyushu University
FUKUOKA, JAPAN
nanri.takeshi.995@m.kyushu-u.ac.jp

Received: February 13, 2020
Revised: April 21, 2020
Accepted: June 1, 2020
Communicated by Ikki Fujiwara

Abstract

Automated tiered storage with fast memory and slow flash storage (ATSMF) is a hybrid storage system located between non-volatile memories (NVMs) and solid state drives (SSDs). ATSMF reduces the average response time for input-output (IO) accesses by migrating concentrated IO-access areas from an SSD to an NVM. However, the current ATSMF implementation cannot sufficiently reduce the average response time because of the bottleneck caused by the Linux brd driver, which is used for the NVM access driver. The response time of the brd driver is more than ten times longer than memory-access speed. To sufficiently reduce the average response time, we developed a block-level driver for an NVM called a "two-mode (2M) memory driver." This driver has a .map-access mode and direct-access mode to reduce the response time while maintaining compatibility with the Linux device-mapper framework. The direct-access mode has a drastically lower response time than the Linux brd driver because the ATSMF driver can directly execute the IO-access function of the 2M memory driver. Experimental results indicate that ATSMF using the 2M memory driver reduced the IO access response time to less than that of ATSMF using the Linux brd driver in most cases.

Keywords: Hybrid storage system, dynamic random-access memory, non-volatile memory (NVM), flash storage; NVM driver, Linux device-mapper

1 Introduction

Automated tiered storage with fast memory and slow flash storage (ATSMF) [22] is a hybrid storage system located between non-volatile memories (NVMs) and SSDs. To reduce the response time, ATSMF should be used on a server system with direct-attached storage and executed using applications on the same server system. ATSMF migrates the area of input-output (IO) concentration from an SSD to an NVM if it predicts that doing so can reduce the average response time. Previous studies [21, 22] investigated the workloads of a virtual desktop infrastructure (VDI), online transaction

processing (OLTP), a shared file server, web server, and mail server and discussed the characteristics of IO concentration. IO concentration takes up only a small percentage of the storage volume and either remains for a long time or shifts to a neighboring area of the storage volume after several minutes on average. Furthermore, this narrow region includes most IO accesses and appears at unpredictable logical block addresses (LBAs). This narrow region is in 1-GB by 1-minute intervals.

In a previous study [22], however, ATSMF could not sufficiently reduce the response time because it used the Linux brd driver as its NVM access driver, and the brd driver had a slow response time and slow migration speed on joining ATSMF. The response time of the brd driver was more than ten times longer than memory-access speed. ATSMF has to use the brd driver because the ATSMF driver is built on the Linux device-mapper framework and should be connected to both the NVM access driver and NVMe driver by using the Linux device-mapper framework.

To reduce the response time of these workloads, we developed a block-level driver for an NVM called a "two-mode (2M) memory driver." This driver has simple memory management, a direct-access mode, and .map-access mode. Simple memory management reduces the overhead of retrieving a target memory area by dividing this area not into pages but into 1-GB units. For the direct-access mode, the 2M memory driver exposes the symbol of its IO access function to other kernel drivers to reduce software overheads for the Linux device-mapper framework. The .map-access mode provides the functions to be added to the Linux device mapper framework [3] and handles an IO request by using the Linux BIO structure [7]. For both modes to coexist, the memory-access code of the 2M memory driver is shared among both modes, and the ATSMF driver calls both modes exclusively for each sub-logical unit number (sub-LUN). The sub-LUN is a portion of storage volume, and ATSMF manages a storage volume by sub-LUN units. The 2M memory driver is effective even when applied to other than ATSMF. This is because the simple memory management can reduce the response time even if the .map-access mode is used. Therefore, the 2M memory driver reduces the response time and becomes compatible with the Linux device-mapper framework.

The key contributions of this work are as follows.

- We developed the 2M memory driver, which has simple memory management, direct-access mode, and .map-access mode, for simultaneously achieving both low IO-access response time and compatibility of the Linux device-mapper framework.
  - The average response time for the 2M memory driver is almost one digit shorter than that for a Linux brd driver by accessing the direct-access mode with simple memory management (1 microsecond on average).
  - The 2M memory driver can support the Linux device-mapper framework by using the .map-access mode with simple memory management.
- We succeeded in reducing the average response time of ATSMF by 27% when we replaced the NVM access driver from the Linux brd driver with the 2M memory driver.
- We also succeeded in reducing the average response time of ATSMF even when the memory capacity for an NVM was set to 10% of the entire volume.

2 ATSMF with Linux brd driver

2.1 Characteristics of IO Concentration

ATSMF reduces the average response time by migrating the whole IO concentration area from an SSD to an NVM dynamically. The IO concentration is included in many workloads, which consist of a VDI server, OLTP, a shared file server, web server, and mail server [21, 11].

The characteristics of IO concentration are summarized in Figure 1. One square denotes a 1-GB by 1-minute interval, and the set of red squares denotes the IO concentration. The size of this concentration is a small percentage of the entire storage volume. Its duration is more than one minute, and its concentration is more than 50 % of the IO access for this interval. Almost all the start LBAs for IO concentration are unpredictable.

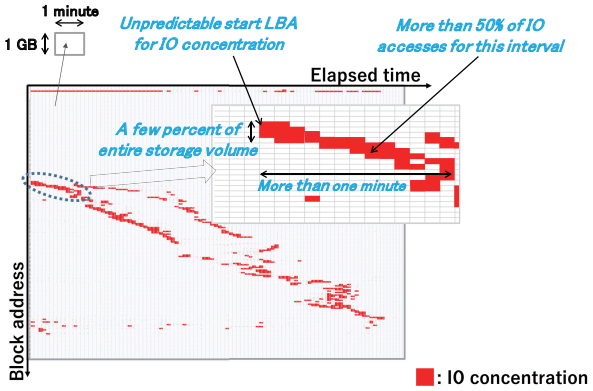

Figure 1: Characteristics of IO concentration

Migration for conventional caching is done by page (4-KB) unit, and its typical migration algorithms are least recently used (LRU), adaptive replacement cache (ARC) [17], and so on. However, its cache hit ratios vary with each workload including IO concentration, but most workloads have low ratios. This is because the IO-concentration area often includes few continuous and few recursive IO-access patterns [23].

2.2 Overview of ATSMF

ATSMF [22, 23] is a hybrid storage system located between NVMs and SSDs to effectively handle IO concentration. Figure 2 shows an overview of ATSMF. Its migration algorithm finds an entire area of IO concentration and migrates it from an SSD to an NVM if the migration is predicted to improve performance for the user. To determine the entire IO concentration, its migration unit should be around 1 GB. The assumed environment is a server with an NVM and SSD, with the user application executed on the server as this reduces the average response time.

The overview of ATSMF's migration algorithm is as follows. The system predicts the effectiveness of migration by monitoring the increase in response time during migration and the decrease in this time after migration. The system continually monitors this increase, this decrease, and the duration of migration then uses the monitored data to predict the average increase and average decrease in the times and duration of each IO concentration. It also continually monitors the occurrence of IO concentrations. When it identifies a concentration, it first predicts the concentration's remaining duration then calculates and compares the response-time increase during migration and response-time decrease after migration by using the predicted duration and monitored values. If the response-time decrease after migration exceeds the response-time increase during migration, the system migrates the data in the target regions to the NVM. It also identifies regions that will likely be targeted in the immediate future and immediately migrates the data in those regions to the NVM.

When the memory consumption of 2M ATSMF reaches the upper limit of memory capacity, the exclusion algorithm of 2M ATSMF is similar to that of LRU. 2M ATSMF selects the least accessed U sub-LUNs and migrates these U sub-LUNs from the memory to the SSD before the replacement judgment from the SSD to memory described in a previous study [22]. We set U to 10 in the evaluation of this paper.

2.3 Overhead of ATSMF using Linux brd driver

Figure 3 shows the overhead of the previous implementation of ATSMF, i.e., ATSMF using the Linux brd driver (brd ATSMF). This ATSMF is composed of a tiering manager and tiering driver. The tiering manager executes the migration algorithm described in Section 2.2. The tiering driver executes the migration between an NVM and SSD when the tiering manager orders the migration

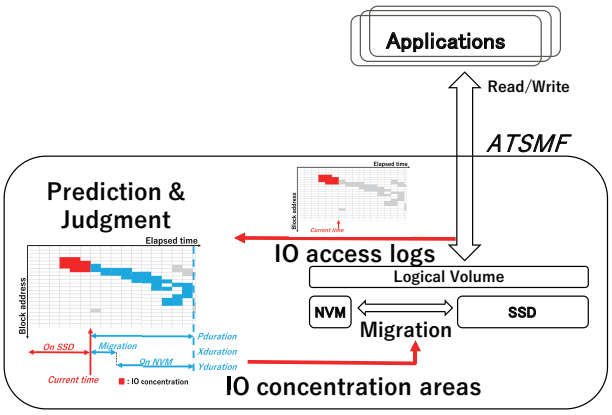

Figure 2: Overview of ATSMF

to the tiering driver. If NVM migration is completed, the tiering driver registers the migration area to the tiering table. This driver also distributes the IO request by referring to the tiering table. The kcopyd driver is a default device-mapper driver [3], and ATSMF orders the kcopyd driver to execute migration. The non-volatile memory express (NVMe) driver [9] is a default Linux driver for handling NVMe SSDs.

The brd driver [2] is installed in the Linux OS by default, and ATSMF uses the driver as its NVM access driver. The brd driver can join the ATSMF tiering driver because it is a device-mapper driver and can be accessed using the Linux BIO structure [7]. However, the brd driver incurs large software overheads and an average response time of around 10 microseconds per IO request, while NVM latency is less than 1 microsecond. Thus, the ATSMF using the brd driver cannot sufficiently reduce the average response time even if its NVM hit ratio is more than 80%. Moreover, when the workload, whose IO concentrations often shift to a neighboring area in a short interval, is executed, the ATSMF cannot sufficiently improve the NVM hit ratio because of the occurrence of frequent migration. Frequent migration often cannot be completed in time since the response time of the brd driver is slow. We want to make the average response time of the NVM access driver to be as close as possible to memory-access speed.

To clarify the reason for the slow response time of the Linux brd driver, we added an internal latency measurement mechanism to the Linux brd driver, built ATSMF with this updated brd driver, and evaluated this ATSMF by replaying a portion of the MSR Cambridge src1_0 workload. Its replay duration was about 90 minutes, and we calculated the average values of these internal latencies by using measured values. As a result, the total latency of the Linux brd driver was 9 microseconds on average and the latency for memory management function was 7 microseconds on average. In the study of this latency of the Linux brd driver using the kernel profiler, lock-wait of the radix-tree [8] is found to occupy the major part of the time. At this point, concentrated IO requests are supposed to be causing frequent lock-waits, since the driver assigns the memory area for each page through the radix tree. The remaining latency was 2 microseconds on average, and the major overhead was the device-mapper handling.

3 NVM driver with low response time and Linux device-mapper compatibility

As explained in Section 2.3, brd ATSMF cannot sufficiently reduce the average response time because of large software overheads for the Linux brd driver. We believe that this is a common problem for device-mappered storage systems with Dual Inline Memory Module (DIMM) attached NVMs. To solve this problems, we developed the 2M memory driver, which dramatically reduces response time

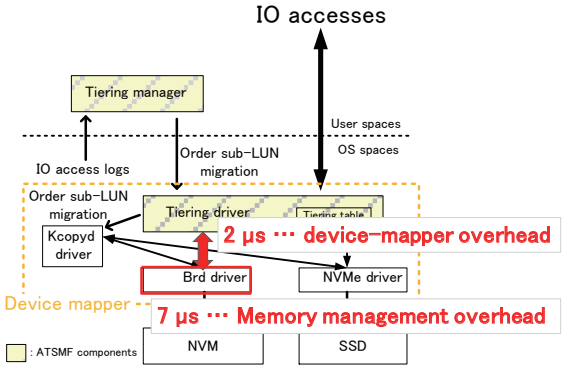

Figure 3: Overhead of ATSMF using Linux brd driver

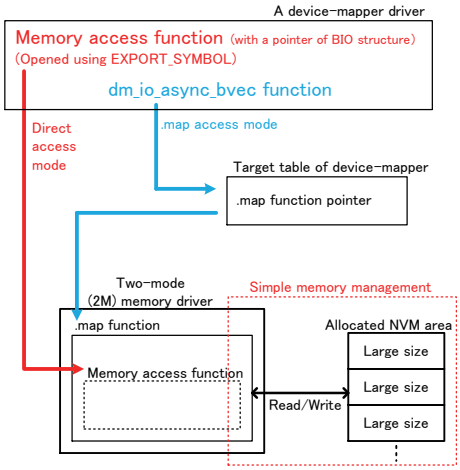

Figure 4: Overview of 2M memory driver

and is compatible with the Linux device-mapper framework.

The 2M memory driver has three features: simple memory management, a direct-access mode, and .map-access mode (see Figure 4). Simple memory management dramatically reduces the memory-management overhead. The direct-access mode reduces the device-mapper overhead between the 2M memory driver and a device-mapper driver by using an original application programming interface (API). The .map-access mode can access data between the 2M memory driver and a device-mapper driver by using the default API for device-mapper framework.

Then, the 2M memory driver's strong point is to achieve low-latency access by using both the simple memory management and the direct-access mode. Its weak point is that programmer should consider how to use properly both the direct-access mode and .map-access mode. The programmer should choose the .map-access mode when he implement the functions for device-mapper framework. He should choose the direct-access mode when he implement the function for data access. He should also use both mode separately by each block of the storage volume.

A traditional device driver for device-mapper framework can also use the 2M memory driver by only using .map access mode. Then, its performance improvement will be limited.

We explain these three features in the following sub-sections.

3.1 Simple memory management

Figure 4 shows an overview of simple memory management, which manages the memory area by not page but by a large unit to dramatically reduce the search latency for retrieving the target memory area. The large unit will be dramatically larger than page size to unuse the radix tree search. During initialization for the 2M memory driver, it allocates several large memory units for the NVM storage region. When handling an IO request, it specifies the memory area by calculating the offset of the allocated NVM regions. The brd driver executes the radix tree search for each IO request to retrieve the target memory area. The search often incurs large overhead (see Section 2.3).

When the brd driver is set by using a large page (ex.1-GB) unit, the overhead for radix tree search may be reduced because the number of leaf for radix tree will be greatly decreased. Then, the latency for brd driver with large page may be almost same as that for 2M memory driver when the NVM capacity is less than 800 GB, which is the capacity of this paper's evaluation. However, the capacity for intel Optane DC Persistent Memory (DCPM) [12] reaches 6 TB when a general 2-socket server is used. And, the DCPM's capacity will be extended in the near future. Therefore, the overhead for radix tree with large page will increase when the brd driver is executed with almost all the DCPM's capacity.

3.2 Direct-access mode

Figure 4 shows an overview of the direct-access mode. This mode can be accessed using an original API and reduces response time by skipping the Linux device-mapper interface. The parameter for this original API is only a pointer of the Linux BIO structure, which is also used by calling the Linux device-mapper interface. When the 2M memory driver is initialized, it registers the memory-access function in the kernel's symbol table by using "EXPORT_SYMBOL". Then, a driver can directly call the memory-access function with only a pointer of the Linux BIO structure when the driver executes IO requests. It then calculates the offset of areas by using simple memory management and copies them between the calculated memory area and IO request when it executes an IO request. The copy code retrieves the multiple memory pointers of an IO request from its BIO structure, maps the pointed areas using "kmap_atomic", and copies them between the calculated memory area and IO request.

When a programmer uses the direct-access mode, he should get the pointer for its memory-access function from the kernel's symbol table and call the memory-access function with the pointer of the Linux BIO structure.

3.3 .map-access mode

The Linux device-mapper framework can generate a virtual storage volume by combining multiple storage drivers. Thus, the hybrid storage structure for ATSMF is built using the device-mapper framework. Figure 4 also shows an overview of the .map access mode. When the 2M memory driver is initialized, it registers the .map function, which handles IO requests, to the target table of the device-mapper by using the "dm_register_target" function. The .map function must be registered when the 2M memory driver is added to the device-mapper, and the dm_register_target function is a device-mapper management function. When handling IO requests, a driver calls the "dm_io_async_bvec" function, which is an IO-request function of the device-mapper, with target information. Then, the device-mapper searches the target table for the .map function of the target driver and calls that function.

The Linux brd driver can also handle an IO request by using the Linux device-mapper interface. However, its response time is higher than that of the 2M memory driver because of simple memory management.

3.4 2M memory driver for applying ATSMF

Figure 5 shows the prototype system for ATSMF with the 2M memory driver on Linux. The 2M memory driver allocates memory area in 1-GB units because 1 GB is the upper limit for the vmalloc

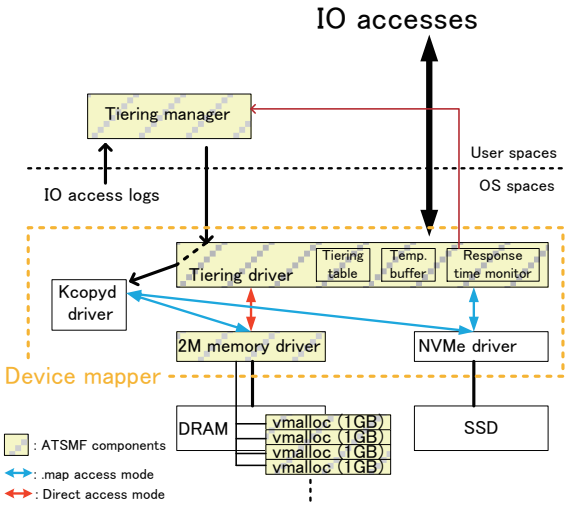

Figure 5: ATSMF with 2M memory driver on Linux

function (Linux 3.10.0-229.el7.x86_64). The system uses dynamic random access memory (DRAM) as a substitute for an NVM.

In the next subsection, we briefly explain ATSMF architecture, and the tiering-driver implementation for accessing the 2M memory driver.

3.4.1 ATSMF architecture

ATSMF consists of a tiering manager, tiering driver, and the 2M memory driver. The tiering manager retrieves the number of IOs per sub-LUN and determines whether the sub-LUN should be migrated by using the algorithms mentioned in [22, 23]. The tiering manager requests that the tiering driver migrates the sub-LUN if the judgment states so.

The tiering driver has three functions. First is distribution of IO request between an NVM and SSD by using the tiering table. Second is the temporary buffer. It keeps the IO request data of a sub-LUN that is migrating. When the migration is completed, its data write back either the NVM or SSD. Third is the response time monitor. The monitored values are used by the migration algorithm mentioned in [22, 23].

3.4.2 Tiering driver implementation for accessing 2M memory driver

The storage volume for ATSMF is divided and managed by sub-LUN unit. Each sub-LUN is assigned to either an NVM region or SSD region. When a sub-LUN is migrating between NVM and SSD regions, the IO requests for migrating sub-LUN are stored to the temporary buffer until the migration is completed. Therefore, the tiering driver accesses the NVM-assigned sub-LUN by direct-access mode.

The sub-LUN migration is executed from the kcopyd driver. The kcopyd driver, which is a default function of the Linux device-mapper, accesses the NVM region by using the .map-access mode of the 2M memory driver. The tiering driver orders the migration to the kcopyd driver when it receives the migration order from the tiering manager. The kcopyd driver moves data of the specified sub-LUN between the NVM and SSD regions. When the sub-LUN migration is completed, the tiering driver moves the written data from the temporary buffer to the NVM region or SSD region and updates the tiering table.

Table 1: Workloads used in experiments

Workload	GB	Write ratio (%)	Comments
VDI	8000	22.98	
DBT2	800	100.00	
src1_0 (MSR)	293	84.24	retrieved from 1920 to 2020 (*1)
src1_1 (MSR)	293	0.91	retrieved from 480 to 600 (*1)
proj1 (MSR)	800	1.16	retrieved from 230 to 550 (*1)
proj2 (MSR)	800	42.29	retrieved from 6760 to 6890 (*1)
proj4 (MSR)	236	0.22	retrieved from 7390 to 7600 (*1)
usr1 (MSR)	800	0.25	retrieved from 2090 to 2350 (*1)
usr2 (MSR)	569	0.22	retrieved from 5970 to 6130 (*1)
web2 (MSR)	182	0.03	retrieved from 5870 to 5930 (*1)
src1_0+ (samba)	293	82.84	created from src1_0
src1_1+ (samba)	293	0.87	created from src1_1

*1: Elapsed time (minutes) from first trace logs

3.4.3 2M memory driver for applying a general device-mapper driver

The 2M memory driver is also effective when applied to a general device-mapper driver. This is because the simple memory management can reduce the response time even if the .map-access mode is used. The major examples for general device-mapper driver is the Linux Logical Volume Manager (LVM). It can attach the 2M memory driver by using .map-access mode.

4 Evaluation

4.1 Experimental system

We implemented the prototype of ATSMF with the 2M memory driver (2M ATSMF) on a Fujitsu PRIMERGY TX300S8, which included two Intel Xeon E5-2660 processors, 896-GB memory, and four 2-TB Intel P3700 SSDs and was installed with Linux CentOS 7.1. We also created an 8-TB logical volume management (LVM) volume by using four 2-TB Intel P3700 SSDs.

The 2M memory driver used a portion of the 896-GB memory, as per the experimental conditions. ATSMF also used the 8-TB LVM as an SSD volume. We used DRAM as a substitute for an NVM because some NVMs [1] have almost the same memory-access latency as DRAM.

The output of the Linux blktrace command was used as block-trace logs, which was the input of the tiering manager. ATSMF migrates the data between an NVM and SSD in sub-LUN units. In this study, a sub-LUN was 1 GB, and the execution interval of the tiering manager was 24 seconds. The interval included the execution time (20 seconds) and post-process (4 seconds) of the block trace. ATSMF then determined the migration at intervals of 24 seconds.

4.2 Methodology

Our intention was to clarify whether 2M ATSMF can handle workloads that generate many IO concentrations. We evaluated its performance in terms of average response time and memory-access ratio. The average response time determines the overall effectiveness of this system including the migration overhead. The memory-access ratio determines its effectiveness without the overhead. We compared the performance of 2M ATSMF with that of brd ATSMF, SSD only, and caching between the memory and SSD by replaying VDI and DBT-2 workloads and several shared fileserver workloads described in Table 1. The VDI workload consisted of the eight 8-TB volumes logs [21]. We chose the log that had the most IO accesses. We also prepared the workloads divided into units of 800 GB (0–800, 800–1600, 1600–2400, 2400–3200, 3200–4000, 4000–4800 GB) because we evaluated the relationship between the number of sub-LUN migrations and memory-access ratio. We did not prepare the workloads whose addresses were more than 4800 GB because these areas had no IO

requests. We used the DBT-2 trace logs described in a previous study [22]. The shared fileserver workloads consisted of the MSR Cambridge [19] and Samba [21] workloads. Most MSR Cambridge workloads appear alternatively when many IO accesses are included and when few IO accesses are included. and these durations are about a week. We then selected one period including many IO accesses from each workload. The duration for each selected period was between 60 to 120 minutes. We created trace logs for the Samba workloads by using portions of the MSR Cambridge src1_0 and src1_1 workloads because their logs were in an aggregate of 1-GB by 1-min units. Oe et al. [20] explained how to create the trace logs.

We used both the Facebook FlashCache [5] and Facebook FlashCache with lazy adaptive replacement (FlashCache-LARC) [15, 4] for the caching implementation. We set the replacement algorithm of FlashCache to least recently used (LRU) and set the replacement algorithm of FlashCache-LARC to LARC. We believe that LARC is one of the most effective cache replacement algorithms because it determines a replacement by using both recency and frequency. We loaded these drivers with a write-back mode and used the Linux brd driver as a block-level access memory driver of FlashCache. The other options of these drivers were the defaults. We downloaded their latest editions and used them for our evaluation.

The trace logs we created were reproduced using the Linux *btreplay* command. Options *-X 1 -W 1* in *btreplay* were specified, where *-X 1* means single speed and *-W 1* means write enable.

We conducted three evaluations. The first was on the effectiveness of the 2M memory driver by using all workloads except for VDI ALL(8000 GB). This is because in a previous study [22], the authors executed brd ATSMF by using the workloads of up to 800 GB. Moreover, we divided these workloads into two groups. One was VDI and DBT-2, and other was MSR Cambridge and Samba. The execution for the VDI and DBT-2 workloads incurred SSD overload because they both included many IO accesses. On the other hand, the execution for the MSR Cambridge and Samba workloads did not incur SSD overload. The second evaluation was on the effectiveness of 2M ATSMF when the upper limit of the memory amount was 10% of the entire volume capacity. The third evaluation was on the effectiveness of 2M ATSMF on executing the VDI ALL workload.

4.3 Evaluation results

A response time for ATSMF depends on the memory-access ratio, response time of the memory part, and that of the SSD part. When the memory-access ratio is high, the response time of its memory part greatly affects the whole response time for ATSMF. When the memory-access ratio is low, the response time of its SSD part greatly affects the whole response time for ATSMF. Therefore, we discuss the evaluation results for ATSMF by using these three data sets.

4.3.1 Effectiveness of 2M memory driver (MSR and Samba)

Figure 6 shows the average response times of the 8-TB LVM SSD volume (SSD only), which consisted of four 2-TB Intel P3700 SSDs, FlashCache-LARC (LARC), FlashCache-LRU (LRU), and ATSMF. Figure 7 shows the average response times of ATSMF's memory driver, and Figure 8 shows the memory-access ratios of the LARC, LRU, and ATSMF for all workloads. This figure also shows the maximum memory consumption (MMC) of ATSMF. The MMC is a percentage of each GB column for Table 1. ATSMF migrated the IO cocentrated areas from SSD to memory (NVM) when it detected the IO concentrations described in a previous paper [22]. It also migrated the areas from memory to SSD when the IO concentration had already been completed. Therefore, the memory consumption of ATSMF changed over time. We also set the memory capacity of both LARC and LRU to the MMC of ATSMF.

We compared the average response times between brd ATSMF and 2M ATSMF. Our 2M ATSMF was the most effective except for usr1 (see Figure 6). We succeeded in reducing the average response time by 27% when we changed the NVM access driver from brd to 2M. Figure 7 shows the average response times for both the 2M memory driver and Linux brd driver. The average response times for the 2M memory driver are almost one digit shorter than those for the Linux brd driver (1 microseconds on average).

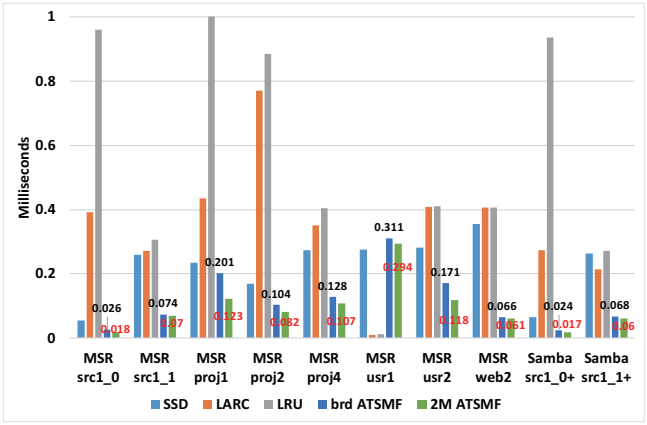

Figure 6: Average response times for MSR and Samba (milliseconds)

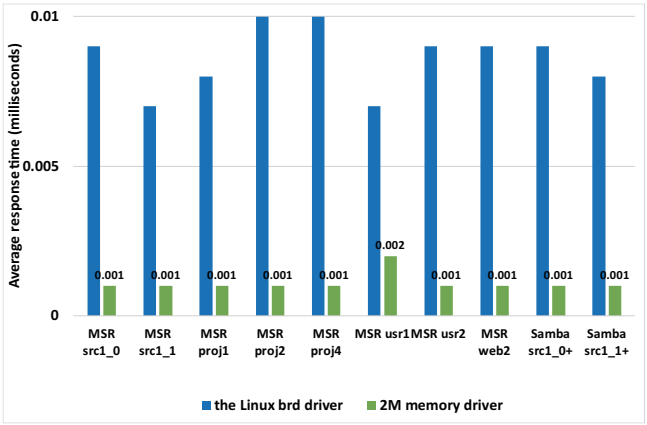

Figure 7: Average response times of ATSMF's memory drivers (MSR and Samba)

Figure 8 shows that the memory-access ratios for 2M ATSMF were almost the same as those for brd ATSMF except for proj1 and usr2. When both proj1 and usr2 were executed, the memory-access ratios for 2M ATSMF surpassed those of brd ATSMF because their IO concentrations often shift to a neighboring area in a short interval, and migration of brd ATSMF often could not be completed in time. Previous studies [22, 23] also discussed the results of proj1 execution.

The average response times for both LARC and LRU were drastically higher than those for ATSMF except for usr1 because of the memory-access ratios. The memory-access ratios of both LARC and LRU were drastically lower (see Figure 8). The poor memory-access ratios of both LARC and LRU mean that their replacement algorithms could not effectively handle the IO accesses of these workloads and most of the IO accesses involved in the IO concentrations. Therefore, we determined that their IO concentrations included few page-level regularities and few repeat accesses to the same pages. On the other hand, ATSMF could handle the IO concentrations effectively because it identified IO accesses not by page but by 1-GB units. Thus, it could determine the entire IO-concentration area.

When usr1 was executed, the average response time of ATSMF was highest and its memory-access ratio was lowest among other systems (see Figure 8). This is because almost all IO accesses appeared in about the 10-GB range of the entire volume, and usr1 had IO and non-IO concentrations that repeated in short intervals. Therefore, the replacement algorithm of ATSMF could not keep the IO concentrations on the memory.

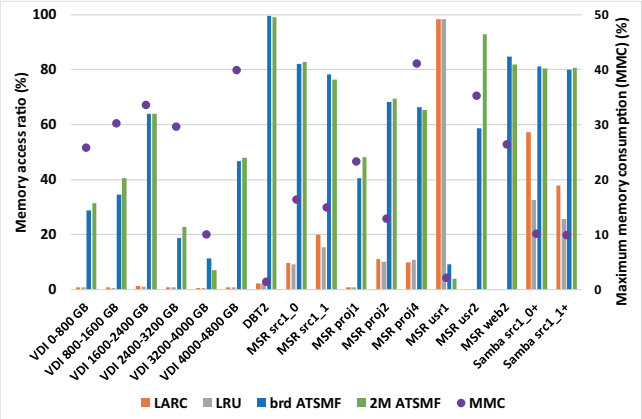

Figure 8: Memory-access ratios for all workloads (%)

4.3.2 Effectiveness of 2M memory driver (VDI and DBT2)

Figure 9 shows the average response times of the 8-TB LVM SSD volume (SSD only), FlashCache-LARC (LARC), FlashCache-LRU (LRU), and both ATSMF implementations. Figure 10 shows the average response times of both ATSMF's memory drivers.

First, we explain the results for SSD only. Figure 9 shows that the average response times for SSD only were more than 100 milliseconds due to SSD overload. Both VDI and DBT-2 included many IO accesses. Thus, the average response times for LARC, LRU, brd ATSMF, and 2M ATSMF also greatly increased compared to those for MSR and Samba's results. However, the average response times for both ATSMF implementations were drastically shorter than those for SSD only, LARC, and LRU when both the DBT2 and VDI 1600-2400 GB were executed because of high memory-access ratios and low average response times for the 2M memory driver. Figure 8 shows that the memory-access ratios for both ATSMF implementations were more than 60%. Figure 10 shows that the average response times for the 2M memory driver were less than 2 micro-seconds when both DBT2 and VDI 1600-2400 GB were executed.

Figure 10 shows that the average response times of the 2M memory driver were almost the same as those of the Linux brd driver probably because of large delay for software interrupt of the 2M memory driver. The 2M memory driver used workqueue, which was prepared by the Linux kernel, when the .map-access mode was executed. The workqueue handling resulted in software interrupt. The 2M memory driver had large delay when the read-dominant workloads were executed because of this interrupt. However, it had small delay when both DBT-2 and VDI 1600-2400 GB were executed because they were write-dominant workloads. We compared the average response times between brd ATSMF and 2M ATSMF when replaying VDI 1600-2400 GB. Both memory-access ratios were almost the same (64%), and the average response time of the 2M memory driver (0.002 ms)was dramatically lower than that of the Linux brd driver (0.013 ms). However, the average response time of the 2M ATSMF (68.17 ms) was slightly higher than that of the brd ATSMF (63.73 ms) because of worse SSD response time. The average response time of the 2M ATSMF's SSD part (186 ms) was slightly higher than that of the brd ATSMF's SSD part (176 ms). The reason for this difference is under investigation.

Figure 8 also includes the memory-access ratios for VDI and DBT-2. The memory-access ratios of both ATSMF implementations were drastically higher than those for LARC and LRU. We assume that the average response times for both ATSMF implementations will be drastically lower than those for SSD only, LARC, and LRU if there is no SSD overload.

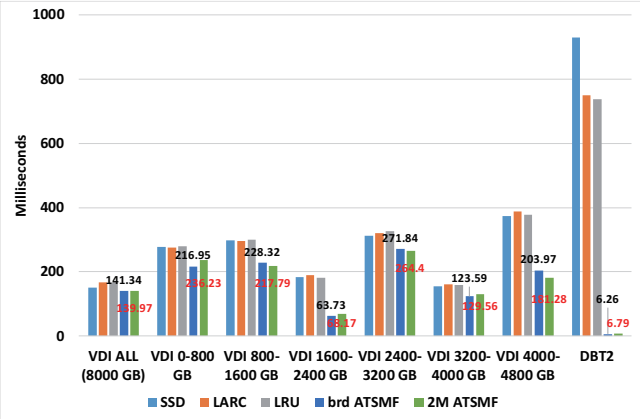

Figure 9: Average response times for VDI and DBT-2 (milliseconds)

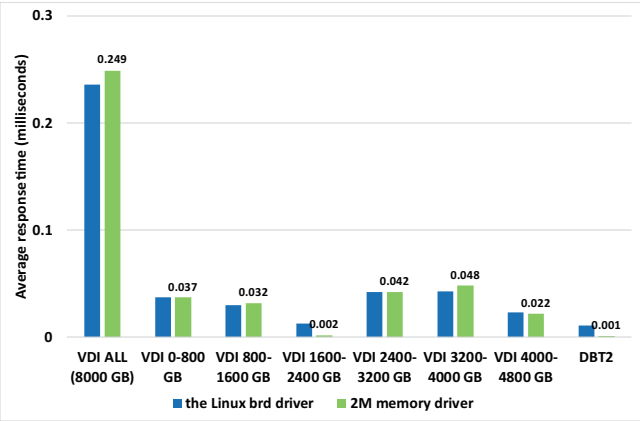

Figure 10: Average response times of ATSMF's memory driver (VDI and DBT-2)

4.3.3 Effectiveness on executing 10-percent memory capacity

ATSMF migrates the IO- concentrated areas from SSD to memory (NVM) when it detects the IO concentrations and migrates the areas from memory to SSD when the IO concentrations have already been completed. In other words, the memory consumption of 2M ATSMF changed in accordance with the number of IO concentrations included in the workloads. Moreover, our current implementation prioritizes memory migration; therefore, it may not execute SSD migration immediately. Figure 8 shows the results for the MMC of the evaluation explained in Sections 4.3.1 and 4.3.2. The results revealed that most of the MMC was about 10–40%.

We then investigated the performance of 2M ATSMF when the upper limit of memory was 10% of the entire volume capacity. We chose VDI 0–800 GB, VDI 800–1600 GB, VDI 1600–2400 GB, VDI 2400–3200 GB, VDI 4000–4800 GB, src1_0, src1_1, proj1, proj4, usr2, and web2 because the MMC of these workloads was more than 10%. We compared the performance of 2M ATSMF with that of SSD only and FlashCache-LARC (LARC) by replaying the above workloads. We also set the memory capacity of LARC to 10% of the entire volume capacity. When the memory consumption of 2M ATSMF reaches the upper limit of memory capacity, the exclusion algorithm of 2M ATSMF, which is described in Section 2.2, is executed.

Figure 11 shows the results of this evaluation. The average response times of 2M ATSMF were lowest among the other systems, and its memory access ratios were also higher than those of LARC. The average response times in this evaluation were slightly higher than those in the other evaluations

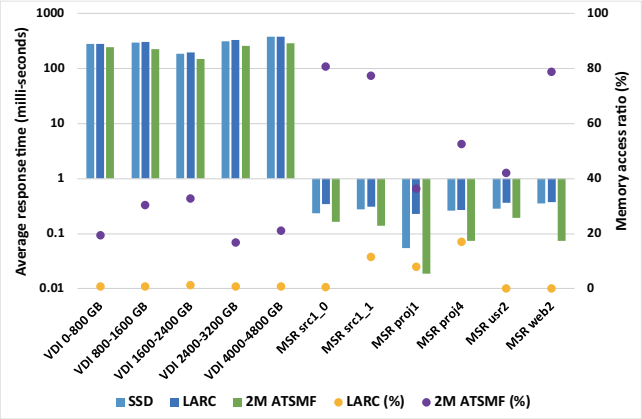

Figure 11: Average response times when MMC was 10%

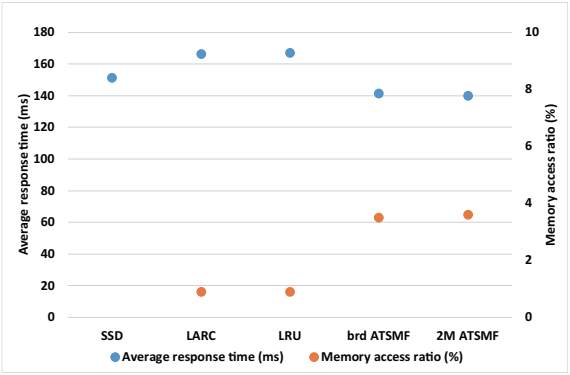

Figure 12: Average response times and memory-access ratios for VDI ALL

(Figures 9 and 6), and the memory-access ratios were slightly lower than those in Figure 8. The size of the difference depended on the migration frequency for each workload. We assumed that the high frequency increased the average response time because not all migrations could be completed in time. To reduce the average response time involving high-frequency migrations, we should accelerate the migration speed between memory and SSDs.

4.3.4 Effectiveness on executing VDI ALL workload

Figure 12 shows the average response times and memory-access ratios for VDI ALL workload. SSD did not have a result for memory-access ratio because SSD is executed using only an SSD. When we executed VDI ALL, 2M ATSMF's average response time was 139.97 milliseconds and its memory-access ratio was only 3.6 %. However, the VDI workloads, divided into units of 800 GB, were from 7.2 to 64.0 % (see Figure 8). This is because the migration of VDI ALL could not be completed in time. The MMC of VDI ALL (2.3%) was smaller than that of the divided workloads, although the capacity of VDI ALL was eight times higher than that of the divided workloads. We also prepared the workloads divided into units of both 2400 and 1600 GB (0–2400, 2400–4800, 0–1600, 1600–3200, 3200–4800 GB) and executed them with 2M ATSMF. Figure 13 shows the results. The memory-access ratios of 2M ATSMF increased as size division decreased. When 1600–2400 GB was executed, 2M ATSMF's average response time was drastically lower than for other 800-GB portions. This is probably because its memory-access ratio is drastically higher and its migration frequency is smaller than for other 800-GB portions. This is evidence that migration cannot be completed in time when

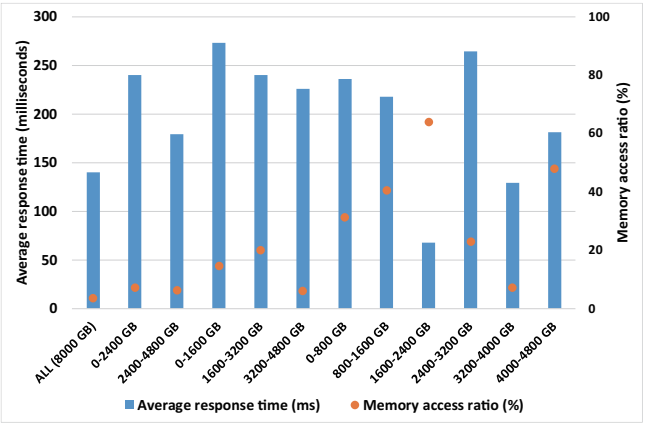

Figure 13: Average response times and memory-access ratios for VDI workloads

Table 2: Estimated average response times with Intel 3D XPoint

Workload	Average response time (ms) (A)	Memory access ratio (%) (B)	*Estimated average response time (ms) (=A + 0.0005*B/100)*
src1_0 (MSR)	0.018	82.9	0.018
src1_1 (MSR)	0.070	76.3	0.070
proj1 (MSR)	0.123	48.3	0.123
proj2 (MSR)	0.082	69.5	0.082
proj4 (MSR)	0.107	65.5	0.107
usr1 (MSR)	0.294	3.9	0.294
usr2 (MSR)	0.118	92.8	0.118
web2 (MSR)	0.061	82.0	0.061
src1_0+ (Samba)	0.017	80.5	0.017
src1_1+ (Samba)	0.060	80.7	0.060

VDI ALL is executed. Therefore, we should divide workloads and operate 2M ATSMF with each divided workload if its migration cannot be completed in time.

5 Discussion

Intel 3D XPoint [6], another NVM under development, uses a new technology that is not flash. Its SSD version has come onto the market, and a DIMM-attached version will come onto the market later in 2019. Therefore, we estimate the average response time of ATSMF with the DIMM-attached Intel 3D XPoint when its access latency is 500 nanoseconds. This is because its write latency was reported as approximately 500 nanoseconds in a prior study [18] [14]. For this estimation, we used the results of executing 2M ATSMF on the MSR and Samba workloads because their average response times were under 1 milli second.

Table 2 lists the estimation results. The estimated average response times were almost the same as the current average response times, because they were on the order of more than ten micro-seconds. This shows that 2M ATSMF is effective at reducing the average response time when its NVM is Intel 3D XPoint instead of DRAM.

Finally, we explained write endurance of Intel 3D XPoint. The latest review document for Intel Optane [12] showed that Intel Optane DC persistent memory, which is the same as Intel 3D XPoint DIMM-attached version, was measured in Petabytes Written (PBW). PBW means that this persistent memory has a 5-year lifetime. Therefore, the 2M memory driver will be able to maximize performance of the persistent memory even if it has no implementation for wear leveling.

6 Related work

We now discuss other block-level NVM device drivers. Bankshot [13] is a caching system between an NVM and disk and has two access paths for the NVM area. To reduce the response time of applications, the NVM area can be directly accessed from a cache library of the user space when a cache hits. If a cache misses, the cache library should access the NVM area by using the Bankshot driver of the OS space. The difference between 2M ATSMF and Bankshot is versatility. An application must use the cache library of Bankshot when operating Bankshot, though the application's response time is drastically reduced when a cache hits. On the other hand, 2M ATSMF's direct-access mode can reduce the average response time without any modification to an application, and its .map-access mode makes it compatible with the Linux device-mapper framework.

"PMEM Block Driver" [16] is a default Linux block driver for Intel 3D-Xpoint DIMM-attached version. The driver can access it like an SSD. ATSMF can be then constructed using the driver. However, the driver has no implementation for the direct-access mode of the 2M memory driver. Therefore, the response time for ATSMF with "PMEM Block Driver" will be higher than that for ATSMF with the 2M memory driver.

The pmem.io web site [10] provides the Persistent Memory Development Kit (PMDK) and many documents explaining how to use it. The PMDK includes libraries to access an NVM device from the user level. The memory-access codes of the 2M memory driver must be re-written when the libraries, which can be accessed in the kernel, are released.

Next, we discuss research on ATSMF. Rajasekaran et al. [24] proposed Multi-Cache, a multi-layer cache-management system that uses a combination of cache devices of varied speeds and costs, such as SSDs and NVMs, to dynamically allocate cache capacities among different virtual machines (VMs). Multi-Cache partitions each device dynamically at runtime in accordance with the workload of each VM and its priority. It uses a heuristic optimization technique that ensures the maximum utilization of caches, resulting in a high hit ratio. In comparison, ATSMF dynamically detects and migrates IO concentrations in one VM; therefore, we can reduce the average response times by combining both ATSMF and Multi-Cache.

On-the-fly automated storage tiering (OTF-AST) [20] is a hybrid storage system located between SSDs and HDDs. The OTF-AST replacement algorithm also identifies IO concentrations lasting for moderately long durations and migrates the data in targeted regions from an HDD to an SSD. The durations are long enough after migration to reduce the overall average response time. This is because the change in HDD latency is very large when migrating between SSDs and HDDs, so the changes in average response time cannot be precisely predicted. The algorithm for identifying IO concentration uses a static parameter for each workload that is determined by analyzing the workload in advance.

7 Conclusion

There are applications, particularly, virtual desktop infrastructures and in-memory database systems, that require storage systems with shorter response times than an SSD's response time. Their workloads were found to contain many IO concentrations.

ATSMF is a hybrid storage system located between NVMs and SSDs. ATSMF migrates the area of IO concentration from an SSD to an NVM if it predicts that doing so will reduce average response time. However, in previous studies, it could not sufficiently reduce the response time because its NVM access driver has a slow response time and slow migration speed on joining ATSMF. The Linux brd driver has been used as the NVM access driver. ATSMF has to use the brd driver because the ATSMF driver joins its NVM access driver by using the Linux device-mapper framework.

To reduce the response time of these workloads, we developed the 2M memory driver, which has two IO access modes, i.e., direct-access and .map-access modes, to balance reduction in response time and compatibility with the Linux device-mapper framework. The response time of the direct-access mode is made drastically lower than that of the Linux brd driver by directly calling the memory access function of the 2M memory driver (reduced from 10 to 1 microseconds on average). We

re-wrote the tiering driver of ATSMF to use the direct-access mode, and its NVM access response time was drastically lower than the previous implementation (brd ATSMF).

We experimentally evaluated ATSMF with the 2M memory driver (2M ATSMF) by measuring the average response time for replaying the VDI, DBT-2, and several shared file-system workloads. 2M ATSMF outperformed brd ATSMF for almost all workloads (17 microseconds under best conditions) and drastically outperformed SSD only and caching between the memory and SSD for almost all workloads. Our 2M ATSMF also reduced the average response time even if its memory amount was 10% of the entire volume capacity. We also evaluated 2M ATSMF with an 8-TB workload and found that its average response time could not be sufficiently reduced because the migration could not be completed in time. We could also reduce 2M ATSMF's workload and operate it with each divided workload. Through these evaluations, we demonstrated the superiority of 2M ATSMF under various workloads including IO concentration, memory-amount limitation, and larger volume.

References

[1] AGIGARAM DDR4 NVDIMM. http://www.agigatech.com/.

[2] brd driver. https://www.kernel.org/doc/Documentation/blockdev/ramdisk.txt.

[3] Device mapper. https://en.wikipedia.org/wiki/Device_mapper.

[4] Facebook FLACHCACHE with LARC. https://github.com/seth-hg/flashcache.

[5] Facebook FLASHCACHE. https://github.com/facebook/flashcache.

[6] intel 3D-Xpoint. https://en.wikipedia.org/wiki/3D_XPoint.

[7] Linux BIO structure. http://lse.sourceforge.net/io/bionotes.txt.

[8] Mastering Linux Kernel Development. ISBN 978-1-78588-305-7.

[9] NVMe Linux driver. http://www.nvmexpress.org/resources/drivers/linux-driver-information/.

[10] Persistent Memory Programming. https://pmem.io.

[11] SNIA trace data MSR Cambridge. http://iotta.snia.org/traces/388.

[12] Intel Optane DC Persistent Memory Module (PMM), April 2019. https://www.storagereview.com/intel_optane_dc_persistent_memory_module_pmm.

[13] Meenakshi Sundaram Bhaskaran, Jian Xu, and Steven Swanson. Bankshot: Caching Slow Storage in Fast Nov-Volatile Memory. In *Interactions of NVM/Flash with Operating-Systems and Workloads (INFLOW'13)*, November 2013.

[14] Subramanya R Dulloor, Amitabha Roy, Zhenguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. Data Tiering in Heterogeneous Memory Systems. In *Proc. the 11th ACM European conference on Computer systems (EuroSys 2016)*, April 2016.

[15] Sai Huang, Qingsong Wei, Jianxi Chen, Cheng Chen, and Dan Feng. Improving flash-based disk cache with Lazy Adaptive Replacement. In *Proc. of 29th IEEE Storage Conference on Massive Data Storage (MSST 2013)*, 2013.

[16] Intel. NVDIMM Namespace Specification (Revision 1.0), April 2015. https://nvdimm.wiki.kernel.org/.

[17] Nimrod Megiddo and Dharmendra S. Modha. ARC: A SELF-TUNING,LOW OVERHEAD REPLACEMENT CACHE. In *Proc. 2th USENIX conference on File and Storage Technorogies (FAST2003)*, March 2003.

[18] Sparsh Mittal and Jeffrey 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*, May 2016.

[19] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write Off-Loading: Practical Power Management for Enterprise Storage. In *in Proc. of 6th USENIX Conf. on File and Storage Tech*, Feb 2008.

[20] Kazuichi Oe, Satoshi Iwata, Takeo Honda, Motoyuki Kawaba, and Koji Okamura. On-The-Fly Automated Storage Tiering (OTF-AST). In *Proc. of The Third Asian Conference on Information Systems – Special Session on Information Storage (ACIS-IS 2014), Nha Trang, Viet Nam*, Dec 2014.

[21] Kazuichi Oe, Takeshi Nanri, and Koji Okamura. Analysis of storage workloads of input-output access locality and designing of hybrid storage system. In *Proc. of the 5th IIAI International Congress on Advanced Applied Informatics*, July 2016.

[22] Kazuichi Oe, Mitsuru Sato, and Takeshi Nanri. Automated tiered storage system consisting of memory and flash storage to improve response time with input-output (IO) concentration workloads. In *Proc. of 5th International Workshop on Computer Systems and Architectures (CSA'17), Aomori, Japan*, November 2017.

[23] Kazuichi Oe, Mitsuru Sato, and Takeshi Nanri. ATSMF: Automated Tiered Storage with Fast Memory and Slow Flash Storage to Improve Response Time with Concentrated Input-Output (IO) Workloads. *IEICE Transactions on Information and Systems (VOL.E101-D No.12)*, December 2018.

[24] Sundaresan Rajasekaran, Shaohua Duan, Wei Zhang, and Timothy Wood. Multi-Cache: Dynamic, Efficient Partitioning for Multi-Tier Caches in Consolidated VM Environments. In *Proc. of Cloud Engineering (IC2E), 2016 IEEE International Conference, Berlin, Germany*, April 2016.