Proposal and Evaluation of IO Concentration-Aware Mechanisms to Improve Efficiency of Hybrid Storage Systems

Kazuichi OE and Takeshi NANRI, Members

SUMMARY Hybrid storage techniques are useful methods to improve the cost performance for input-output (IO) intensive workloads. These techniques choose areas of concentrated IO accesses and migrate them to an upper tier to extract as much performance as possible through greater use of upper tier areas. Automated tiered storage with fast memory and slow flash storage (ATSMF) is a hybrid storage system situated between non-volatile memories (NVMs) and solid-state drives (SSDs). ATSMF aims to reduce the average response time for IO accesses by migrating areas of concentrated IO access from an SSD to an NVM. When a concentrated IO access finishes, the system migrates these areas from the NVM back to the SSD. Unfortunately, the published ATSMF implementation temporarily consumes much NVM capacity upon migrating concentrated IO access areas to NVM, because its algorithm executes NVM migration with high priority. As a result, it often delays evicting areas in which IO concentrations have ended to the SSD. Therefore, to reduce the consumption of NVM while maintaining the average response time, we developed new techniques for making ATSMF more practical. The first is a queue handling technique based on the number of IO accesses for NVM migration and eviction. The second is an eviction method that selects only write-accessed partial regions in finished areas. The third is a technique for variable eviction timing based on the number of IO accesses for NVM consumption and average response time. Experimental results indicate that the average response times of the proposed ATSMF are almost the same as those of the published ATSMF, while the NVM consumption is three times lower in best case.

key words: hybrid storage system, ATSMF, non-volatile memory (NVM), solid state drive (SSD), dynamic tiering

1. Introduction

Hybrid storage techniques, such as caching [1], [2] and tiering [3]–[6], are useful methods to improve the cost performance for input-output (IO) intensive workloads. These techniques choose areas of concentrated IO accesses and migrate them to an upper tier to extract as much performance as possible through greater use of upper tier areas.

Some IO intensive workloads include IO concentrations [7]–[9], which are aggregations of IO accesses. They occur in narrow regions of a storage volume and may persist for up to an hour. These narrow regions occupy only a few percent of the logical unit number (LUN) capacity (several gigabytes or more in many cases), are the targets of most IO accesses, and appear at unpredictable logical block addresses. After the IO concentrations cease, the affected regions often include some continued IO accesses for a while. The amount of these IO accesses depend on each workload. We also investigated these IO concentrations from the viewpoint of page-level access characteristics and found that page-level regularity varies for each workload and IO accesses are often concentrated in certain pages. Such IO concentrations are included in shared file servers, mail servers, web servers, online transaction processing (OLTP), virtual desktop infrastructure (VDI), and commercial cloud systems. Most of these systems are real operational systems accessed by many users. Therefore, it is important to improve performance for workloads including IO concentrations.

An approach called “automated tiered storage with fast memory and slow flash storage” (ATSMF) [6], [10] can effectively handle IO concentrations, because its migration algorithm finds an entire area of IO concentration and migrates it from a solid-state drive (SSD) to a non-volatile memory (NVM) if the migration will improve performance for the user. However, the ATSMF reported in [6], [10] is a prototype system for assessing its performance improvement; its implementation focuses only the functions for improving performance. As such, it uses large amounts of NVM area temporarily, because its migration algorithm always prioritizes NVM migration over eviction. A published evaluation showed that the maximum NVM consumption temporarily reached about 40% of the total volume. It is important for the ATSMF to reduce its consumption of NVM when the ATSMF is executed on a virtual machine (VM) environment. This environment includes many storage volumes, and many ATSMFs are executing in parallel to handle these storage volumes.

To dramatically reduce the consumption of NVM, we have developed four new efficient techniques for building a practical ATSMF. These techniques include changing the priority of some of the NVM migrations and evictions, reducing the NVM eviction time, and changing the eviction algorithm for the case of fully consumed NVM.

In summary, the contributions of this paper are as follows.

- We developed new eviction techniques for ATSMF to reduce the consumption of NVM capacity while maintaining the average response time:
  - A queue handling technique based on the number of IO accesses for NVM migration and eviction.
  - An eviction method that selects only write-
accessed partial regions in finished areas.
- Variable evict timing to balance the NVM consumption and average response time.
- An eviction method for full NVM consumption.

- Experimental evaluation showed that the average response times of the proposed ATSMF are almost the same as those of the published ATSMF, while the NVM consumption is drastically lower. Moreover, the NVM access ratios of the proposed ATSMF are almost the same as those of the published ATSMF, again with drastically lower NVM consumption.

This work was originally presented in a proceedings of a conference [11].

The rest of this paper is organized as follows. Section 2 explains IO concentrations and ATSMF. In Sect. 3, we describe the new techniques for ATSMF. In Sect. 4, we describe how to implement them on a Linux platform. In Sect. 5, we describe the experimental evaluation of the improved ATSMF, and in Sect. 6, we discuss the results. Section 7 provides a comparison with related work, before we summarize our work in Sect. 8.

2. IO Concentration and ATSMF

2.1 IO Concentration of Workloads

Published studies [6], [8] investigated the workloads of virtual desktop infrastructure (VDI), online transaction processing (OLTP), a shared file server, web server, and mail server and showed the features of IO concentrations (Fig. 1). It was found that regions of concentrated IOs take up only a small percentage of the storage volume and either remain for a long time or shift to neighboring regions after several minutes on average. Furthermore, this narrow region includes most IO accesses and appears at unpredictable logical block addresses (LBAs). Its space allocation and time duration are on the order of giga-bytes and minutes, respectively. To clarify IO concentration features, the published studies investigated the workloads by taking both macroscopic view and microscopic view. The macroscopic view divided the workloads into 1-GB unit, and the microscopic view divided the workloads into page unit. By taking a macroscopic view, we regard the above characteristics as IO concentration features. Figure 1 (a) shows an example of this macroscopic view.

To observe IO concentration features from a microscopic view, published studies investigated the cache hit ratios of the above-mentioned workloads. The cache was migrated to a storage area by page unit, and a replacement algorithm such as first in, first out (FIFO), least recently used (LRU), and so on was often used. In those studies, the cache hit ratios varied with the workload, but almost all the workloads had low ratios. This means that the workloads included few page-level regularities and could not be handled effectively with caching. In other words, there are many accesses to nearby pages, but there are little repeated accesses to the same pages. The published studies also investigated the page-level access locality in the regions of concentrated IOs and found that IO accesses are often concentrated in some pages of IO concentration region. Figure 1 (b) shows an example of this microscopic view.

After an IO concentration ceases, the affected regions often include some continued IO accesses for a while. Figure 2 shows an example. Series of red squares indicate the IO concentration and series of gray square indicate the affected regions. The amount of these IO accesses varies depending on each workload. We call these IO accesses “subsequent IO concentrations”.

2.2 Overview of ATSMF

We begin by defining the terms used in this section. A logical unit number (LUN) is a logical volume of a storage system and is recognized by the operating system (OS). A sub-LUN is a portion of a LUN consisting of many continuous blocks. The size of a sub-LUN is typically around 1 GB, and the size of a block is normally 512 bytes.

Automated tiered storage with fast memory and slow flash storage (ATSMF) [6], [10] is a hybrid storage system that uses both non-volatile memories (NVMs) and solid-state drives (SSDs). Figure 3 shows an overview of ATSMF. Unfortunately, the implementation of the published ATSMF focuses only the NVM migration for improving performance. In particular, the NVM eviction for saving NVM capacity is inefficient, and the maximum NVM consumption
temporarily reaches about 40% of the total volume. In the evaluation section of this paper, we call the previous ATSMF “pATSMF”.

2.2.1 Overview of NVM Migration

The ATSMF detects the IO concentration by checking the number of IO accesses for sub-LUN unit. Then, the system predicts the effect of migration by using previously monitored values of both the increase in response time during migration and the decrease in response time after migration. If the response-time decrease after migration exceeds the response-time increase during migration, the system migrates the data in the target sub-LUNs to an NVM.

ATSMF has another mechanism, which can be called “proactive migration” [12], for improving performance. It also calculates the sub-LUNs that will likely be targeted in the immediate future by using the candidate sub-LUNs for the described NVM migration and movement speed for IO concentration. Then, it migrates the data in those sub-LUNs to an NVM immediately.

2.2.2 Problems of NVM Eviction

The current NVM eviction mechanism has three problems. The NVM eviction should be executed after the subsequent IO concentration is completed. The duration of the subsequent IO concentration depends on the workload. However, ATSMF executes NVM eviction after a pre-defined time period elapses. For this reason, it cannot synchronize the completion of the subsequent IO concentration with the execution of the NVM eviction.

The NVM eviction procedure is executed by sub-LUN unit. ATSMF only cancels an eviction when the target sub-LUN does not include a dirty page. If the target sub-LUN has few dirty pages, this eviction includes many wasteful pages.

When requests for NVM migration and NVM eviction appear simultaneously, ATSMF executes NVM migration preferentially, because it prioritizes reducing the average response time over consumption of NVM, as shown in Fig.4. Therefore, it consumes a lot of NVM temporarily. The figure shows an example of NVM consumption for a VDI workload. The NVM consumption temporarily reached around 200 GB, or around 25% of VDI volume (the VDI volume was 800 GB). A published paper [6] also showed that the NVM consumption of ATSMF could temporarily reach over 40% of the volume.

3. Optimization of Eviction Algorithm in ATSMF

3.1 Overview

To reduce the consumption of NVM while maintaining the average response time, we developed new techniques for applying ATSMF.

- The first is a control technique for when requests for NVM migration and NVM eviction occur simultaneously. This technique facilitates NVM eviction by treating low-priority NVM migration and NVM eviction fairly (technique1).
- The second is a technique for reducing the NVM eviction time by moving not a whole sub-LUN but only some dirty portions of a sub-LUN (technique2).
- The third is an optimization of the NVM eviction timing (technique3).
- The fourth is a technique for evicting fully consumed NVM (technique4).

The technique1 is useful for almost all workloads. The technique2 is useful only for the workloads including many write operations. The technique3 is useful only for the workloads that the duration for subsequent IO concentration is often changed. The technique4 is useful only when the NVM consumption reaches the upper limit of the NVM capacity.

When read-dominated workload is executed, the technique1 is useful and the technique3 may be useful. When read-write mixed or write-dominated workload is executed, both the technique1 and technique2 are useful and the technique3 may be useful. The technique4 is useful for each workload only when the NVM consumption reaches the upper limit of the NVM capacity. The following sections de-
scribe detailed information of these. In addition, these techniques do not include reliable features. Proposals for these features are future work.

3.2 Queue Handling Technique Based on Number of IO Accesses

Figure 5 shows the new internal structure of the tiering manager for applying ATSMF. The tiering manager consists of a workload analyzer, sub-LUN queue, and sub-LUN mover. The workload analyzer consists of functions for handling proactive migration, the occurrence of IO concentrations (Sect. 2.2.1), and the ends of IO concentrations (Sect. 2.2.2). The sub-LUN queue consists of a high-priority NVM-up queue, a low-priority NVM-up queue, and an NVM-down queue. The tiering manager repeatedly retrieves the number of IOs per sub-LUN at a certain time interval by analyzing the IO access logs of the executing system.

To reduce the consumption of NVM while maintaining the average response time, we created a new queue handling technique for ATSMF. The occurrence of an IO concentration induces retrieval of continuous sub-LUNs, as illustrated in Fig. 6. The concentration in the figure occupies three sub-LUNs. ATSMF first retrieves the number of IOs per sub-LUN. It then sorts sub-LUN IDs by the number of IOs. Next, it combines adjacent sub-LUNs, calculates the number of IOs by adding up the numbers of combined sub-LUNs, and sorts the combined sub-LUNs by the number of IOs from highest to lowest. We call these combined sub-LUNs a “sub-LUN group.” It then retrieves sub-LUN groups by the number of IOs, calculates the total number of IOs of the retrieved sub-LUN groups, and calculates the IO rate (the total number of group IOs/the total number of all IOs). If the IO rate is more than \( M \), this step stops. The retrieved sub-LUN groups are the new IO concentrations. The sub-LUNs belonging to these groups are candidate for NVM migration. \( M \) should be set to be more than 50 because most IO concentrations include more than half of all IOs (see Sect. 2.1).

The portion of these candidate sub-LUNs may include a small number of IO accesses. Therefore, ATSMF divides these sub-LUNs into those with many IO accesses and those with few IO accesses by using the IO ratio between the sub-LUN IO accesses and all IO accesses. For the evaluation in this paper, we set the IO ratio to 5% by referring to published workload studies [7]–[9]. Then, ATSMF inserts the sub-LUNs with many IO accesses into the high-priority NVM-up queue and those with few IO accesses into the low-priority NVM-up queue. In addition, ATSMF calculates the sub-LUNs for proactive migration by using the above candidate sub-LUNs and movement speed for IO concentration and speculatively inserts the calculated sub-LUNs into the high-priority NVM-up queue. The movement speed is calculated by using the change for sub-LUNs of IO concentration ([12]).

Next is how to insert sub-LUNs into the NVM-down queue. When the IO concentration is ended, ATSMF keeps the sub-LUNs for the ended IO concentration till a certain time. When the certain time is passed, ATSMF inserts these sub-LUNs into the NVM-down queue. Section 3.4 shows how to calculate the certain time.

Following this, the sub-LUN mover retrieves from the high-priority NVM-up queue and sends a request for sub-LUN migration to the tiering driver. When the high-priority NVM-up queue becomes empty, the sub-LUN mover alternately retrieves from both the low-priority NVM-up queue and NVM-down queue and send a request for sub-LUN migration to the tiering driver. After a certain time elapses, it stops the migration and clears both the high- and low-priority NVM-up queues. This is because the sub-LUNs for the IO concentration in this interval may move to other sub-LUNs in the next interval.

When the tiering driver receives the migration request, it executes the requested sub-LUN migration immediately. The migrating sub-LUN can be accepted IO requests by using the kernel memory buffer [13].
OE and NANNRI: PROPOSAL AND EVALUATION OF IO CONCENTRATION-AWARE MECHANISMS TO IMPROVE EFFICIENCY OF HYBRID STORAGE SYSTEMS

3.3 NVM Eviction for Write-Accessed Partial Regions

When the tiering driver receives a request for sub-LUN migration from the tiering manager, it migrates the sub-LUNs between the NVM and SSD. When a migration with a direction of NVM up is completed, the tiering driver assigns a bitmap and starts to redirect IO accesses to the NVM when the address is in the migrated sub-LUN. Each bit of the bitmap corresponds to part of a region in the migrated sub-LUN.

As a result, when the tiering driver receives a write IO access to the migrated address, it redirects the IO access to the NVM and sets the appropriate bit in the bitmap to one. Then, when it receives a request for sub-LUN eviction (or migration from the NVM to the SSD) from the tiering manager, it migrates only the partial regions whose bits are in the bitmap.

When the number of partial regions reaches a threshold, ATSMF migrates not these partial regions but the whole sub-LUN because the overheads due to division of sub-LUN surpass the migration delay for a whole sub-LUN. The threshold should be measured beforehand.

By using this technique, sub-LUN evictions can finish in a shorter time when write IO accesses have high locality in a sub-LUN.

We select 1 MB as a partial region of this paper’s implementation. A bitmap size per sub-LUN is 128 bytes because the sub-LUN size is 1 GB. Each sub-LUN is assigned one bitmap entry in current implementation. Then, we prepare one thousand bitmap entries (128 KB) because the workload size of this paper’s evaluation is up to 880 GB. The bitmap entry is used only when sub-LUN is on NVM. To reduce the number of bitmap entries, we should assign bitmap entry to sub-LUN only when a sub-LUN is on NVM.

3.4 Variable Eviction Timing

Many workloads induce subsequent IO concentrations like those shown in Fig. 2. Section 2.2.2 explained that the published implementation of ATSMF migrates sub-LUNs from an NVM to an SSD when a pre-defined time elapses after the IO concentration ends. The pre-defined time is statically set by analyzing individual workloads and calculating average times. The effective time might vary, however, because the characteristics of individual workloads change with the passage of time. Here, we instead dynamically update the eviction timing by analyzing the most recent IO access logs.

The method of calculating the current eviction timing is as follows. For a fixed interval after an IO concentration has ended, ATSMF continuously checks the number of subsequent IO accesses against the sub-LUNs for that IO concentration. If the number suddenly drops, ATSMF saves the time difference between the concentration end time and the drop time. The time difference is a candidate value for a new eviction timing. As time differences accumulate, ATSMF calculates the average of these timings and updates the eviction timing from the average. The setting for the eviction timing is thus determined by analyzing each workload and calculating the averages of these times.

3.5 Eviction of Fully Consumed NVM

When ATSMF runs out of NVM capacity, a different eviction algorithm is run before the normal migration and eviction algorithms described in Sects. 2.2.1 and 2.2.2.

This eviction algorithm is similar to the LRU algorithm. When the NVM consumption reaches the upper limit of the NVM capacity, ATSMF selects the least accessed $U$ sub-LUNs and migrates them from the NVM to the SSD before applying normal NVM migration. We set $U$ to 10 in the evaluation described in this paper, because ATSMF executed less than 10 NVM migrations in one interval for the workload set used here. In general, $U$ should be set according to the appearance of IO concentrations.

4. Implementation on Linux OS

Figure 7 shows a block diagram of a prototype ATSMF system for Linux, which consists of a tiering manager, a tiering driver, and the 2M memory driver [14]. The 2M memory driver has both direct IO access mode and .map IO access mode. The direct IO access mode dramatically reduces the response time for IO requests (from 10 to 1 us on average), because it allows any driver to access the NVM area directly. However, the IO access function for the driver should be modified. On the other hand, the .map IO access mode maintains the Linux device-mapper framework. In this way, the driver, whose IO interface is a device-mapper, can access the 2M memory driver by using .map IO access mode.

The tiering manager retrieves the number of IO accesses per sub-LUN and determines whether migration
should be done by using the algorithms described in a prior paper [6]. If this judgement requires migration, the tiering manager requests that the tiering driver migrate the sub-LUNs. The tiering driver receives the IO requests and distributes them to the 2M memory driver or NVM driver by using a tiering table, which manages the sub-LUNs that have been migrated to NVM. When the memory address of an IO request hits the tiering table, the tiering driver calls the 2M memory driver with the IO request by using the direct IO access mode. On the other hand, the tiering driver requests migration via the `kcopyd` driver, which is one of the default functions of the device mapper [15], if it receives such a migration request from the tiering manager. The `kcopyd` driver accesses the 2M memory driver by using the .map IO access mode. When a sub-LUN is migrating between the NVM and the SSD, the tiering driver temporarily writes IO requests that appear in the migrating sub-LUN in its buffer area. Then, the tiering driver moves the written data from its buffer area to the SSD after the sub-LUN migration has been completed [13].

Finally, we explain the software overhead for operating the ATSMF. The ATSMF gathers IO access logs by using Linux blktrace command. The previous research [16] showed that the execution for the Linux blktrace command used tiny CPU time and write IO traffic. The executions for the tiering manager and tiering driver have little effect on user’s IO accesses because these executions are second-order interval.

5. Evaluation

5.1 Experimental System

We implemented an ATSMF system with the proposed features on a Fujitsu PRIMERGY TX300S8 server. The server included two Intel Xeon E5-2660 processors, 896 GB of memory, four 2-TB Intel P3700 SSDs, and one 800-GB Intel P3700 SSD, and it ran Linux CentOS 7.1. We also created an 8-TB volume for logical volume management (LVM) by using four 2-TB Intel P3700 SSDs. The configuration for these SSDs is just bunch of disks (JBOD).

In the experiments, the 2M memory driver used part of the 896-GB memory. ATSMF also used the 8-TB LVM volume as the SSD volume. We used dynamic random-access memory (DRAM) as a substitute for NVM, because the memory access latency of NVM is almost the same as that of DRAM. An alternative is AGIGARAM [17], which implements both DRAM and flash in a dual in-line memory module (DIMM) and copies all data from DRAM to flash when the power to the DIMM is turned off. The average response time of AGIGARAM is almost the same as that of DRAM and much less than that of an SSD. As another alternative, Intel 3D-XPoint [18] consists of NVM elements. Subramanya et al. [19] reported that its memory access latency is two to four times greater than that of DRAM.

The output of the Linux blktrace command was used as the IO access logs. The size of the sub-LUN was 1 GB, and the execution interval of the tiering manager was 24 seconds. That interval included the execution time (20 seconds) and post-processing time (4 seconds) of the block trace. We set this execution interval as 24 seconds to keep the enough migration time. From our prior experiment, a migration time was a few seconds. When the execution interval was set 24 seconds, ATSMF could migrate up to ten sub-LUNs.

5.2 Methodology

Our intention was to clarify whether the modified ATSMF system with the proposed features can handle workloads that generate many IO concentrations when it is operated in a realistic system configuration. Table 1 lists the VDI workload, DBT2 workload, and several shared fileserver workloads from MSR Cambridge [20, 21] and Samba [7, 8]. For the MSR Cambridge workloads, we chose periods that include many IO accesses, because those duration are about a week, and periods with few IO accesses often appear. The authors of [22] explained how to create the samba0 and samba1 workloads. The published papers [6, 8] showed that these workloads contained many IO concentrations. MSR Cambridge is a set of workloads from decade ago, while VDI is only 3 years old. Moreover, cloud workloads [9] includes many IO concentrations. Therefore, we think that the results of this evaluation will be useful for current computer system.

We evaluated the performance in terms of the average response time and NVM access ratio by replaying these workloads when the NVM capacity was set to 30, 60, and 90 GB. The NVM capacities are each around 10% of workload sizes listed in the table. The average response time determines the overall effect of the ATSMF system including the migration overhead. The NVM access ratio determines the effect without overhead.

We compared the performance of the modified ATSMF with that of an SSD only, caching between the NVM and SSD, and the published ATSMF by running the workloads listed in Table 1. Here, the “published ATSMF” means a combination of the published implementation of ATSMF [6] with the 2M memory driver [14]. Hereafter, we will abbreviate “ATSMF with the proposed features” as “ATSMF” and “the published ATSMF” as “pATSMF.”

Table 1 Workloads used in the experiments

| Workload | GB | WR% | Comments |
|----------|----|-----|----------|
| vdi     | 800 | 23.0 |          |
| db2    | 800 | 100 |          |
| src1/2 (MSR) | 293 | 84.2 | retrieved between 1920 and 2020(*2) |
| proj1 (MSR) | 293 | 0.9 | retrieved between 480 and 600 (*2) |
| proj2 (MSR) | 880 | 1.2 | retrieved between 230 and 550 (*2) |
| proj3 (MSR) | 880 | 42.3 | retrieved between 6760 and 6890 (*2) |
| usz (MSR) | 880 | 0.2 | retrieved between 7380 and 7660 (*2) |
| usz2 (MSR) | 569 | 0.2 | retrieved between 5970 and 6130 (*2) |
| web2 (MSR) | 182 | 0.1 | retrieved between 5870 and 5930 (*2) |
| samba0 (Samba) | 293 | 82.8 | created from src1_0 |
| samba1 (Samba) | 293 | 0.9 | created from src1_0 |

*1: Write ratios (%)

*2: Elapsed time (minutes) from first trace logs
For the caching implementation, we used Facebook FlashCache [1] in its base form or with a lazy adaptive replacement cache (FlashCache-LARC) [2]. We set the replacement algorithm as the LRU algorithm or LARC, respectively. We also loaded these FlashCache drivers with NVM and SSD. These driver options were set to the defaults (write-back mode included); the NVM handling driver was the Linux brd driver. We used the latest editions in the evaluation. When several write requests occur on a cache block at one time, these FlashCache drivers write back the cache block even though free cache blocks exist.

The NVM capacity for pATSMF is no limitation, because pATSMF assigns NVM areas when it detects an IO concentration and releases the areas in which IO concentrations have ended. The NVM consumption example shown in Fig. 4 is for pATSMF. We also set the NVM capacity for both LARC and LRU to the maximum NVM consumption of pATSMF.

Both ATSMF and pATSMF use a prediction table to decide migration. We created prediction tables with the method used in a published study [6].

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

We always delete all data stored in ATSMF on starting each evaluation.

5.3 Results of Evaluation

The average response times for both VDI and DBT-2 were more than 150 milliseconds because the SSD was overloaded. Accordingly, we will discuss these overloading workloads separately from the other workloads.

5.3.1 VDI and DBT-2

Figures 8 and 9 show the average response times and NVM access ratios of the 8-TB LVM SSD volume, which consisted of four 2-TB Intel P3700 SSDs, FlashCache-LARC, FlashCache-LRU, pATSMF, and ATSMF. The figures also indicate the NVM sizes for LARC, LRU, and pATSMF (right y-axis). The NVM consumption for pATSMF varied drastically according to the occurrences and extinctions of IO concentrations. The NVM sizes were the maximum NVM consumptions for each workload.

(1) Overview of the results

First, the VDI results for pATSMF and ATSMF indicate that the average response times were almost the same among all methods, because the SSD was overloaded and the NVM access ratios were low. The maximum NVM size for pATSMF exceeded 200 GB; however, the NVM access ratio for ATSMF reached about 72% of pATSMF’s result when the NVM capacity for ATSMF was 60 GB.

Next, the DBT-2 results for pATSMF and ATSMF show that the average response times were almost the same, because the maximum NVM size for pATSMF was 12 GB and the NVM access ratios for pATSMF and ATSMF were each almost 100%.

Finally, the results for LARC and LRU show that the NVM access ratios of LARC and LRU were much lower than those of pATSMF and ATSMF. The previous works [8], [10] showed that both the VDI and DBT-2 included the IO concentrations. The poor NVM access ratios of both LARC and LRU mean that their replacement algorithms could not handle the IO accesses of these workloads so well, and most of the IO accesses were involved in IO concentrations. Therefore, we determined that the IO concentrations included few page-level regularities. We predict that such IO concentrations will often appear near block IO accesses rather than in the same area.

(2) Effect of each technique

Both the technique1 and technique4 are effective for the VDI. From the previous paragraph, the average response time for ATSMF (60 GB) was about 72% of pATSMF’s results though the NVM capacity for ATSMF (60 GB) was about 30% of pATSMF’s NVM capacity. Studies on the contributions of technique1 and technique4 are future works.

With VDI, the effects of technique2 and technique3 are negligible because read operations occupies most of its
workload and the

5.3.2 MSR Cambridge and Samba

Figure 10 and 11 show the average response times and NVM access ratios. Moreover, Fig. 12 also shows the average response times for all workloads.

(1) Overview of the results

First, let us examine the results for src1_0, src1_1, proj2, proj4, web2, Samba0, and Samba1. The average response times were almost same for both pATSMF and ATSMF. In particular, pATSMF consumed more than 100 GB of NVM area when proj2 and proj4 were executed. Then, both ATSMF (60 GB) and ATSMF (30 GB) needed to evict the NVM area on executing these workloads. However, the average response times for ATSMF (30 GB) were almost the same as those for pATSMF because of the effect of proposed technique1 and 4. The technique1 saved the consumption for NVM area, and the technique4 could make a free NVM area when the ATSMF consumed all the NVM area. The technique2 was also effective for proj2 workload because it is read-write mixed workload. The NVM access ratios for ATSMF (30 GB) were little lower than those for pATSMF; however, the NVM access ratios were almost the same when the NVM capacity for ATSMF was more than 60 GB. In brief, ATSMF (60 GB) achieved the same NVM access ratio as pATSMF with half of pATSMF’s NVM consumption.

Next let us take a look at the results for proj1 and usr2. The average response times for pATSMF were the shortest and the NVM access ratios for pATSMF were the longest even when the NVM capacity for ATSMF was 90 GB. This is because the region of IO concentrations for these workloads often shifted to a neighboring region after several minutes. Consequently, the maximum NVM consumption for pATSMF exceeded 200 GB, and the NVM migration and NVM eviction for ATSMF often could not finish in time. As the NVM capacity of ATSMF was increased from 30 GB to 90 GB, the NVM access ratios of ATSMF also increased. This is because increasing the capacity of NVM amplifies the effectiveness of the proposed technique1 for ATSMF. Both proj1 and usr2 were read major workloads.

Next, let us examine the results for usr1. The differing results were because almost all IO accesses appeared in about a 10-GB range of the entire volume, and the workload had IO and non-IO concentrations that repeated at short intervals. Therefore, the average response times for LARC and LRU were the shortest because their NVM access ratios were almost 100%, while the average response time for pATSMF was the longest because the replacement algorithm of pATSMF could not keep the IO concentrations in the NVM. The NVM access ratio of ATSMF, however, was nearly equal to that of LARC and LRU, because of the effectiveness of the variable eviction timing described in Sect. 3.4 (technique3). ATSMF kept sub-LUNs appearing in IO and non-IO concentrations repeatedly by automatically changing their eviction timings.

Finally, let us examine the results for LARC and LRU. Except for usr1, the average response times were longer than those for pATSMF and ATSMF, and the NVM access ratios were lower than those for pATSMF and ATSMF for the same reason as stated in Sect. 5.3.1.

(2) Effect of each technique

Table 2 shows the ratios of average response time and capacity of the examined workloads over pATSMF. The column of recommended techniques is checked when the tech-
Table 2  Recommended techniques and the ratios of response time and capacity of ATSMF(30 GB) over pATSMF

| workload    | Recommended techniques | Relative value |    |
|-------------|------------------------|----------------|----|
|             | 1                      | 2              | 3  | 4  | Average | Capacity |
| src1 (MSR) | ✓                      | ✓              | ✓  | ✓  | 1.06    | 0.63     |
| src2 (MSR) | ✓                      | ✓              | ✓  | ✓  | 1.31    | 0.68     |
| proj1 (MSR)| ✓                      | ✓              | ✓  | ✓  | 1.65    | 0.16     |
| proj2 (MSR)| ✓                      | ✓              | ✓  | ✓  | 1.02    | 0.29     |
| proj3 (MSR)| ✓                      | ✓              | ✓  | ✓  | 1.16    | 0.31     |
| usr1 (MSR) | ✓                      | ✓              | ✓  | ✓  | 1.84    | 0.15     |
| web1 (MSR) | ✓                      | ✓              | ✓  | ✓  | 1.26    | 1.05     |
| samba0 (Samba)| ✓                   | ✓              | ✓  | ✓  | 1.00    | 1.00     |
| samba1 (Samba)| ✓                    | ✓              | ✓  | ✓  | 0.93    | 1.05     |

Technique is logically expected. Technique 1 is effective on most of the workloads except for usr1. On the other hand, technique 2 is suitable for the workloads in which write operation dominates. Technique 3 is effective only with the workload usr1. This is because, in this workload, the duration of the subsequent IO concentration changes frequently. Technique 4 does not show good effect on usr1, samba0, and samba1 because the capacity for ATSMF(30 GB) is bigger than that for pATSMF.

These results show that, in most of the workloads, ATSMF could reduce NVM consumption with limited increase of response time. Detailed studies on the effect of each technique will be done as future works.

5.3.3 Effectiveness of 2M Memory Driver

We evaluated the effectiveness of the 2M memory driver by executing pATSMF with the Linux brd driver. Hereafter, we will abbreviate “pATSMF with the Linux brd driver” as “brd pATSMF.” We evaluated brd pATSMF, pATSMF, ATSMF(30 GB), ATSMF(60 GB), and ATSMF(90 GB) by replaying the MSR Cambridge and Samba workloads.

Figure 13 and 14 show the average response times and NVM access ratios. Moreover, Fig. 15 also shows the average response times for all workloads. The average response times for pATSMF were always shorter than those for brd pATSMF because of the effectiveness of the 2M memory driver. The average response times for ATSMF were shorter than those for brd pATSMF whereas the NVM access ratios for ATSMF were almost the same as those for brd pATSMF, again because of the effectiveness of the 2M memory driver.

On the other hand, the average response times for ATSMF were longer than those for brd ATSMF when the NVM access ratios for ATSMF were shorter than those for brd pATSMF. Taking src1 as an example, the average response time for ATSMF(30 GB) was longer than that for brd pATSMF. However, the average response times for ATSMF(60 GB and 90 GB) were shorter than that for brd pATSMF because of the increase in ATSMF’s NVM access ratio.

6. Discussion

Intel 3D XPoint uses a new technology, and its DIMM-attached version (Intel Optane DC Persistent Memory, or DCPM) already arrived on the market (in 2019, April). Mital et al. [23] showed that write latency for DCPM was approximately 500 ns. Izraelevitz et al. [24] showed that its latency varied from 100 ns to 800 ns depending on the execution conditions. Most cases for its latency were less than 500 ns. However, the latency was nearly equal to 800 ns when its maximum bandwidth was reached. Moreover, this paper discussed how bandwidth varied with access size. In
et al. accessed the same DCPM heavily at the same time. Nanri et al. [26] showed that its latency reached about 2 us when two different applications accessed the same DCPM heavily at the same time. The 2 us latency assumed the worst case and 200 ns latency was centered around 100 ns depending on its access pattern. Moreover, its peak bandwidth can be reached when a multiple of the block size (4 cache lines = 256 byte) is used. Satoshi et al. [27] showed that its latency reached about 2 us when two different applications accessed the same DCPM heavily at the same time. Nanri et al. [27] also showed its latency for two DCPM implementations. First case was set only DCPM on a CH bus, and second case was set both DCPM and DRAM on a CH bus.

It was about 200 ns when first case, and its latency was about 800 ns when second case.

As shown in these papers [24]–[27], the relationship between the access size and the latency of DCPM is not easy to model. However, its latency varies between 100 ns and 2 us depending on how DCPM is accessed. Therefore, in this paper, 200 ns, 800 ns and 2 us are chosen as possible samples of the latency, without consideration of access sizes. The 2 us latency assumed the worst case and 200 ns latency assumed the best case when the DCPM was applied. For this estimation, we used the results of executing ATSMF(30 GB) on the MSR and Samba workloads, because their average response times were under 1 millisecond. More detailed performance model that consists sizes will be considered in the future work.

Table 3 lists the 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. For example, the case for samba0 and 2 ns DCPM latency increases the average response time from 0.0167 ms to 0.0183 ms. Its rate of increase is only 9.71%. This shows that ATSMF is effective at reducing the average response time when its NVM is Intel DCPM instead of DRAM.

| Workload | Average access size (KB) | Average response time (ms) | Memory access ratio (%) | Estimation1 (ms) | Estimation2 (ms) | Estimation3 (ms) |
|----------|--------------------------|----------------------------|-------------------------|-----------------|-----------------|-----------------|
| src1_0   | 30.0                     | 0.0191                     | 82.9                    | 0.0193          | 0.0198          | 0.0208          |
| src1_1   | 35.3                     | 0.0224                     | 67.7                    | 0.0225          | 0.0229          | 0.0238          |
| proj1    | 43.2                     | 0.0208                     | 23.1                    | 0.0209          | 0.0210          | 0.0213          |
| proj2    | 21.3                     | 0.0839                     | 64.3                    | 0.0840          | 0.0041          | 0.0852          |
| prop4    | 25.9                     | 0.1247                     | 57.8                    | 0.1248          | 0.1252          | 0.1269          |
| usr1     | 62.7                     | 0.0421                     | 86.0                    | 0.0423          | 0.0428          | 0.0438          |
| usr2     | 57.7                     | 0.2109                     | 32.2                    | 0.2170          | 0.2172          | 0.2175          |
| web2     | 60.5                     | 0.0769                     | 81.1                    | 0.0770          | 0.0774          | 0.0778          |
| samba0   | 30.0                     | 0.0167                     | 81.1                    | 0.0169          | 0.0171          | 0.0183          |
| samba1   | 35.3                     | 0.0561                     | 80.3                    | 0.0563          | 0.0567          | 0.0577          |

\[ C = A + 0.0002B/100, \quad D = A + 0.00008B/100, \quad E = A + 0.002B/100 \]

7. Related Work
First, we describe another piece of research in which cache capacities were dynamically assigned in accordance with changes in workload. Sundaresan et al. [28] proposed Multi-Cache, a multi-layer cache management system that uses a combination of cache devices of varied speed and cost, 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 maximum utilization of caches, resulting in a high hit ratio. In comparison, ATSMF dynamically detects and migrates IO concentrations on one VM (or one workload). Therefore, the total response time can be reduced if ATSMF and Multi-Cache are operated in combination.

Second, we explain hybrid storage systems consisting of memory and SSDs. Subramanya et al. [19] proposed data tiering between NVM and DRAM. Their research showed that only a small part of the memory area allocated by an application requires DRAM speed. Therefore, their proposed system judges whether each memory area requires DRAM speed by executing the X-Mem profiler. The system assigns a DRAM area in accordance with the profiler’s output.

Panruo et al. [29] proposed an algorithm-managed hybrid NVM/DRAM system for balancing across multiple dimensions (performance, energy, and resilience). Their system focuses on high-performance computing (HPC) applications and aims to leverage knowledge of numerical algorithms to direct data placement without extensive hardware changes. It requires an implementation of algorithm-directed data placement (ADDP) to support the hardware. It also includes a customized direct memory access (DMA) mechanism for bulk data movement.

Jiazin et al. [30] proposed HiNFS, a high-performance file system for non-volatile main memory (NVMM). It has two access modes to hide the system’s slow write latency. In the first mode, HiNFS uses an NVMM-aware write buffer policy to buffer lazy-persistent file writes in DRAM and lazily pass them to NVMM to hide its long write latency. In the second mode, HiNFS directly accesses NVMM for eager-persistent file writes and directly reads file data from both DRAM and NVMM, as they have similar read performance.

Jian et al. [31] presented NOVA, a file system designed to maximize performance on hybrid volatile/non-volatile memory systems while providing strong consistency guarantees. NOVA is a log-structured file system (LFS) for leveraging NVMs. To achieve its objective, it keeps complex metadata structures in DRAM to accelerate lookup operations, while keeping logs in NVM. These logs provide metadata, data, and mmap atomicity, and do not need to be contiguous because NVM supports fast, highly concurrent random access.

Third, we investigated prefetching studies, because
ATSMF includes a prefetching method. Gokul et al. [32] proposed a context-aware prefetching technique that captures application contexts, such as transaction or query, and leverages them. The technique focuses only on block-level access patterns, and as a result, it cannot retrieve the IO concentrations described in this paper.

Mingju et al. [33] proposed a technique that they call “Table-based Prefetching” (TaP), which is used to record request addresses and determine the optimum prefetch cache size. The optimum size is defined for each workload to achieve the best read-ahead hit rate. This technique also focuses on block-level sequential accesses, so it cannot retrieve IO concentrations.

Finally, we discuss other studies on building NVM block device drivers. Bankshot [34] is a caching system between NVM and disks; it also has two NVM access paths. To reduce an application’s response time, it can directly access the NVM area from a cache library of the user space when a cache hit occurs. As for a cache miss, the cache library accesses the NVM area by using the Bankshot driver in the OS space. The difference between our 2M memory driver for ATSMF and Bankshot is versatility. To reduce an application’s response time significantly in the case of a cache hit, it must be modified to use the Bankshot cache library. On the other hand, the 2M memory driver’s direct IO access mode can reduce the average response time without modifying the application, and its .map IO access mode also makes it compatible with the Linux device-mapper framework.

The Intel persistent memory block driver (PMBD) [35] is implemented with a Linux block device driver for persistent memory. This PMBD has three major features: (1) data protection, (2) data persistence, and (3) write ordering. It does not, however, support the features of our 2M memory driver.

The pmem.io web site [36] provides the Persistent Memory Development Kit (PMDK) as well as documentation on how to use it. The PMDK includes libraries to access NVM devices from the user level. Our 2M memory driver will have to be re-written with memory access code when the libraries, which can be accessed in the kernel, are released.

8. Conclusion

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 aims to reduce the average response time for IO accesses by migrating concentrated IO access areas from an SSD to an NVM. When the concentrated IO accesses finish, the system migrates these areas from the NVM back to the SSD. The published ATSMF implementation consumes much of the capacity of the NVM in temporarily migrating the concentrated IO access areas, because its algorithm executes NVM migration with high priority. Therefore, it often delays evicting areas in which IO accesses have finished to the SSD.

We developed four eviction techniques for ATSMF to reduce the consumption of NVM while maintaining the application’s average response time. The first is a queue handling technique based on the number of IO accesses for NVM migration and eviction. The second is an eviction method that selects only write-accessed partial regions in the areas in which IO concentrations have finished. The third is a technique for variable eviction timing to balance the NVM consumption and average response time. The fourth is an eviction method for the case of fully consumed NVM.

We evaluated this improved ATSMF in experiments measuring the average response time and NVM access ratio while running various workloads, including a VDI, in-memory database, and several shared file system workloads. We found that the improved ATSMF outperformed a plain SSD, caching between the NVM and SSD, and the published ATSMF on almost all the workloads. Furthermore, the NVM consumption was much lower than that of the published ATSMF on most workloads.

For VDI workload, the average response time for ATSMF was almost as same as that for pATSMF even if the NVM consumption for ATSMF was about three times lower than that for pATSMF. For MSR proj2 and 4 workloads, the average response time for ATSMF were a little higher than that for pATSMF even if the NVM consumption for ATSMF was about three times lower than that for pATSMF.

9. Future Work

We will execute quantitative evaluation for ATSMF using a real NVM (ex. intel DCPM). Moreover, we will clarify the contribution ratio for each technique of ATSMF and correlation between NVM size and the effect of ATSMF.

References

[1] “Facebook FLASHCACHE.” https://github.com/facebook/flashcache.
[2] “Facebook FLASHCACHE with LARC.” https://github.com/seth-hg/flashcache.
[3] “EMC White Paper, EMC FAST VP for Unified Storage Systems A Detailed Review,” March 2011.
[4] “FUJITSU Automated Storage Tiering,” http://storage-system.fujitsu.com/jp/products/diskarray/feature/003/.
[5] F. Chen, D. Koufaty, and X. Zhang, “Hystor: Making the best use of solid state drives in high performance storage systems,” Proc. ICS, pp.22–32, May 2011.
[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,” Proc. 3rd International Workshop on Computer Systems and Architectures (CSA’17), Aomori, Japan, Nov. 2017.
[7] K. Oe, T. Honda, and M. Kawaba, “Samba workload analysis and consideration for hybrid storage system,” IPSJ SIGOS, Tokyo, Japan (in Japanese), Dec. 2012.
[8] K. Oe, T. Nanri, and K. Okamura, “Analysis of storage workloads of input-output access locality and designing of hybrid storage system,” Proc. 5th IIAI International Congress on Advanced Applied Informatics, July 2016.
[9] K. Oe, K. Oghihara, and T. Honda, “Analysis of commercial cloud workload and study on how to apply cache methods,” IEICE CPSY,
Kumamoto, Japan (SWoPP 2018), CPSY2018-14, Aug. 2018.

[10] K. Oe, M. Sato, and T. Nauri, “ATSMF: Automated tiered storage with fast memory and slow flash storage to improve response time with concentrated input-output (IO) workloads,” IEICE Trans. Inf. & Syst. vol.E101-D, vol.12, Dec. 2018.

[11] K. Oe and T. Nauri, “Hybrid storage system to achieve efficient use of fast memory area,” The Seventh Int. Symp. Computing and Networking (CANDAR’19), Nagasaki, Japan, Nov. 2019.

[12] K. Oe, T. Nauri, and K. Okamura, “On-the-fly automated storage tiering with caching and both proactive and observational migration,” Proc. 3rd International Workshop on Computer Systems and Architectures (CSA’15), Sapporo, Japan, Dec. 2015.

[13] K. Oe, S. Iwata, T. Nauri, and K. Okamura, “Proposal and evaluation for hybrid storage system which can choose and migrate IO concentration areas to SSD automatically when their duration and number of IO access are long enough to surpass the migration overhead,” IPSJ Trans. Advanced Computing Systems (No.53) (in Japanese), March 2016.

[14] K. Oe and T. Nauri, “Non-volatile memory driver for applying automated tiered storage with fast memory and slow flash storage,” Proc. 6rd International Workshop on Computer Systems and Architectures (CSA’18), Takayama, Japan, Nov. 2018.

[15] “Device mapper.” https://en.wikipedia.org/wiki/Device_mapper.

[16] K. Oe, K. Oghiha, Y. Noguchi, and T. Ozawa, “Proposal for a hierarchical storage system that can move a spike to high-speed storage in real time,” IPSJ Trans. Advanced Computing Systems (No.40) (in Japanese), Oct. 2012.

[17] “AGIGARAM DDR4 NVDIMM.” http://www.agigatech.com/.

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

[19] S.R. Dulloor, A. Roy, Z. Zhao, N. Sundaram, N. Satish, R. Sankaran, J. Jackson, and K. Schwan, “Data tiering in heterogeneous memory systems,” Proc. 11th ACM European conference on Computer systems (EuroSys 2016), pp.1–16, April 2016.

[20] D. Narayanana, A. Donnelly, and A. Rowstron, “Write Off-Loading: Practical Power Management for Enterprise Storage,” Proc. 6th USENIX Conf. on File and Storage Tech, Feb. 2008.

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

[22] K. Oe, S. Iwata, T. Honda, M. Kawaba, and K. Okamura, “On-the-fly automated storage tiering (OTF-AST),” Proc. The Third Asian Conference on Information Systems – Special Session on Information Storage (ACIS-IS 2014), Nha Trang, Viet Nam, Dec. 2014.

[23] M. Mittal and J.S. Vetter, “A survey of software techniques for using non-volatile memories for storage and main memory systems,” IEEE Trans. Parallel Distrib. Syst., vol.27, no.5, pp.1537–1550, May 2016.

[24] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y.J. Soh, Z. Wang, Y. Xu, S.R. Dulloor, J. Zhao, and S. Swanson, “Basic performance measurements of the Intel Optane DC Persistent Memory Module.” https://arxiv.org/abs/1903.05714.

[25] A. van Renen, L. Vogel, V. Leis, T. Neumann, and A. Kemper, “Persistent memory I/O primitives,” Proc. 15th International Workshop on Data Management on New Hardware (DaMoN’19), Amsterdam, Netherlands, July 2019.

[26] S. Imamura and E. Yoshida, “The analysis of inter-process interference on a hybrid memory system,” HPC Asia2020, Jan. 17, 2020, Fukuoka, Japan, Jan. 2020.

[27] T. Nauri, K. Oe, E. Yoshida, H. Otsuji, and E. Hayashi, “Preliminary study of message queuing system based on RDMA over non-volatile DIMM,” Academic eXchange for Information Environment and Strategy (AXIES) 2020, Dec. 9, 2020, Japan, Dec. 2020.

[28] S. Rajasekaran, S. Duan, W. Zhang, and T. Wood, “Multi-cache: Dynamic, efficient partitioning for multi-tier caches in consolidated VM environments,” Proc. Cloud Engineering (IC2E), 2016 IEEE International Conference, Berlin, Germany, April 2016.

[29] P. Wu, D. Li, Z. Chen, J.S. Vetter, and S. Mittal, “Algorithm-directed data placement in explicitly managed non-volatile memory,” Proc. 25th ACM Int. Symp. High Performance Parallel and Distributed Computing (HPDC 2016), pp.141–152, May 2016.

[30] J. Ou, J. Shu, and Y. Lu, “A high performance file system for non-volatile main memory,” Proc. 11th ACM European conference on Computer systems (EuroSys 2016), April 2016.

[31] J. Xu and S. Swanson, “NOVA: A log-structured file system for hybrid volatile/non-volatile main memories,” Proc. 14th USENIX Conf. on File and Storage Tech (FAST’16), Feb. 2016.

[32] G. Soundararajan, M. Mihaiescu, and C. Amza, “Context-aware prefetching at the storage server,” the 2008 USENIX Annual Technical Conference (ATC 2008), June 2008.

[33] M. Li, E. Varki, S. Bhatia, and A. Merchant, “TaP: Table-based prefetching for storage caches,” Proc. 6th USENIX Conf. on File and Storage Tech, Feb. 2008.

[34] M.S. Bhaskaran, J. Xu, and S. Swanson, “Bankshot: Caching slow storage in fast non-volatile memory,” Interactions of NVM/Flash with Operating-Systems and Workloads (INFLOW’13), Nov. 2013.

[35] “Intel persistent memory block driver.” https://github.com/NVMe/pmbd/pmbd.

[36] “Persistent memory programming.” https://pmem.io.

Kazuichi Oe graduated from School of Computer Science from Kyushu University, Japan in 1988 and worked at FUJITSU LABORATORIES LTD. He got Ph.D. Degree from Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University, Japan in 2016. He retired from FUJITSU LABORATORIES LTD. on June 30, 2020 and became a member of National Institute of Informatics (NII) from July 1, 2020. He is member of IEICE and ACM SIGOPS. Current his research interests are system software for hybrid memory system including storage system, and storage workload analysis. Moreover, he is researching activation of FlashStorage and non-volatile memory.

Takeshi Nauri received the B.E. and M.E. from Kyushu University. After joining the Computer Center of the university, he received the Ph.D. in Computer Science in 2000. Since 2001, he has been an Associate Professor at the Research Institute for Information Technology of Kyushu University. His major interest is the middleware for largescale parallel computing.