ATSMF: Automated Tiered Storage with Fast Memory and Slow Flash Storage to Improve Response Time with Concentrated Input-Output (IO) Workloads

Kazuichi OE†a, Member, Mitsuru SATO†b, and Takeshi NANRI††c, Nonmembers

SUMMARY The response times of solid state drives (SSDs) have decreased dramatically due to the growing use of non-volatile memory express (NVMe) devices. Such devices have response times of less than 100 micro seconds on average. The response times of all-flash-array (AFA) systems have also decreased dramatically due to the growing use of non-volatile memory express (NVMe) devices. Such devices have response times of less than 100 micro seconds on average. The response times of all-flash-array (AFA) systems have also decreased dramatically due to the growing use of NVMe SSDs. Storage interface response times have been reduced due to the recent introduction of the hyper-converged infrastructure (HCI) [2], in which server systems are equipped with directly-attached storage and appear as a purely software-defined environment. However, some applications, including virtual desktop infrastructure (VDI) [3], in-memory database systems [4], and certain data analysis systems, require storage systems with even shorter response times.

Various vendors have recently developed dual inline memory module (DIMM) attached non-volatile memory (NVM) [5], and the response times of some byte-accessible NVM devices are much less than those of SSDs. An NVM device, that combines dynamic random-access memory (DRAM) and flash storage into a single DIMM, has the same response time as DRAM because the NVM flash storage uses a back-up device only when power is lost [6]. Intel plans to release 3D Xpoint DIMMs in the second half of 2018 [7]. Their write latency will be about 500 ns, and their read latency will be about 125 ns [8]. Therefore, their response time will be more than that of DRAM and much less than those of SSDs. They will cost much more than SSDs, and their capacity will be restricted by the number of DIMM slots.

On-the-fly automated storage tiering (OTF-AST) [9], [10] is a hybrid storage system located between SSDs and HDDs. The OTF-AST replacement algorithm identifies input-output (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 changes of HDD latency are very big when migrating between SSDs and HDDs, so the changes in average response time cannot be predicted exactly. The algorithm for identifying IO concentration uses a static parameter for each workload that is determined by analyzing the workload in advance.

To design an efficient hybrid storage system midway between NVM devices and SSDs, we investigated the workloads of applications requiring storage systems with shorter response times. We found that their workloads tended to contain many IO concentrations, which are aggregations of IO accesses. They access narrow regions (a small percentage) of the storage volume and continue for up to about an hour. They either target one region for a long time or shift...
to a neighboring region of the volume after several minutes on average. These narrow regions are the target of most IO accesses and appear at unpredictable logical block addresses (LBAs). The OTF-AST replacement algorithm can efficiently handle these IO concentrations for the OTF-AST with SSD and HDD. However, this algorithm is inefficient on executing the OTF-AST with NVM and SSD because changes of both NVM and SSD latencies are almost stable on migration and we can predict the changes exactly by monitoring these latencies.

We also investigated the page-level regularity of the workloads and found that they included few regularities. The page-level regularity means IO access repeatability per page unit. Therefore, caching [11]–[13] cannot reduce the response times of such applications because the caching-replacement algorithm uses page-level regularity.

To drastically reduce the response times of applications requiring storage systems with shorter response times, we developed an automated tiered storage system called “automated tiered storage with fast memory and slow flash storage” (ATSMF) in which the memory component is non-volatile memory (Fig.1). The assumed environment is a server with non-volatile memory and directly attached SSDs, with the user application executed on the server as this reduces the average response time. The system predicts the effect of migration by using the previously monitored values of the increase in response time during migration and the decrease in response time after migration. These values are consistent for each type of workload if the system is built using both non-volatile memory and SSDs. In particular, the system continually monitors the increase in response time during migration, the decrease in response time after migration, and the duration of migration and uses the data to predict the average increase and the average decrease in the times and duration of each IO concentration. Moreover, it continually monitors the occurrence of IO concentrations. When it identifies a concentration, it first predicts its remaining duration and 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, it migrates the data in the target regions to NVM. It also identifies regions that will likely be targeted in the immediate future and migrates the data in those regions to NVM immediately.

The key contributions of this work are as follows.

- We clarified the features of VDI and online transaction processing (OLTP) workloads. In particular, by taking a macroscopic view, we demonstrated that these workloads contain many IO concentrations. By taking a microscopic view, we found that these concentrations include few page-level regularities. Shared file servers, web servers, and mail servers also have such features.
- We developed the ATSMF system, which reduces the IO access response time to less than that of flash storage only and of caching. Its migration algorithm predicts the effect of migration by using the previously monitored values of the increase in response time during migration and the decrease in response time after migration and using the duration predicted using the statistical information for previous IO concentrations. OTF-AST cannot use this approach because the changes in HDD latency are very big when migrating between SSDs and HDDs, so the changes cannot be predicted exactly. The differences between the work reported here (journal paper) and we previously reported [14] (latest conference paper) are as follows.
  - We have now evaluated our ATSMF system against Facebook FlashCache with lazy adaptive replacement (FlashCache-LARC) and OTF-AST to better clarify its effectiveness.
  - We also evaluated it when the upper limit of the NVM size was 10% of the total volume capacity.
- We demonstrated that our ATSMF system reduces the IO access response time to less than that of flash storage only and of caching in most cases.
- We also demonstrated that the system’s NVM access ratios are substantially higher than those of caching in most cases.

2. Flash Storage and Memory

2.1 Flash Storage

Let us begin by defining the term “flash storage” as used in this paper. Flash storage means some type of commercial AFA [15] or NVMe SSD [16]. The average response times of such storage devices are typically less than
100 micro-seconds. However, the times are much more than micro-second order. We evaluated an Intel P3700 SSD [16]. Its average response time was mostly between 40 and 80 micro-seconds: around 40 micro-seconds on read/predominant workloads and around 80 micro-seconds on read/write mixed workloads. For example, Sudan et al. [17] reported response times of SSD: 47 micro-seconds on read and 15 micro-seconds for write for SSDs. Therefore, we assume that the SSD response time is more than 10 micro-seconds and less than 100 micro-seconds in this paper. Hereafter, we call a flash memory device or flash storage simply an SSD.

### 2.2 Memory

The focus here is a hybrid storage system with NVM and SSDs. The NVM is used as the upper layer of the SSDs. The NVM response time must be at least one order of magnitude less than that of the SSDs in order to reduce the average response time. Since we assume here that the SSD response time is between 10 and 100 micro-seconds, the NVM response time must be less than 1 micro-seconds.

There are two solutions besides DRAM that satisfy this NVM performance requirement. One is the AGIGARAM nonvolatile RAM system [6], which implements both DRAM and flash in a DIMM and copies all data from the DRAM to the flash when power to the DIMM is turned off. The average response time of AGIGARAM is almost the same as that of DRAM and is much less than that of an SSD.

The other is Intel’s 3D-Xpoint NVM currently under development [18]. Dulloor et al. [8] reported that its memory access latency is two to four times greater than that of DRAM. We estimate that its memory access latency is less than that of an SSD because the column access strobe latency (read response time) of DRAM is about 15 nanoseconds [19].

In this study, we used DRAM as a substitute for NVM because both AGIGARAM and 3D-Xpoint are difficult to obtain. We refer to NVM as simply memory.

### 3. IO Concentrations in Storage Workloads

#### 3.1 Definition of Terms

Let us begin by defining the terms used in this paper. A logical unit number (LUN) is a logical volume of a storage system that is recognized by the operating system. A sub-LUN is a portion of an 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. Logical block addresses are assigned by block and are used to retrieve data, which are stored in terms of LUNs. Moreover, workload means the number of IO accesses that are targeted at a block unit during short intervals.

#### 3.2 Overview of Investigated Workloads

We investigated the workloads on a VDI server, a DBT-2 benchmark, a shared file server, a web server, and a mail server in order to clarify the features of IO concentrations.

Oe et al. analyzed a VDI server workload and a Microsoft Exchange (MS EX) server workload [20]. The VDI server was operated as a company server by the research department, accessed by approximately 300 users, and controlled using the CITRIX Xen-Desktop desktop virtualization software platform [21]. The VDI consisted of eight volumes, each with a capacity of 8 TB. Therefore, the data capacity for the VDI workloads was 64 TB.

The MS EX server was also operated as a company server, and had a data capacity of 112.5 TB. A Samba workload [22] was accessed by approximately 3000 users. Its data capacity was 4.4 TB.

Narayanan et al. analyzed the Microsoft Research (MSR) Cambridge workload [23], [24]. They input real-world logs of accesses to file servers, print servers, and Web/SQL servers, which included block IO trace data obtained from 13 servers, 36 workloads (volumes), and 179 disks on the MSR servers over the course of approximately one week. For our study, we used the reported results for these workloads, except for the DBT-2 workload.

To investigate the OLTP workload, we executed the DBT-2 benchmark [25], which is an OLTP transactional performance test. It simulates a wholesale parts supplier where several workers access a database, update customer information, and check on parts inventories. It is a fair usage implementation of the Transaction Processing Performance Council’s (TPC’s) TPC-C benchmark specification [26]. We retrieved and studied trace logs obtained when we executed the DBT-2 benchmark. The DBT-2 parameters were the database size (50 GB), the number of connections (10), the database size (50 GB), and the total volume size (800 GB).

Figure 2 shows the DBT-2 heatmap. The IO concentrations targeted about 5% of the storage volume and accounted for more than 80% of all IO accesses. The write ratio was 100% and almost all IO accesses targeted certain LBAs. When the values of the DBT-2 parameters were changed to enable investigation of other conditions, different LBAs were targeted. These results indicate that the LBAs targeted by IO concentrations change during OLTP operation.

#### 3.3 How Workloads were Investigated

We aimed to clarify both the spatial and temporal locality of concentrated IO accesses in workloads by taking both macroscopic and microscopic views.

To investigate the workloads from a macroscopic view, we divided the LUNs for the above-mentioned workloads into 1-GB sub-LUNs, counted the number of IO accesses to each sub-LUN during one-minute intervals, and calcu-
3.4 Features of IO Concentrations

The features of the IO concentrations are summarized in Table 1. “IO access ratio” shows the ratio against the total number of IO accesses. “LUN capacity ratio” shows the ratio of the LUN capacity targeted in IO concentrations against the total LUN capacity. “Average duration” shows the average duration of the IO concentrations. “Percentage of unpredictable LBAs” shows the ratio of unpredictable start LBAs against the IO concentration. A start LBA is the first LBA targeted in an IO concentration. To judge whether the start LBAs were unpredictable, we summed the number of occurrences of IO concentrations for each start LBA for the workload and calculated the percentage of occurrences for each LBA against the total number of occurrences. We judged that an LBA was unpredictable if the percentage was less than 5% [20]. “Write ratio” shows the write ratio against the total number of accesses in an IO concentration. “LRU” and “ARC” show the cache hit ratios when LRU and ARC were used as the sim-ideal replacement algorithm.

### Table 1 Features of IO concentrations

| Workload     | Samba | proj1 | proj2 | proj4 | src1_0 | src1_1 | usr1 | usr2 | web2 | MS EX | DBT-2 | VDI |
|--------------|-------|-------|-------|-------|--------|--------|------|------|------|-------|-------|-----|
| IO access ratio (%) | 58    | 77    | 84    | 89    | 95     | 82     | 77   | 82   | 92   | 75    | 80    | 81  |
| LUN capacity ratio (%) | 0.14  | 0.45  | 0.46  | 0.46  | 0.53   | 0.53   | 0.53 | 0.53 | 0.53 | 0.53  | 0.53  | 0.53 |
| Average duration (minutes) | 30    | 60    | 54    | 40    | 87     | 66     | 86   | 32   | 58   | 20    | —     | 3   |
| Percentage of unpredictable LBAs (%) | 71    | 75    | 88    | 75    | 90     | 89     | 66   | 84   | 100  | 92    | —     | 80  |
| Write ratios (%) | 70    | 11    | 12    | 2     | 44     | 5      | 9    | 19   | 1    | 50    | 100   | 20  |
| LRU (sim-ideal,16GB) (%) | 57.34 | 0.43  | 7.41  | 7.62  | 7.11   | 31.15  | 98.45| 0.09 | 0.35 | —     | 77.67 | 12.71 |
| LRU (sim-ideal,64GB) (%) | 57.41 | 0.63  | 11.33 | 11.53 | 11.62  | 41.11  | 98.45| 0.11 | 0.38 | —     | 77.67 | 12.71 |
| ARC (sim-ideal,16GB) (%) | 54.35 | 0.54  | 7.47  | 10.80 | 7.15   | 28.56  | 98.45| 0.11 | 0.35 | —     | 77.67 | 12.71 |
| ARC (sim-ideal,64GB) (%) | 57.41 | 0.67  | 11.33 | 11.56 | 11.62  | 41.11  | 98.45| 0.11 | 0.38 | —     | 77.67 | 12.71 |

3.4.1 Macroscopic View

We explain the threshold for each workload before discussing the results. For the VDI, DBT-2, Samba, and MSR Cambridge workloads, we defined the threshold as 10 IOPS. We used the proj1, proj2, proj4, src1_0, src1_1, usr1, usr2, and web2 MSR Cambridge workloads because they had several consecutive hours of concentrated IO requests intermittently targeting sub-LUNs. All, except web2, were file-server workloads. For the MS EX workloads, we defined the threshold as 1 IOPS.

As shown in Table 1, the accesses in IO concentrations accounted for more than 58% of the IO accesses and targeted regions in less than 5% of the LUN capacity. These concentrations lasted from 3 to 87 minutes, and at least 66% of them targeted unpredictable LBAs. The write ratios in the table show there were read-predominant, read-write mixed, and write-predominant workloads. Figure 3 (a) shows an example macroscopic view.

The average duration and percentage of unpredictable LBAs for DBT-2 are not shown because DBT-2 is not an operational workload log but a benchmark workload log. This means that its values change with the parameter settings.
3.4.2 Microscopic View

Both the LRU and ARC cache hit ratios were calculated by sim-ideal. They varied with the workload, but all workloads, except for the Samba, usr1, and DBT-2 workloads, had low cache hit ratios. Moreover, these ratios were almost the same even when the cache size was increased from 16 to 64 GB. This means that these workloads included few page-level regularities and thus could not be handled effectively with caching. In addition, all of the workloads included IO concentrations, which were detected when taking a macroscopic view. Figure 3(b) shows an example microscopic view.

The results for MS EX are not shown because we were unable to obtain the MS EX trace logs.

4. Automated Tiered Storage with Fast Memory and Slow Flash Storage (ATSMF) System

4.1 Overview

The results presented in Sect. 3 led us to develop the automated tiered storage with fast memory and slow flash storage (ATSMF) system for handling IO concentrations.

As illustrated in Fig. 1 and briefly described in the Introduction, it divides a storage volume into sub-LUNs and monitors the volume for IO accesses at the sub-LUN level. At the same time, it monitors the increase in response time during migration (magnitude), decrease in response time after migration (magnitude), and duration of migration per sub-LUN (magnitude). When it identifies an IO concentration, it predicts its remaining duration and calculates the time needed to migrate the data in the targeted regions by using the migration duration per sub-LUN and the number of sub-LUNs targeted in the IO concentration. The system also calculates and compares the response-time increase during migration and response-time decrease after migration by using the predicted remaining duration, calculated duration of migration, and monitored values. It immediately migrates the data in the targeted sub-LUNs if the response-time decrease after migration exceeds the response-time increase during migration. The system also identifies sub-LUNs likely to be targeted in the immediate future and migrates the data in those sub-LUNs to memory immediately by using a previously reported technique [28]. When an IO concentration ends, the system migrates the data in the targeted sub-LUNs to an SSD.

We previously developed an on-the-fly automated storage tiering (OTF-AST) system consisting of SSDs and HDDs. Its replacement algorithm also handles IO concentrations, but it differs from the ATSMF one in how replacement is determined. Because changes in HDD latency are very big when migrating between SSDs and HDDs and because the changes cannot be predicted exactly, the OTF-AST algorithm judges whether to initiate migration by using a static parameter under the assumption that the HDD was the latest device accessed. The static parameter for each workload is determined by analyzing the workload in advance. In contrast, the ATSMF replacement algorithm predicts the effect of migration by using both the memory and SSD response times and the statistics for IO concentration duration. This is feasible because changes in memory and SSD latency are consistent during migration between memory and SSDs.

4.2 Judgement of Migration from an SSD to Memory

The ATSMF system counts the number of IO accesses per sub-LUN and uses the number for each short interval to detect IO concentrations. Figure 4 illustrates the detection of an IO concentration. This concentration accessed data in three sub-LUNs. The technique used to detect an IO concentration was previously reported [9] and [10] and is described below.

The ATSMF system first retrieves the number of IOs per sub-LUN and uses the number for each sub-LUN to detect IO concentrations. Figure 4 illustrates the detection of an IO concentration. This concentration accessed data in three sub-LUNs. The technique used to detect an IO concentration was previously reported [9] and [10] and is described below.

The ATSMF system 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 the number 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 for 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 $M$, the system stops. The retrieved sub-LUN groups correspond to a new IO concentration. For the results presented in Sect. 3, most of the IO concentrations included more than half of all IOs. Therefore, we set $M$ to be more than 50. The $M$ for the results shown in Fig. 4 was 60.

When the system detects an IO concentration, it monitors the concentration, as illustrated in Fig. 5. An IO concentration targeting sub-LUNs 4, 5, and 6 is detected in the

![Detection of IO concentration](image)
The ATSMF system predicts the remaining duration of an IO concentration by using data in a prediction table, like the one shown in Table 2. From this table, the remaining duration for a concentration continuing for three intervals is 43.90 seconds. As the number of intervals increases, the remaining duration also increases because the long-continued IO concentrations are remained.

Figure 6 shows the number of IO concentrations by duration. We created this figure by using the execution logs of an ATSMF prototype system that stored the duration of each IO concentration. The majority of IO concentrations continued for short periods. We created Table 2 by using collected workload data. First, we collected and saved the durations of each IO concentration. Then, we retrieved the IO concentrations lasting more than N intervals and calculated the average duration by using the collected data. Finally, we obtained the remaining duration by subtracting N from the average duration. For example, the remaining duration is 43.90 when N is 3.

To update the prediction table automatically for an actual implementation, we need to implement a feature for updating the table by using the durations saved at regular intervals, e.g., half a day. The system can then create an appropriate table for each workload and PC environment.

ATSMF continuously monitors the migration durations and uses them to calculate the average duration ($AveMgDuration$). It retrieves the number of sub-LUNs (subLUNs) targeted by IO concentrations, and obtains the predicted remaining duration ($Pduration$) by using the prediction table (Table 2). The duration during migration ($Xduration$) and that in memory ($Yduration$) are calculated as illustrated in Fig. 7.

$$Xduration = AveMgDuration \times subLUNs \quad (1)$$
$$Yduration = Pduration - Xduration \quad (2)$$

To determine whether to migrate the data in a region in an SSD targeted by an IO concentration, ATSMF continually collects the average response time of the SSD ($Avef$), that of the SSD when the migration is executed ($AvefWithMg$), and that of the memory ($Avm$). $Y$ is the total response-time decrease after migration, and $X$ is the total response-time increase during migration. The system calculates $X$ and $Y$ using the following expressions.

$$X = (AvefWithMg - Avef) \times Xduration \quad (3)$$
$$Y = (Avef - Avm) \times Yduration \quad (4)$$

If $Y$ exceeds $X$, the system migrates the data in the region targeted by the IO concentration. If $Y$ does not exceed
X, it puts the migration on hold. If the IO concentration continues into the next interval, the system repeats the determination process.

4.3 Judgment of Migrating from Memory to an SSD

The ATSMF system migrates the data in sub-LUNs from memory to an SSD when $Z$ seconds have elapsed since the IO concentration ended. This is because any subsequent IO concentrations targeting the same sub-LUNs and lasting for a short duration often appear within $Z$ seconds.

The $Z$ parameter is set as follows. A higher value may increase the ATSMF’s memory access ratios but also increase its memory consumption. The value should thus be set using the frequency information for subsequent IO concentrations. After an IO concentration has ended, the number of subsequent IO accesses is continuously checked against the sub-LUNs for the IO concentration that just ended for a fixed interval. If the number suddenly falls, ATSMF migrates the targeted sub-LUNs from memory to an SSD and sets $Z$ to the time difference between the concentration end time and the fall time. These time setting for $Z$ are determined by analyzing each workload and calculating the average times.

5. Evaluation

5.1 Experimental System

The experimental system (Fig. 8) was implemented on a Fujitsu PRIMERGY TX300S8 server. It included two Intel Xeon E5-2660, 256-GB memories, and an 800-GB Intel P3700 SSD and had a Linux CentOS 7.1 operating system. The migration algorithm described in Sects. 4.2 and 4.3 was separated from the system’s kernel module (device driver) and implemented as a user-level program to facilitate implementation. The prototype ATSMF system was composed of a tiering manager and tiering driver. The tiering manager retrieved the IO access logs by using the Linux blktrace command. The retrieved logs were used to determine whether the data in the sub-LUNs targeted by an IO concentration should be migrated to memory. The tiering manager also matched the IO concentration to the sub-LUN numbers and initiated migration of the data in the sub-LUNs to the tiering driver. The tiering driver then migrated the data in the sub-LUNs. It also received IO requests and distributed them to the brd driver or NVMe driver by using a tiering table. The tiering table managed data-migrated sub-LUNs. The brd driver serves as a block driver of the Linux kernel for accessing memory.

The overhead for the Linux block trace command should be considered because it is executed when retrieving IO access logs. A previous study found that its execution requires little CPU time and few write IO accesses [29]. We thus judged that its overhead can be neglected.

The sub-LUN size was 1 GB, and the execution interval of the tiering manager was 24 seconds. The interval included the execution time of the block trace (20 seconds) and post-processing of the block trace (4 seconds). The execution interval is determined as follows. A shorter execution interval improves system responsiveness but may reduce the number of detected IO concentrations because some concentrations may contain short periods during which there are few IO accesses. Moreover, a shorter execution interval may restrict the number of sub-LUN migrations because short intervals with few IO accesses can break up a continuous IO concentration. Therefore, a moderately short execution interval should be set for each workload.

The default setting of the $Z$ parameter (Sects. 4.3) was 74 seconds ($=24$ seconds x 3 intervals). We used the MSR src1_0 workload for calculating the $Z$ parameter.

5.2 Methodology

Our intention was to clarify whether ATSMF can handle workloads that generate many IO concentrations. We evaluated its performance in terms of the average response time and memory access ratio. The average response time determines the overall effect of the system including the migration overhead. The memory access ratio determines this effect without overhead. We compared the performance of ATSMF with that of an SSD only, caching, and on-the-fly automated storage tiering (OTF-AST) [9], [10] between memory and SSDs by replaying the VDI workload, DBT-2 workload, and several shared fileserver workloads described in Table 3. The VDI workload consisted of the eight 8-TB-volume logs. We used the log with the most IO accesses, and extracted 800 GB of data from the log because the size of the Intel P3700 SSD was 800 GB. The selected 800-GB area was the most concentrated area of the volume. We used the DBT-2 trace logs described in Sect. 3.2. The shared fileserver workloads were the MSR Cambridge workloads [23], [24] together with the Samba workloads [20], [22]. We selected periods with many IO accesses because the durations of the MSR Cambridge workloads were about a week and included periods with few IO
Table 3 Workloads used in experiments

| Workload  | GB | Comment |
|-----------|----|---------|
| VDI       | 800|         |
| DBT-2 (OLTP) | 800|         |
| src1_0 (MSR) | 293| retrieved between 1920 and 2020 * |
| src1_1 (MSR) | 293| retrieved between 480 and 600 * |
| proj1 (MSR) | 800| retrieved between 230 and 350 * |
| proj2 (MSR) | 800| retrieved between 6760 and 6890 * |
| proj4 (MSR) | 236| retrieved between 7390 and 7600 * |
| us1 (MSR)  | 800| retrieved between 2090 and 2350 * |
| us2 (MSR)  | 569| retrieved between 5970 and 6130 * |
| web2 (MSR) | 182| retrieved between 5870 and 5930 * |
| src1_0+ (Samba) | 293| created from src1_0 |
| src1_1+ (Samba) | 293| created from src1_1 |

*: Elapsed time (minutes) from first trace logs

Table 4 Average response time (milli-seconds) and memory access ratio (%)

| Workload  | SSD (A) | LARC (B) | B/A | LRU (C) | C/A | OTF-AST (D) | D/A | ATSMF (E) | E/A | MMC* (GB) | MMC** (%) |
|-----------|---------|----------|-----|---------|-----|-------------|-----|-----------|-----|-----------|-----------|
| VDI       | 12.60   | 20.27 (0.8) | 1.61 | 23.15 (0.8) | 1.84 | 12.4 (3.0) | 0.98 | 13.04 (55.1) | 1.04 | 221         | 27.6       |
| DBT2      | 0.084   | 0.079 (77.0) | 0.94 | 0.447 (58.8) | 5.32 | 0.040 (65.0) | 0.48 | 0.034 (89.3) | 0.41 | 93          | 11.6       |
| MSR src1_0 | 0.070   | 0.289 (8.2) | 4.13 | 0.615 (8.8) | 8.79 | 0.030 (75.1) | 0.43 | 0.025 (82.7) | 0.36 | 43          | 14.7       |
| MSR src1_1 | 0.042   | 0.247 (28.4) | 5.88 | 0.319 (29.8) | 7.60 | 0.033 (55.0) | 0.79 | 0.025 (86.4) | 0.64 | 74          | 25.3       |
| MSR proj1 | 0.032   | 0.350 (0.9) | 0.43 | 0.198 (11.6) | 7.94 | 0.050 (46.3) | 1.09 | 0.031 (82.8) | 0.58 | 144         | 18.0       |
| MSR proj2 | 0.046   | 0.354 (11.2) | 10.24 | 1.310 (1.0) | 31.19 | 0.099 (12.4) | 2.36 | 0.052 (74.7) | 1.23 | 220         | 27.5       |
| MSR proj4 | 0.053   | 0.975 (10.2) | 18.40 | 0.746 (10.5) | 14.08 | 0.039 (43.2) | 0.74 | 0.031 (82.8) | 0.58 | 144         | 18.0       |
| MSK us1   | 0.004   | 0.017 (98.4) | 0.43 | 0.018 (98.3) | 0.45 | 0.001 (29.8) | 1.28 | 0.030 (86.6) | 0.65 | 184         | 78.9       |
| MSK us1** | 0.040   | 0.017 (98.4) | 0.43 | 0.018 (98.3) | 0.45 | 0.051 (29.8) | 1.28 | 0.030 (85.4) | 0.38 | 49          | 6.1        |
| MSK us2   | 0.041   | 0.473 (0.1) | 11.54 | 0.51 (0.2) | 12.44 | 0.065 (13.7) | 1.59 | 0.037 (84.3) | 0.90 | 231         | 40.6       |
| MSR web2  | 0.062   | 0.243 (0.0) | 3.92 | 0.256 (0.0) | 4.13 | 0.057 (32.1) | 0.92 | 0.043 (81.7) | 0.69 | 75          | 41.2       |
| Samba src1_0+ | 0.065 | 0.312 (41.2) | 4.80 | 0.597 (41.2) | 9.18 | 0.025 (69.9) | 0.38 | 0.021 (88.4) | 0.32 | 54          | 18.4       |
| Samba src1_1+ | 0.048 | 0.128 (43.7) | 2.67 | 0.312 (31.7) | 6.5 | 0.020 (75.4) | 0.42 | 0.019 (87.9) | 0.40 | 46          | 15.7       |

MMC*: Maximum memory consumption of ATSMF [capacity (GB) and percentage of that shown in Table 3]
Number in parentheses of B, C, D, and E: memory access ratios (%), us1**: Z parameter (Sects. 4.3) set to 480 seconds.

Accesses. We could not replay the Samba workloads [20] since the workload logs are in aggregates of 1-GB 1-min units; they can only be replayed as statistical data. We therefore created trace logs for the Samba workloads by using portions of the MSR Cambridge src1_0 and src1_1 workloads, as described in a previous report [9].

We used both the Facebook FlashCache [11] and Facebook FlashCache drivers with lazy adaptive replacement (FlashCache-LARC) [30] for the caching implementation. We set the replacement algorithm of FlashCache to LRU and set the replacement algorithm of FlashCache-LARC to LARC. Simulation has shown that the hit ratio of LARC is at least 16.5% higher than that of ARC [31]. We set these drivers to have a write-back mode and used the Linux btrfs driver as a block-access memory driver. The other options of these drivers were set to the defaults setting. We downloaded their latest editions and used them for our evaluation.

OTF-AST is a hybrid storage system located between SSDs and HDDs. Its replacement algorithm identifies IO concentrations and migrates them from an HDD to an SSD. However, its migration algorithm is based on handling HDDs, so it may have been slow in our evaluation when OTF-AST was executed with memory and SSD. This is because it restricts the number of migrations per unit time to prevent sharp increases in HDD latency. Moreover, it requires a large safety margin because it cannot predict sharp increase values.

We created prediction tables of the type shown in Table 2 by analyzing the trace log of each workload listed in Table 3. It has been shown that the distributions of the IO-concentration durations were almost uniform in all portions of every workload [20]. While the prediction tables should be created using trace logs collected immediately before the ATSMF begins to operate, previous results [20] show that prediction tables created using the trace logs of each workload listed in Table 3 are almost the same as those created using trace logs collected immediately before the ATSMF begins to operate.

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

5.3 Evaluation Results

Table 4 shows both the average response time and memory access ratio of the Intel P3700 SSD volume, FlashCache LARC, FlashCache LRU, OTF-AST, and ATSMF. It also shows the maximum memory consumption (MMC) of ATSMF. ATSMF assigns sub-LUNs from memory when it determines that an IO concentration has appeared, as described in Sect. 4.2, and it releases those sub-LUNs when Z seconds have elapsed after the concentration has ended, as described in Sect. 4.3.

Figure 9 shows memory consumption for the VDI workload as a function of elapsed time in seconds. The memory capacities for both LARC and LRU were set to the maximum amount of memory, as it was for ATSMF. The FlashCache memory capacity was 221 GB during execution of the VDI workload.

5.3.1 VDI

The average response time of ATSMF (13.04 ms) for the
VDI workload was 50% less than that of LARC and 77% less than that of LRU, and 55.1% of the IO accesses were to memory. However, the average response times of SSD and ATSMF were almost the same because the average response time of ATSMF’s SSD part was drastically higher than that of SSD only (Intel P3700 only).

Figure 10 shows the cumulative distribution functions (CDFs) of the VDI response time. The more to the upper-left the curve, the shorter the response time. The average response time of ATSMF’s SSD part was 28.9 milli second while the average response time of SSD only was 12.6 milli second. This is because the VDI workload overloaded the Intel P3700 SSD, and the migration load of ATSMF increased this overload.

The average response time of OTF-AST (12.4 ms) was a bit less than that of ATSMF and almost equal to that of SSD because OTF-AST’s memory access ratio was only 3.0%. OTF-AST mainly used its SSD part since it could not migrate the targeted data.

About 1% of the IO accesses were to memory when both FlashCache LARC and LRU were used. The poor memory access ratios of both FlashCache LARC and LRU mean that their replacement algorithms cannot handle the IO accesses in the VDI workload well. Table 1 shows that the IO concentrations in the VDI workload targeted about 1% of the storage volume and included about 80% of all IO accesses. This means that its IO concentrations included few page-level regularities. This finding indicates that its IO concentrations will often appear not in the same but in near pages.

5.3.2 DBT-2

The average response time of ATSMF (0.034 milli second) for the DBT-2 workload was dramatically less than those of LARC, LRU, and SSD only. The memory access ratio of ATSMF (89.3%) was also higher than those of LARC and LRU.

Figure 11 shows the CDFs. ATSMF (ALL) is on the ATSMF (MEMORY) side between ATSMF (SSD) and ATSMF (MEMORY) because of the high memory access ratio (89.3%). The average response time of LRU was 5.32 times that of SSD only though its memory access ratio was 58.8%. This may be due to an implementation problem when FlashCache LRU was installed between memory and the SSD.

The average response time of ATSMF was also less than that of OTF-AST (0.040 milli second), and its memory access ratio was higher than that of OTF-AST (65.0%). The memory access ratio of OTF-AST was lower because the replacement judgment of OTF-AST is slower and less accurate than that of ATSMF when OTF-AST manages storage devices without HDDs.

5.3.3 MSR and Samba

The average response times of ATSMF for the MSR and Samba workloads were the lowest among those of SSD only, LARC, LRU, and OTF-AST and not for the proj1 and usr1 workloads. The memory access ratios of ATSMF were the highest among those of LARC, LRU, and OTF-AST and not for the usr1 workload.

The memory access ratios of OTF-AST were lower than that of ATSMF and the OTF-AST’s average response times were higher than ATSMF’s average response times due to the lower memory access ratio of OTF-AST, as described in Sect. 5.3.2.

The memory access ratios of both LARC and LRU were much lower than that of ATSMF except for the usr1
workload. This is because their replacement algorithms cannot handle the IO accesses in the MSR and Samba workloads well. Moreover, their average response times were much higher than that of ATSMF except for the src1 workload and also much higher than that of SSD only even though their memory access ratios were 25% or better (src1-1, src1-0+, and src1-1+). An implementation problem may have occurred when both FlashCache LARC and FlashCache LRU were installed between memory and the SSD.

Figure 12 shows the CDFs of the proj1 response time. ATSMF (ALL) is in the middle between ATSMF (SSD) and ATSMF (MEMORY). ATSMF (SSD) is more to the right than P3700 only because of the high-frequency migration of ATSMF. Therefore, the average response time of ATSMF was higher than that of SSD only (P3700 only). To reduce the average response time for the proj1 workload, we should suspend ATSMF migrations when the SSD part of ATSMF is busy. Observational migration [32] periodically sends small amounts of data and checks the response time to determine whether a device is busy. When the device is busy, it puts migrations on hold. Therefore, the average response times of ATSMF might be less than those of the Intel P3700 if we were to replace the current technique with observational migration.

Finally, the average response time of ATSMF for the src1 workload was the highest among SSD only, LARC, LRU, and OTF-AST, and its memory access ratio was the lowest among all methods. This is because almost all IO accesses targeted an approximately 10-GB range of the entire volume, and the src1 workload had IO and non-IO concentrations that repeated at short intervals. Therefore, the ATSMF replacement algorithm could not keep all the data for the IO concentrations in memory.

To keep the data targeted by the IO concentration in memory, we changed the Z parameter (Sects. 4.3) from 74 seconds (default value) to 480 seconds. The results shown for “MSR usr1***” in Table 4 are those obtained for 480 seconds. The average response time of ATSMF (0.015 ms) was the lowest among SSD only (0.040 ms), LARC (0.017 ms), LRU (0.018 ms), and OTF-AST (0.051 ms). The memory access ratio of ATSMF (85.4%) was near that of LARC (98.4%) and LRU (98.3%). This means that the Z parameter should be set in accordance with the features of the workload.

6. Discussion

6.1 Memory Consumption of ATSMF

ATSMF assigns sub-LUNs from memory when it determines that an IO concentration has appeared, as described in Sect. 4.2, and it releases those sub-LUNs when Z seconds have elapsed after the concentration has ended, as described in Sect. 4.3. In other words, the memory consumption of ATSMF varies with the number of IO concentrations in the workload. Since our current implementation prioritizes memory migration, SSD migration may not be executed immediately if there are many pending migrations.

We thus investigated the performance of ATSMF when the upper limit of the memory size was 10% of total volume capacity. We used all workloads except for the src1 because their MMCs were more than 10% (See Table 4). We compared the performance of ATSMF with those of SSD only and FlashCache LARC by replaying those workloads. We also set the memory capacity of LARC to 10% of total volume capacity.

The ATSMF algorithm for excluding sub-LUNs when memory consumption reaches the upper limit of memory capacity is similar to that of LRU. When the memory consumption reaches the upper limit, the least accessed U sub-LUNs are selected and migrated from memory to the SSD before migration judgment from SSD to memory described in Sect. 4.2 is performed. We set U to 10.

As shown in Table 5, the average response times of ATSMF were less than those of LARC except for the DBT-2 workload. However, these average response times were more than those of SSD only except for the src1-0 and src1-0+ workloads because the restriction on memory size increased the number of migrations and reduced the memory access ratio.

Figure 13 shows the CDFs of the proj1 response time...
for a memory size of 80 GB. Compared with the SSD curve in Fig. 12 (0.171 ms), the average response time of the SSD curve was much higher (0.406 ms). Moreover, compared with the memory access ratio in Table 4 (74.7%), the memory access ratio of ATSMF for MSR proj1 in Table 5 was much lower (45.6%). To reduce the average response time of ATSMF, we need to speed up migration between memory and the SSD with minimal overhead. This should increase the memory access ratio and reduce the average response time.

7. Related work

Previous research has also considered dynamic assignment of cache capacity in accordance with the change in workload. Rajasekaran et al. [33] proposed Multi-Cache, a multilayer cache management system that uses a combination of cache devices with various 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 cache, resulting in a high hit ratio. In comparison, ATSMF dynamically detects and migrates the data in regions targeted by IO concentrations in one VM (or one workload). Therefore, we can reduce the total response time if we operate both ATSMF and Multi-Cache in combination.

Other research has investigated hybrid storage systems consisting of memory and SSDs. Dulloor et al. [8] proposed data tiering between NVM and DRAM. They found that a small part of the memory area allocated by applications needs DRAM speed. Their proposed system judges whether each memory area needs DRAM speed by executing the X-Mem profiler. The system assigns a DRAM area in accordance with the output of the profiler.

Wu et al. [34] proposed an algorithm-managed hybrid NVM/DRAM system to balance the load across multiple dimensions (performance, energy, and resilience). Their proposed system is focused on high-performance computing applications and is aimed at leveraging the knowledge of numerical algorithms for direct data placement without extensive hardware changes. Their system implements algorithm-directed data placement to support the hardware. It includes a customized DMA mechanism for bulk data movement.

Ou et al. [35] proposed HiNFS, a high performance file system for non-volatile main memory (NVMM). HiNFS has two access modes that hide a system’s low write latency. In the first, HiNFS uses an NVMM-aware write buffer policy to buffer the lazy-persistent file writes in DRAM and pass them to NVMM lazily to hide the long write latency of NVMM. In the second, HiNFS directly accesses NVMM for eager-persistent file writes and directly reads file data from both DRAM and NVMM as they have a similar read performance.

Xu and Swanson [36] 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 for leveraging non-volatile memories. It keeps complex metadata structures in DRAM to accelerate lookup operations and keeps its 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.

Among state-of-the-art commercialized technique, Intel’s Optane SSD is substantially faster than SSD using flash memory because it uses Intel’s 3D-Xpoint NVM cell. Intel’s Memory Drive Technology (IMDT) [37] extends memory capacity beyond DRAM limitations by combining DRAM and Intel’s Optane SSD. This memory extension technique improves cost-performance as does ATSMF, which is a hybrid storage technique. Since IMDT is a memory extension technique while ATSMF is a storage layer technique, we cannot compare their performance.

Last, we also investigated prefetching studies that have been made because ATSMF uses a prefetching, and there have been several studies on prefetching Soundararajan et al. [38] reported a context-aware prefetching technique that captures application contexts, such as a transaction or query, and leverages them. However, the technique focuses on block-level access patterns and cannot detect IO concentrations like ATSMF does.

Li et al. [39] proposed a technique they call “Table-based Prefetching” (TaP), which is used to record request addresses and determine the optimum prefetch cache size for TaP. The optimum size is defined for each workload in order to achieve the best read-ahead hit rate. However, this technique also focuses on block-level sequential access and cannot detect IO concentrations.

8. Conclusion

The response time of solid state drives (SSDs) has decreased dramatically with the growing use of non-volatile memory express (NVMe) devices, which have response times of less than 100 micro seconds on average. However, there are ap-
lications, particularly, virtual desktop infrastructure and in-memory database systems, that require storage systems with even shorter response times.

Various vendors developed non-volatile memory (NVM) devices, and some of them have response times much less than those of SSDs. An NVM device combining DRAM and flash storage in a single DIMM has the same response time as DRAM because the NVM’s flash storage uses back-up device only when power is lost. Intel’s 3D Xpoint DIMMs, scheduled for release in the second half of 2018, will have a write latency of about 500 ns and a read latency of about 125 ns. Their response time will thus be more than that of DRAM and much less than that of SSDs. However, they will cost much more than SSDs, and their capacity will be restricted by the number of DIMM slots.

Our automated tiered storage with fast memory and slow flash storage (ATSMF) system addresses this problem by greatly reducing the response time for various types of workload without substantially increasing the cost.

It identifies IO concentrations in workloads that last for moderately long periods and migrates the data in the targeted regions to NVM. It also identifies regions that will likely be targeted in the immediate future and migrates the data in those regions to NVM immediately.

ATSMF monitors the increase in response time during migration and change in response time after migration. If it detects an IO concentration, it first predicts its duration, calculates and compares the response-time increase during migration and the response-time decrease after migration, and migrates the data in the targeted regions if the expected response-time decrease after migration exceeds the expected response-time increase during migration. When an IO concentration has disappeared, the area used to store the targeted data is released and the data is migrated to an SSD.

We evaluated ATSMF experimentally by replaying virtual desktop infrastructure, in-memory database, and several shared filesystem workloads. We measured the average response times and NVM access ratios. ATSMF outperformed both flash storage only and caching between NVM and flash storage for most of the workloads.

References

[1] “NVM Express.” http://www.nvme.org/.
[2] “Hyper-converged infrastructure.” https://en.wikipedia.org/wiki/Hyper-converged_infrastructure.
[3] “Desktop virtualization.” https://en.wikipedia.org/wiki/Desktop_virtualization.
[4] “List of in-memory databases.” https://en.wikipedia.org/wiki/List_of_in-memory_databases.
[5] K. Oe, T. Nani, and K. Okamura, “Feasibility study for building hybrid storage system consisting of non-volatile DIMM and SSD,” Proc. of 4rd International Workshop on Computer Systems and Architectures (CSA’16), pp.454–457, Hiroshima, Japan, Dec. 2016.
[6] “AGIGARAM DDR4 NVDIMM.” http://www.agigatech.com/.
[7] “Intel to Launch 3D XPoint DIMMs in 2H 2018.” https://www.anandtech.com/show/12041/intel-to-launch-3d-xpoint-dimms-in-2h-2018.
[8] 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.
[9] K. Oe, S. Iwata, T. Honda, M. Kawaba, and K. Okamura, “On-The-Fly Automated Storage Tiering (OTF-AST),” Proc. Third Asian Conference on Information Systems – Special Session on Information Storage (ACIS-IS 2014), Nha Trang, Viet Nam, pp.537–544, Dec. 2014.
[10] K. Oe, S. Iwata, T. Nani, 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 Transactions on Advanced Computing Systems (no.53) (in Japanese), vol.9, no.1, pp.1–16, March 2016.
[11] “Facebook FLASHCACHE.” https://github.com/facebook/flashcache.
[12] “Fusion IO Directcache.” http://www.fusionio.com/data-sheets/directcache/.
[13] N. Megiddo and D.S. Modha, “ARC: A SELF-TUNING, LOW OVERHEAD REPLACEMENT CACHE,” Proc. 2th USENIX conference on File and Storage Technologies (FAST2003), pp.115–130, March 2003.
[14] K. Oe, M. Sato, and T. Nani, “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), pp.311–317, Aomori, Japan, Nov. 2017.
[15] “DELL EMC DSSD D5.” https://www.emc.com/collateral/datasheet/h14828-ds-dssd-d5-rack-scal-fape-ad.pdf.
[16] “Intel SSD DC P3700 Series.” http://www.intel.com/content/www/us/en/solid-state-drives/ssd-dc-p3700-spec.html.
[17] K. Sadan, A. Badam, and D. Nellans, “NAND-Flash: Fast Storage or Slow Memory?,” Proc. 3rd Annual Non-Volatile Memories Workshop 2012 (NVMW 2012), San Diego, USA, March 2012.
[18] “intel 3D-Xpoint.” https://en.wikipedia.org/wiki/3D_XPoint.
[19] “DDR4 SDRAM.” https://www.3d-xpoint.com.
[20] K. Oe, T. Nani, 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, pp.971–974, July 2016.
[21] “CITRIX XenDesktop.” https://www.citrix.com/products/xendesktop/overview.html.
[22] K. Oe, T. Honda, and M. Kawaba, “Samba workload analysis and consideration for hybrid storage system,” IPSJ SIGOS, Tokyo, Japan (in Japanese), vol.2012-OS-123, no.1, Dec. 2012.
[23] D. Narayanan, A. Donnelly, and A. Rowstron, “Write Off-Loading: Practical Power Management for Enterprise Storage,” Proc. 6th USENIX Conf. on File and Storage Tech, vol.4, no.3, pp.1–23, Feb. 2008.
[24] “SNIA trace data MSR Cambridge.” http://iotta.snia.org/traces/388.
[25] “Database Test Suite.” http://osdl.sourceforge.net/.
[26] “TPC Benchmark C.” http://www.tpc.org/tpcc/.
[27] “sim-ideal.” https://github.com/arh/sim-ideal.
[28] K. Oe, T. Nani, 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), pp.371–377, Sapporo, Japan, Dec. 2015.
[29] K. Oe, K. Ogihara, Y. Noguchi, and T. Ozawa, “Proposal for a Hierarchical Storage System that can Move a Spike to High-speed Storage in Real Time,” IPSJ Transactions on Advanced Computing Systems (no.40) (in Japanese), vol.5, no.5, pp.118–127, Oct. 2012.
[30] “Facebook FLASHCACHE with LARC.” https://github.com/seth-hg/flashcache.
[31] S. Huang, Q. Wei, J. Chen, C. Chen, and D. Feng, “Improving Flash-based Disk Cache with Lazy Adaptive Replacement,” Proc. IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST 2013), pp.1–10, Long Beach, CA, USA, May 2013.
[32] K. Oe, T. Nani, and K. Okamura, “On-The-Fly Automated Storage
Tiering with Proactive and Observational Migration,” IEICE CPSY, Oita, Japan (SWoPP 2015), CPSY2015-30, Aug. 2015.

[33] 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, pp.182–191, Berlin, Germany, April 2016.

[34] 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 International Symposium on High Performance Parallel and Distributed Computing (HPDC 2016), pp.141–152, May 2016.

[35] 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), pp.1–16, April 2016.

[36] 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), pp.323–338, Feb. 2016.

[37] “Intel Optane SSD DC P4800X Series with Intel Memory Drive Technology,” https://www.intel.co.jp/content/dam/www/public/us/en/documents/product-briefs/oprane-ssd-dc-p4800x-mdt-brief.pdf.

[38] G. Soundararajan, M. Mhiailescu, and C. Amza, “Context-Aware Prefetching at the Storage Server,” the 2008 USENIX Annual Technical Conference (ATC 2008), pp.377–390, June 2008.

[39] 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.