An Efficient Hybrid I/O Caching Architecture Using Heterogeneous SSDs

Reza Salkhordeh, Mostafa Hadizadeh, and Hossein Asadi

Abstract—Storage subsystem is considered as the performance bottleneck of computer systems in data-intensive applications. *Solid-State Drives* (SSDs) are emerging storage devices which unlike *Hard Disk Drives* (HDDs), do not have mechanical parts and therefore, have superior performance compared to HDDs. Due to the high cost of SSDs, *entirely* replacing HDDs with SSDs is not economically justified. Additionally, SSDs can endure a limited number of writes before failing. To mitigate the shortcomings of SSDs while taking advantage of their high performance, SSD caching is practiced in both academia and industry. Previously proposed caching architectures have *only* focused on either performance or endurance and neglected to address *both* parameters in suggested architectures. Moreover, the cost, reliability, and power consumption of such architectures is not evaluated. This paper proposes a hybrid I/O caching architecture that while offers higher performance than previous studies, it also improves power consumption with a similar budget. The proposed architecture uses DRAM, *Read-Optimized SSD* (RO-SSD), and *Write-Optimized SSD* (WO-SSD) in a three-level cache hierarchy and tries to efficiently redirect read requests to either DRAM or RO-SSD while sending writes to WO-SSD. To provide high reliability, dirty pages are written to at least two devices which removes any single point of failure. The power consumption is also managed by reducing the number of accesses issued to SSDs. The proposed architecture reconfigures itself between performance- and endurance-optimized policies based on the workload characteristics to maintain an effective tradeoff between performance and endurance. We have implemented the proposed architecture on a server equipped with industrial SSDs and HDDs. The experimental results show that as compared to state-of-the-art studies, the proposed architecture improves performance and power consumption by an average of 8% and 28%, respectively, and reduces the cost by 5% while increasing the endurance cost by 4.7% and negligible reliability penalty.

Index Terms—Solid-State Drives, I/O Caching, Performance, Data Storage Systems.

1 Introduction

Hard Disk Drives (HDDs) are traditional storage devices that are commonly used in storage systems due to their low cost and high capacity. The performance gap between HDDs and other components of computer systems has significantly increased in the recent years. This is due to HDDs have mechanical parts which puts an upper limit on their performance. To compensate the low performance of HDDs, storage system designers proposed several hybrid architectures consists of HDDs and faster storage devices such as *Solid-State Drives* (SSDs).

SSDs are non-mechanical storage devices that offer higher performance in random workloads and asymmetric read/write performance as compared to HDDs. SSD manufacturers design and produce several types of SSDs with different performance and cost levels to match a wide range of user requirements. The relative performance and costs of SSDs compared to HDDs and *Dynamic Random Access Memory* (DRAM) is shown in Fig. 1. Due to the relatively very high price of SSDs, replacing the entire disk array in data storage systems with SSDs is not practical in *Big Data* era. In addition, SSDs have restricted lifetime due to the limited number of reliable writes which can be committed to SSDs. The power outage can also cause data loss in SSDs as reported in [1]. Although SSDs have such shortcomings, they have received a significant attention from both academic and industry and many architectures for I/O stack based on SSDs have been proposed in recent years.

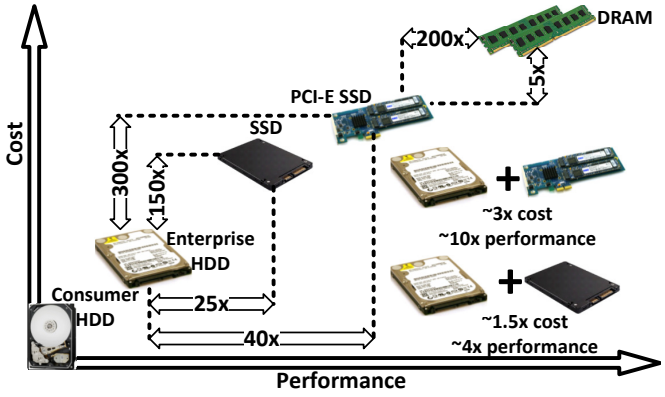

Fig. 1: Storage Devices Characteristics

One promising application of SSDs emerged in recent years is to alleviate low performance of HDDs with minimal cost overhead by using SSDs as a caching layer for HDD-based storage systems. The main focus of previous studies in caching architecture is on improving

• *Reza Salkhordeh, Mostafa Hadizadeh, and Hossein Asadi (corresponding author) are with the Department of Computer Engineering, Sharif University of Technology, Emails: salkhordeh@ce.sharif.edu, mhadizadeh@ce.sharif.edu, and asadi@sharif.edu .*

performance and/or endurance. Three main approaches have been proposed in previous studies to this end: a) prioritizing request types such as filesystem metadata, random, and read requests, b) optimizing baseline algorithms, and c) modifying the traditional single-level cache. As shown in Fig. 1, caching architectures offer various performance levels with significantly different costs depending on their choice of SSDs. Previous studies neglected to consider the effect of choosing different SSDs on the performance. Additionally, they are mostly focused on the performance, while other major system metrics such as power consumption, endurance, and reliability also need to be considered in the caching architectures. To our knowledge, none of previous studies considered all the mentioned parameters, simultaneously.

This paper proposes a *Three-level I/O Cache Architecture* (TICA) which aims to improve the performance and power consumption of SSD-based I/O caching while having minimal impact on the endurance. TICA employs *Read-Optimized SSD* (RO-SSD), *Write-Optimized SSD* (WO-SSD), and DRAM as *three levels* of I/O cache. Employing heterogeneous SSDs decreases the probability of correlated failure of SSDs since such SSDs either belong to different brands or have different internal data structures/algorithms. To enhance the performance of both read and write requests, TICA is configured in the write-back mode such that it buffers the most frequent read and write intensive data pages in DRAM to reduce the number of writes committed to SSDs and to increase their lifespan. In order to guarantee the reliability of write requests, all write requests are committed to both DRAM and WO-SSD before the write is acknowledged to the user. Dirty data pages in DRAM are asynchronously flushed to RO-SSD to free up allocated DRAM space for future requests.

In order to efficiently optimize the proposed architecture for read- and write-intensive applications, we offer two cache policies where evicted data pages from DRAM can be either moved to SSDs or removed from the cache. The first policy, called *Write to Endurance Disk* (TICA-WED), improves performance since the next access to the data page will be supplied by SSDs instead of HDD. The shortcoming of TICA-WED is reducing SSDs lifetime due to the extra writes for moving the data page from DRAM to SSD. To alleviate such shortcoming, the second policy, called *Endurance Friendly* (TICA-EF), can be employed. In TICA-EF, performance is slightly decreased while the lifetime of SSDs is significantly extended. To select between TICA-WED and TICA-EF, we propose a state-machine which analyzes the running workload and dynamically selects the most effective policy for TICA. With such data flow, TICA improves performance and power consumption of I/O cache while having negligible endurance overhead and no cost and reliability impact.

To verify the efficiency of TICA, we have first extracted I/O traces from a server equipped with two Intel Xeon, 32GB memory, and 2x SSD 512GB. I/O traces are extensively analyzed and characterized to help optimize parameters of TICA towards higher performance. Experimental setup consists of a rackmount server equipped with a RO-SSD, a WO-SSD, and 128GB memory. The benchmarking suites for experiments consist of over 15 traces from Microsoft research traces [2], HammerDB [3], and FileBench [4]. Experimental results show that despite reducing the cost by 5%, as compared to the state-of-the-art architectures, TICA enhances performance and power consumption, on average, by 8% (and up to 45%), and by 28% (and up to 70%), respectively, while having only 4.7% endurance overhead and negligible reliability penalty.

To our knowledge, we make the following contributions:

- By carefully analyzing state-of-the-art SSDs available in the market and their characteristics, we select two types of SSDs to design a low-cost hybrid caching architecture capable of providing high performance in both read- and write-intensive applications.
- We propose a three-level caching architecture, called TICA, employing DRAM, RO-SSD, and WO-SSD to improve performance and power consumption of storage systems while having negligible endurance penalty.
- TICA reduces the correlated failure rate of SSDs in I/O caching architectures by using heterogeneous SSDs while the performance is not limited by the slower SSD, unlike traditional heterogeneous architectures.
- To balance performance and endurance, we propose *Endurance-Friendly* (TICA-EF) and *Write to Endurance Disk* (TICA-WED) policies for TICA, where the first policy prioritizes endurance and the second policy tries to further improve performance.
- We also propose a state-machine model to select one of TICA-EF or TICA-WED policies based on the workload characteristics. Such model can identify the most effective policy for TICA with negligible overhead while running I/O intensive applications.
- We have implemented TICA on a physical server equipped with enterprise SSDs and HDDs and conducted an extensive set of experiments to accurately evaluate TICA, considering all optimization and buffering in storage devices and *Operating System* (OS).

The remainder of this paper is organized as follows. Previous studies are discussed in Section 2. The motivation for this work is presented in Section 3. Section 4 introduces the proposed caching architecture. In Section 5, the experimental setup and results are presented. Finally, Sec. 6 concludes the paper.

2 Previous Studies

Previous studies in SSD-based I/O caching can be categorized into three groups: a) prioritizing various re-

quest types based on storage device characteristics, b) optimizing baseline eviction and promotion policies, and c) proposing multi-level caching architectures. The first category tries to characterize performance of HDDs and SSDs. Based on the characterization, request types which have higher performance gap between HDDs and SSDs are prioritized to be buffered. A comprehensive study on the workload characteristics and request types is conducted in [5]. The different response time of SSDs on read and write requests is considered in [6] to prioritize data pages. The locality of data pages is employed in RPAC [7] to improve both performance and endurance of caching architecture. ReCA [8] tries to characterize several requests and workload types and selects suitable data pages for caching. Filesystem metadata is one of the primary request types which is shown to be very efficient for caching [9], [10]. OODT [9] considers randomness and frequency of accesses to prioritize the data pages. To reduce the migrations between HDD and SSD, [11] considers the dirty state of the data pages in memory buffers. ECI-Cache [12] prioritizes data pages based on the request type (read/write) in addition to the reuse distance. The optimization of the previous studies in this category is mostly orthogonal to TICA and can be employed in the eviction/promotion policies of the SSDs in TICA.

The studies in the second category try to optimize the eviction policy of caching architectures. To prevent cache pollution, *Lazy Adaptive Replacement Cache* (LARC) [13] is suggested which promotes data pages to cache on the second access to the data page. This technique, however, cannot perform in a timely fashion when workload is not stable. mARC [14] tries to select the more suitable option from ARC and LARC based on the workload characteristics. In [15], various management policies based on ARC for DRAM-SSD caching architectures are compared. A more general approach to prevent repetitive replacement of data pages in SSDs is suggested in [16] which provides buffered data pages a more chance to be accessed again and therefore stay in the cache. S-RAC [17] characterizes workloads into six groups. Based on the benefit of buffering requests in each category, it decides which data pages are best suited for caching. S-RAC tries to reduce the number of writes in SSD to improve its lifetime with minimal impact on the cache performance. H-ARC [18] partitions the cache space into *clean* and *dirty* sections where each section is maintained by ARC algorithm. D-ARC [19] also tries to improve ARC by prioritizing the data pages based on the clean/dirty state. Me-CLOCK [20] tries to reduce the memory overhead of SSD caching architectures by using bloom filter. RIPQ [21] suggests a *segmented-LRU* caching algorithm, which aggregates small random writes and also places user data with the same priority close to each other. In WEC [22], write-efficient data pages are kept in cache for longer periods to reduce the writes due to the cache replacements. This category of previous studies is also orthogonal to TICA and such policies can be employed jointly with TICA to further improve performance and/or endurance.

Among previous studies that try to enhance performance of I/O caching by utilizing multi-level cache hierarchies, LLAMA [23] employs a DRAM-SSD architecture for designing an *Application Programming Interface* (API) suitable for database management systems. FASC [24] suggests a DRAM-SSD buffer cache, which tries to reduce the cost of evictions from buffer cache as well as write overheads on the SSD. Employing exclusive DRAM-SSD caching is investigated in [15] which shows the impact of such technique on improving SSD endurance. In [25], separate promotion/demotion policies for DRAM/SSD cache levels are replaced with a unified promotion/demotion policy to improve both performance and endurance. uCache [26] also employs a DRAM-SSD architecture and tries to reduce the number of writes in the SSD due to the read misses. In case of a power loss, all dirty data pages in DRAM will be lost which significantly reduces the reliability of uCache. Additionally, no redundant device is employed and both DRAM and SSD are single points of failure. MDIS [27] uses a combination of DRAM, SSD, and NVM to improve performance of I/O caching. Although performance and energy consumption are improved in MDIS, the cost and reliability have not been taken into account. Graphene [28] suggests a DRAM-SSD architecture to improve performance of graph computing for large graphs. SSD caching is also suggested in distributed and *High Performance Computing* (HPC) environments [29], [30], [31], [32], [33].

Optimizing SSDs for key-value store is discussed in previous studies. DIDACache [34] allows the key-value SSD cache to directly manage the internal SSD structure to improve both performance and endurance. WiscKey [35] separates key and value storage in SSD to improve random lookups and database loading. Deduplication and compression can also be employed to extend the SSDs lifetime [36], [37], [38]. Modifying the existing interface between OS and SSDs is also suggested in previous studies to design efficient caching architectures [39], [40]. In [40], a new interface for SSDs is designed, which does not allow overwriting of data pages, to reduce the size of the required DRAM in SSD and also to improve performance. F2FS [41] employs an append-only logging approach to reduce the need for overwriting data pages in SSDs. KAML [42] suggests a customized interface for SSDs for storing and retrieval of key-value data. FStream [43] employs *streamID* to hint *Flash Translation Layer* (FTL) on lifetime of user data so that FTL places the data pages with the same lifetime on a physical block. Optimizing SSD caching architectures by leveraging information from SSDs internal data structures such as FTL is also suggested in previous studies [44], [45]. FLIN [46] provides a fair scheduler for SSDs servicing multiple applications simultaneously. A scheduler to maximize the efficiency of parallelism inside of the SSD is also proposed in [47]. SHRD [48] tries to optimize the physical placement of data pages in SSD to reduce the

TABLE 1: Power Consumption, Cost, Reliability, and Endurance of Storage Devices in the Cache Architectures

Device	MTTF (h)	$/GB	Writes/GB	Read/Write/Idle Power (w)
DRAM	4M	7.875	∞	4/4/4
C-SSD	1.5M	0.375	750	3.3/3.4/0.07
RO-SSD	2M	0.74	1,171	3.3/3.4/0.07
WO-SSD	2M	0.842	6,416	2.4/3.1/1.3

FTL overheads on random write requests. AGCR [49] characterizes the workload behavior and increases the program time of read-intensive data pages in the flash chips so that their read time can be decreased. Such architectures require hardware modifications which is not in the scope of this paper.

In general, one of the main shortcomings of previous studies is neglecting to consider the difference between various SSD brands and models in terms of cost and read/write performance. Many types of SSDs are optimized towards read operations while others are optimized to provide higher write performance. In addition, the tradeoff between performance, power consumption, endurance, reliability, and cost has not been considered in previous works which is crucial for I/O caching architectures.

3 MOTIVATION

In this section, we detail the three shortcomings of state-of-the-art caching architectures which motivates us to propose three-level caching architecture employing SSDs in addition to DRAM. First, we show the diverse characteristics of the SSDs in the market and the performance impact of employing such SSDs as the caching layer for HDDs. Second, we evaluate the write overhead of caching read misses in SSDs. Finally, we investigate the performance of mirrored heterogeneous SSDs employed to overcome the correlated SSDs failure.

SSD manufacturers employ *Single-Level Cell* (SLC), *Multi-Level Cell* (MLC), or *Three-Level Cell* (TLC) NAND chips in their products. SLC SSDs have the highest performance and endurance at the cost of more than 2x of MLC SSDs. The read performance of MLC SSDs, however, is comparable to the SLC SSDs due to the nature of the NAND flashes. Table 1 reports the performance and endurance of several types of SSDs. Using high cost SSDs is not economically justifiable in several workload types. Fig. 2 shows the read and write IOPS per $ for various SSDs. In read-intensive workloads employing RO-SSD or *Consumer-SSD* (C-SSD) results in higher performance per cost. RO- or C-SSDs, however, fail to provide high performance per cost in write-intensive workloads. This experiment reveals that high-cost and low-cost SSDs can be efficient in different workload types and using only one SSD type cannot provide suitable performance per cost in all workload types.

In *Write-Back* (WB) cache policy which is commonly practiced in previous studies, each read miss requires

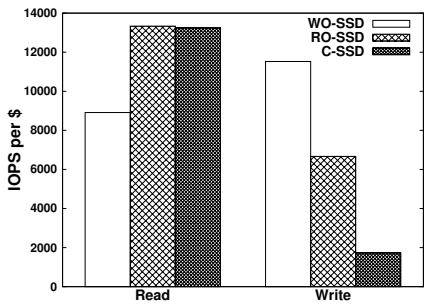

Fig. 2: Performance per Cost for various SSDs

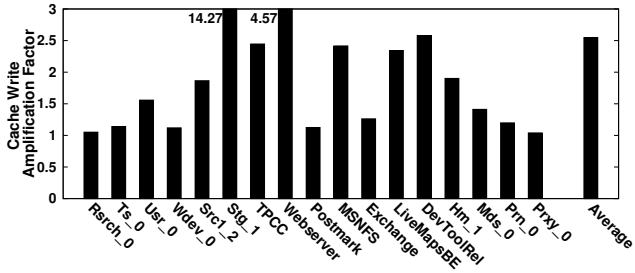

Fig. 3: CWAF for various workloads

writing a data page to the SSD while all write requests are directed to the SSD. Therefore, the total number of writes in SSD will be higher than the number of write requests in the workload. This will result in reduced lifetime of SSDs employed as a WB cache. To evaluate the amplification of writes in previous studies, we introduce *Cache Write Amplification Factor* (CWAF) parameter which is calculated based on Equation 1. Fig. 3 shows CWAF parameter for various workloads. In *Stg_1* and *Webserver* workloads, CWAF is greater than 4.5 which shows the importance of read misses on the SSDs lifetime. By reducing the number of writes due to the read misses on SSDs, we can significantly improve the SSD endurance.

$$CWAF = \frac{Writes_{ssd}}{Writes_{workload}} \quad (1)$$

One of the reliability concerns of employing SSDs, specially in *Redundant Array of Independent Disks* (RAID) configurations is correlated failures due to the either software or hardware defects [50]. Since SSDs in the RAID configuration are identical and in mirrored RAID configurations they receive the same accesses, any software defect probably will trigger on both SSDs resulting in data loss. Additionally, due to the same aging pattern and lifetime, both SSDs are expected to fail in a close time interval which also results in data loss. To mitigate such problem and reduce the probability of double disk failures, employing heterogeneous SSDs with different internal algorithms and/or from different brands can be practiced. Here, we investigate the effect of employing such technique on various MLC-TLC SSD combinations. Fig. 4 shows the normalized performance of various mirrored (RAID-1) configurations for heterogeneous SSDs compared to the performance of homogeneous mirrored

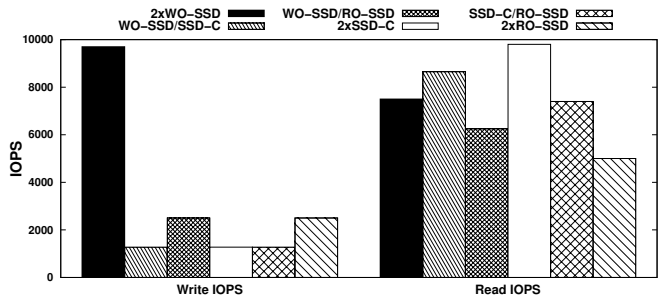

Fig. 4: Performance of Heterogeneous RAID Architectures

SSDs. As can be seen in this figure, the performance is limited by the slower SSD, specially in write requests which results in overall lower performance per cost. For instance, replacing a SSD in a mirrored WO-SSD with a RO-SSD results in almost 5x performance degradation in write requests. Write performance of two mirrored RO-SSDs is equal to the performance of mirrored WO-SSD and RO-SSD while the cost and power consumption of the latter architecture is higher. In read requests, the performance degradation of employing heterogeneous architectures is lower compared to write requests since the performance gap of different SSDs is smaller in read requests. This experiment shows that there is a need for heterogeneous SSD architectures with high performance per cost to simultaneously improve both performance and reliability.

4 Proposed Architecture

An efficient I/O caching architecture should provide high performance, endurance, and reliability with reasonable cost overhead in order to be integrated in storage and high-performance servers. Previous caching architectures have neglected to simultaneously consider such important parameters of I/O caching and focused on improving *only* one of the parameters without investigating the corresponding impact on the other parameters. The proposed architecture is motivated by the lack of a comprehensive caching architecture which is able to mitigate the shortcomings of previous studies discussed in Section 3.

For this purpose, we try to improve performance, power consumption, and lifetime of I/O cache by using a DRAM and high performance SSDs and reducing the number of committed writes to SSDs. To address the reliability concern, TICA is designed such that it does not have any single point of failure and in addition, a failure in any of the caching devices will not result in data loss. This is while the cost overhead is kept as small as possible compared to the traditional caching architectures. TICA is also architected in such a way that the optimizations proposed in previous studies for increasing the cache hit ratio and prioritizing request types can be directly integrated with TICA in order to further improve performance and/or endurance.

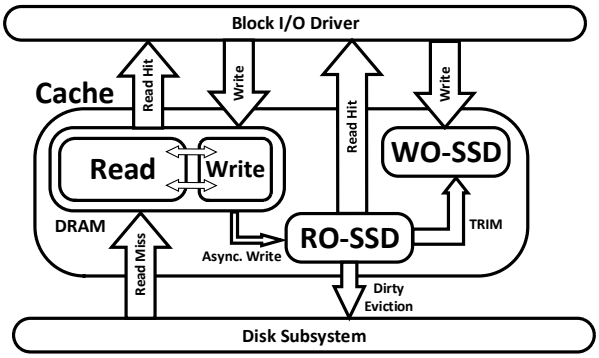

Fig. 5: Proposed Architecture

4.1 High-Level Architecture

To design an efficient caching architecture, we leverage the traditional cache architecture and use three different storage devices for I/O caching. A DRAM module alongside a RO-SSD and a WO-SSD form the three-levels of the proposed architecture. In order to decrease the probability of data loss, a small battery-backup unit is added to DRAM which can sustain a cold system reboot. Such heterogeneous architecture improves the reliability by reducing the probability of double disk failures due to the correlated failure between SSDs of the same model. Fig. 5 depicts the proposed architecture consists of three hardware modules. The data migration inside the I/O cache or between the cache and the main storage device is done using *Direct-Memory Access* (DMA) unit to reduce the CPU overhead. Since a data page might exist in more than one caching device at any time, they are looked up based on the device priority which are prioritized as DRAM, RO-SSD, and then WO-SSD for read requests. TICA works in write-back mode and as such, all write requests will be buffered. If the old data page resides in any of the caching devices, it will be invalidated. In addition to invalidation in mapping data structures, a TRIM[1] request is sent to SSDs to improve its performance on write requests.

The proposed architecture also employs a DRAM in addition to SSDs in the caching layers where it is partitioned into read and asynchronous write cache sections. The read cache partition is used for caching read miss requests. The requested data page is moved to DRAM using DMA and afterwards the data page will be copied from DRAM cache to the destination memory address in the user space. The user write requests arriving to the cache will be redirected to both WO-SSD and DRAM where they will be stored in the second partition of DRAM. An asynchronous thread goes through the second partition and sends the data pages to the RO-SSD and removes them from DRAM. The size of partitions is adjusted dynamically in the runtime based on the percentage of the write requests arrived to DRAM.

To enhance the performance of the proposed architecture, RO-SSD and WO-SSD are configured in such a way that they reside in the critical path of responding

1. Informs disk about data blocks which are no longer in use by OS.

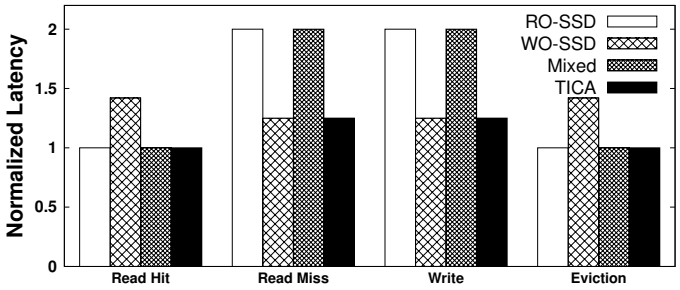

Fig. 6: Average Response Time of Cache Operations Normalized to RO-SSD Read Latency

to those requests that can be handled more efficiently. This way TICA can have optimal performance on both read and write requests without having to use ultra high-performance SSDs which significantly reduces the total cost of I/O cache. In order to show the difference between the proposed architecture and the traditional RAID 1 configurations, the normalized average response time under various cache operations is depicted in Fig. 6. All configurations use two SSDs where in the first two configurations, SSDs in RAID 1 are the same and in the third configuration (mixed) and TICA, one RO-SSD and one WO-SSD are employed. In order to have a fair comparison in Fig. 6, the DRAM module in the proposed architecture is ignored in this experiment. As shown in Fig. 6, TICA has near optimal performance on every cache operation since the critical path of operations and the optimal operation for each SSD is considered.

4.2 Detailed Algorithm

Algorithm 1 depicts the workflow of the proposed architecture in case of a request arrival. If the request is to write a data page and the data page exists in the cache, it will be invalidated. Lines 5 through 8 check the DRAM write cache partition for free space. If there is no space available, the size of the write cache partition will be extended. The calculation for extending the write cache size considers a baseline cache size called $defwritecachesize$ and if the current write cache size is greater than this value, the write cache size will be extended by smaller values. This technique prevents write cache partition from over extending which will reduce the number of read hits from DRAM. In addition to DRAM, WO-SSD will be checked for free space and if there is no space left, a victim data page will be selected and discarded from both SSDs (lines 9 through 11). The victim data page will be removed from RO-SSD since leaving a dirty data page in RO-SSD has a risk of data loss in case of failure of this SSD. After allocating a page in both DRAM and WO-SSD, the write request will be issued. The request for flushing from DRAM to RO-SSD will be issued after completion of the user request.

If an incoming request is for reading a data page, the caching devices will be searched based on their read performance (DRAM, RO-SSD, and WO-SSD, in order). If the request is served from DRAM, LRU_{DRAM} will

Algorithm 1 Proposed Caching Algorithm

```
1: procedure ACCESS(Request)
2:     capacityEstimator(Request)
3:     if Request.iswrite then
4:         IssueDiscards(Request.address)
5:         if DRAMwritecache.isfull then
6:             writecachesize ← writecachesize+
                   2^-(writecachesize−defwritecachesize)
7:             DiscardfromDRAM(writecachesize)
8:             waitforFreeup
9:         if WOSSD.isfull then
10:            FreeupWOSSD
11:            FreeupROSSD
12:        Issue writes to WOSSD and DRAM
13:        Wait for issued writes
14:        update LRU_DRAM and LRU_WOSSD
15:        Issue async. write to ROSSD
16:    else
17:        if inDRAM(Request.address) then
18:            ReadfromDRAM(Request.address)
19:            Update LRU_DRAM
20:        else if InROSSD(Request.address) then
21:            ReadfromROSSD(Request.address)
22:            Update LRU_ROSSD
23:            Update LRU_WOSSD
24:        else if InWOSSD(Request.address) then
25:            ReadfromWOSSD(Request.address)
26:            Update WOSSDLRU
27:        else
28:            if DRAMReadcache.isfull then
29:                writecachesize ← max(defwritecachesize,
                       writecachesize − 2^(writecachesize−defwritecachesize))
30:                DiscardfromDRAM(writecachesize)
31:                if TICA is in WED mode then
32:                    Copy evicted page to WOSSD
33:                Issue page fault for Request.address
```

be updated and if the request is hit in either of SSDs, the LRU queue for both SSDs will be updated. If the request is missed in the cache while DRAM read cache is full and the DRAM write cache size is greater than $defwritecachesize$, the DRAM write cache size will be shrunk. In order to shrink the write cache, it is required to wait for completion of one of the ongoing asynchronous writes to RO-SSD; this will make the current request being stalled. On the other hand, evicting a data page from DRAM read cache imposes no I/O overhead. Therefore, in the proposed architecture, a request is sent to the disk for reading the data page and a watcher thread waits for completion of one of the asynchronous writes to RO-SSD. If one of the asynchronous writes is finished before disk I/O, its place will be allocated for the data page and if not, a data page will be evicted from DRAM read cache in order to make the required space available. TICA allocates a few data pages from DRAM and SSDs for internal operations such as migrating data pages. The default behavior of TICA is to discard the evicted data page from DRAM which we call TICA-EF. There is an alternative approach which copies the evicted data page to WO-SSD which is called TICA-WED. TICA-WED and the algorithm for selecting the TICA policy are detailed next.

4.3 TICA-EF vs. TICA-WED

As mentioned earlier, the endurance of the SSD caching architectures is penalized by the read misses. TICA-EF eliminates the writes in the SSDs due to the read

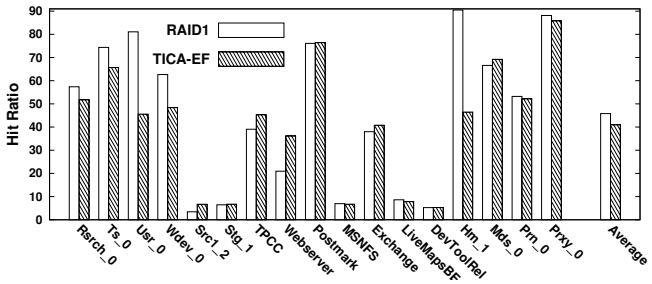

Fig. 7: TICA-EF Hit Ratio

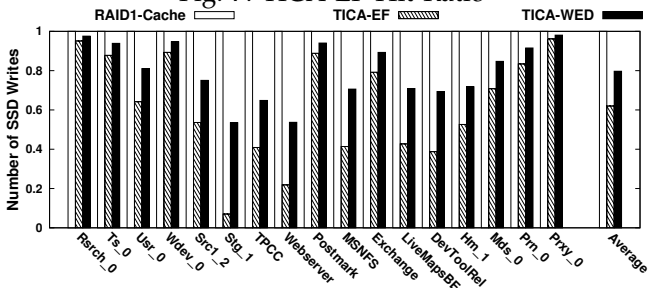

Fig. 8: Total Number of Writes Committed to SSDs

misses and therefore, is called *Endurance-Friendly*. Such approach, however, imposes performance cost since the data pages are evicted early from the cache and cache hit ratio is decreased. Fig. 7 shows the evaluation of the TICA-EF in terms of the cache hit ratio compared to the baseline RAID-1 configuration. TICA-EF fails to provide high performance in several workloads such as *Usr_0*, *Hm_1*, and *Wdev_0*. Our investigation reveals that this is due to the large working set size of the read-intensive data pages. Such data pages can be only buffered in DRAM and since DRAM has a small size, data pages are evicted before re-referencing. Therefore, TICA-EF needs to access HDD more often to bring back evicted data pages to DRAM.

Copying the evicted data pages from DRAM to SSD can improve performance at the cost of reducing endurance. To show the effect of such technique, we propose TICA-WED which copies the data pages on eviction from DRAM to WO-SSD. As mentioned in the motivation section (Section 3), this will decrease the endurance of SSDs. Fig. 8 shows the number of writes committed to SSDs in TICA-WED compared to TICA-EF. In read-intensive workloads with small working set size, TICA-EF has close endurance efficiency to TICA-WED. In other workloads, however, TICA-WED has higher endurance efficiency. We can conclude here that both TICA-EF and TICA-WED policies can provide a suitable policy for a specific workload type and a general approach is required to select one of these two policies based on the workload characteristics.

4.4 Adaptive TICA

To select an effective policy for TICA, we have analyzed the performance of TICA-EF. The performance behavior of TICA-EF in *Wdev_0*, *Usr_0*, *Ts_0* and *Rsrch_0* workloads reveals that there are two reasons for low performance of TICA-EF: 1) DRAM size is less than the working set size, and 2) cold data pages are trapped in SSDs. To mitigate the performance degradation of TICA-EF, we propose *TICA-Adaptive* (TICA-A) which switches the policy from TICA-EF to TICA-WED when one of the two above conditions is detected. In this section, the algorithms for identifying the mentioned two conditions are detailed.

Algorithm 2 DRAM Low Capacity Identifier

```
1: windowSize ← 2 * DRAM_size
2: requestCounter, EQHit, DRAMReadHit ← 0
3: procedure CAPACITYESTIMATOR(request)
4:     requestCounter ← requestCounter + 1
5:     if request.isRead then
6:         if Hit in DRAM then
7:             DRAMReadHit ← DRAMReadHit + 1
8:         else if Hit in EQ then
9:             EQHit ← EQHit + 1
10:    if requestCounter == windowSize then
11:        if (EQHit + DRAMReadHit) > T_max then
12:            Switch to TICA-WED
13:        else if EQHit > T_min then
14:            Switch to TICA-WED
15:        else
16:            Switch to TICA-EF
17:        requestCounter, EQHit, DRAMReadHit ← 0
```

4.4.1 DRAM Low Capacity Identifier

Due to the small size of DRAM in TICA, the thrashing problem [51] is likely to happen if the working set size of the workload is larger than DRAM size. To identify such condition, we keep a queue of evicted data pages from DRAM, called *Evicted Queue* (EQ). Evicted data pages from DRAM enter EQ if they are copied to the DRAM due to a read miss. The hit ratio *with* and *without* considering the EQ is calculated periodically and if their difference is greater than a predefined threshold (T_{min}), TICA will switch to the TICA-WED policy. Employing the threshold for minimum difference between hit ratios prevents constantly switching between the two policies.

Since TICA-EF lowers the *Total Cost of Ownership* (TCO) by extending the SSDs lifetime, we prefer it over TICA-WED. Therefore, if TICA-EF has high hit ratio, regardless of the hit ratio of EQ, we switch to TICA-EF. The threshold (T_{max}), however, should be set conservatively to ensure negligible performance degradation. Modifying the thresholds enables us to prefer one of the two policies based on the I/O demand of the storage system and/or the overall system status. For instance, when most SSDs in the array are old, we would prefer TICA-EF to prolong their lifetime and reduce the probability of data loss. Algorithm 2 shows the flow of identifying thrashing in DRAM. Switching between the policies is conducted once the number of incoming requests to the cache becomes twice the size of DRAM memory. For each request, the counter for hits in DRAM and EQ are updated in Lines 5 through 9. In Lines 11 to 17, the hit ratios are checked and TICA policy is changed if required.

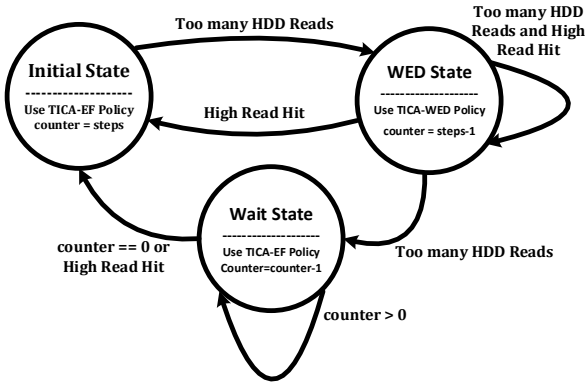

Fig. 9: State Machine for Preventing Cold Data Trapped in SSD

4.4.2 Preventing Cold Data Trapped in SSDs

In TICA-EF, only write accesses are redirected to SSDs and all read accesses are supplied by either DRAM or HDD. Therefore, in read-intensive workloads, SSDs become idle and previously hot data pages which are now cold reside in SSDs without any means to evict such data pages. To prevent such problem, we propose a *State Machine Based Insertion* (SMBI) to *conservatively* switch from TICA-EF to TICA-WED in order to replace the cold data pages in SSDs. The simplified model of SMBI is shown in Fig. 9. We identify two conditions 1) too many HDD reads and 2) high hit ratio. When both conditions are met in the workload, TICA switches to TICA-WED until one of the conditions is no longer valid. *Too many HDD reads* shows that the read working set size is larger than DRAM size. In such condition, we allow evicted data pages from DRAM to enter WO-SSD to increase its hit ratio and reduce the number of HDD reads. We cannot rely solely on the number of HDD reads for switching to WED since in workloads with low locality, the number of HDD reads is also high and copying the evicted data pages from DRAM to WO-SSD will only impose endurance cost without any performance improvement. Therefore, SMBI stays in the WED state as long as both number of HDD reads and hit ratio are high. Having high hit ratio and low number of HDD reads shows that the working set size is smaller than DRAM size and SMBI switches back to the EF policy.

If the hit ratio is decreased while the number of HDD reads is still high, SMBI enters a waiting state which prevents re-entering WED mode in the next few windows. This state prevents constantly switching between the two policies. Algorithm 3 shows the detailed flow of SMBI. Line 13 switches the policy to TICA-WED if the number of HDD read requests in the current window is greater than the T_{hdd} threshold. In lines 18 through 28, SMBI checks both conditions and if one of them is no longer valid, it switches back to TICA-EF policy.

Algorithm 3 State Machine Based Insertion

```
1: counter ← steps
2: currentState ← initialState, nextState ← initialState
3: diskRead, readHit, requestCounter ← 0
4: procedure SMBI(request)
5:     requestCounter ← requestCounter + 1
6:     if request.isRead then
7:         if Hit in cache then
8:             readHit ← readHit + 1
9:         else
10:            diskRead ← diskRead + 1
11:    if requestCounter == sampleSize then
12:        if currentState == initialState then
13:            if diskRead > T_hdd then
14:                Switch to TICA-WED policy
15:                counter ← steps − 1
16:                currentState ← WEDState
17:        else if currentState == WEDState then
18:            if diskRead > T_hdd then
19:                if readHit > T_read then
20:                    Switch to TICA-WED policy
21:                    currentState ← WEDState
22:                else
23:                    Switch to TICA-EF policy
24:                    currentState ← waitState
25:            else if readHit > T_read then
26:                Switch to TICA-EF policy
27:                counter ← steps
28:                currentState ← initialState
29:        else if currentState == waitState then
30:            if (counter == 0 or readHit) > T_read then
31:                Switch to TICA-EF policy
32:                counter ← steps
33:                currentState ← initialState
34:            else
35:                counter ← counter − 1
```

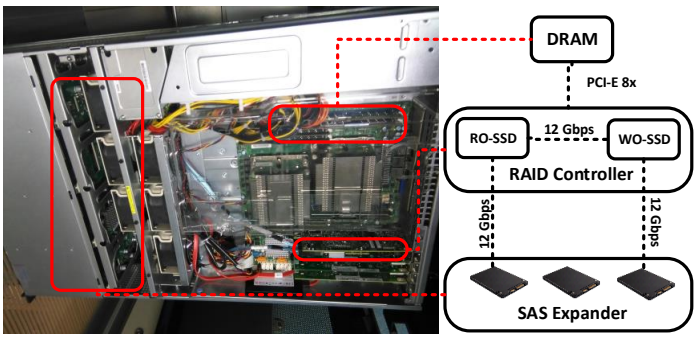

Fig. 10: Hardware Architecture of Experimental Setup

5 Experimental Results

In this section, the performance, power consumption, endurance, and reliability of the proposed architecture is evaluated. We compare TICA with a state-of-the-art multi-level caching architecture (uCache [26]) and a state-of-the-art SSD caching architecture (S-RAC [17]). In order to have a fair comparison, uCache is modified and all single points of failure are removed to improve its reliability. Support for RAID1 SSDs is also added to uCache. Since S-RAC is a single-level cache architecture, a first level DRAM cache is added so that all three examined architectures benefit from both DRAM and SSD. To show the effect of different TICA policies, in addition to TICA-A, both TICA-EF and TICA-WED are also evaluated. The detailed characteristics of the workloads are reported in Table 2.

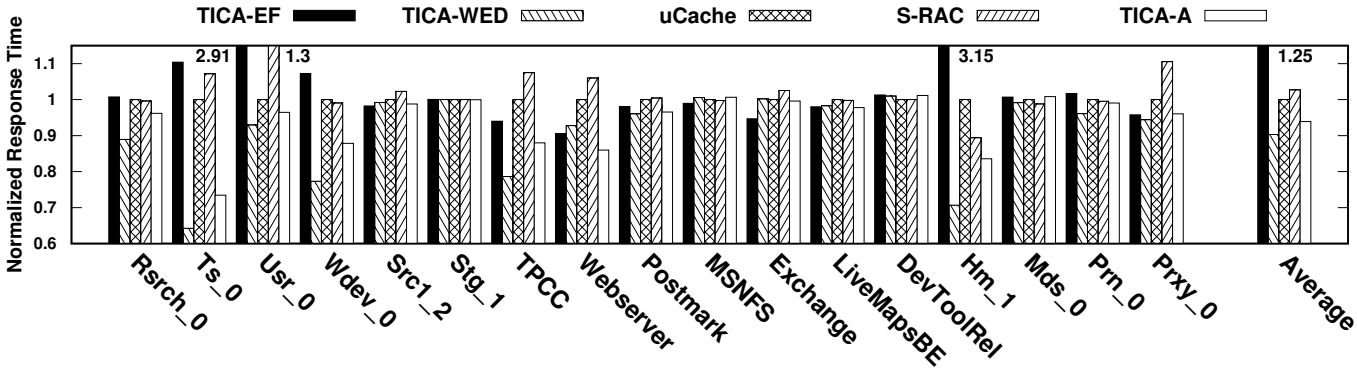

Fig. 11: Normalized Response Time: TICA vs. Conventional Caching Architectures

5.1 Experimental Setup

To conduct the experiments, we employed a rackmount server equipped with Intel Xeon, 128GB memory, and a SSD for the operating system to reduce the effect of the operating system and the other running applications on the obtained results. Fig. 10 shows the actual server running experiments and the interfaces between I/O cache layers. *SAS expander* is capable of supporting both SATA and SAS disk drivers. *RAID controller* is configured in *Just a Bunch Of Disks* (JBOD) mode where disks are directly provided to the OS without any processing by the controller. Employing *SAS expander* and *RAID controller* enables us to run experiments on various SATA/SAS SSDs without need for disk replacement or server reboot.

WO- and RO-SSDs are selected from enterprise-grade SSDs employed in the datacenters. We warm up SSDs before each experiment by issuing requests until the SSD reaches a stable latency. The traces are replayed on the devices using our in-house trace player which is validated by blktrace [52] tool. The requests sent to the disk by our trace player are compared to the original trace file to ensure it has the expected behavior. The characteristics of DRAM and SSDs employed in the experimental results is reported in Table 1. In the experiments, size of SSDs and DRAM is set to 10% and 1% of the working set size, respectively. The value of T_{min}, T_{max}, T_{hdd}, and T_{read} are set to 0.15, 0.25, 0.2, and 0.2, respectively.

TABLE 2: Workload Characteristics

Workload	Total Requests Size	Read Requests	Writes Requests
TPCC	43.932 GB	1,352,983 (70%)	581,112 (30%)
Webserver	7.607 GB	418,951 (61%)	270,569 (39%)
DevToolRel	3.133 GB	108,507 (68%)	52,032 (32%)
LiveMapsBE	15.646 GB	294,493 (71%)	115,862 (28%)
MSNFS	10.251 GB	644,573 (65%)	349,485 (35%)
Exchange	9.795 GB	158,011 (24%)	502,716 (76%)
Postmark	19.437 GB	1,272,148 (29%)	3,172,014 (71%)
Stg_1	91.815 GB	1,400,409 (64%)	796,452 (36%)
Rsrch_0	13.11 GB	133,625 (9%)	1,300,030 (91%)
Src1_2	1.65 TB	21,112,615 (57%)	16,302,998 (43%)
Wdev_0	10.628 GB	229,529 (20%)	913,732 (80%)
Ts_0	16.612 GB	316,692 (18%)	1,485,042 (82%)
Usr_0	51.945 GB	904,483 (40%)	1,333,406 (60%)
Hm_1	9.45 GB	580,896 (94%)	28,415 (6%)
Mds_0	11.4 GB	143,973 (31%)	1,067,061 (69%)
Prn_0	63.44 GB	602,480 (22%)	4,983,406 (78%)
Prxy_0	61.03 GB	383,524 (5%)	12,135,444 (95%)

5.2 Performance

Fig. 11 shows the normalized response time of TICA compared to uCache and S-RAC, all normalized to uCache. TICA-WED which is optimized toward higher performance, reduces the response time by 12% on average compared to uCache and S-RAC. The highest performance improvement of TICA belongs to *Ts_0* workload with 45% reduction in response time (compared to S-RAC). Although TICA-WED and TICA-EF differ in read miss policy and *Ts_0* is a write-dominant workload (80% write requests), TICA-WED still performs better than TICA-EF with 42% less response time. This shows the significant impact of writing read misses on the SSDs and therefore, forcing the dirty data pages to be evicted from cache. TICA-WED also improves performance in read-dominant workloads such as TPCC by copying the evicted data pages from DRAM to WO-SSD.

TICA-EF, optimized toward better endurance, outperforms TICA-WED in few workloads such as *Webserver* and *Exchange*. Our investigation reveals that this is due to a) the limited space of SSDs and b) forcing the eviction of dirty data pages from SSD which is conducted aggressively in TICA-WED. In *Webserver* workload, TICA-A also identifies such problem and manages copying evicted data pages from DRAM to WO-SSD. Therefore, it has better performance-efficiency in *Webserver* workload compared to both TICA-EF and TICA-WED policies. By managing the evicted data pages from DRAM, TICA-A improves performance compared to previous studies by up to 45% and 8% on average. We can conclude here that TICA-A is performance-efficient in both read- and write-intensive workloads by managing the evicted data pages from DRAM.

5.3 Power Consumption

To evaluate the power consumption of TICA, we estimate the total consumed energy for workloads. In addition to the read and write requests, idle power consumption of the devices is also considered in the energy consumption to further increase its accuracy. The read and write operations for background tasks such as

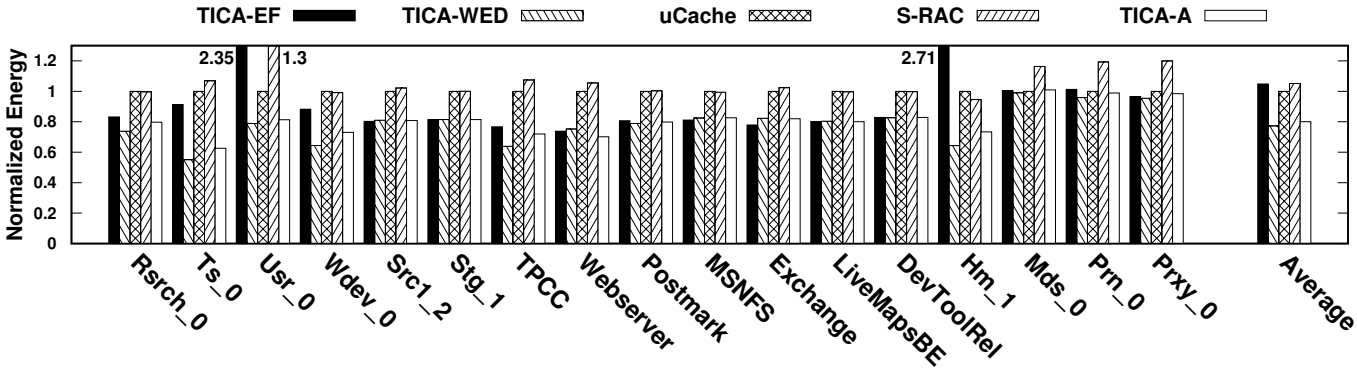

Fig. 12: Normalized Power Consumption: TICA vs. Conventional Caching Architectures

TABLE 3: Parameters Description

Parameter	Description
$Read_{wo}$	WO-SSD Total Read Requests
$Write_{wo}$	WO-SSD Total Write Requests
$Read_{ro}$	RO-SSD Total Read Requests
$Write_{ro}$	RO-SSD Total Write Requests
$Read_D$	DRAM Total Read Requests
$Write_D$	DRAM Total Write Requests
$RLat_{wo}$	WO-SSD Read Latency
$WLat_{wo}$	WO-SSD Write Latency
$RLat_{ro}$	RO-SSD Read Latency
$WLat_{ro}$	RO-SSD Write Latency
Lat_{wo}	DRAM Latency
RP_{wo}	WO-SSD Read Power
WP_{wo}	WO-SSD Write Power
RP_{ro}	RO-SSD Read Power
WP_{ro}	RO-SSD Write Power
P_D	DRAM Power
IP_{wo}	WO-SSD Idle Power
IP_{ro}	RO-SSD Idle Power
IP_D	DRAM Idle Power
$Idle_{wo}$	Total WO-SSD Idle Time
$Idle_{ro}$	Total RO-SSD Idle Time
$Idle_D$	Total DRAM Idle Time
R_{Device}	Device Reliability
U_{Device}	Device Unreliability
$MTTF_{Device}$	Device Mean Time To Failure

copying the dirty data pages from DRAM to RO-SSD and flushing such data pages from RO-SSD to disk are also included in the energy consumption formula. Equation 2 shows the formula for estimating the total energy consumption. All parameters are detailed in Table 3.

$$
\begin{aligned}
Energy = & \\
& \sum^{Read_{wo}} (RLat_{wo} * RP_{wo}) + \sum^{Write_{wo}} (WLat_{wo} * WP_{wo}) + \\
& (Idle_{wo} * IP_{wo}) + \sum^{Read_{ro}} (RLat_{ro} * RP_{ro}) + \\
& \sum^{Write_{ro}} (WLat_{ro} * WP_{ro}) + (Idle_{ro} * IP_{ro}) + \\
& \sum^{(Read_D + Write_D)} (Lat_D * P_D) + (Idle_D * IP_{ro}) \qquad (2)
\end{aligned}
$$

TICA improves the power consumption by a) employing power-efficient SSDs while maintaining the performance and b) reducing the number of accesses to the SSDs. Previous studies employ two identical SSDs in a mirrored RAID configuration to provide high reliability while as discussed in Section 3, heterogeneous SSDs are not performance-efficient in traditional mirrored RAID configurations. As such, state-of-the-art architectures such as uCache and S-RAC need to employ two WO-SSDs, which have high power consumption. TICA on the other hand, employs a WO-SSD and a RO-SSD in its architectures which results in lower power consumption compared to using two WO-SSDs. Additionally, by reducing the response time of the requests, SSDs more often enter idle state and therefore, the total power consumption is decreased. Fig. 12 shows the normalized consumed energy of TICA compared to uCache and S-RAC, normalized to uCache. In all workloads, TICA policies improve power consumption which shows the effectiveness of replacing a WO-SSD with a RO-SSD. The only exceptions are *Usr_0* and *Hm_1* workloads where TICA-EF has 2.35x and 2.71x higher power consumption compared to uCache, respectively. This is due to the high response time of the requests in this workload, which prevents SSDs from entering the idle state. TICA-A improves power consumption by an average of 28% and the maximum improvement in the power consumption (70%) belongs to *Ts_0* workload in comparison with S-RAC.

5.4 Endurance

TICA-WED redirects all evicted data pages from DRAM to WO-SSD and therefore, significantly increases the number of writes in SSDs. TICA-EF on the other hand, does not copy such data pages to SSDs to preserve their lifetime. Fig. 13 shows the normalized number of writes in the SSDs compared to uCache and S-RAC, normalized to uCache. uCache, S-RAC, and TICA-EF have almost the same number of writes in SSDs since they limit writes in the SSDs. TICA-EF improves SSDs lifetime by an average of 1.3% compared to uCache and S-RAC.

TICA-WED places all evicted data pages from DRAM in SSD and therefore, increases the number of writes in SSDs. For instance, in *Stg_1* workload, TICA-WED reduces the SSDs lifetime by more than 7x. TICA-A which tries to balance the performance and endurance has only 4.7% on average lifetime overhead compared to uCache and S-RAC. Since TICA employs an unbalanced writing scheme between WO-SSD and RO-SSD, the lifetime of the RO-SSD is not affected by the increase in the number of writes. Note that TICA-A still improves the SSDs

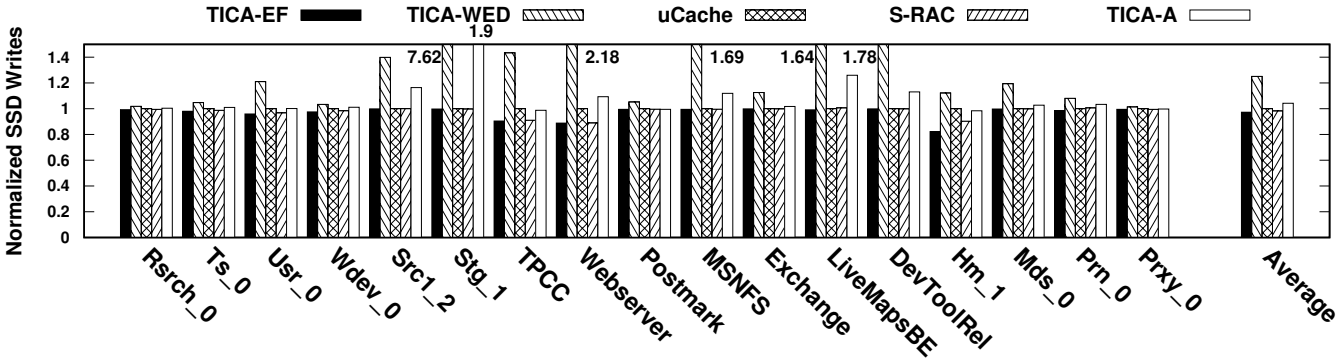

Fig. 13: Normalized Number of Writes Committed to SSDs

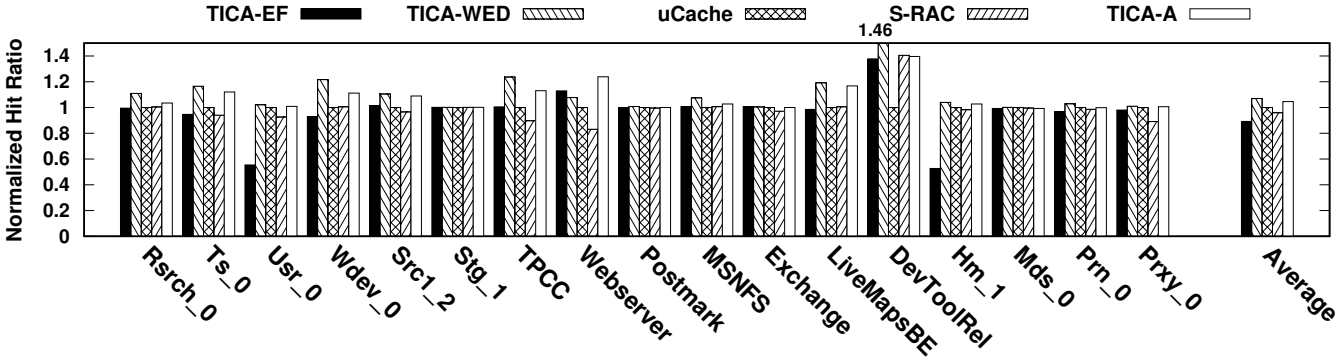

Fig. 14: Hit Ratio of Caching Architectures

lifetime by an average of 38% compared to the single-level SSD architectures (not shown in Fig. 13).

5.5 Hit Ratio

TICA, uCache, and S-RAC do not comply with a simple LRU algorithm (there is not a global LRU queue for all data pages). Hence, the hit ratio of such multi-level caching architectures needs to be evaluated. Although the performance of such architectures is already investigated in Section 5.2, the hit ratio evaluation enables us to predict the performance in other hardware setups such as using RAID or *Storage Area Network* (SAN) storages as the backend of the caching architecture.

Fig. 14 shows the normalized hit ratio of TICA-EF, TICA-WED, TICA-A, and S-RAC compared to uCache. TICA-WED which is optimized toward performance, has the highest average hit ratio. TICA-A improves hit ratio compared to uCache and S-RAC in all workloads by an average of 6.4%. The highest improvement belongs to *DevToolRel* workload where TICA-WED and TICA-A improve hit ratio by 46% and 39%, respectively. S-RAC, however, has comparable hit ratio with TICA in this workload. This is due to the ghost queue employed in S-RAC to identify requests with higher benefit for caching. Due to the DRAM eviction policy of TICA-EF, it has lower hit ratio compared to uCache and other TICA policies. The hit ratio degradation of TICA-EF, however, is negligible in most workloads.

5.6 Reliability

uCache and S-RAC employ two WO-SSDs while TICA uses a WO-SSD alongside a RO-SSD in its architecture. Both uCache and S-RAC will fail only when both WO-SSDs are failed. There are two conditions which can result in failure of TICA: a) failure of WO-SSD and RO-SSD, and b) failure of WO-SSD and DRAM. Since RO-SSD has lower *Mean Time To Failure* (MTTF) [53] compared to WO-SSD, TICA might reduce the overall reliability of the system. DRAM, on the other hand, has higher MTTF compared to WO-SSD. Therefore, the actual reliability of TICA depends on the duration of keeping dirty data pages in DRAM and RO-SSD.

The reliability of caching architectures is calculated based on *Reliability Block Diagram* (RBD) [54]. To calculate the system reliability, RBD uses 1) the reliability of system components (storage devices in our use-case) and 2) the dependency of system failure to the failure of components. The reliability of storage devices is computed based on MTTF [53]. This is done via considering the exponential distribution for faults in SSDs, which is formulated in Equation 3. The MTTF value for storage devices is extracted from their datasheets. Although other distributions such as Weibull might be more suitable, they require additional parameters to MTTF to model reliability. Field studies in SSD failure models do not disclose the brands of SSDs [50], [55], [56], and therefore, we cannot use such models. If such field studies become available, we can employ a more accurate MTTF for

devices in the exponential distribution. This can be done by estimating the real MTTF of the device, based on the *Cumulative Distribution Function* (CDF) of Weibull distribution, as discussed in [57]. The description of parameters in Equation 3 is available in Table 3. Equation 4 and Equation 5 show the formula for calculating the reliability of TICA and uCache, respectively. Note that the reliability of S-RAC is calculated using the same formula as uCache. The α variable denotes the weight of each failure scenario. In the traditional RAID architectures, α is equal to one. In TICA, α depends on the running workload and number of write requests. Since TICA employs a RO-SSD instead of WO-SSD, compared to uCache and S-RAC, it is expected that TICA slightly reduces the reliability. Considering 0.8 as the value of α, which is close to the actual value of α in our experiments, TICA will have unreliability of $1.27 * 10^{-5}$ while unreliability of uCache and S-RAC is $1.14 * 10^{-5}$. Note that the cost of hardware in TICA is lower than uCache and TICA will have the same reliability compared to uCache if the same hardware is employed for both architectures.

$$R_{Device} = e^{-\frac{1}{MTTF_{Device} * 365 * 24}} \quad (3)$$

$$R_{TICA} = \alpha * (1 - (1 - R_{\text{WO-SSD}}) * (1 - R_D)) + (1 - \alpha) * (1 - (1 - R_{\text{WO-SSD}}) * (1 - R_{\text{RO-SSD}})) \quad (4)$$

$$R_{uCache} = 1 - (1 - R_{\text{WO-SSD}}) * (1 - R_{\text{WO-SSD}}) \quad (5)$$

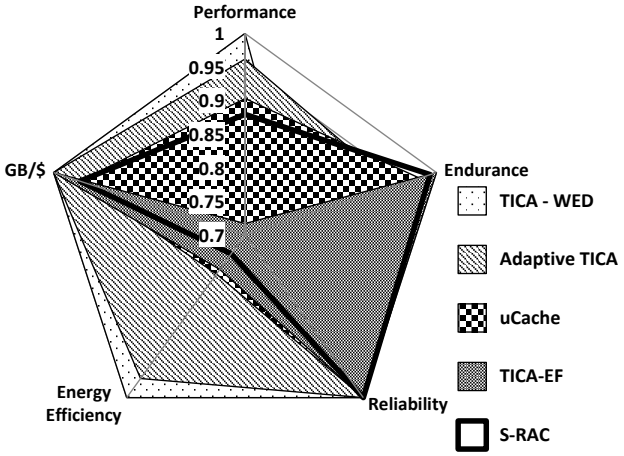

Fig. 15: Overall Comparison of Caching Architectures (Higher values are better)

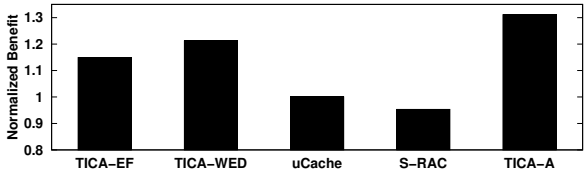

Fig. 16: Normalized Benefit of Various Caching Architectures

5.7 Overall

We can conclude that the experimental results with following observations: 1) TICA improves performance and hit ratio compared to previous state-of-the-art architectures. 2) The power consumption is also improved in TICA by reducing the number of accesses to the SSDs. 3) Lifetime of SSDs is extended in TICA compared to single-level SSD caching architectures while the lifetime is negligibly reduced compared to uCache and S-RAC. 4) The reliability of TICA is the same as previous studies when the same hardware is employed. Reducing the total cost in TICA can result in slightly less reliability. Fig 15 shows the overall comparison of TICA policies with uCache and S-RAC. All parameters are normalized to the highest value where higher values are better in all parameters. Fig. 16 also shows the overall benefit of caching architectures. Benefit is computed by multiplying normalized performance, endurance, cost, and power consumption. uCache and S-RAC, which focus on optimizing only one parameter have lower benefit compared to TICA variants. TICA-A provides the highest benefit since it considers all mentioned parameters in designing caching architecture and balances the performance and endurance, based on the workload characteristics.

6 Conclusion

In this paper, we demonstrated that simultaneously employing different SSDs in traditional architectures is not performance-efficient. In addition, state-of-the-art architectures neglected to consider all aspects of the caching architectures. To mitigate such problems, we proposed a three-level caching architecture, called TICA, which by employing RO-SSD and WO-SSD tries to reduce the cost and improve the performance and power consumption. TICA does not have any single point of failure offering high reliable I/O cache architecture. This is while the endurance cost of the proposed architecture is only 4.7% higher than state-of-the-art caching architectures. Additionally, the hardware cost of TICA is 5% less than conventional architectures. The SSDs lifetime is extended by up to 38% compared to single-level SSD caching architectures. The experimental results demonstrated that our architecture can improve performance and power consumption compared to previous studies, by up to 8% and 28%, respectively.

Acknowledgments

This work has been partially supported by *Iran National Science Foundation* (INSF) under grant number 96006071 and by HPDS Corp.

References

[1] S. Ahmadian, F. Taheri, M. Lotfi, M. Karimi, and H. Asadi, "Investigating power outage effects on reliability of solid-state drives," in *to appear in Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018.

[2] Storage Networking Industry Association, "Microsoft enterprise traces," http://iotta.snia.org/traces/130, accessed: 2015-08-10.

[3] S. Shaw, *HammerDB: the open source oracle load test tool*, 2012, accessed: 2017-08-10. [Online]. Available: http://www.hammerdb.com/

[4] V. Tarasov, E. Zadok, and S. Shepler, "Filebench: A flexible framework for file system benchmarking," *USENIX; login*, vol. 41, 2016.

[5] M. Tarihi, H. Asadi, A. Haghdoost, M. Arjomand, and H. Sarbazi-Azad, "A hybrid non-volatile cache design for solid-state drives using comprehensive I/O characterization," *IEEE Transactions on Computers (TC)*, vol. 65, no. 6, pp. 1678–1691, 2016.

[6] X. Wu and A. L. N. Reddy, "Managing storage space in a flash and disk hybrid storage system," in *IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sept 2009, pp. 1–4.

[7] F. Ye, J. Chen, X. Fang, J. Li, and D. Feng, "A regional popularity-aware cache replacement algorithm to improve the performance and lifetime of SSD-based disk cache," in *IEEE International Conference on Networking, Architecture and Storage (NAS)*, Aug 2015, pp. 45–53.

[8] R. Salkhordeh, S. Ebrahimi, and H. Asadi, "ReCA: an efficient reconfigurable cache architecture for storage systems with online workload characterization," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. PP, no. 99, pp. 1–1, 2018.

[9] R. Salkhordeh, H. Asadi, and S. Ebrahimi, "Operating system level data tiering using online workload characterization," *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1534–1562, 2015.

[10] S. Liu, J. Jiang, and G. Yang, "Macss: A metadata-aware combo storage system," in *Proceedings of the International Conference on Systems and Informatics (ICSAI)*, May 2012, pp. 919 –923.

[11] M. Lin, R. Chen, J. Xiong, X. Li, and Z. Yao, "Efficient sequential data migration scheme considering dying data for HDD/SSD hybrid storage systems," *IEEE Access*, vol. 5, pp. 23 366–23 373, 2017.

[12] S. Ahmadian, O. Mutlu, and H. Asadi, "ECI-Cache: A high-endurance and cost-efficient I/O caching scheme for virtualized platforms," in *in Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*. ACM, 2018.

[13] S. Huang, Q. Wei, D. Feng, J. Chen, and C. Chen, "Improving flash-based disk cache with lazy adaptive replacement," *ACM Transactions on Storage (TOS)*, vol. 12, no. 2, pp. 8:1–8:24, Feb. 2016.

[14] R. Santana, S. Lyons, R. Koller, R. Rangaswami, and J. Liu, "To ARC or Not to ARC," in *Proceedings of the 7th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage)*, 2015, pp. 14–14.

[15] R. Appuswamy, D. C. van Moolenbroek, and A. S. Tanenbaum, "Cache, cache everywhere, flushing all hits down the sink: On exclusivity in multilevel, hybrid caches," in *IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, May 2013, pp. 1–14.

[16] Y. Liang, Y. Chai, N. Bao, H. Chen, and Y. Liu, "Elastic Queue: A universal SSD lifetime extension plug-in for cache replacement algorithms," in *Proceedings of the 9th ACM International on Systems and Storage Conference (SYSTOR)*. ACM, 2016, pp. 5:1–5:11. [Online]. Available: http://doi.acm.org/10.1145/2928275.2928286

[17] Y. Ni, J. Jiang, D. Jiang, X. Ma, J. Xiong, and Y. Wang, "S-RAC: SSD friendly caching for data center workloads," in *Proceedings of the 9th ACM International on Systems and Storage Conference*. ACM, 2016, pp. 8:1–8:12. [Online]. Available: http://doi.acm.org/10.1145/2928275.2928284

[18] Z. Fan, D. Du, and D. Voigt, "H-ARC: A non-volatile memory based cache policy for solid state drives," in *Mass Storage Systems and Technologies (MSST)*, June 2014, pp. 1–11.

[19] X. Chen, W. Chen, Z. Lu, P. Long, S. Yang, and Z. Wang, "A duplication-aware SSD-Based cache architecture for primary storage in virtualization environment," *IEEE Systems Journal*, vol. 11, no. 4, pp. 2578–2589, Dec 2017.

[20] Z. Chen, N. Xiao, Y. Lu, and F. Liu, "Me-CLOCK:a memory-efficient framework to implement replacement policies for large caches," *IEEE Transactions on Computers (TC)*, vol. 65, no. 8, pp. 2665–2671, Aug 2016.

[21] L. Tang, Q. Huang, W. Lloyd, S. Kumar, and K. Li, "RIPQ: Advanced photo caching on flash for facebook," in *Proceedings of the 13th Usenix Conference on File and Storage Technologies (FAST)*, 2015, pp. 373–386.

[22] Y. Chai, Z. Du, X. Qin, and D. Bader, "WEC: Improving durability of ssd cache drives by caching write-efficient data," *IEEE Transactions on Computers (TC)*, vol. PP, no. 99, pp. 1–1, 2015.

[23] J. Levandoski, D. Lomet, and S. Sengupta, "LLAMA: A cache/storage subsystem for modern hardware," *Proceedings of the VLDB Endowment*, vol. 6, no. 10, pp. 877–888, Aug. 2013. [Online]. Available: http://dx.doi.org/10.14778/2536206.2536215

[24] J. Wang, Z. Guo, and X. Meng, "An efficient design and implementation of multi-level cache for database systems," in *Database Systems for Advanced Applications*. Springer International Publishing, 2015, pp. 160–174.

[25] C. Yuxia, C. Wenzhi, W. Zonghui, Y. Xinjie, and X. Yang, "AMC: an adaptive multi-level cache algorithm in hybrid storage filesystems," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4230–4246, 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3530

[26] D. Jiang, Y. Che, J. Xiong, and X. Ma, "uCache: A utility-aware multilevel SSD cache management policy," in *IEEE 10th International Conference on High Performance Computing and Communications, IEEE International Conference on Embedded and Ubiquitous Computing*, Nov 2013, pp. 391–398.

[27] S. K. Yoon, Y. S. Youn, S. J. Nam, M. H. Son, and S. D. Kim, "Optimized memory-disk integrated system with DRAM and nonvolatile memory," *IEEE Transactions on Multi-Scale Computing Systems*, vol. PP, no. 99, pp. 1–1, 2016.

[28] H. Liu and H. H. Huang, "Graphene: Fine-grained IO management for graph computing," in *Proceedings of the 15th Usenix Conference on File and Storage Technologies (FAST)*. USENIX Association, 2017, pp. 285–299.

[29] S. He, Y. Wang, and X. H. Sun, "Improving performance of parallel I/O systems through selective and layout-aware SSD cache," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 27, no. 10, pp. 2940–2952, Oct 2016.

[30] E. Kakoulli and H. Herodotou, "OctopusFS: A distributed file system with tiered storage management," in *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2017, pp. 65–78.

[31] L. Wu, Q. Zhuge, E. H. M. Sha, X. Chen, and L. Cheng, "BOSS: An efficient data distribution strategy for object storage systems with hybrid devices," *IEEE Access*, vol. 5, pp. 23 979–23 993, 2017.

[32] S. He, Y. Wang, Z. Li, X. H. Sun, and C. Xu, "Cost-aware region-level data placement in multi-tiered parallel I/O systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 28, no. 7, pp. 1853–1865, July 2017.

[33] D. Arteaga, J. Cabrera, J. Xu, S. Sundararaman, and M. Zhao, "CloudCache: On-demand flash cache management for cloud computing," in *Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST)*, ser. FAST'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 355–369. [Online]. Available: http://dl.acm.org/citation.cfm?id=2930583.2930610

[34] Z. Shen, F. Chen, Y. Jia, and Z. Shao, "DIDACache: a deep integration of device and application for flash based key-value caching," in *Proceedings of the 15th Usenix Conference on File and Storage Technologies (FAST)*. USENIX Association, 2017, pp. 391–405.

[35] L. Lu, T. S. Pillai, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Wisckey: Separating keys from values in ssd-conscious storage," in *Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST)*, 2016, pp. 133–148.

[36] W. Li, G. Jean-Baptise, J. Riveros, G. Narasimhan, T. Zhang, and M. Zhao, "CacheDedup: In-line deduplication for flash caching," in *Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST)*. USENIX Association, 2016, pp. 301–314.

[37] H. Wu, C. Wang, Y. Fu, S. Sakr, K. Lu, and L. Zhu, "A differentiated caching mechanism to enable primary storage deduplication in clouds," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 29, no. 6, pp. 1202–1216, June 2018.

[38] X. Zhang, J. Li, H. Wang, K. Zhao, and T. Zhang, "Reducing solid-state storage device write stress through opportunistic in-place delta compression," in *Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST)*. USENIX Association, 2016, pp. 111–124.

[39] M. Saxena and M. M. Swift, "Design and prototype of a solid-state cache," *Transactions on Storage (TOS)*, vol. 10, no. 3, pp. 1–34, 2014.

[40] S. Lee, M. Liu, S. Jun, S. Xu, J. Kim, and Arvind, "Application-managed flash," in *Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST)*, 2016, pp. 339–353.

[41] C. Lee, D. Sim, J. Hwang, and S. Cho, "F2FS: A new file system

for flash storage," in *Proceedings of the 13th Usenix Conference on File and Storage Technologies (FAST)*, 2015, pp. 273–286.
[42] Y. Jin, H. Tseng, Y. Papakonstantinou, and S. Swanson, "KAML: A flexible, high-performance key-value SSD," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 373–384.
[43] E. Rho, K. Joshi, S.-U. Shin, N. J. Shetty, J. Hwang, S. Cho, D. D. Lee, and J. Jeong, "FStream: Managing flash streams in the file system," in *Proceedings of the 16th Usenix Conference on File and Storage Technologies (FAST)*, 2018, pp. 257–264.
[44] Q. Xia and W. Xiao, "High-performance and endurable cache management for flash-based read caching," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 27, no. 12, pp. 3518–3531, Dec 2016.
[45] J. Wan, W. Wu, L. Zhan, Q. Yang, X. Qu, and C. Xie, "DEFT-Cache: A cost-effective and highly reliable SSD cache for RAID storage," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2017, pp. 102–111.
[46] A. Tavakkol, M. Sadrosadati, S. Ghose, J. Kim, Y. Luo, Y. Wang, N. M. Ghiasi, L. Orosa, J. Gmez-Luna, and O. Mutlu, "FLIN: enabling fairness and enhancing performance in modern NVMe solid state drives," in *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 397–410.
[47] N. Elyasi, M. Arjomand, A. Sivasubramaniam, M. T. Kandemir, C. R. Das, and M. Jung, "Exploiting intra-request slack to improve SSD performance," in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2017, pp. 375–388.
[48] H. Kim, D. Shin, Y. H. Jeong, and K. H. Kim, "SHRD: Improving spatial locality in flash storage accesses by sequentializing in host and randomizng in device," in *Proceedings of the 15th Usenix Conference on File and Storage Technologies (FAST)*, 2017, pp. 271–283.
[49] Q. Li, L. Shi, C. J. Xue, K. Wu, C. Ji, Q. Zhuge, and E. H.-M. Sha, "Access characteristic guided read and write cost regulation for performance improvement on flash memory," in *Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST)*, 2016, pp. 125–132.
[50] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," in *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM, 2015, pp. 177–190.
[51] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, "The evicted-address filter: A unified mechanism to address both cache pollution and thrashing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 355–366. [Online]. Available: http://doi.acm.org/10.1145/2370816.2370868
[52] A. D. Brunelle, "Block I/O layer tracing: blktrace," in *Gelato-Itanium Conference and Expo (gelato-ICE)*, 2006.
[53] J. Lienig and H. Bruemmer, *Fundamentals of Electronic Systems Design*. Springer International Publishing, 2017.
[54] E. Dubrova, *Fault-Tolerant Design*. Springer Publishing Company, Incorporated, 2013.
[55] I. Narayanan, D. Wang, M. Jeon, B. Sharma, L. Caulfield, A. Sivasubramaniam, B. Cutler, J. Liu, B. Khessib, and K. Vaid, "SSD failures in datacenters: What? when? and why?" in *Proceedings of the 9th ACM International on Systems and Storage Conference (SYSTOR)*, 2016, pp. 7:1–7:11.
[56] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash reliability in production: The expected and the unexpected," in *14th USENIX Conference on File and Storage Technologies (FAST)*, 2016, pp. 67–80.
[57] M. Kishani and H. Asadi, "Modeling impact of human errors on the data unavailability and data loss of storage systems," *IEEE Transactions on Reliability (TR)*, vol. 67, no. 3, pp. 1111–1127, Sept 2018.

Reza Salkhordeh received the B.Sc. degree in computer engineering from Ferdowsi University of Mashhad in 2011, and M.Sc. degree in computer engineering from Sharif University of Technology (SUT) in 2013. He has been a member of *Data Storage, Networks, and Processing* (DSN) lab since 2011. He was also a member of Iran National Elites Foundation from 2012 to 2015. He has been the director of Software division in HPDS corporation since 2015. He is currently a Ph.D. candidate at SUT. His research interests include operating systems, solid-state drives, memory systems, and data storage systems.

Mostafa Hadizadeh received the B.Sc. degree in computer engineering from Shahid Beheshti University (SBU), Tehran, Iran, in 2016. He is currently pursuing the M.Sc. degree in computer engineering at Sharif University of Technology (SUT), Tehran, Iran. He is a member of Data Storage, Networks, and Processing (DSN) Laboratory from 2017. From December 2016 to May 2017, he was a member of Dependable Systems Laboratory (DSL) at SUT. His research interests include computer architecture, memory systems, dependable systems and systems on chip.

Hossein Asadi (M'08, SM'14) received the B.Sc. and M.Sc. degrees in computer engineering from the SUT, Tehran, Iran, in 2000 and 2002, respectively, and the Ph.D. degree in electrical and computer engineering from Northeastern University, Boston, MA, USA, in 2007.

He was with EMC Corporation, Hopkinton, MA, USA, as a Research Scientist and Senior Hardware Engineer, from 2006 to 2009. From 2002 to 2003, he was a member of the Dependable Systems Laboratory, SUT, where he researched hardware verification techniques. From 2001 to 2002, he was a member of the Sharif Rescue Robots Group. He has been with the Department of Computer Engineering, SUT, since 2009, where he is currently a tenured Associate Professor. He is the Founder and Director of the *Data Storage, Networks, and Processing* (DSN) Laboratory, Director of Sharif *High-Performance Computing* (HPC) Center, the Director of Sharif *Information and Communications Technology Center* (ICTC), and the President of Sharif ICT Innovation Center. He spent three months in the summer 2015 as a Visiting Professor at the School of Computer and Communication Sciences at the Ecole Poly-technique Federele de Lausanne (EPFL). He is also the co-founder of HPDS corp., designing and fabricating midrange and high-end data storage systems. He has authored and co-authored more than eighty technical papers in reputed journals and conference proceedings. His current research interests include data storage systems and networks, solid-state drives, operating system support for I/O and memory management, and reconfigurable and dependable computing.

Dr. Asadi was a recipient of the Technical Award for the Best Robot Design from the International RoboCup Rescue Competition, organized by AAAI and RoboCup, a recipient of Best Paper Award at the 15th CSI International Symposium on *Computer Architecture & Digital Systems* (CADS), the Distinguished Lecturer Award from SUT in 2010, the Distinguished Researcher Award and the Distinguished Research Institute Award from SUT in 2016, and the Distinguished Technology Award from SUT in 2017. He is also recipient of Extraordinary Ability in Science visa from US Citizenship and Immigration Services in 2008. He has also served as the publication chair of several national and international conferences including CNDS2013, AISP2013, and CSSE2013 during the past four years. Most recently, he has served as a Guest Editor of IEEE Transactions on Computers, an Associate Editor of Microelectronics Reliability, a Program Co-Chair of CADS2015, and the Program Chair of CSI National Computer Conference (CSICC2017).