LBICA: A Load Balancer for I/O Cache Architectures

Saba Ahmadian, Reza Salkhordeh, and Hossein Asadi
Data Storage, Networks, and Processing (DSN) Lab, Department of Computer Engineering
Sharif University of Technology, Tehran, Iran
Email: {ahmadian, salkhordeh}@ce.sharif.edu, and asadi@sharif.edu

***Abstract*—In recent years, enterprise *Solid-State Drives* (SSDs) are used in the caching layer of high-performance servers to close the growing performance gap between processing units and storage subsystem. SSD-based I/O caching is typically not effective in workloads with burst accesses in which the caching layer itself becomes the performance bottleneck because of the large number of accesses. Existing I/O cache architectures mainly focus on maximizing the cache hit ratio while they neglect the average queue time of accesses. Previous studies suggested bypassing the cache when burst accesses are identified. These schemes, however, are *not* applicable to a general cache configuration and also result in significant performance degradation on burst accesses.**

In this paper, we propose a novel I/O cache load balancing scheme (LBICA) with adaptive write policy management to prevent the I/O cache from becoming performance bottleneck in burst accesses. Our proposal, unlike previous schemes, which disable the I/O cache or bypass the requests into the disk subsystem in burst accesses, selectively reduces the number of waiting accesses in the SSD queue and balances the load between the I/O cache and the disk subsystem while providing the maximum performance. The proposed scheme characterizes the workload based on the type of in-queue requests and assigns an effective cache write policy. We aim to bypass the accesses which 1) are served faster by the disk subsystem or 2) cannot be merged with other accesses in the I/O cache queue. Doing so, the selected requests are responded by the disk layer, preventing from overloading the I/O cache. Our evaluations on a physical system shows that LBICA reduces the load on the I/O cache by 48% and improves the performance of burst workloads by 30% compared to the latest state-of-the-art load balancing scheme.

I. Introduction

Increasing number of I/O intensive applications such as *Online Transaction Processing* (OLTP), *High Performance Computing* (HPC), web, and email applications arises the demand in data-centers for high-performance storage systems. The most common approach to improving the performance of storage systems is to employ *Solid-State Drives* (SSDs) [1] in the caching layer of the disk subsystems [2], [3], [4], [5], [6], which are mainly built upon low-performance and low-reliable *Hard Disk Drives* (HDD) [7], [8], [9] or mid-range SSDs (as shown in Fig. 1). Inclusion of SSDs in the I/O caching layer of systems improves the response time of the requests supplied by the cache, and hence, a wide range of enterprise and academic I/O cache architectures are proposed with the purpose of maximizing the hit ratio of the caching layer [10], [11], [12], [2], [13], [3], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28]. In these I/O caching schemes, mainly based on *datapath* or *push mode* cache architectures, the entire accesses are directed to the caching layer [29] and as such, the highest number of requests is responded via the caching layer to achieve the highest performance in terms of hit ratio. In addition, *non-datapath* caching schemes [29] have been proposed, which mainly focus on improving the endurance of the SSD cache with considerable performance overhead.

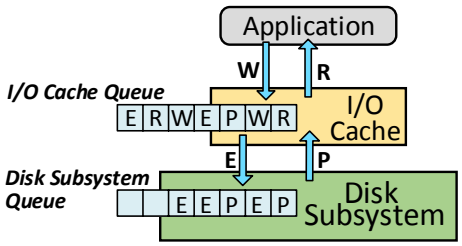


Fig. 1. I/O cache and disk subsystem queue in a write-back cache (R: Read, W: Write, P: Promote, and E: Evict).

To obtain the highest performance for both read and write accesses, storage architects offer assigning *Write Back* (WB) policy on the caching layer. Doing so, both read and write accesses are buffered in the cache, which results in improving the performance of future accesses to the same blocks. However, recent studies suggested using cache write policies other than WB, such as *Write Through* (WT), *Read Only* (RO), and *Write Only* (WO) to provide higher reliability or improve the endurance of the SSD cache while usually have performance overheads on the running workloads [2]. Using WB policy (as enterprise approaches do [10], [11], [12]), in contrast, imposes unwanted I/O load on both the caching layer and the disk subsystem (as shown in Fig. 1). For instance, each missed read request is supplied at the cost of imposing one read access on the disk subsystem and one write access to the caching layer (due to promoting new data block). This becomes worse when an eviction is required to provide free cache blocks before promoting the newer ones.

Existing caching architectures, mainly with the purpose of providing the highest hit ratio, cannot overcome the heavy I/O load of the burst accesses of applications such as boot storms, OLTP, TPC-C, mail, web, and database servers [30]. The burst accesses of such heavy workloads remain in the queue of the caching layer (since they will be served by cache, i.e., cache hit) while the disk subsystem is idle without serving any request. Thus, the caching layer becomes the performance bottleneck of the system [30]. Such poor load balancing between layers of storage hierarchy is due to 1) ignoring the queue time of the I/O requests in the caching layer and 2) not considering the impact of promotion and eviction of data blocks on the cache I/O load. Recent studies suggest I/O bypassing schemes, in which they estimate the wait time of in-queue requests and selectively direct the requests with the highest estimated latencies to the disk subsystem

[30]. Such schemes suffer from three main shortcomings: 1) they *only* consider a specific caching policy where they would not be applicable to the other types of I/O caches. In addition, such cache policy (WT and WO) is not employed in the caching layer of enterprise storage systems. This is because it only can improve the performance of read after write accesses while read and write requests experience the latency of disk subsystem. 2) The way how they select to-be-bypassed requests is inefficient, which imposes a considerable performance overhead on the operation of the queue. 3) Such schemes may bypass the requests, which were supposed to be hit in the cache while they may keep the requests in the cache queue which would not improve the hit ratio resulting in a significant performance degradation in burst accesses.

In this paper, we propose LBICA, a novel I/O cache load balancing scheme, which adaptively assigns an effective write policy to the caching layer. LBICA 1) prevents the caching layer from becoming the performance bottleneck and 2) improves the performance of the workloads with burst I/O accesses. Unlike previous load balancing schemes, which are optimized to a specific cache policy, our proposal is applicable to different caching architectures. Furthermore, LBICA 1) detects burst workloads and 2) characterizes the requests of the workloads, then adaptively assigns effective write policies to the caching layer. Doing so, we eliminate the performance overhead of selecting the to-be-bypassed requests and only bypass the accesses which 1) have not considerable impact on the overall performance and 2) cannot be merged with the other accesses in the caching layer. LBICA, using kernel level tools such as *iostat* [31] and *blktrace* [32], detects the burst intervals and also characterizes the workloads and puts them in a) *random read*, b) *random write*, c) *sequential read*, d) *sequential write*, and e) *mixed read and write* groups. We assign efficient write policy to the caching layer based on the characteristics of the running workloads in different burst intervals, which results in enhanced performance and highly utilized storage subsystem. We evaluate LBICA on a physical system including 4TB SAS 7.2K Seagate HDD in the disk subsystem level and 1TB 863a Samsung SSD in the I/O cache level. We use EnhanceIO [33], as an open-source I/O caching kernel module, to develop the proposed caching architecture. Our experimental results show that the proposed I/O cache load balancing scheme reduces the load on the I/O cache by 48% and improves the performance by 30% compared to the latest state-of-the-art load balancing scheme. We make the following contributions:

1) We propose LBICA, a novel I/O cache load balancing scheme, which effectively improves the performance and prevents the bottleneck effect of I/O caching layer in burst accesses.
2) LBICA considers the requests as a) application read, b) application write, c) cache promote, and d) cache evict, and characterizes the running workloads based on the in-queue requests type.
3) Our proposed load balancing scheme effectively assigns efficient cache write policy in different burst intervals based on the I/O characteristics of the running workloads.
4) Our proposal effectively bypasses the I/O requests from the cache while serving such request from the disk subsystem would not have any negative impact on the overall performance resulting in reduced I/O cache queue time.

II. Related Work

Existing SSD-based I/O caching schemes such as Janus [17], Hystor [13], ReCA [3], KAML [26], DIDACache [27], and ALACC [28] mainly focus on maximizing performance (in terms of hit ratio) of the running workloads. Such schemes aim to direct all I/O requests into the cache, and hence reduce the response time of the accesses, which are served by the caching layer. In addition, recent studies consider the impact of such I/O caching schemes on the lifetime of SSDs by proposing the I/O caching schemes to reduce the number of writes on the SSDs with the minimum performance overhead. Such I/O caching architectures do not provide the mechanisms to overcome the heavy load of burst I/O accesses due to OLTP or database applications. Thus, by neglecting the queue time of the requests in both I/O cache and disk queues, the caching layer behaves as a performance bottleneck resulting in a significantly large latency. However, only few studies consider the impact of queue time of the requests in the performance provided by the I/O cache. Such studies aim to enhance the average I/O latency of the workloads by balancing the load on the I/O cache and disk subsystem. *Selective I/O Bypass* (SIB) [30] is the latest state-of-the-art load balancing schemes, which aims to balance the load of I/O requests between SSD and disk subsystem preventing the I/O cache to become performance bottleneck. This scheme is designed in a way that *only* works for WT and WO caches in which only write accesses are buffered in the I/O cache while they are propagated to the disk subsystem at the same time. Such cache policy mainly aims to preserve the reliability with the following shortcomings which prevents the storage designers from employing such I/O cache scheme: 1) it only improves the performance of read accesses, which would be supplied by the cache (i.e., read after write accesses), 2) there is no any performance improvement on other accesses such as read after read (due to WO policy in which no read access is buffered in the cache) and write accesses (due to WT policy in which the accesses experience the latency of disk subsystem).

SIB selectively bypasses the I/O requests of the burst accesses into the disk subsystem. To do so, this scheme estimates the latency of in-queue requests and directs them to the disk subsystem [30]. The major disadvantages of SIB which we aim to resolve are: 1) employing the WT policy on the SSD cache. In such WT cache, since all write requests are supplied by both cache and disk in the same time, the queue size of both the cache and disk becomes the same in write-intensive workloads. In addition, due to higher delay of the disk compared to the SSD, the queue size of the disk becomes larger than that of SSD. In such condition, in case of burst accesses of a write-intensive workload, both SSD and disk become overloaded where no load balancing is possible. This is because the bypassed requests from the SSD cache will experience much larger delay in the disk queue. 2) selecting to-be-bypassed requests imposes performance and computational overhead on the system. 3) Such I/O cache policy (WT and WO) is not applicable on enterprise systems due to negligible performance improvement which only affects read after write accesses. In summary, SIB is the most close approach to LBICA with the above-mentioned shortcomings where our proposal aims to resolve them by 1) applying adaptive write policies on the I/O cache and 2) preventing unnecessary promotion (or eviction) to (or from) the cache which leads to eliminating significant

unwanted I/O load on both SSD cache and disk subsystem.

III. Proposed Method

In this section, we propose our I/O cache load balancing scheme (LBICA). As shown in Fig. 2, LBICA consists of three main procedures in which it 1) detects heavy I/O load, 2) characterizes the running workload, and 3) balances the I/O load between storage hierarchy. LBICA gets the SSD and HDD queue size and based on the in-queue requests characteristics, it decides an efficient write policy for the cache to provide both load balance and I/O performance. In Section III-A and Section III-B, we show how LBICA detects burst I/O accesses and characterizes the workload, respectively. Then in Section III-C, we elaborate the proposed load balancing scheme.

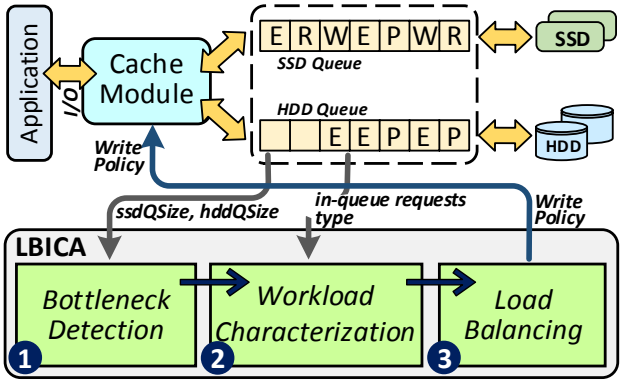


Fig. 2. The proposed I/O cache load balancing scheme (R: Read, W: Write, P: Promote, and E: Evict).

A. Bottleneck Detection

To detect the burst interval (i.e., heavy I/O load) on the storage, LBICA uses *iostat* [31] from *sysstat* package as an comprehensive kernel level I/O tool which reports the I/O statistics in the block layer. Periodically, using the given information by *iostat*, we calculate the maximum queue time of both the I/O cache and disk subsystem based on Eq. 1.

$$\begin{aligned} cache\ Qtime &= ssdQSize \times ssdLatency \\ disk\ Qtime &= hddQSize \times hddLatency \end{aligned} \quad (1)$$

Where $cache\ Qtime$ and $disk\ Qtime$ are the maximum queue time of the I/O cache and disk subsystem, respectively. $ssdQSize$ and $hddQSize$ are the current queue size of the cache and disk subsystem (i.e., the number of pending requests in the queue), respectively. $ssdLatency$ and $hddLatency$ are the average read/write latency (i.e., service time) of the employed SSD and HDD in the caching and disk subsystem level. We detect the I/O caching layer as performance bottleneck when the maximum latency of the in-queue requests in I/O cache ($cache\ Qtime$) is greater than that of the disk subsystem ($disk\ Qtime$). In this case, bypassing the in-queue requests from the cache to the disk subsystem results in improved response time. In the following we elaborate on how we select and bypass the group of the requests to the disk subsystem.

B. Workload Characterization

In case of burst I/O intervals where the I/O cache becomes the performance bottleneck, LBICA aims to characterize the running workload based on in-queue requests. We use *blktrace* [32] as a block level I/O tracing tool to get the list of in-queue requests. The in-queue requests in the I/O cache can be either read or write, where each of them may be due to 1) accesses of application (shown as R: Read and W: Write) or 2) accesses due to promotion and eviction of data blocks (shown as P and E). Based on the ratio of requests type (R: Read, W: Write, P: Promote, or E: Evict), we put the workloads in the following groups:[1]

1) **Group_1**: mainly includes the requests from R and P types (shown in Fig. 3a). Such accesses represent the behavior of a workload with *random read* access pattern in which most of the accesses are served by the cache (hit) and remaining are supplied by the disk subsystem (miss) and are promoted to the caching layer.
2) **Group_2**: mainly includes R and W types of the requests (shown in Fig. 3b). Such accesses are due to running a *mixed read write* workload in the application level where the written data blocks in the I/O cache are accessed (read) by the future requests (i.e., hit).
3) **Group_3**: mainly includes W and E requests (shown in Fig. 3c), which shows the accesses of a *write intensive* workload. In case of higher ratio of W compared to E, we detect this workload as *random write*. Otherwise, the workload is categorized as *sequential write*.
4) **Group_4**: mainly includes the requests from P type (shown in Fig. 3d). Such accesses represent a *sequential read* workload in which all read accesses are missed from cache and are promoted to the caching layer.

The remaining groups are the set of accesses with 1) majority of R and E and 2) majority of W and P. Such set of accesses may not occur during a workload execution on a storage subsystem with I/O caching architecture, and hence, we do not consider them in our proposed workload characterization.

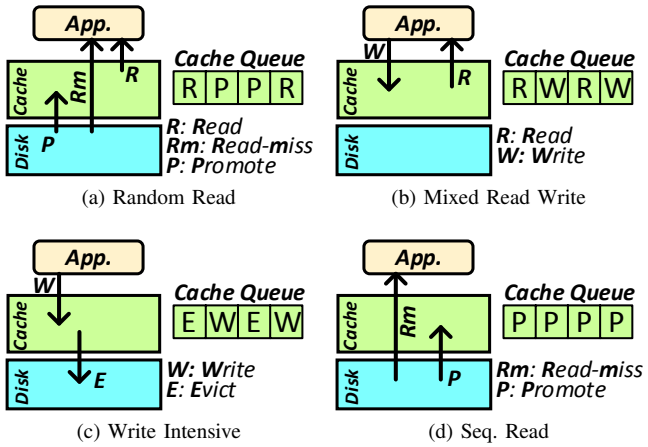


Fig. 3. Workload characterization based on in-queue requests type by LBICA.

C. Load Balancing

In case of burst accesses, to balance the I/O load between the caching layer and disk subsystem, LBICA assigns an efficient write policy to the cache based on the characteristics of the running workload (as elaborated in Section III-B). In the following, we first provide our proposed efficient cache write

[1] We assume that the workload has passed its warm-up interval.

policy assignment and then show how such scheme 1) prevents the I/O cache bottleneck effect and 2) provides maximum performance (in terms of latency) for the running workload.

1) We set the *Write Only* (WO) policy on the I/O cache when the running workload is from Group_1 category (i.e., random read workload). Such policy assignment provides the following benefits: 1) the I/O cache serves the read accesses (i.e., hit) and 2) read misses which are supplied by the disk subsystem are not promoted to the cache, reducing heavy load of writes due to promotions on the I/O cache.
2) When the running workload on the disk subsystem is categorized in Group_2 (i.e., mixed read write workload), we set *Read Only* (RO) policy on the cache. Doing so, we reduce the load on the I/O cache with bypassing the write accesses to the disk subsystem and only serve read accesses from the caching layer. The reason behind this policy is the higher priority of read accesses over writes.
3) The write policy of the cache is set to WB in the workloads from Group_3 and only the requests from the tail of the cache queue are bypassed to the disk subsystem. Such approach 1) provides the highest available performance for the requests of the cache queue which are below of the bottleneck threshold and 2) supplies the requests which are in the bottleneck threshold of the I/O cache by disk subsystem and provides smaller latency compared to the queue time of the I/O cache.
4) We set the WB policy on the cache when the running workload is from Group_4. This is because the caching layer has no impact on the supplying of sequential read accesses, which are fully responded by the disk subsystem (i.e., cache miss). In this case, the I/O cache never becomes performance bottleneck.

IV. Experimental Results

In this section, we evaluate LBICA and show how our proposed I/O cache load balancing scheme provides higher performance compared to the previous load balancing schemes. We compare LBICA with 1) baseline WB cache without load balancing and 2) SIB as the latest state-of-the-art load balancing scheme [30]. To do so, we develop both schemes and perform the same set of experiments in our physical test platform.

A. Experimental Setup

We evaluate LBICA on a physical system, where we use 4TB SAS 7.2K Seagate HDD in the disk subsystem and 1TB SSD Samsung 863a in the I/O caching layer. We implement I/O cache level by an open-source, EnhanceIO [33], kernel level module. We run different types of workloads from TPC-C, mail server, and web server with burst I/O accesses and compare the I/O load and performance of different architectures.

B. I/O Load Comparison

In this section, we compare the I/O load of the cache (in terms of queue size) in different architectures (baseline WB cache, SIB, and LBICA). Fig. 4 and Fig. 5 show the I/O load on the cache and disk subsystem for different workloads provided by WB cache, SIB, and LBICA. We monitor and report the queue time and maximum latency in intervals of 10 minutes.

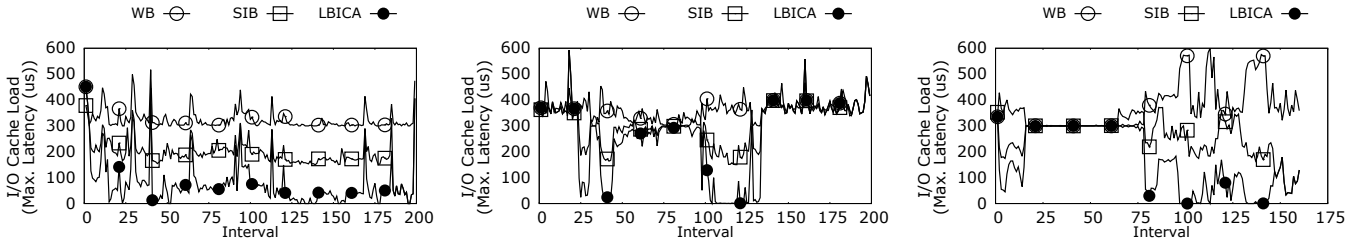
We make two main observations:

1) WB cache fails in balancing the I/O load in which the SSD cache becomes the performance bottleneck of the requests in all intervals. This is because WB policy directs all requests to the cache to achieve the maximum hit ratio. In contrast, such scheme imposes significant high I/O load on the caching layer, resulting in high I/O latency.
2) LBICA, compared to SIB, reduces the load on the I/O cache by 30%, on average. The reason behind this is that LBICA, unlike SIB, assigns an efficient write policy to the I/O cache and hence, bypasses large number of requests to the disk subsystem. The bypassed requests by LBICA to the disk subsystem are served with smaller latency than that of by I/O cache (due to the large queue time of the I/O cache).

C. Workload Characterization and Policy Assignment

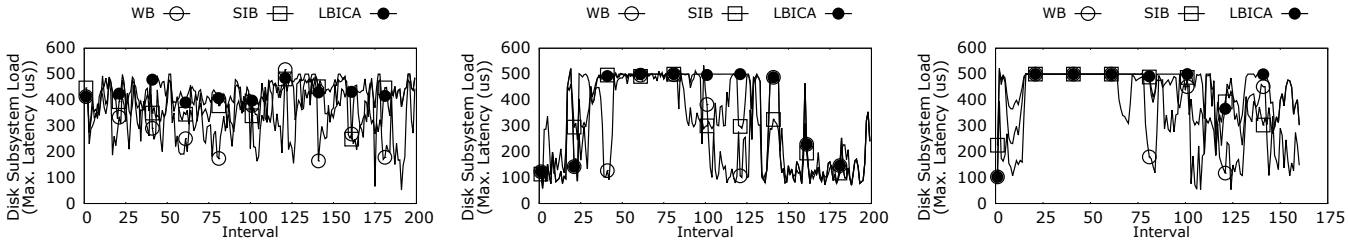
In this section, we show how LBICA detects the characteristics of the running burst workload and decides an efficient write policy in different intervals. The initial write policy of the cache is set to WB, which in the burst accesses, LBICA sets different write policies (WO and RO) based on the characteristics of the running workload. Fig. 6 shows the I/O load on the cache and disk subsystem by LBICA which a) the burst intervals, b) detected characteristics of the workload, and c) assigned write policy by LBICA are provided in this figure. We make the following observations:

1) For the TPC-C workload, at the interval of 3, LBICA reports a burst interval (in which the queue time of the I/O cache is greater than that of by disk subsystem). As shown in Fig. 6a, LBICA characterizes the running workload as *random read* due to the ratio of the accesses (R: 44%, W: 2.2%, P: 51%, and E: 2.8%). Thus, it assigns WO policy to the I/O cache. In this case, LBICA prevents buffering read misses on the cache, which contributes to 51% of the load on the I/O cache (i.e., promotions (P)). Doing so, the I/O load on the SSD is reduced by more than 50% with a negligible performance overhead.
2) For the mail server workload, at interval 23, a burst interval is detected by LBICA. Then the characterization of the workload is set to *mixed read write* because of the majority of R and W operations on the I/O cache queue (R: 13.9%, W: 70.4%, P: 3.9%, and E: 11.8%). In this case, LBICA sets the RO policy on the cache, in which the read accesses are served by cache, while the write accesses are bypassed to the disk subsystem. Thus, LBICA reduces the I/O load on the cache by 70%.
   Then at interval 128, a burst interval is detected, which is mainly due to R and P operations. Such workload is from *random read* type which LBICA assigns WO policy on the cache. In this case, read hits and read misses are served by the cache and disk subsystem, respectively, while we prevent buffering read misses in the I/O cache, resulting in about 50% load reduction on the I/O cache.
   At the rest of the experiment, at interval of 134, the I/O cache becomes the performance bottleneck. In this case, since the majority of operations in the I/O cache queue are from W and E type (about 90%), the workload is detected



(a) TPC-C (b) Mail Server (c) Web Server

Fig. 4. I/O load (in terms of max. latency) on the I/O cache by WB, SIB, and LBICA.



(a) TPC-C (b) Mail Server (c) Web Server

Fig. 5. I/O load (in terms of max. latency) on disk subsystem by WB, SIB, and LBICA.

as *write intensive*, and hence, LBICA assigns WB policy on the cache.

3) For the web server workload, at the first interval, LBICA detects the I/O cache as performance bottleneck. The majority of the accesses in the I/O cache queue are from R and W type (R: 17.9%, W: 63.8%, P: 7.9%, and E: 10.4%), which the workload is detected as *mixed read write*. LBICA sets the RO policy on the cache, and hence, reduces 63% of the load on the cache.

We conclude that LBICA, in the burst accesses, assigns an efficient write policy on the cache and reduces the I/O load on the cache up to 70% (48%, on average).

D. Performance Improvement

In this section, we compare the average performance of the running workloads by WB cache, SIB, and LBICA. Fig. 7 shows the overall latency of the workloads during the experiments. We make two main observations:

1) LBICA improves the average latency up to 22% and 11.7% compared to WB cache and SIB, respectively (14% and 7%, on average).
2) The highest performance improvement is achieved for TPC-C workload while LBICA only improves the performance of mail server by 4%. The reason behind is that LBICA assigns RO policy in the intervals 23 to 128 and bypasses 70% of requests (write accesses) to the disk subsystem, resulting in a poor performance improvement.

V. Conclusion

In this paper, we proposed LBICA, a novel I/O cache load balancing scheme, which effectively detects burst I/O accesses and assigns efficient cache write policy to prevent the I/O cache from becoming the bottleneck. Using kernel level I/O tracing tools, LBICA analyzes the I/O load on both cache and disk subsystem. Doing so, it detects the burst intervals and characterizes the running workloads based on the type of in-queue requests. We put the residing requests in the I/O cache queue into four groups: Write (W), Read (R), Promote (P), and Evict (E). Then we characterize the running workload based on the ratio of different request types in the I/O cache queue. Based on the workload characteristics, we set an efficient write policy on the I/O cache with the purpose of balancing load on both cache and disk subsystem and providing the highest performance (in terms of latency). Our evaluations on a physical system shows that LBICA reduces the load on the I/O cache by 48% and improves the performance of burst accesses by 30% compared to the latest state-of-the-art I/O cache load balancing scheme.

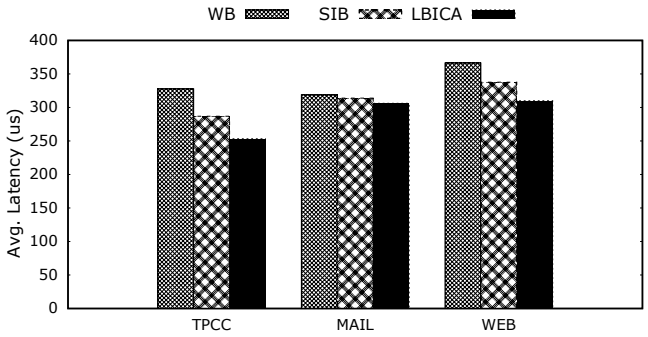


Fig. 7. Average latency achieved by WB cache, SIB, and LBICA.

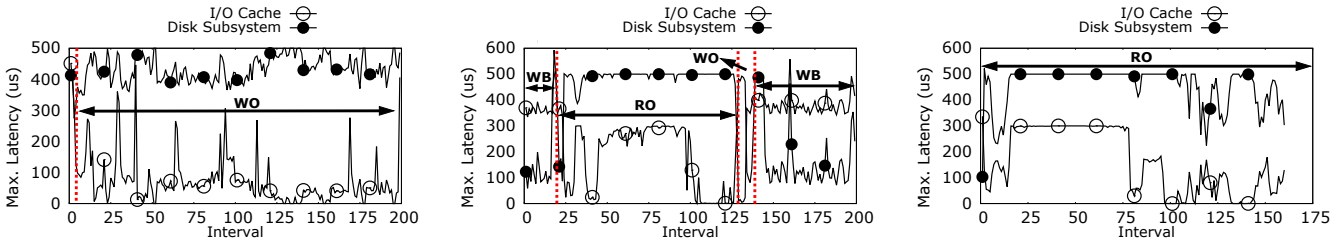
Acknowledgment

This work has been partially supported by *Iran National Science Foundation* (INSF) under grant number 9606071 and by HPDS Corp.

References

[1] 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*, vol. 65, no. 6, pp. 1678–1691, 2016.

[2] S. Ahmadian, O. Mutlu, and H. Asadi, "ECI-Cache: A High-Endurance and Cost-Efficient I/O Caching Scheme for Virtualized Platforms," *Proc. ACM Meas. Anal. Comput. Syst. (POMACS)*, vol. 2, no. 1, pp. 9:1–9:34, Apr. 2018.



(a) TPC-C (b) Mail Server (c) Web Server

Fig. 6. Workload characterization and policy assignment by LBICA in burst intervals.

[3] 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. 29, no. 7, pp. 1605–1620, July 2018.

[4] R. Salkhordeh, M. Hadizadeh, and H. Asadi, "An Efficient Hybrid I/O Caching Architecture Using Heterogeneous SSDs," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, pp. 1–1, 2018.

[5] 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.

[6] 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.

[7] 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, 2018.

[8] M. Kishani, R. Eftekhari, and H. Asadi, "Evaluating Impact of Human Errors on the Availability of Data Storage Systems," in *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*. Lausanne, Switzerland: European Design and Automation Association, 2017, pp. 314–317.

[9] M. Kishani, M. Tahoori, and H. Asadi, "Dependability Analysis of Data Storage Systems in Presence of Soft Errors," *IEEE Transactions on Reliability (TR)*, 2018 (in press).

[10] Dell EMC Corp. (2018) Dell EMC Unity: FAST Technology Overview. https://www.emc.com/collateral/white-papers/h15086-emc-unity-fast-technology-overview.pdf. [Online]. Available: https://www.emc.com/collateral/white-papers/h15086-emc-unity-fast-technology-overview.pdf

[11] HP. (2018) HPE Smart Array SR SmartCache. [Online]. Available: https://h20195.www2.hpe.com

[12] NetApp. (2017) SSD Cache Feature. https://library.netapp.com. [Online]. Available: https://library.netapp.com

[13] F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '11, 2011, pp. 22–32.

[14] F. Meng, L. Zhou, X. Ma, S. Uttamchandani, and D. Liu, "vCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment," in *Proceedings of USENIX Annual Technical Conference (USENIX ATC)*, 2014.

[15] R. Barik, J. Zhao, and V. Sarkar, "S-CAVE: Effective SSD Caching to Improve Virtual Machine Storage Performance," in *Proceedings of Parallel Architectures and Compilation Techniques (PACT)*, 2013.

[16] R. Koller, A. J. Mashtizadeh, and R. Rangaswami, "Centaur: Host-Side SSD Caching for Storage Performance Control," in *IEEE International Conference on Autonomic Computing (ICAC)*, 2015.

[17] C. Albrecht, A. Merchant, M. Stokely, M. Waliji, F. Labelle, N. Coehlo, X. Shi, and E. Schrock, "Janus: Optimal Flash Provisioning for Cloud Storage Workloads," in *Proceedings of USENIX Annual Technical Conference (USENIX ATC)*, 2013.

[18] S. Byan, J. Lentini, A. Madan, L. Pabon, M. Condict, J. Kimmel, S. Kleiman, C. Small, and M. Storer, "Mercury: Host-Side Flash Caching for the Data Center," in *IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2012.

[19] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating Server Storage to SSDs: Analysis of Tradeoffs," in *ACM European Conference on Computer Systems (EuroSys)*, 2009.

[20] 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.

[21] J. Matthews, S. Trika, D. Hensgen, R. Coulson, and K. Grimsrud, "Intel&Reg; Turbo Memory: Nonvolatile Disk Caches in the Storage Hierarchy of Mainstream Computer Systems," *Trans. Storage (TOS)*, vol. 4, no. 2, pp. 4:1–4:24, May 2008.

[22] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "HybridStore: A Cost-Efficient, High-Performance Storage System Combining SSDs and HDDs," in *IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, July 2011, pp. 227–236.

[23] C. Li, P. Shilane, F. Douglis, H. Shim, S. Smaldone, and G. Wallace, "Nitro: A Capacity-Optimized SSD Cache for Primary Storage," in *Proceedings of USENIX Annual Technical Conference (USENIX ATC)*, 2014.

[24] Y. Klonatos, T. Makatos, M. Marazakis, M. D. Flouris, and A. Bilas, "Azor: Using Two-Level Block Selection to Improve SSD-Based I/O Caches," in *IEEE Sixth International Conference on Networking, Architecture, and Storage (NAS)*, July 2011, pp. 309–318.

[25] Q. Xia and W. Xiao, "Flash-Aware High-Performance and Endurable Cache," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE 23rd International Symposium on*. IEEE, 2015, pp. 47–50.

[26] Y. Jin, H.-W. Tseng, Y. Papakonstantinou, and S. Swanson, "KAML: A Flexible, High-Performance Key-Value SSD," in *High Performance Computer Architecture (HPCA), IEEE International Symposium on*. IEEE, 2017, pp. 373–384.

[27] Z. Shen, F. Chen, Y. Jia, and Z. Shao, "DIDACache: A Deep Integration of Device and Application for Flash Based Key-Value Caching." in *File and Storage Technologies (FAST)*, 2017, pp. 391–405.

[28] Z. Cao, H. Wen, F. Wu, and D. H. Du, "ALACC: Accelerating Restore Performance of Data Deduplication Systems Using Adaptive Look-Ahead Window Assisted Chunk Caching," in *Proceedings of the 16th USENIX Conference on File and Storage Technologies*. USENIX Association, 2018, pp. 309–323.

[29] Y. Chai, Z. Du, X. Qin, and D. A. Bader, "WEC: Improving Durability of SSD Cache Drives by Caching Write-Efficient Data," *IEEE Transactions on Computers (TC)*, vol. 64, no. 11, pp. 3304–3316, Nov 2015.

[30] J. Kim, H. Roh, and S. Park, "Selective I/O Bypass and Load Balancing Method for Write-Through SSD Caching in Big Data Analytics," *IEEE Transactions on Computers (TC)*, vol. 67, no. 4, pp. 589–595, April 2018.

[31] IOSTAT. (2018) IOSTAT. https://linux.die.net/man/1/iostat. [Online]. Available: https://linux.die.net/man/1/iostat

[32] Blktrace. (2006) Blktrace: Block Layer IO Tracing Tool. https://linux.die.net/man/8/blktrace. [Online]. Available: https://linux.die.net/man/8/blktrace

[33] EnhanceIO. (2012) EnhanceIO. https://github.com/stec-inc/EnhanceIO. [Online]. Available: https://github.com/stec-inc/EnhanceIO