Microprocessors and Microsystems 78 (2020) 103217

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Cascaded Write Amplification of LSM-tree-based Key-Value Stores underlying Solid-State Disks

Hui Sun [a,b], Shangshang Dai [a], Jianzhong Huang [c,*]

[a] School of Computer Science and Technology, Anhui University, China
[b] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, China
[c] Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, China

ARTICLE INFO

Keywords:
Log-structured merge tree
Write amplification
Solid state disks
Key-value stores

ABSTRACT

Log-structured merge tree (*i.e.,* LSM-tree)-based key–value stores (i.e., KV stores) are widely used in big-data applications and provide high performance. NAND Flash-based Solid-state disks (*i.e.,* SSDs) have become a popular storage device alternative to hard disk drives (*i.e.,* HDDs) because of their high performance and low power consumption. LSM-tree KV stores with SSDs are deployed in large-scale storage systems, which aims to achieve high performance in the cloud. Write amplification in LSM-tree KV stores and NAND Flash memory in SSDs are defined as WA1 and WA2 in this paper. The former, which is attributed to compaction operations in LSM-tree-based KV stores, is a burden on I/O bandwidth between the host and the device. The latter, which results from out-place updates in NAND Flash memory, blocks user I/O requests between the host and NAND Flash memory, thereby degrading the SSD performance. Write amplification impairs the overall system performance. In this study, we explored the two-level cascaded write amplification in LSM-tree KV stores with SSDs. The cascaded write amplification is represented as WA. Our primary goal is to comprehensively study two-level cascaded write amplification on the host-side LSM-tree KV stores and the device-side SSDs. We quantitatively analyze the impact of two-level write amplification on overall performance. The cascaded write amplification is 16.44 (WA1 is 16.55; WA2 is 0.99) and 35.51 (WA1 is 16.6; WA2 is 2.14) for SSD-I and SSD-S with LevelDB's default setting under DB_bench. The larger cascaded write amplification of KV stores has a bad impact on SSD performance and lifetime. The throughput of SSD-S and SSD-I under an 80%-write workload is approximately 0.28x and 0.31x of that under a 100%-write workload. Therefore, it is important to design a novel approach to balance the cost of an SSD lifetime caused by cascaded write amplification and its high performance under the read-write-mixed workloads. We attempt to reveal details of cascaded write amplification and hope that this study is useful for developers of LSM-tree-based KV stores and SSD software stacks.

1. Introduction

LSM-tree-based key–value stores (KV stores for short) were designed to improve system performance under random-write-intensive workloads. LSM-tree can sequentially batch key–value pairs (KV pairs) to disk and sort KV pairs by multi-merge operations compared to other indexing structures, such as B-tree. A great many random writes occur, and there is even an update in B-tree [1]. KV stores, which exhibit advantages in write performance, have been widely used in big-data applications [2] in the industry, such as Bigtable [3], LevelDB [4], Hbase [5], Cassandra [6], and RocksDB [7].

Although KV stores exhibit high performance in large-scale storage systems, a compaction procedure can determine the overall system performance. A compaction procedure aims to perform merge operations for overlapped key–value ranges among sorted string table files (*i.e.,* SSTables), which merges and sorts SSTables for the following data queries. However, compaction can result in large write amplification, which is defined as the ratio of the total data written to the disk and written data in a user-level application. The performance of KV stores is largely dependent on write amplification, particularly under random-write-intensive workloads. The ideal value of write amplification is one; however, large write amplification commonly causes space costs and lower performance. In this study, write amplification that results from compaction in KV stores is defined as the first-level cascaded write amplification called WA_1.

Several large-scale cloud data centers employ NAND flash memory-based solid-state disks (*i.e.,* SSDs) as the primary storage or cache [8] to improve performance and power consumption [9]. An SSD exhibits

* Corresponding author.
E-mail addresses: sunhui@ahu.edu.cn (H. Sun), shangshangdai@stu.ahu.edu.cn (S. Dai), hjzh@hust.edu.cn (J. Huang).

https://doi.org/10.1016/j.micpro.2020.103217
Received 9 March 2020; Received in revised form 20 June 2020; Accepted 30 July 2020
Available online 8 August 2020

high write/read performance compared to a traditional hard disk (*i.e.,* HDD) [10]. An HDD is composed of magnetic components, which incur a long latency to access sectors. However, an SSD has no seek time because of the electronic NAND Flash memory inside SSDs. Therefore, SSDs exhibit high performance and reliability [11].

However, the features of erase-before-write and out-place-update operations in a NAND flash causes write amplification. Write amplification is defined as the ratio of data volume written to flash memory and that written to an SSD from the host-side operating system. Write amplification exists in the process of garbage collection (*i.e.,* GC), which will begin when the user space is smaller than the threshold. The goal of garbage collection is to reclaim invalid data space by means of copy-rewrite and block-erase operations. High-frequency GC causes lots of copy-rewrite operations in flash memory and conflicts with the arrival of external user I/O requests for data buses, thereby degrading performance and prolonging the response time of user requests. In this study, write amplification inside an SSD is referred to as the second-level cascaded write amplification, *i.e.,* WA2 for short.

For a KV store with an SSD, the overall performance is largely impaired by the two-level cascaded write amplification, *i.e.,* WA1 in the KV store and WA2 in the SSD. High WA1 means that the compaction that rewrites much more data than required impacts the performance. Read/write performance of NAND Flash-based SSDs is largely degraded by the conflicts between user I/O requests and copy-rewrite operations inside the NAND Flash memory. The write amplification of an SSD (i.e., WA2) can impair the overall performance as it increases under a random write-intensive workload.

It is meaningful to exploit two-level write amplification and its impact on the performance of the overall system. Experiment results are useful for developers who intend to integrate KV stores and firmware software inside an SSD to design efficient collaborative software stacks for KV stores with SSD, thereby fully utilizing the benefits of KV stores and SSDs.

In this study, we conducted extensive tests to study the two-level cascaded write amplification in KV stores and underlying SSDs. LevelDB and Wisckey, two popular KV stores, are employed in our work. YCSB and DB_bench are used as the macro- and micro-benchmarks that run on two types of SSDs and an HDD. We are targeted at revealing the impact of the two-level cascaded write amplification on performance in terms of write amplification and throughput by means of the following configurations: (1) two statuses (i.e., steady and initial status) in the tested SSD and HDD; (2) two types of KV stores, LevelDB and Wisckey; (3) synthetic and realistic workloads; and (4) different parameters in LSM-tree KV stores and workloads. The overall write amplification (*i.e., WA*) represents the write amplification of the overall system, which multiplies a KV store's write amplification (WA1) by the SSD-level write amplification (WA2).

We carry out extensive experiments in this paper. Then, we find that the cascaded write amplification has a significant impact on the performance and lifetime of an SSD with KV stores under DB_bench and YCSB-C.

Under DB_bench, the overall cascaded write amplification (i.e., WA) of LevelDB is 62.50 (WA1 is 18.44; WA2 is 3.39), 22.80 (WA1 is 18.46; WA2 is 1.24), and 18.15 (WA1 is 18.15; WA2 is 1.00). Meanwhile, the cascaded WA of Wisckey is 10.44 (WA1 is 5.20; WA2 is 2.01), 5.17 (WA1 is 5.15; WA2 is 1.00), and 5.16 (WA1 is 5.16; WA2 is 1.00) on the SSD-S, SSD-I, and HDD under the fillrandom-load workload, respectively. In addition, the throughput of the SSD-S and SSD-I with LevelDB is 2.2 MB/s and 3.2 MB/s, respectively. The throughput of the two tested SSDs with Wisckey is 5.6 MB/s and 5.9 MB/s, respectively. Under YCSB-C, the cascaded WA of SSD-S and SSD-I with LevelDB is 33.89 (WA1 is 9.44; WA2 is 3.59) and 15.07 (WA1 is 9.40; WA2 is 1.60), respectively. The throughput of SSD-S and SSD-I with LevelDB is 31.59 kops/s and 52.72 kops/s under Zipfian workload with 100%-write configuration. Under Zipfian workload with 100%-write configuration, the WA of SSD-S and SSD-I with Wisckey is 6.44 (WA1 is 3.11; WA2 is 2.07) and 3.48 (WA1 is 3.13; WA2 is 1.10), respectively. The throughput of the two SSDs is 73.22 kops/s and 82.36 kops/s, respectively.

The cascaded write amplification of the overall system is determined by the application-level write amplification (WA1). However, WA2 is also important for the overall write amplification. This write amplification of KV stores has a largely bad impact on the SSD lifetime. Please refer to Sections 5 and 6 for more details on the experimental results.

2. Background

Log-structured merge tree (i.e., LSM-tree), which is a disk-based data structure, sorts random-write data by grouping them as a data component in memory. LSM-tree is consisted of one component in memory and multiple ones on disk. LSM-tree-based key–value stores are employed to manage unstructured data because of their sequential high-performance [12] in the big data era. Numerous prominent IT companies have built their data infrastructures using LSM-tree-based key–value stores. In addition, NAND Flash-based solid-state disks (SSDs) can also provide high performance and low latency under sequential-write workloads, compared with random-write workloads. To take advantage of LSM-tree and SSDs, LSM-tree-based key–value stores are widely used in large-scale storage systems based on SSDs [13, 14]. However, write amplification exists in key–value stores and NAND Flash-based SSDs as mentioned in Section 1. In this section, we present details of KV stores and write amplification in KV stores and NAND Flash-based SSDs.

2.1. LSM-tree-based KV stores

Fig. 1(a) illustrates the architecture of the Log-structured Merge Tree (*i.e.,* LSM-tree) comprising one component (C_0) in memory and multiple components on disk (C_1... C_{n-1}, and C_n).

The memory component (C_0) is used to store data written by the user in the main memory and to convert a large number of random write operations into batch sequential write operations, which can improve the random write performance on the disk. When the memory component approaches its predefined size, it is dumped into a Sorted-String Table (i.e., SSTable) in disk component C_1 for persistent storage, while a new memory component is created to continue processing requests from applications.

The disk component provides persistent storage space for C_0, which has been designed as a hierarchical structure (C_1, C_2, …, C_n). Each level has a different number of SSTables, which are organized into a series of levels that grow exponentially in size.

When a new KV pair is inserted into a KV store, data is appended to the log file on the disk. Afterwards, the KV pair is written into the component C_0 in which the recent data research can be conducted in a fixed size in memory. When the component size is larger than the threshold, data is dumped to component C_1 on the disk by means of a rolling merge procedure.

LSM-tree provides high-performance retrieval operations because of the latest data existing in C_i, followed by C_1 and C_2. To retrieve a KV pair, LSM-tree searches for it from C_0 to C_k until it finds or reaches the last component. LSM-tree executes many more read operations to obtain the targeted KV pairs. Thus, LSM-tree is suitable for application scenarios with many write operations but few read operations.

When the size of component C_i reaches its threshold, entries in component C_i having a key-range that overlaps with that in C_{i+1} move to memory and perform rolling merge processes. Afterwards, these newly no-overlapped entries are dumped into component C_{i+1}. This process is called compaction, and it can trigger write amplification and degrade overall performance.

Using the benefits of the log-structured merge tree, the advantage of LSM-tree KV stores significantly improves disk write performance because random data is sequentially written to disk. However, LSM-tree KV stores can produce a large number of write I/Os in the process of compaction, which influences the ongoing write/read operations, thereby degrading the overall performance of storage systems.

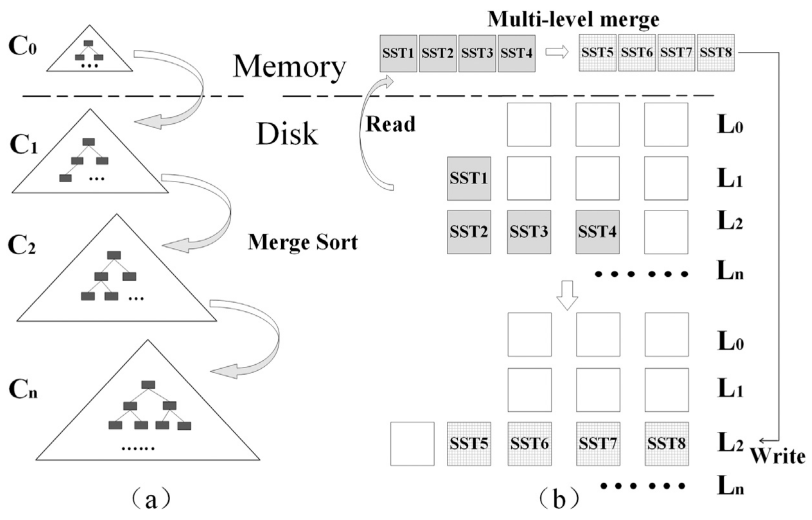


Fig. 1. The architecture (a) and compaction process (b) in LSM-tree based key–value stores and LevelDB. It must be noted that the notation SST means a Sorted-String Table or SSTable. These SSTables SST1, SST2, SST3, and SST4 are involved in the process of compaction. Meanwhile, SST5, SST6, SST7, and SST8 represent new SSTables after the compaction procedure.

2.2. LevelDB and compaction

LevelDB is a popular extension of LSM-tree. We demonstrate the structure and compaction procedure in LevelDB.

In LevelDB, one component is in memory, and multiple components are stored on the disk, as shown in Fig. 1, LevelDB includes six types of files, i.e., MemTable and Immutable MemTable in memory and the current file, Manifest file, log file, and SSTables on the disk. An arrival KV pair is first inserted into a log file and then written into a MemTable in memory. MemTable turns into Immutable Memtable when it reaches the threshold; then, it is configured into a Sorted-String Table (i.e., SSTable) and dumped to the disk. LevelDB stores SSTables on the disk in a multiple-level manner, from level 0 to level L_k. The threshold at a low level is ten times that at a high level. When the size of a level exceeds its threshold, LevelDB selects SSTables from a low level and compacts them with SSTables that have overlapped key-ranges at a high level. A new SSTable is placed at a high level, such as L_0. In terms of performance, different SSTables exhibit overlapping ranges in level L_0 rather than other levels. An SSTable is migrated from level L_k to level L_{k+1} through a process called **compaction**.

In Fig. 1, three SST files overlap with a victim SST in key range, namely, SST2, SST3, and SST4. LevelDB moves these SSTs into memory, sorts them by a multi-level merge, and produces new SSTables, such as SST5, SST6, SST7, and SST8. Then, they are rewritten to level L2. In a compaction procedure, the SST overlapped with the victim SSTs is rewritten, the process of which causes write magnification. LSM-tree achieves high write performance by moving the batch order of KV to the disk. To support the retrieval and provide reliable read performance, LSM-tree continuously compacts KV throughout the entire life cycle to achieve sorted KV pairs. As the scale of the data increases, write amplification rises up caused, the compaction procedure [15,16].

2.3. Garbage collection in SSDs

Compared with an HDD, an SSD, having no mechanical components, consists of a controller, DRAM, flash memory packages, etc. An SSD has no seek latency and directly accesses data by page-, block- or hybrid-level mapping schemes. High performance is provided by SSDs, but the latency of read operations is one order of magnitude lower than that of write and erase operations [17].

In NAND Flash memory, a block, which consists of a fixed number of pages, is the basic unit of an erase operation. The unit of a read/write (program) operation is a page, which is empty and valid before writing data to it.

Updated data is written in an erased block, and the old data page is set to be invalid, which is called out-place update rather than in-place update. In Fig. 2, when there is no efficient free page in a block (block x and block y), the valid pages in these blocks move to an erased block (block z), and then the blocks with invalid data pages are erased. This process is called garbage collection and can cause write amplification in SSDs.

2.4. Write amplification

Write amplification initially appeared in publications of Intel and SiliconSystems [18]. The term is usually referred to as a key factor restricting random write performance and data persistence in NAND Flash-based SSDs [19].

Write amplification is a determining factor of the performance of NAND Flash-based SSDs. Write amplification is one when an SSD is in the initial state or under a sequential workload. Then, the amount of data written by the host to the SSD is similar to that written to flash memory. When garbage collection frequently occurs and the user data space reduces, out-place updated operations increase. Thus, the amount of data written to the flash memory enlarges, thereby causing large write amplification. There are a limited number of program and erase operations in Single-Level Cell (SLC), Multi-Level Cell (MLC), and Triple-Level Cell (TLC)-based NAND Flash memory [20]. The costs of the program and erase operations shorten SSD lifetime [21]. In addition, copy-rewrite operations conduct data transfer from/to flash memory, which aggregates the burden on bandwidth between the host and device. The real-time response of user requests is then blocked to reduce write performance [22].

3. Related work

The idea of log-structured first comes from Rosenblum's work in 1992 [23]. An entire disk is considered as a log space to store persistent data and perform a search in the log. Newly inserted data is appended to the log, such that the access mode of data is sequential, which increases the bandwidth of the disk. O'Neil et al. [24], inspired by the data structure of B-Tree, proposed the LSM-tree structure to improve random write performance.

To address the compaction of LSM-tree KV stores, researchers have conducted much research on applications, such as Wisckey, Pebblesdb, LSM-tire, etc.

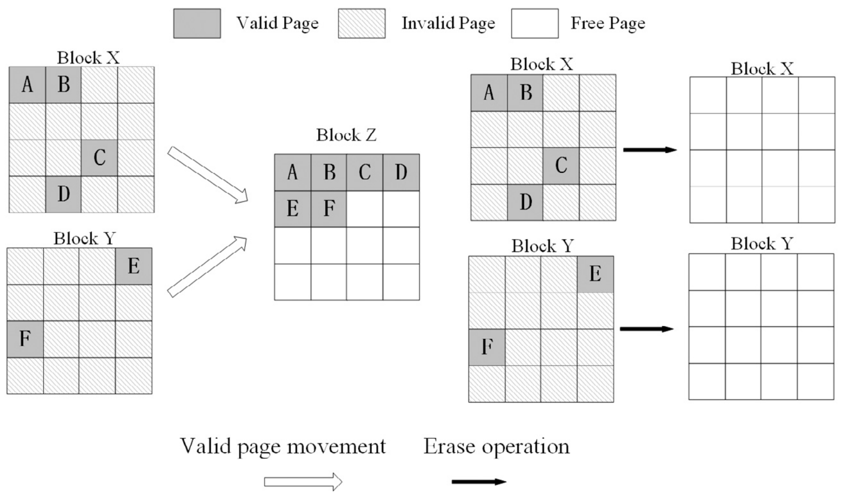


Fig. 2. The process of garbage collection (GC), which is the main reason for Write amplification inside SSD. The block X and Y are the victim blocks in GC. During GC, the valid data moves to the free block Z, then, the victim blocks will be erased.

Wisckey [13] is proposed to optimize the performance of key–value stores using high-performance SSDs. It separately stores key and value to reduce write amplification. Generally, the size of a key in a KV pair is smaller than that of the value. Wisckey involves keys in a compaction procedure, and the values are stored in log files that are not involved in compaction, thereby reducing write amplification.

Pebblesdb [25] is a new KV store based on the skip list and fragmented log-structured merge trees (FLSM). Except for the level L0, FLSM divides each level into different segments, each of which is set to be a guard. There are no overlapping key ranges between two guards, which keeps them in order, but the keys in a guard can be disordered. This scheme enables SSTables in a given guard to be sorted and then fragmented at level i. The guard at level (i+1) only receives a new SSTable that fits into the key range and does not rewrite all SSTables at level (i+1).

LSM-trie [12], a KV store, is proposed to reduce write amplification. LSM-trie constructs a trie or a prefix tree that stores data in a hierarchical structure and reorganizes them using an efficient compaction method.

Researchers have studied key–value stores on the basic different storage medium in the devices, such as BerkeleyDB [26], HashCache [27], memcachedb [28], and MongoDB [29], which have been proposed to improve the performance of key–value store with HDD being employed as the storage devices. GearDB aims to improve the SMR-based key–value stores. Meanwhile, SILT [30], FAWN [31], SkimpyStash [32], LOCS, Flashield, and FlashStore are proposed to improve NAND flash memory-based key–value stores. However, NVMKV and KVFTL are targeted at implementing a new flash translation layer for key–value stores. Some special work is presented as follows.

Due to the high capacity and low price of SMR, it can be an effective method for deploying LSM-tree-based key–value stores in SMR. However, the degradation of performance in SMR, which results from the garbage collection, cannot be ignored in real-world applications. Yao et al. [33] proposed a GC-free KV store tailored for HMSMR drives, GearDB, which largely improves the overall performance of key–value stores in SMR.

To incorporate a multi-channel SSD in KV stores, Wang [34] designed a KV store by means of a configurable open-channel SSD called LOCS. The KV store can directly access data in NAND flash memory by multiple channels.

Meanwhile, the scheduling and dispatching scheme is optimized for concurrent I/O requests, which improves the performance of data access. A scheduler existing between APIs and LevelDB is designed to dispatch requests from LevelDB to channels of a customized SSD. In addition, the erase operation and garbage collection are also managed by this scheduler. Requests from LevelDB call APIs to conduct read, write, and erase operations. The scheduler takes over the scheduling and dispatching functions, which simplify execution units in the SSD controller. Thus, the high-performance multi-channel SSD improves overall performance in KV stores.

Assaf Eisenman et al. [35] revealed that the limited lifetime of SSDs prevents their worldwide employment, for example, key-value cache. This is because key-value cache frequently performs insert, update, and deletion of small objects, which causes excessive write and erase operations that shorten the SSD lifetime. Flashield, which is a hybrid key-value cache, uses DRAM as a filter to improve the data volume written to an SSD. Flashield performs lightweight machine learning admission control to predict objects that may be frequently read without updates and ones that are prime candidates to be stored on SSDs.

Biplob Debnath et al. proposed FlashStore [36], which is a high-performance and persistent key-value store that uses NAND flash memory as the non-volatile cache between DRAM and HDD. NAND Flash is used to store the hot data of KV pairs. For each lookup operation for keys, only one read operation on NAND FLASH is performed. The hot data set changes with the runtime of workloads. Flashstore moves the cold data in NAND flash into HDD, which can reserve space for the new hot data.

Leonardo Marmo et al. proposed NVMKV [37], which is a scalable and lightweight KV store that leverages advanced capabilities that are becoming available in modern FTLs.

Yen et al. [38] found that the space management strategies for the fix-sized KV pair can cause low utilization of space in SSDs. Using key–value-specific solid state drives (KVSSD), they proposed a key–value flash translation layer (i.e., KVFTL). The partitioning method in the storage level is designed. The space inside an SSD is variable based on the fixed pages and can store KV pairs with different sizes.

Currently, there is research work to optimize the system performance based on KV store and storage medium, such as KVSSD, KVFTL, owing to the characteristics of the host end and device. However, work is still in the initial stage. For this reason, we study the cascaded write amplification between the host and SSD-based devices for KV stores.

Samsung first released key–value SSDs in 2017. The existing storage architecture is considered to be transformed to the KV interface instead of the current block interface. It needs to transform the KV request to LBA and, then converting to PBA. This process consumes storage performance and resources. The KV SSD cuts off unnecessary software and hardware stacks. Therefore, applications can directly access the SSD, which greatly improves the performance of SSD-based storage systems.

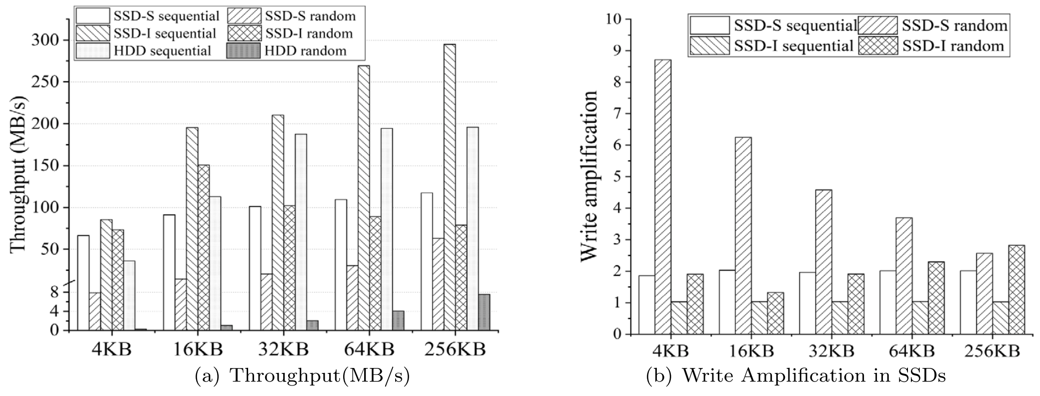


Fig. 3. Throughput and Write Amplification inside SSDs of SSD-I, SSD-S, and HDD in sequential and random work load tested by iometer.

Sung et al. [39] proposed KVSSD to decrease cascaded write amplification caused by the LSM-tree, host file system, and garbage collection inside an SSD. KVSSD integrates an LSM-tree and flash translation layer inside an SSD to improve write amplification in terms of two levels. LSM-tree with SSDs produces three-level write amplification. The first is write amplification of the LSM-tree due to the compaction of SSTables with overlapped key ranges. The second is caused by the out-place update operations in flash memory inside an SSD. The last one is the copy-rewrite during garbage collection. KVSSD performs copy-free compaction in LSM-tree by means of a new mapping scheme in FTL, which can directly distribute data in flash memory, thus achieving an efficient garbage collection and low write amplification.

Shannon System [40] is a storage company that has designed a key–value SSD [41] based on the open channel system. This was done to eliminate the write amplification overhead caused by compaction, file system, block level, etc. in KV stores. The traditional storage architecture of KV stores is changed to reduce the overhead between each storage level. Key–value pairs are directly written to the SSD, bypassing the database, file system, block device layer, etc. The key–value stores' performance is 40x better than that of Rocksdb.

4. Experimental setup

In this section, we illustrate the experimental setup and method for studying cascaded write amplification.

4.1. Software and hardware

The platform is composed of a server and three types of tested disks. The server is configured with four Intel Core(TM) i5-7400 CPU @ 3.00 GHz, 4 GB of DRAM, and a 1 TB TOSHIBA HDD. Ubuntu 16.04 LTS with the 4.4.0-142-generic version kernel is employed as the operating system, and ext4 is used as the file system. Two types of SSDs and an HDD are employed in the test. Details of the tested disks are listed in Table 1. We used Iometer [42] to conduct experiments on raw tested disks to study the performance and write amplification, see Fig. 3. Under sequential-write workloads with 256-KB I/Os, the maximum throughput is 295 MB/s, 117 MB/s, and 196 MB/s for the SSD-I, SSD-S, and HDD, respectively. There is a difference of more than one order of magnitude in throughput for the HDD between sequential- and random-write workloads. Fig. 3(b) shows that write amplification of an SSD is always approximately one under a sequential-write workload for the SSD-I and SSD-S. Under the random-write workload, write amplification is larger than that under the sequential-write workload for the SSD-I and SSD-S.

DB_bench and YCSB are employed as workloads while LevelDB and Wisckey are used as tested key–value stores. Snappy in both KV stores is closed to study the impact of the written data on write amplification and throughput. DB_bench, a default micro-benchmark in LevelDB, is a popular tool for testing KV stores. It provides various benchmarks and state information of databases. Meanwhile, db_bench can easily be configured by parameters for a tested database. To implement our work in realistic workloads, YCSB [43] is used to evaluate the performance of SSDs with KV stores. To accurately evaluate the effects of a write-intensive workload on the LSM-tree, we reconfigured several parameters in YCSB. Two types of user-defined parameters in the workload are used, *e.g.,* the ratio of read and write and the record size. We used a low-cost C++ version of YCSB-C [44,45] in this study.

Table 1
Tested disks in this study.

Brand	Model	Medium	Interface	Capacity (GB)
Intel	545s Series	TLC	SATA-III	256
Sandisk	SSD Plus	TLC	SATA-III	240
Toshiba	DT01ACA100	–	SATA-III	512

LevelDB, which is a widely popular KV store extended from BigTable in Google, supports range queries, snapshots, batch sequential write, etc. LevelDB is open source, and many researchers have worked on it. Wisckey, which is an extended version of LevelDB, aims to improve the write amplification of LSM-tree on SSDs. Wisckey separates key and value in the LSM-tree so that the value does not participate in compaction. Only keys are involved in compaction; thus, the performance of compaction is improved under random-intensive workloads.

4.2. Evaluation method

We obtain the host-level write amplification by modifying the KV store source code. It is an intrinsic property of the LSM-tree KV stores. The device-level write amplification that is caused by the out-place-update operations in flash memory inside SSDs can be measured by S.M.A.R.T information. Cascaded write amplification is consist of two types of write amplification. We study the influence on the write amplification of the overall KV stores.

An SSD has two states [46], *i.e.,* initial and steady states. In the initial state, SSDs exhibit good performance because there is sufficient user space and log space, which well accommodates user write I/Os. In the steady state, an SSD evolves into a state of steady performance, which is determined by workloads. The performance of a two-state SSD is different. Therefore, we set tested SSDs in the initial and the steady states to study the performance of KV stores with SSDs.

Write amplification includes three types of values, WA1, WA2, and WA, WA1 which results from a compaction procedure, indicates the write amplification of a KV store. WA1, which is considered as an application-level metric, is denoted as

$$WA1 = \frac{HostWriteData}{UserWriteData}, \tag{1}$$

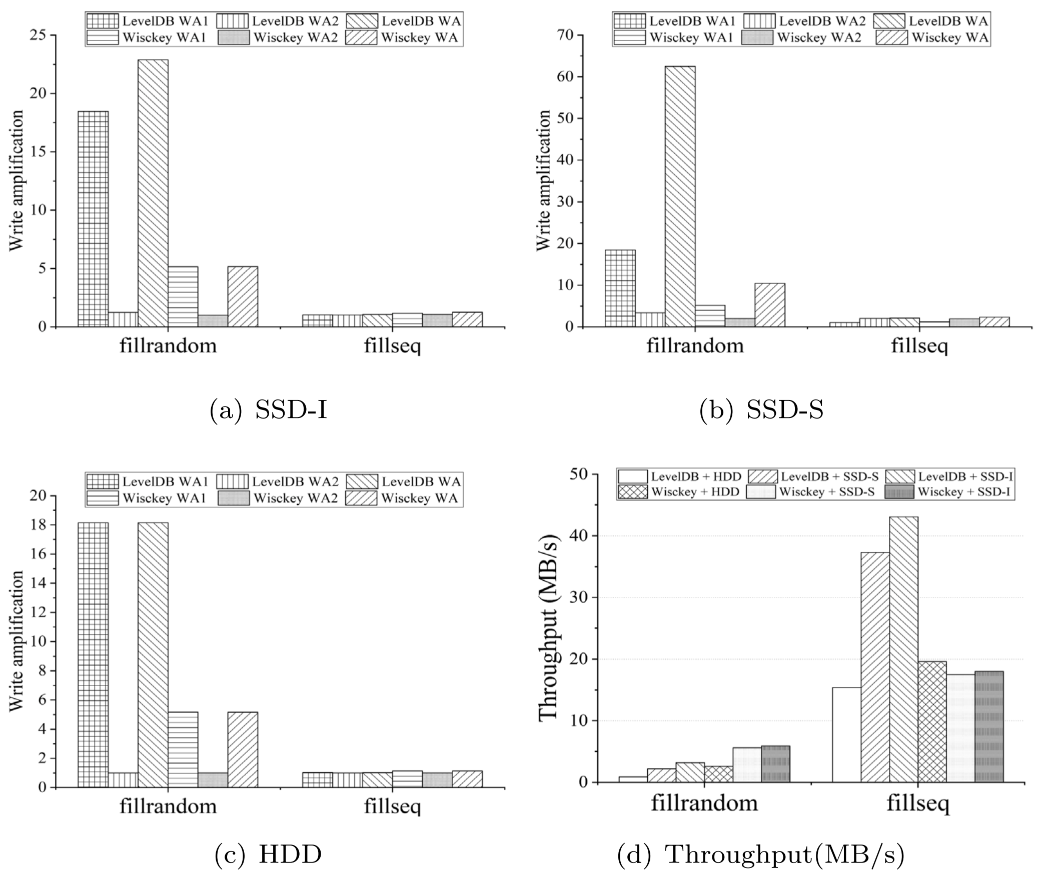


Fig. 4. Write Amplification and Throughput of the SSD-I, SSD-S, and HDD under DB_bench. It must be noted that two types of LSM-tree-based key–value stores (i.e., LevelDB and Wisckey), two types of workloads (i.e., Fillseq and Fillrandom), three tested metrics for write amplification (i.e., WA1, WA2, and WA), and throughput (MB/s) are employed in this test. The notations WA1, WA2, and WA represent application-level write amplification, write amplification of a tested SSD (or HDD), and the cascaded write amplification of the overall system, respectively. In Fig. 4(d), the notation LevelDB+HDD means that LevelDB runs on the tested HDD.

Table 2
Workload characteristics in DB_BENCH.

DB_bench type	Benchmarks	Data volume (GB)	Value_size	Write_buffer_size (MB)	SSTable_size (MB)	Block_size (KB)
dbw1	fillseq	54	100 B	4	2	4
dbw2	fillrandom					
dbw3	fillrandom	27	100 B	4	2	4
dbw4		54				
dbw5	fillrandom	30	16 B	4	2	4
dbw6			100 B			
dbw7			16 KB			
dbw8			64 KB			
dbw9			1 MB			
dbw10	fillrandom	30	100 B	4	2	4
dbw11				16		
dbw12				32		
dbw13				64		
dbw14	fillrandom	30	100 B	4	0.5	4
dbw15					2	
dbw16					8	
dbw17					16	
dbw18	fillrandom	30	100 B	4	2	4
dbw19						9
dbw20						18
dbw21						24
dbw22						36

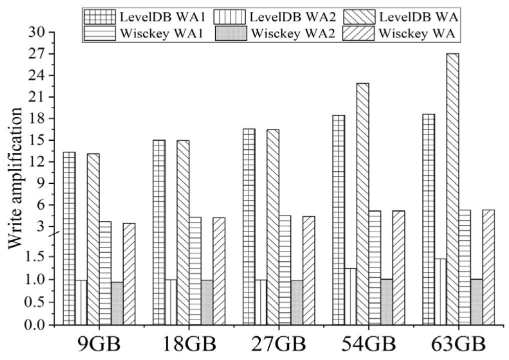
where *HostWriteData* represents the data volume issued to disk. Its value can be obtained by a tool named *iostate* in Linux. *UserWriteData*, which is the size of a KV pair multiplied by the number of KV pairs, is the total written by an application.

WA2, which represents the write amplification of a tested SSD, is a device-level write amplification and given as

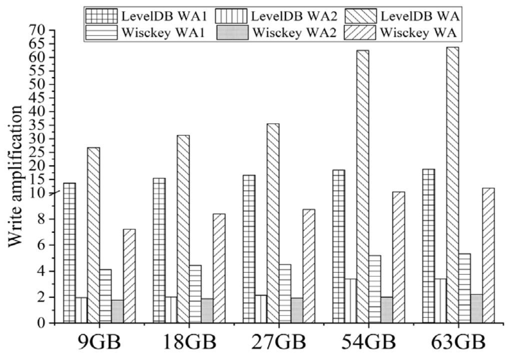
$$WA2 = \frac{NAND\ Write\ Data}{Host\ Write\ Data}, \tag{2}$$

Table 3
Workload characteristics in YCSB.

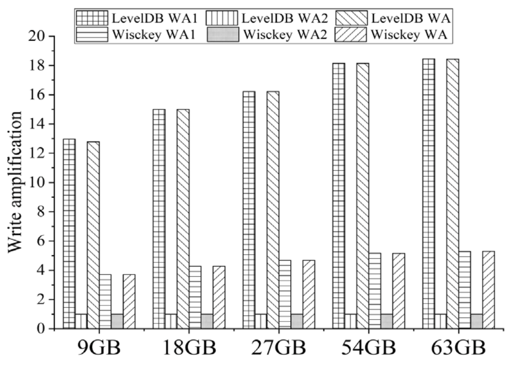
YCSB type	Distribution	Data volume (GB)	Value_size	Write (%)
ycsbw1			16 B	
ycsbw2			100 B	
ycsbw3	zipfian	30	16 KB	100
ycsbw4			64 KB	
ycsbw5			1 MB	
ycsbw6	zipfian	30	100 B	100
ycsbw7	uniform			
ycsbw8				100
ycsbw9	zipfian	30	100 B	80
ycsbw10				60
ycsbw11				40



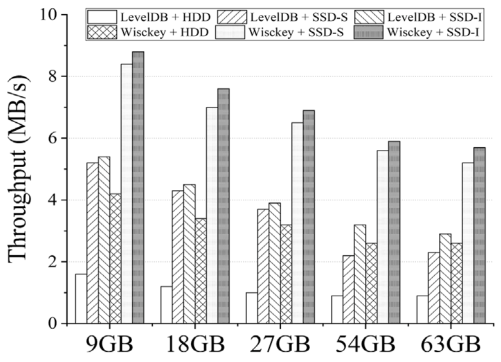
(a) SSD-I



(b) SSD-S



(c) HDD



(d) Throughput(MB/s)

Fig. 5. WA and Throughput of the SSD-I, SSD-S, and HDD under different Data Volume in DB_bench. It must be noted that two types of LSM-tree-based key–value stores (i.e., LevelDB and Wisckey), five types of data volume (i.e., 9 GB, 18 GB, 27 GB, 54 GB and 63 GB), three tested metrics for write amplification (i.e., WA1, WA2, and WA), and throughput (MB/s) are employed in this test. Notations WA1, WA2, and WA represent application-level write amplification, write amplification of a tested SSD (or HDD), and the cascaded write amplification of the overall system, respectively. In Fig. 5(c), the notation LevelDB+HDD means that LevelDB runs on the tested HDD.

where *NANDWriteData*, which is the data volume written by the SSD controller to NAND Flash memory, is obtained by S.M.A.R.T information.

In addition, data in an HDD can be overwritten, which is considered to have no write amplification compared with an SSD; thus, write amplification of the tested HDD is set to one.

WA, which is the cascaded write amplification, is a system-level metric that is used to evaluate write amplification of the overall system. WA is defined as

$$WA = WA1 \times WA2 \quad (3)$$

We evaluate the performance of KV stores with SSDs in terms of throughput, i.e., MB/s for DB_bench and Kops/s under YCSB, respectively.

Various parameters were configured in DB_bench and YCSB. To obtain the same tested environment, we conducted ATA Secure Erase commands [47] on SSDs to recover the initial state before each test.

5. Performance evaluation on DB_Bench

5.1. Fillseq and fillrandom workload

In this test, we evaluate write amplification in two types of KV stores with three disks under 54-GB sequential- and random-load workloads (see Fig. 4). An HDD is used as the baseline in our experiments. In LevelDB and Wisckey, WA1 is around one under fillseq-load workload on the SSD-S, SSD-I, and HDD. The result is attributed to there being no compaction triggered under this workload. Thus, the user-written data volume is similar to that written by the host. WA1 of LevelDB with

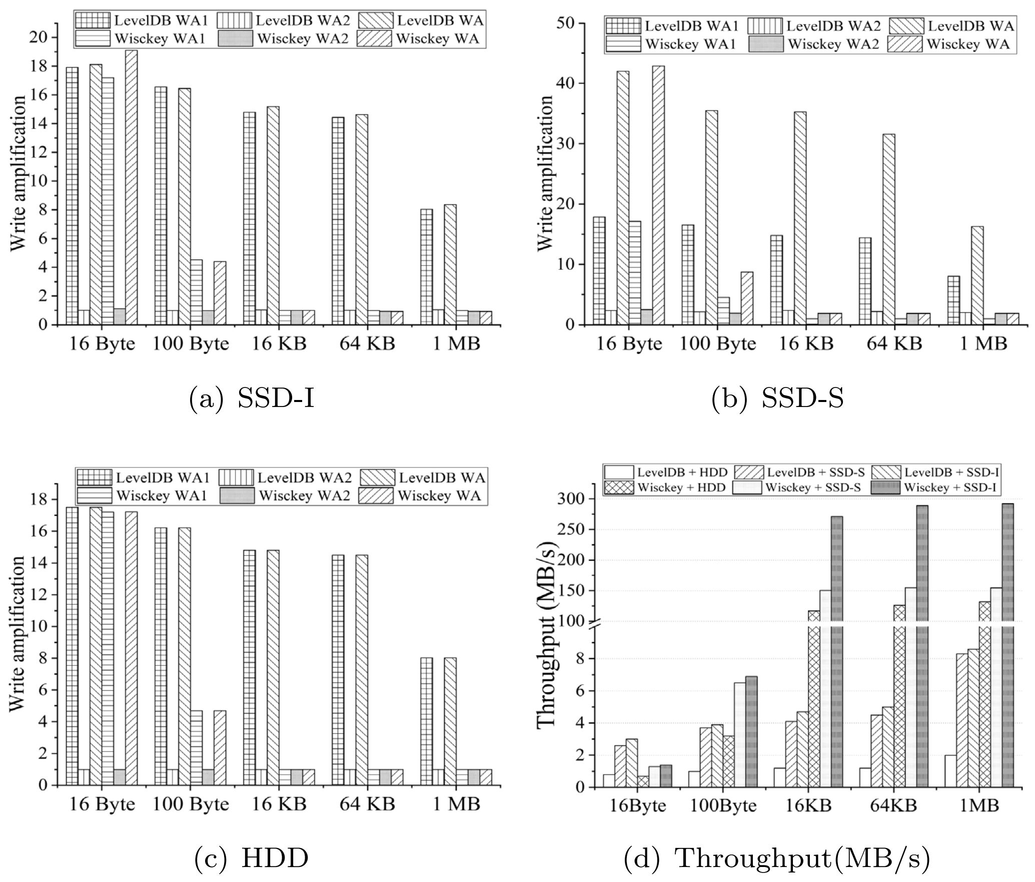


Fig. 6. WA and Throughput of the SSD-I, SSD-S, and HDD under different value size in DB_bench. It must be noted that two types of LSM-tree-based key–value stores (i.e., LevelDB and Wisckey), five types of value size (i.e., 16 B, 100 Byte, 16 KB, 64 KB and 1 MB), three tested metrics for write amplification (i.e., WA1, WA2, and WA), and throughput (MB/s) are employed in this test. Notations WA1, WA2, and WA represent application-level write amplification, write amplification of a tested SSD (or HDD), and the cascaded write amplification of the overall system, respectively. In Fig. 6(c), the notation LevelDB+HDD means that LevelDB runs on the tested HDD.

the SSD-I, SSD-S, and HDD approaches 18.46, 18.44, and 18.15, respectively under the random-load workload. In Wisckey, write amplification presents 5.15, 5.20, and 5.16 on the SSD-I, SSD-S, and HDD under the same workload. The reason is that compaction is triggered, which increases the write amplification. It is observed that KV stores with the SSD-I reduce write amplification by up to approximately 50.0% under the random-load workload compared to the SSD-S. Afterwards, we primarily focus on experiments under random-load workload to study the performance of KV stores with SSDs in terms of write amplification and throughput. The controller of the SSD-I is featured with data compression when data is written into flash memory [48,49], which further reduces write amplification.

Figs. 4(a) and 4(b) show that KV stores with SSDs exhibit good performance in the initial state of SSDs. For example, WA2 of the SSD-I is around one under the sequential-load workload; WA2 of the SSD-S is between two and three.

Fig. 4(b) present that WA2 of LevelDB with SSDs is larger than that of Wisckey under the random-load workload because LevelDB has a higher WA1 than Wisckey; therefore, much data is loaded onto the disk. Garbage collection is frequently triggered because of there being less user space on the disk. LevelDB has a write amplification close to that of Wisckey under the sequential-load workload.

In Fig. 4, the WA of LevelDB and Wisckey is 62.5 and 10.0 on the SSD-S, respectively. LevelDB and Wisckey presents 22.8 and 5.17 on the SSD-I, respectively.

We observe that the overall cascaded write amplification (i.e., WA) for the SSD-I and SSD-S is different even though WA1 and WA2 have no significant changes. Then, the overall performance is impaired by the cascaded write amplification.

5.2. *Various data volume (value = 100 Bytes, key = 16 Bytes)*

In this test, we exploit the performance of KV stores with SSDs according to various data volume in DB_bench. It must be noted that the written data volume is represented by the number of KV pairs, such as 5.0×10^8 (i.e., 54-GB data volume) and 2.5×10^8 (i.e., 27-GB data volume), respectively.

Fig. 5 shows that the values of WA1 on the SSD-I, SSD-S, and HDD with LevelDB and Wisckey present similar results. WA1 increases with the size of the data volume for two KV stores. LevelDB has 1.1x, 1.1x, and 1.1x increments of write amplification for the SSD-I, SSD-S, and HDD, respectively, under the 54-GB workload. However, Wisckey with the SSD-I, SSD-S, and HDD presents 1.1x, 1.2x, and 1.1x improvements in write amplification under the same workload. A large data volume can enlarge WA2 on SSDs. For LevelDB with the SSD-S, WA2 is 2.15 and 3.39 under the 27- and 54-GB data volume workloads, respectively. These results can be attributed to the compaction procedure at different levels in LevelDB and Wisckey. When the data volume in a level reaches its threshold, the compaction procedure is triggered to compact SSTables with overlapped key ranges. The data volume written to disk increases, thereby enhancing write amplification in a KV store. Under random-intensive workloads, the possibility of garbage collection in SSDs becomes high, which enlarges write amplification, particularly as the data volume written to SSDs increases.

Fig. 5 shows that the cascaded write amplification (i.e., WA) of LevelDB is 35.5 and 16.4 under the 27-GB data volume workload on the SSD-S and SSD-I, respectively. This metric of LevelDB increase to 62.5 and 22.8 under 54-GB data volume workload on the SSD-S and SSD-I, respectively. It must be noted that even though the value of WA1

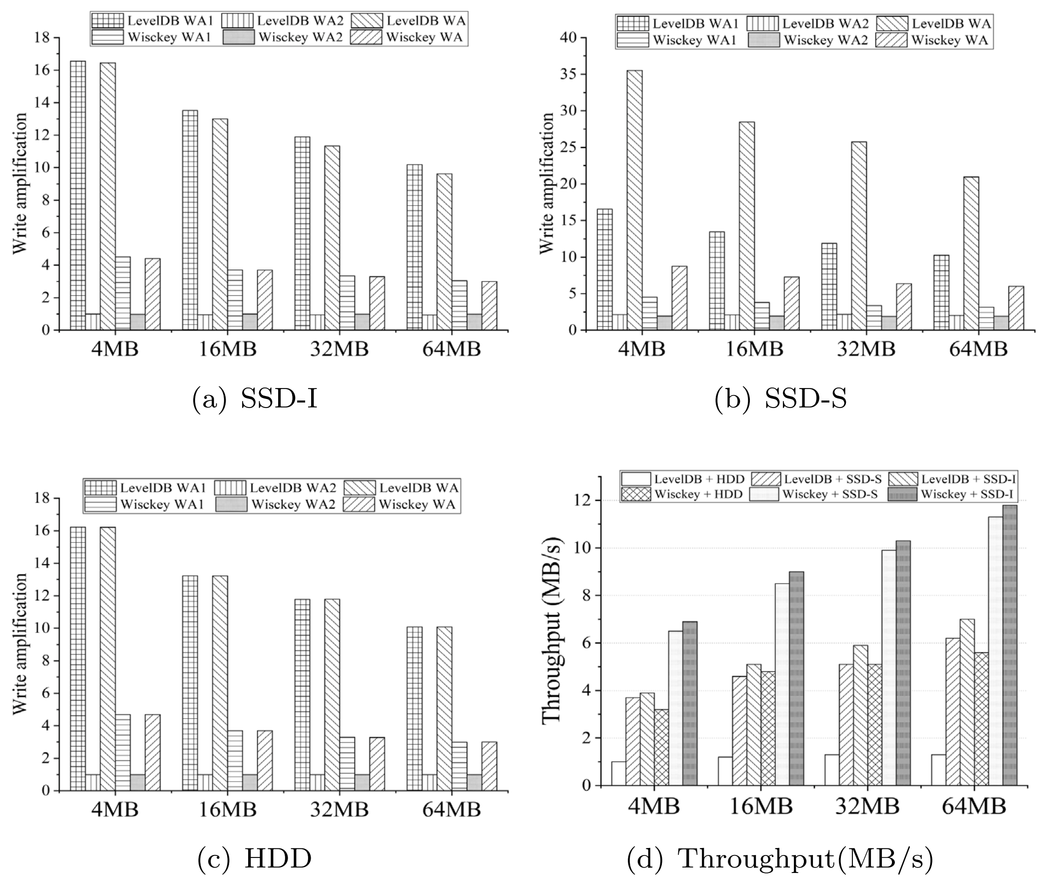


Fig. 7. WA and Throughput of the SSD-I, SSD-S, and HDD under different Write buffer size in DB_bench. It must be noted that two types of LSM-tree-based key–value stores (i.e., LevelDB and Wisckey), four types of write buffer size (i.e., 4 MB, 16 MB, 32 MB and 64 MB), three tested metrics for write amplification (i.e., WA1, WA2, and WA), and throughput (MB/s) are employed in this test. Notations WA1, WA2, and WA represent application-level write amplification, write amplification of a tested SSD (or HDD), and the cascaded write amplification of the overall system, respectively. In Fig. 7(c), the notation LevelDB+HDD means that LevelDB runs on the tested HDD.

or WA2 are similar, the overall WA exhibits a large difference under workloads.

Fig. 5(d) presents the throughput of KV stores with SSDs. As the data volume increases, the performance decreases because of the increment of cascaded write amplification, which is obvious in compaction-intensive cases under random-written workloads.

5.3. Various value size in key–value pairs

The default size of a key and value in LevelDB and Wisckey is 16 B and 100 B. The size of the key and value is an essential parameter in workloads. The size of the value, which is variable, is bigger than that of a key. Five types of value sizes are configured in DB_bench, *i.e.,* 16 B, 100 B, 16 KB, 64 KB, and 1 MB. To keep the same data volume (30 GB), amounts of KV pairs vary with the size of a value, which aims to keep the raw size.

In Fig. 6, WA1 in KV stores decreases as the size of a value increases. When the value size is 16 Bytes, the LevelDB-based WA1 of the SSD-I, SSD-S, and HDD is 17.92, 17.85, and 17.51 (highest), respectively. WA1 decreases to 8.03, 8.03, and 8.02 as the size of a value increases to 1 MB.

In Wisckey with the SSD-I, SSD-S, and HDD, WA1 exhibits 17.20, 17.15, and 17.22 using a 16-B value, respectively. But WA1 decreases to 4.52, 4.52, and 4.69 as the value size approaches 100 B (default), respectively. When the value size rises to 16 KB, WA1 drops to one. It can be found that Wisckey's write amplification is significantly improved as the value size increases. When the value size is 16 B, which is similar to the size of a key, LevelDB has the same write amplification or even slightly worse.

The number of KV pairs decreases with the increment of value size because the data volume is 30 GB in this test. Moreover, the requests of the random-load workload in DB_bench have a uniform distribution; thus, it has a same probability to read SSTables in each level for LevelDB and Wisckey. The density of KV pairs is higher at a low level than that at a high level, which causes high-frequency compaction at a low level. Write amplification then decreases as the size of the value increases.

Wisckey separately stores the key and the value; then, only keys are involved in a compaction procedure. When the value size is larger than that of the key, the key negligibly impairs write amplification in compaction compared to the key written in log files. Thus, write amplification that results from compaction becomes small in this case.

When the written data volume is unchanged, WA1 does not present an obvious difference. The amount of data written to disk exhibits a relatively stable value; meanwhile, WA2 has the same trend on the SSD-I and SSD-S.

In this test, the cascaded write amplification declines with the increment of KV pairs. The overall throughput improves when the cascaded write amplification decreases.

5.4. Various buffer size in key–value pairs

The write buffer size, which is the buffer size for the memtable is employed in our test. This buffer size is identified as that of the log files written to the underlying storage device. The default of the write buffer size is set to 4 MB. The larger the buffer size is, the better performance the database system presents. As shown in Fig. 7, the cascaded write amplification of the KV stores is smaller because the buffer is set larger;

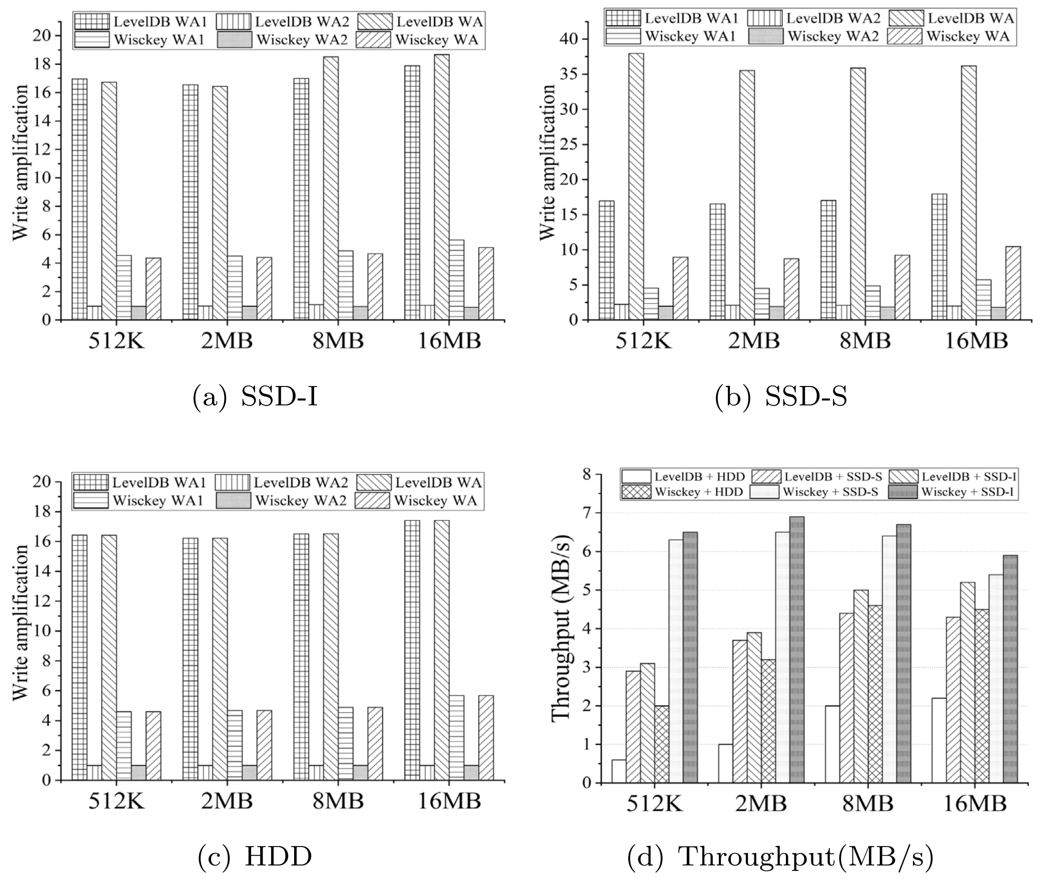


Fig. 8. WA and Throughput of the SSD-I, SSD-S, and HDD under different SSTable sizes in DB_bench. It must be noted that two types of LSM-tree-based key–value stores (i.e., LevelDB and Wisckey), four types of SSTable sizes (i.e., 512 KB, 2 MB, 8 MB, and 16 MB), three tested metrics for write amplification (i.e., WA1, WA2, and WA), and throughput (MB/s) are employed in this test. Notations WA1, WA2, and WA represent application-level write amplification, write amplification of a tested SSD (or HDD), and the cascaded write amplification of the overall system, respectively. In Fig. 8(c), the notation LevelDB+HDD means that LevelDB runs on the tested HDD.

meanwhile, throughput gradually increases. When the buffer reaches 4 MB, the cascaded write amplification of KV stores achieves its largest value; then, the throughput becomes the smallest. The WA1 of the SSD-I, SSD-S, and HDD with LevelDB is approximately 16.56, and the throughput is 3.9, 3.7, and 1.0, respectively. The WA1 of the SSD-I, SSD-S, and HDD with Wisckey is 4.52, 4.52, and 4.69, respectively. The throughput is 6.9, 6.5, and 3.2, respectively. When the buffer is 64 MB, the cascaded write amplification of the KV store becomes the smallest, while the throughput reaches its largest value. The WA1 of the SSD-I, SSD-S, and HDD with LevelDB is 10.2, 10.3, and 10.1, respectively. The throughput of KV stores becomes 7.0, 6.2, and 1.3, respectively.

We can conclude the reasons for these results. As the memtable increases, the key–value pair is written to the disk in the size of the Immtable. SSTables are directly dumped to the following level. When levels 0 and 1 have not overlapped ranges of KV pairs in the Immtable, the KV pair in the Immtable will be directly dumped to level two. Then, the sizes of SSTables that are generated by these direct-dumps operations will become larger while the write amplification from level 0 compaction to level 2 reduces. Moreover, when the Immtable dumps to level 0, the size of a new SSTable increases. Because level 0 triggers compaction by the number of SSTables, the larger SSTable will reduce the trigger frequency of the compaction. Thus, as the write buffer size increases, the host-side write amplification can be much smaller.

The change in the buffer size cannot significantly reduce the amount of data written to the underlying SSD. WA2 does not change significantly with the increment of buffer size. For example, in LevelDB, the WA2 of the SSD-S is 2.2, 2.2, 2.2, and 2.0 under 4-MB, 16-MB, 32-MB, and 64-MB buffer sizes, respectively. The WA2 of the SSD-I is 0.99, 0.96, 0.95, and 0.95 under the four buffer sizes, respectively.

5.5. *Various SSTable size in key–value pairs*

The default size of an SSTable is 2 MB. We set it to 512 KB, 2 MB, 8 MB, and 16 MB, respectively, in this test. In Fig. 8, the change in the SSTable size has little effect on the cascaded write amplification. For SSD-I, the size of SSTable varies from 512 KB to 2 MB, and the WA1 of LevelDB and Wisckey presents slight decrements. For example, LevelDB's WA1 drops to 16.56 from 16.96, while Wisckey's WA1 decreases to 4.52 from 4.55. When the size of an SSTable rises to 16 MB from 2 MB, the WA1 of LevelDB and Wisckey rises from 16.56 to 17.89 and from 4.52 to 5.64, respectively. It must be noted that the change of WA1 for the SSD-S has the same trend as that of the HDD, i.e., a weak downward trend for WA1 from 512 KB to 2 MB and a slight upward trend from 2 MB to 16 MB.

The reason is that the size of the SSTable directly impacts the number of SSTables at each level. The larger the SSTable size is, the more KV pairs are configured. Then, the number of SSTables at each level is relatively few. For level 0, the compaction operation is triggered by the number of files at this level. It is less likely to trigger the compaction when there are fewer SSTables. Thus, when the size of the SSTable increases from 512 KB to 2 MB, the number of compaction operations decreases, thereby resulting in a weak drop in WA1. When the size of an SSTable is large, SSTables that are involved in the overlapped key range become large. In the process of compaction, the data volume is much greater than that of the small SSTables. The value of WA1 increases. In addition, a large SSTable will reduce the frequency of compaction but increase the write data volume on the disk when a compaction process is triggered. Therefore, the different sizes of an SSTable have little effect on the cascaded write amplification.

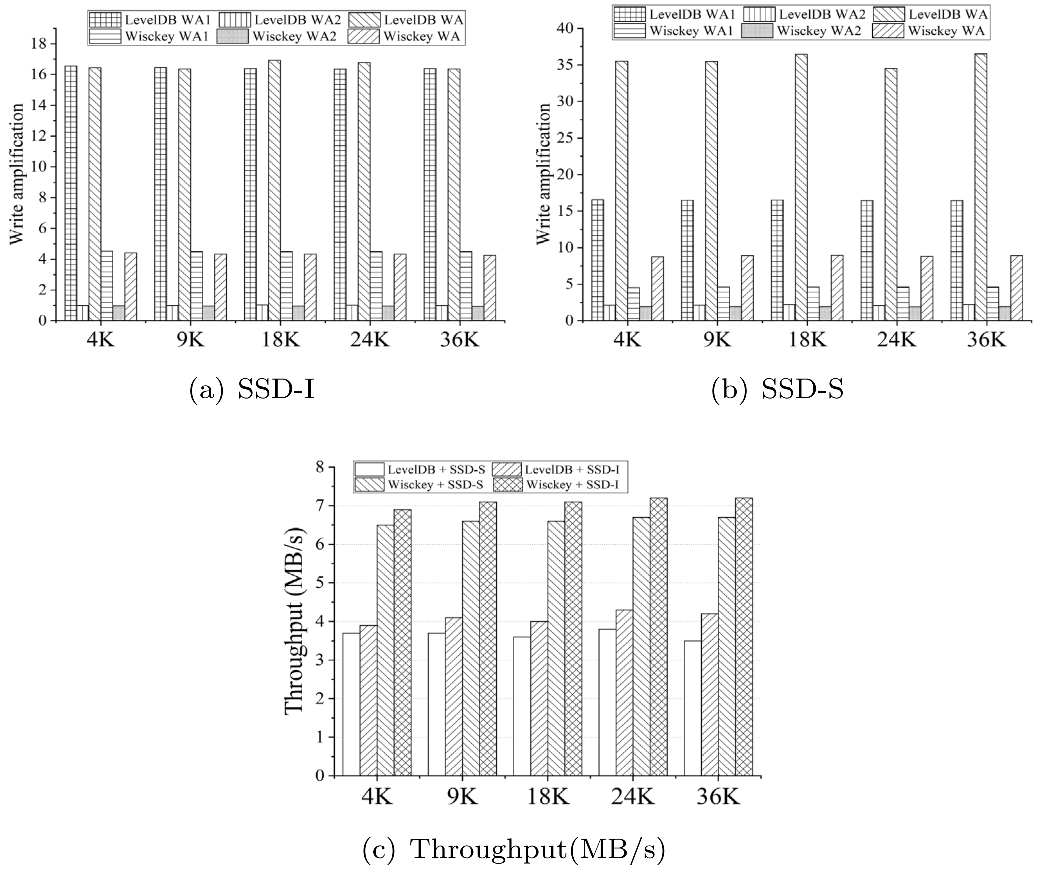


Fig. 9. WA and Throughput of the SSD-I and SSD-S under different block sizes in DB_bench. It must be noted that two types of LSM-tree-based key–value stores (i.e., LevelDB and Wisckey), five types of block sizes (i.e., 4 KB, 9 KB, 18 KB, 24 KB and 36 KB), three tested metrics for write amplification (i.e., WA1, WA2, and WA), and throughput (MB/s) are employed in this test. Notations WA1, WA2, and WA represent application-level write amplification, write amplification of a tested SSD, and the cascaded write amplification of the overall system, respectively. In Fig. 9(a), the notation LevelDB+SSD-I means that LevelDB runs on the tested SSD-I.

Because the size of the SSTable rarely affects the amount of data written to the SSD, the change of the SSTable rarely impairs the WA2. Then, the cascaded write amplification has the same trends as those of WA1.

In Fig. 8(a), the cascaded write amplification of LevelDB decreases by 0.02% as the size of an SSTable increases from 512 KB to 2 MB. From 2 MB to 16 MB, the cascaded write amplification increases by 0.05%. However, the throughput of LevelDB has an upward trend, as shown in Fig. 8(d), and the throughput of LevelDB with 2-MB, 8-MB, and 16-MB SSTables increases by 25%, 28%, and 4% compared with that of 512-KB SSTables, respectively, because increasing the size of an SSTable enlarges the size of an I/O request which improves the throughput of the underlying storage device.

5.6. Block size in SSTable

This metric (block size) is the size of user data in a block. Its default value is 4 KB, which is the basic unit of an SSTable on the disk. In Fig. 9, the block size has little effect on write amplification and the throughput of LevelDB and Wisckey on tested disks. In Fig. 9(a), the variant of cascaded write amplification of LevelDB on the SSD-I is −0.4%, 3.4%, −0.08%, and −2.4%, respectively, as the block size increases. Wisckey has a change of −1.8%, 0%, 0%, and 1.6%, respectively, In Fig. 9(b), the trend of change in the cascaded write amplification on SSD-S is small. This is because the variation in cascaded write amplification is rarely obvious, which is similar to that of the throughput. The change in throughput of LevelDB on SSD-I is 5.1%, −2.5%, 7.5%, and 2.3%, respectively. In this case, the throughput has the same variation.

The reason we can observe this is presented as follows. First, the block size is a threshold to guarantee that the number of key–value pairs in the data block does not exceed the threshold. Thus, it only impacts the number of blocks in an SSTable rather than compaction. Second, when a block is full, the KV store immediately flushes the block and appends it to the SSTable. Therefore, the block size influences the I/O size and frequency of I/O transfers to the SSD. However, the impact will be degraded by the I/O scheduling and FTLs on the SSD. Then, this metric has no significant effect on the WA2 of the SSD. As shown in Fig. 9, for the block size increments, the WA2 of LevelDB on the SSD-I is 0.99, 0.99, 1.03, 1.03, and 1.0, respectively. This metric presents write amplification as 2.15, 2.15, 2.21, 2.10, and 2.22 on the SSD-S, respectively.

Observation. The results under DB_bench reveal that WA1 is determined by sequential- and random-load workloads. For example, the sequential-load workload rarely produces write amplification. Second, the record size impacts on WA1, the value of which decreases when the record size becomes large. We change the number of KV pairs with the same data volume, and then we find that the values of WA1 and WA2 increase as the written data volume increases.

Second, the change of record size has a stable impact on WA1, which decreases as the record size increases. We can find that much written data can cause the increment of WA1 and WA2.

Third, the cascaded write amplification is largely dependent on WA1 and WA2. In KV stores with SSDs, there is a slight difference in WA2, but the cascaded write amplification has a significant gap, which means that WA1 is a dominated metric compared to WA2. This result is attributed to cascaded write amplification of KV stores with SSDs.

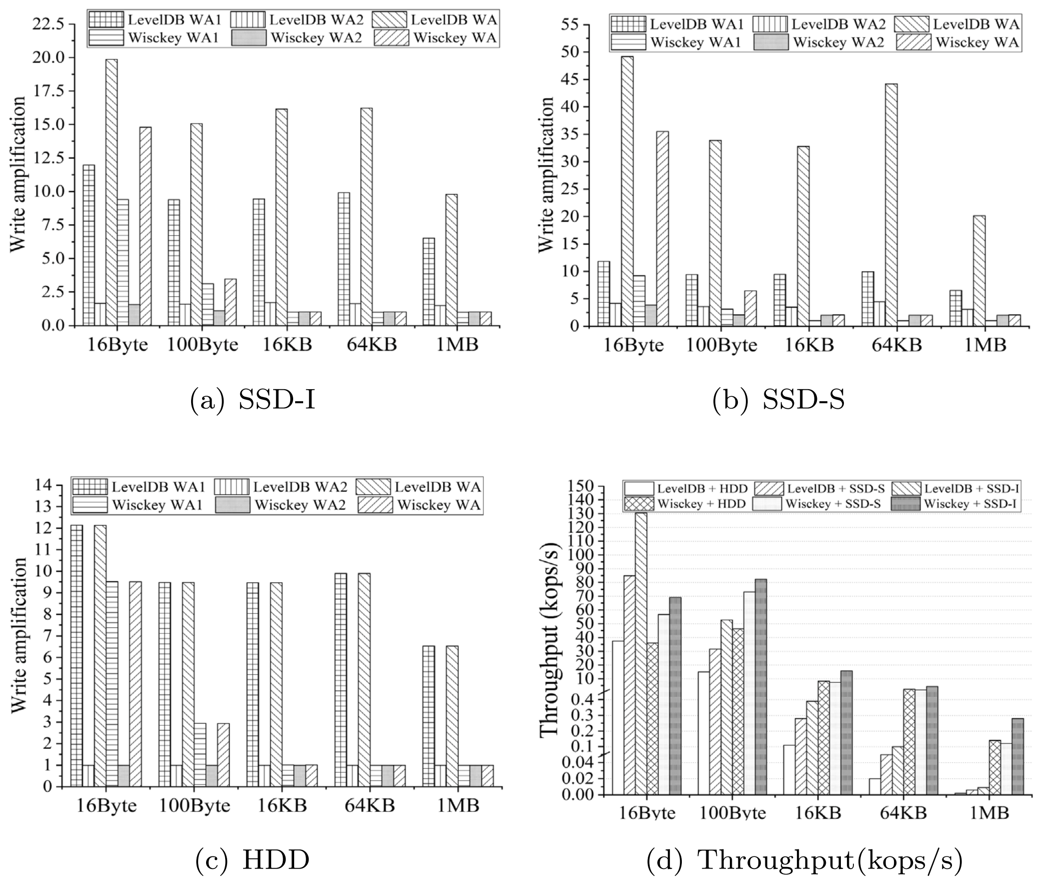


Fig. 10. WA and Throughput of the SSD-I, SSD-S, and HDD under various value sizes in YCSB. It must be noted that two types of LSM-tree-based key–value stores (i.e., LevelDB and Wisckey), five types of value sizes (i.e., 16 B, 100 B, 16 K, 64 K and 1 MB), three tested metrics for write amplification (i.e., WA1, WA2, and WA), and throughput (kops/s) are employed in this test. Notations WA1, WA2, and WA represent application-level write amplification, write amplification of a tested SSD (or HDD), and the cascaded write amplification of the overall system, respectively. In Fig. 10(c), the notation LevelDB+HDD means that LevelDB runs on the tested HDD.

6. Performance evaluation on YCSB

6.1. Various value sizes in YCSB

In this test, the notations *fieldlength* and *fieldcount* represent the size and number of a field for a value, respectively. The size of the value is multiplied by fieldlength and fieldcount. We set the value size to be 16 B, 100 B, 16 KB, 64 KB, and 1 MB (See Table 3.). Fig. 10 shows that WA1 generally decreases, which is similar to DB_bench. As the record size increases, WA becomes largest when the value is 16 B, while its smallest value exists when the 1-MB value is configured in key–value pairs. The cascaded write amplification decreases by 40%, 35%, and 31% for SSD-S, SSD-I and HDD for Leveldb, respectively, when the size of the value is 1 MB compared with when it is 16-B value. All tested disks present that WA1 decreases as the value size increases from 100 B to 16 KB. In the case of 64-KB values, WA1 is larger than that with 16-KB and 1-MB value. In Wisckey, WA1 approaches one when the record size is no less than 16 KB. For the shaking phenomenon at 64 KB, we think that DB_bench and YCSB have different request distributions. The distribution of DB_bench is uniform, while YCSB employs Zipfian. In Zipfian, several records that have a large probability of being accessed, are called hot data or popular data. Uniform distribution provides the same probability for each record. When the distribution is not uniform, WA1 then cannot increase as the size of a value becomes large.

The performance (i.e., kops) decreases as the size of the value increases. The reason is that it costs much time to transfer an I/O request into the underlying storage device when the size of the value is larger. It must be noted that the number of I/Os per second will decrease.

6.2. Various request distributions in YCSB

Under YCSB, two types of workload distributions (i.e., Zipfian and uniform) are employed in the experiment. In Fig. 11, the WA1, WA2, and WA under the Zipfian distribution are significantly lower than that under the Uniform distribution.

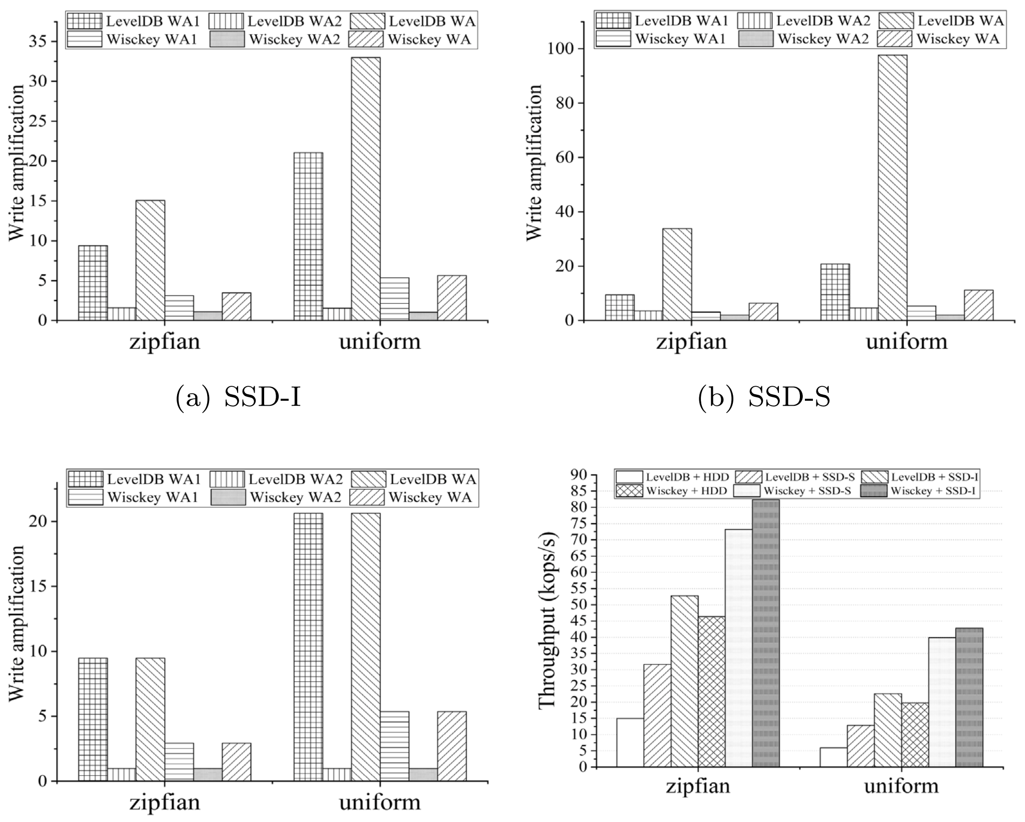
For the SSD-I, SSD-S, and HDD, LevelDB's WA1 increases by 11.67, 11.41, and 11.15, respectively under the Uniform distribution. Wisckey has a WA1 of 2.24, 2.26, and 2.43 under the same distribution, respectively. The reason is that each row in this workload has an equal probability of being read, while other rows have much more potential to be targeted by read or scan operations. Those rows are defined as a hot set. Thus, the KV store triggers more compaction under the Uniform distribution. Then, this workload causes a larger write amplification in the applications.

In the Uniform distribution, the workload generator provides much data that is written into the SSD. For the SSD-I and SSD-S, the WA2 of LevelDB under the Uniform distribution increases by 0.03 and 1.1 compared with that of the Zipfian distribution. Wisckey rises by 0.01 and 0.05 under two types of distribution, respectively. The cascaded write amplification has the same trend as WA1.

Under the Zipfian, the SSD-I, SSD-S and HDD with both KV stores present lower cascaded write amplification than that under the Uniform distribution. Then, the performance (kops) under Zipfian is also higher than that under Uniform, see Fig. 11(d).

6.3. Various ratios of read and write

In a real-world workload, NoSQL databases run a workload that is a mix of both read and write I/Os. Therefore, we employ YCSB to



(a) SSD-I (b) SSD-S

(c) HDD (d) Throughput(kops/s)

Fig. 11. WA and Throughput of the SSD-I, SSD-S, and HDD under various request distribution in YCSB. It must be noted that two types of LSM-tree-based key–value stores (i.e., LevelDB and Wisckey), two types of request distribution (i.e., zipfian and uniform), three tested metrics for write amplification (i.e., WA1, WA2, and WA), and throughput (kops/s) are employed in this test. Notations WA1, WA2, and WA represent application-level write amplification, write amplification of a tested SSD(or HDD), and the cascaded write amplification of the overall system, respectively. In Fig. 11(c), the notation LevelDB+HDD means that LevelDB runs on the tested HDD.

generate four types of different ratios of read and writes (i.e., R:W is set to 0:1, 2:8, 4:6, and 6:4 in Table 2) in the workload to study the effect of the read-write mixed workload on the cascaded write amplification. Without loss of generality, we set the data volume generated by the YCSB-C to 10 GB. Fig. 12 shows that different read-write ratios have an impact on write amplification and performance.

When the ratio of read and write is 0:1 (full write), the KV store has the smallest cascaded write amplification and the highest throughput on the three storage devices. The values of WA1, WA2, and WA3 increase with the read ratio, and the throughput decreases.

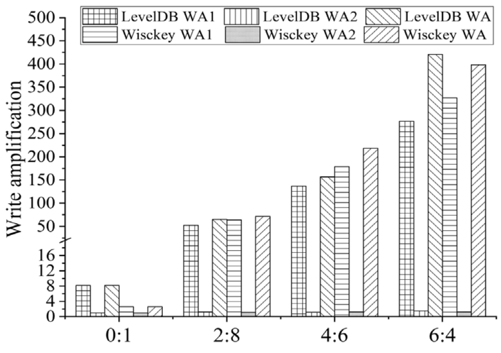
In Fig. 12(a), the WA1 of the SSD-I with LevelDB is 8.18, 52.5, 136.83, and 276.65 under 0%-, 20%-, 40%-, and 60%-read ratios, respectively. Meanwhile, WA2 is 1, 1.14, 1.24, and 1.52 in the case of four types of read ratio, respectively. The cascaded write amplification under the 60%-read-ratio workload is 51 times that under the 0%-read-ratio workload, while the throughput decreases by 94.0%. The cascaded write amplification of the SSD-S and HDD increases by 74 times and 44 times from the 60%-read-ratio workload to the 0%-read-ratio workload, respectively. The throughput decreases by 98.0% and 98.04%. The throughput of LevelDB on the SSD-I, SSD-S, and HDD is 1.41, 4.08, and 0.36 (MB/s) under 60%-read-ratio workloads. Wisckey also shows the same trend. The cascaded write amplification of the SSD-I, SSD-S, and HDD increases by 153, 239, and 124 times, respectively. The throughput of Wisckey decreases by 99.2%, 99.1% and 99.23% in the case of the three devices, respectively.

This can be seen for the following three reasons: First, the compaction at level 0 is not only triggered by the limited number of SSTables but also caused by the seek_compaction because of the read operation. A large number of SSTables that are read from disk perform compaction at level 0. Thus, under the read-intensive workload, a high frequency of compaction is triggered, thereby enlarging the WA1.

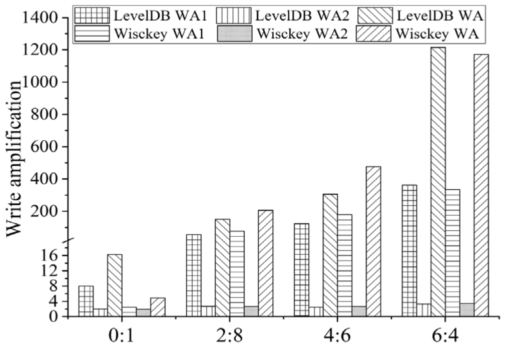
Second, LevelDB and Wisckey, which are based on LSM-tree, are popularized as the write-optimized KV storage systems but have poor read performance.

Third, owing to the mechanical structure of an HDD, its read performance is very different from that of an SSD under the random workload. When the size of a request is 4 KB (4-KB block being read from an SSTable), there is an obvious difference from that of an SSD. However, the performance of an HDD and SSD has little difference under the 4-KB-sequential-written workload. The performance of LevelDB with an HDD is very obvious under sequential- and random-write workloads. This can cause much seek_compaction at level 0, thereby resulting in cascaded write amplification. We find that it is necessary to avoid very read-intensive workload on HDD-based storage systems.

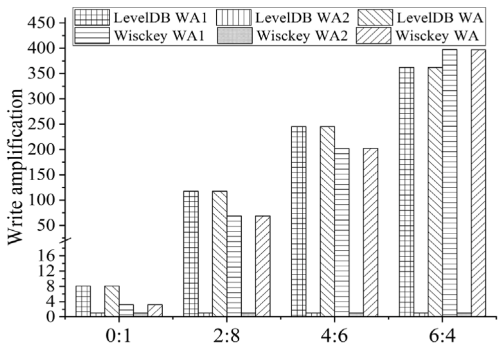
Observation. There are different features between YCSB and DB_bench. First, YCSB has two phases, i.e., load and run. This workload includes at least twice as much write data as that in DB_bench under the same load. Second, WA1 is much larger than that under db_bench, and the default distribution of the workload in YCSB-C is *Zipfian*. These features in YCSB cause much data to be written onto a disk under YCSB compared to DB_bench. In this case, WA2 of the SSD-S and SSD-I has a significant improvement under YCSB compared with that under DB_bench. This proves that the data volume that is written to disks enlarges WA2 for the same workload distribution. Therefore, we conducted an extensive experiment in Section 7.



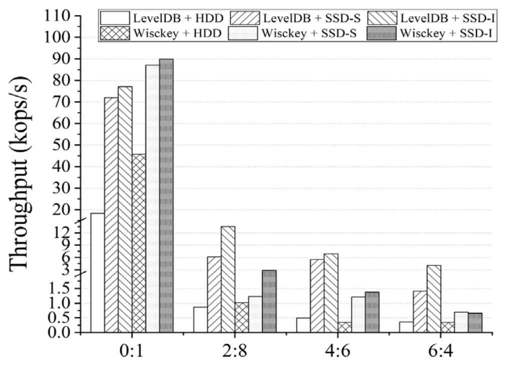
(a) SSD-I



(b) SSD-S



(c) HDD



(d) Throughput(kops/s)

Fig. 12. WA and Throughput of the SSD-I, SSD-S, and HDD under different read/write ratio in YCSB. It must be noted that two types of LSM-tree-based key–value stores (i.e., LevelDB and Wisckey), four types of read/write ratio (i.e., 0:1, 2:8, 4:6 and 6:4), three tested metrics for write amplification (i.e., WA1, WA2, and WA), and throughput (kops/s) are employed in this test. Notations WA1, WA2, and WA represent application-level write amplification, write amplification of a tested SSD (or HDD), and the cascaded write amplification of the overall system, respectively. In Fig. 12(c), the notation LevelDB+HDD means that LevelDB runs on the tested HDD.

Moreover, the various ratios of the read and write have an impact on KV stores with SSDs in terms of write amplification and throughput. Inside an SSD, a cascaded write amplification has a negative impact on SSD lifetime. However, the performance of an HDD is different from that of an SSD. The approach to balancing this situation is a fundamental technology that should be studied by researchers. LevelDB exhibits a slightly different performance with various sizes in values in YCSB-C and DB_bench, which validates that the small change of value size does not significantly impact WA1 in realistic cases.

7. Scalability evaluation

In the above experiment, tested SSDs are set to the initial state by means of ATA Secure Erase before each test. This method can remove all data in SSDs and set the SSD to the initial state.

Therefore, WA2 of an SSD always shows a stable value. We change the write buffer size to execute the experiment. The large write buffer provides a reduction in cascaded write amplification. At the beginning of a test, we close the SSD TRIM and employ Iometer to sequentially load 4-KB write requests to the disk for two hours. At the end of each test, we do not delete the database files in the disk; immediately, the subsequent experiment is conducted. After each test, the remaining space in the SSD is smaller than in the last test, which causes the SSD to be in the steady state.

In Fig. 13, WA1 decreases with the increment in write buffer size, but WA2 increases in the steady state. In the stable state, the WA2 of the SSD-I rises by 5%, 75%, 77%, and 73% compared to the initial state using the 4-MB, 16-MB, 32-MB, and 64-MB write buffer, respectively. SSD-s exhibits a change of 15%, 88%, 170%, and 199% in the stable state compared to the initial state, respectively, under the four types of write buffer sizes.

The SSD-S presents obvious increments and SSD-I shows a stable trend when the value of WA2 remains 1approximately 1.5, which indicates that the SSD-I has a better improvement in write amplification. In this test, we can find that the state of an SSD influences write amplification. In Fig. 13(c), the throughput of steady-state SSDs is lower than that in the initial-state because large WA2 in SSDs degrades overall system performance.

8. Summary

In this section, we summarize the following key findings from our extensive experimental results:

(1) We find that WA1 is largely impacted by sequential- and random-load workloads. Under the random workloads, it will produce application-level write amplification. Second, the read-write ratio has a great impact on WA1 and WA2. Third, the number of KV pairs and the size of a KV pair will have a significant impact on WA1.
(2) In most cases, Wisckey has superior benefits to LevelDB in terms of write amplification and performance. In realistic workloads with mixed read, the performance of LevelDB and Wisckey are very poor, especially on the HDD. SSDs have performance benefits compared with HDDs in KV stores. However, the hundreds of cascaded write amplification largely impair the SSD lifetime under read-write-mixed workloads.

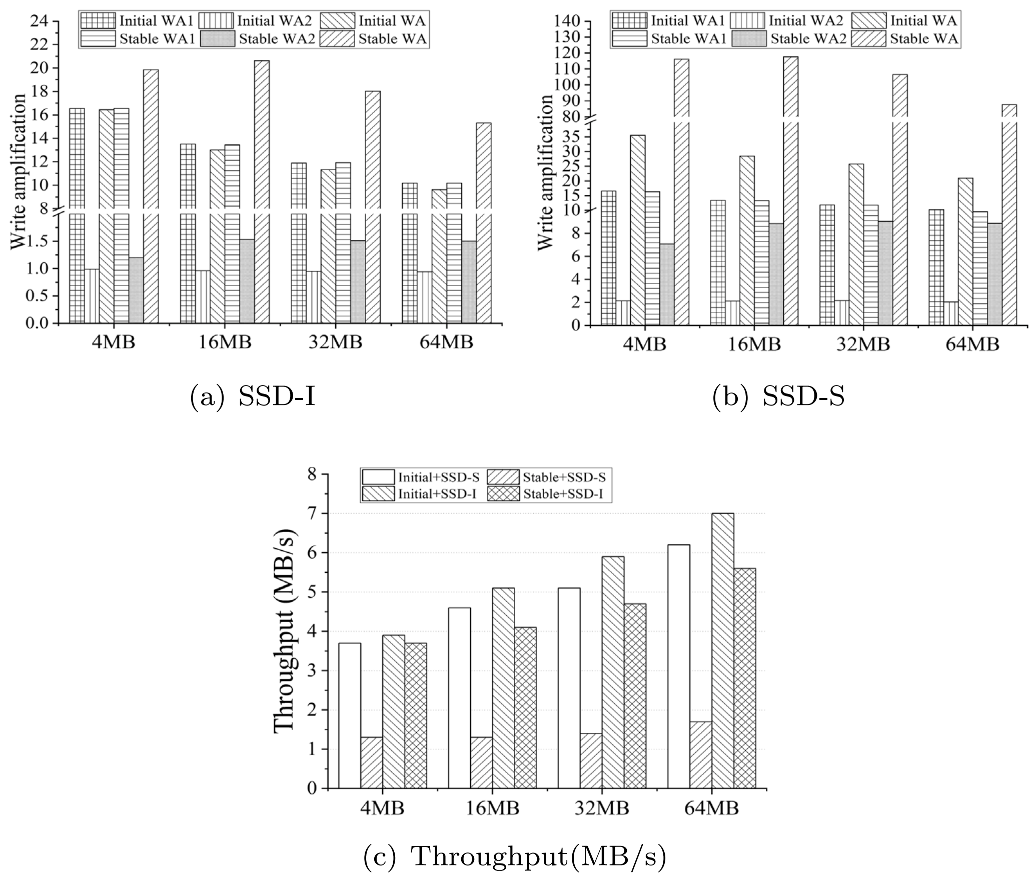


Fig. 13. WA and Throughput of different SSD states under different write buffer sizes and SSD States in DB_bench. It must be noted that two types of SSD states (i.e., Initial and Stable), four types of write buffer sizes (i.e., 4 MB, 16 MB, 32 MB and 64 MB), three tested metrics for write amplification (i.e., WA1, WA2, and WA), and throughput (MB/s) are employed in this test. Notations WA1, WA2, and WA represent application-level write amplification, write amplification of a tested SSD, and the cascaded write amplification of the overall system, respectively. In Fig. 13(a), the notation Initial+SSD-I means that the tested SSD-I is set to the initial state before each test.

(3) The cascaded write amplification of the overall system is determined by the application-level write amplification (WA1). But, WA2 is also important for the cascaded write amplification. For the SSD-I and SSD-S, it can be found that there is a small difference in WA2 between the two SSDs, but significant WA can exist because of the cascaded effect between WA1 and WA2. This write amplification of KV stores has a largely bad impact on the SSD lifetime.

(4) According to the experimental results, the throughput of an HDD is worse than that of the two SSDs, e.g., from 0.2x to 0.5x the performance of that in SSDs in most cases. The SSD-I presents better performance than the SSD-S. Fig. 12 shows that when the workload is mixed with read and write, the performance of an HDD is largely lower than that of SSDs. At this time, it can reveal the advantage of an SSD in KV stores in terms of performance. However, the throughput of the SSD-S and SSD-I under the 80%-write workload is around 0.28x and 0.31x of that under the 100%-write workload. Therefore, researchers are encouraged to design a novel approach to balance the cost of an SSD lifetime caused by cascaded write amplification and its high performance under the read-write-mixed workload, which is a key issue impacting the overall performance under realistic workloads.

(5) In Fig. 3(a), we find that the throughput of both LevelDB and Wisckey with the tested disks are lower than those on raw disks, which indicates that KV stores cannot make full use of the performance of disks. However, the results in Fig. 6(d) show that the throughput of Wisckey on the SSD-I and SSD-S is close to that on raw SSDs when value size reached 16 KB. This is attributed to the following two reasons. (1) An SSD exhibits the best performance in the initial state. (2) With more than 16-KB value size, Wisckey loads 30-GB data, which is much less than that on the raw disk. The performance of SSDs increases as the data volume written to the SSD is reduced.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

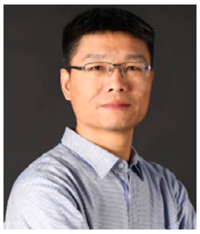
Acknowledgments

This work is supported in part by National Natural Science Foundation of China under Grants 61702004, 61572209. Sun's work is partly supported by State Key Laboratory of Computer Architecture (ICT, CAS), China under Grant No. CARCH201915.

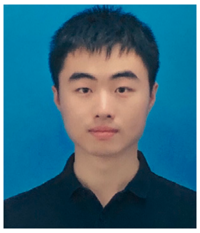
References

[1] D. Comer, Ubiquitous B-tree, ACM Comput. Surv. 11 (2) (1979) 121–137.
[2] Z. Deng, K. Li, K. Li, J. Zhou, A multi-user searchable encryption scheme with keyword authorization in a cloud storage, Future Gener. Comput. Syst. 72 (2017) 208–218.
[3] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber, Bigtable: A distributed storage system for structured data, ACM Trans. Comput. Syst. (TOCS) 26 (2) (2008) 4.

[4] S. Ghemawat, J. Dean, Leveldb, 2011.
[5] M.N. Vora, Hadoop-hbase for large-scale data, in: Proceedings of 2011 International Conference on Computer Science and Network Technology, vol. 1, IEEE, 2011, pp. 601–605.
[6] A. Lakshman, P. Malik, Cassandra: a decentralized structured storage system, ACM SIGOPS Oper. Syst. Rev. 44 (2) (2010) 35–40.
[7] D. Borthakur, RocksDB: A persistent key-value store, 2014.
[8] F. Chen, D.A. Koufaty, X. Zhang, Hystor: making the best use of solid state drives in high performance storage systems, in: Proceedings of the International Conference on Supercomputing, ACM, 2011, pp. 22–32.
[9] Y. Wang, K. Li, J. Zhang, K. Li, Energy optimization for data allocation with hybrid SRAM+NVM SPM, IEEE Trans. Circuits Syst. I. Regul. Pap. 65 (1) (2018) 307–318.
[10] P. Desnoyers, Empirical evaluation of NAND flash memory performance, ACM SIGOPS Oper. Syst. Rev. 44 (1) (2010) 50–54.
[11] P. Cappelletti, C. Golla, P. Olivo, E. Zanoni, Flash Memories, Springer Science & Business Media, 2013.
[12] X. Wu, Y. Xu, Z. Shao, S. Jiang, LSM-Trie: An LSM-tree-based ultra-large key-value store for small data items, in: 2015 USENIX Annual Technical Conference (USENIX ATC 15), USENIX Association, Santa Clara, CA, ISBN: 978-1-931971-225, 2015, pp. 71–82.
[13] L. Lu, T.S. Pillai, H. Gopalakrishnan, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, Wisckey: Separating keys from values in ssd-conscious storage, ACM Trans. Storage (TOS) 13 (1) (2017) 5.
[14] F. Mei, Q. Cao, H. Jiang, L. Tian, LSM-tree managed storage for large-scale key-value store, IEEE Trans. Parallel Distrib. Syst. 30 (2) (2018) 400–414.
[15] T. Yao, J. Wan, P. Huang, X. He, Q. Gui, F. Wu, C. Xie, A light-weight compaction tree to reduce I/O amplification toward efficient key-value stores, in: Proceedings of the 33rd International Conference on Massive Storage Systems and Technology (MSST 2017), 2017.
[16] P. Raju, R. Kadekodi, V. Chidambaram, I. Abraham, Pebblesdb: Building key-value stores using fragmented log-structured merge trees, in: Proceedings of the 26th Symposium on Operating Systems Principles, ACM, 2017, pp. 497–514.
[17] G. Wu, X. He, Reducing SSD read latency via NAND flash program and erase suspension, in: FAST, vol. 12, 2012, 10–10.
[18] R. Lucchesi, SSD Flash drives enter the enterprise, Silverton Consulting 8 (2011) 2008, accessed on.
[19] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, R. Pletka, Write amplification analysis in flash-based solid state drives, in: Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference, ACM, 2009, p. 10.
[20] B. Schroeder, R. Lagisetty, A. Merchant, Flash reliability in production: The expected and the unexpected, in: 14th USENIX Conference on File and Storage Technologies FAST 16, 2016, pp. 67–80.
[21] T.-Y. Chen, Y.-H. Chang, Y.-H. Kuan, Y.-M. Chang, Virtualgc: Enabling erase-free garbage collection to upgrade the performance of rewritable SLC NAND flash memory, in: Proceedings of the 54th Annual Design Automation Conference 2017, ACM, 2017, p. 25.
[22] C. Gao, L. Shi, M. Zhao, C.J. Xue, K. Wu, E.H. Sha, Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives, in: 2014 30th Symposium on Mass Storage Systems and Technologies (MSST), 2014, pp. 1–11.
[23] M. Rosenblum, J.K. Ousterhout, The design and implementation of a log-structured file system, ACM Trans. Comput. Syst. (TOCS) 10 (1) (1992) 26–52.
[24] P. O'Neil, E. Cheng, D. Gawlick, E. O'Neil, The log-structured merge-tree (LSM-tree), Acta Inform. 33 (4) (1996) 351–385.
[25] E.D. Sneed, R.L. Folk, Pebbles in the lower colorado river, texas a study in particle morphogenesis, J. Geol. 66 (2) (1958) 114–150.
[26] M.A. Olson, K. Bostic, M.I. Seltzer, Berkeley DB, in: USENIX Annual Technical Conference, FREENIX Track, 1999, pp. 183–191.
[27] A. Badam, K. Park, V.S. Pai, L.L. Peterson, HashCache: Cache storage for the next billion, in: NSDI, vol. 9, pp. 123–136.
[28] G.W. Poerwawinata, A.I. Kistijantoro, Memcachedb persistent implementation using leveldb, in: 2015 International Conference on Electrical Engineering and Informatics (ICEEI), IEEE, 2015, pp. 283–287.
[29] K. Chodorow, MongoDB: The Definitive Guide: Powerful and Scalable Data Storage, " O'Reilly Media, Inc.", 2013.
[30] H. Lim, B. Fan, D.G. Andersen, M. Kaminsky, SILT: A memory-efficient, high-performance key-value store, in: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ACM, 2011, pp. 1–13.
[31] D.G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, V. Vasudevan, FAWN: A fast array of wimpy nodes, in: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, ACM, 2009, pp. 1–14.
[32] B. Debnath, S. Sengupta, J. Li, Skimpystash: RAM space skimpy key-value store on flash-based storage, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, ACM, 2011, pp. 25–36.
[33] T. Yao, J. Wan, P. Huang, Y. Zhang, Z. Liu, C. Xie, X. He, GearDB: a GC-free key-value store on HM-SMR drives with gear compaction, in: 17th USENIX Conference on File and Storage Technologies FAST 19, 2019, pp. 159–171.
[34] P. Wang, G. Sun, S. Jiang, J. Ouyang, S. Lin, C. Zhang, J. Cong, An efficient design and implementation of LSM-tree based key-value store on open-channel SSD, in: Proceedings of the Ninth European Conference on Computer Systems, ACM, 2014, p. 16.
[35] A. Eisenman, A. Cidon, E. Pergament, O. Haimovich, R. Stutsman, M. Alizadeh, S. Katti, Flashield: a hybrid key-value cache that controls flash write amplification, in: Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation, USENIX Association, 2019, pp. 65–78.
[36] B. Debnath, S. Sengupta, J. Li, Flashstore: high throughput persistent key-value store, Proc. VLDB Endowment 3 (1–2) (2010) 1414–1425.
[37] L. Marmol, S. Sundararaman, N. Talagala, R. Rangaswami, S. Devendrappa, B. Ramsundar, S. Ganesan, NVMKV: A scalable and lightweight flash aware key-value store, in: 6th USENIX Workshop on Hot Topics in Storage and File Systems HotStorage 14, 2014.
[38] Y.-T. Chen, M.-C. Yang, Y.-H. Chang, T.-Y. Chen, H.-W. Wei, W.-K. Shih, Co-optimizing storage space utilization and performance for key-value solid state drives, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 38 (1) (2019) 29–42.
[39] S.-M. Wu, K.-H. Lin, L.-P. Chang, KVSSD: Close integration of LSM trees and flash translation layer for write-efficient KV store, in: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2018, pp. 563–568.
[40] Shannon-Systems, Innovating solid state storage solutions, 2018, URL http://en.shannon-sys.com/product_detail?id=4973859839646911603.
[41] shannon sys, Kv_libcpp, 2019, URL https://github.com/shannon-sys/kv_libcpp.
[42] J. Sievert, Iometer: The i/o performance analysis tool for servers, 2004.
[43] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, Benchmarking cloud serving systems with YCSB, in: Proceedings of the 1st ACM Symposium on Cloud Computing, ACM, 2010, pp. 143–154.
[44] A. Papagiannis, G. Saloustros, P. Gonzlez-Frez, A. Bilas, Tucana: Design and implementation of a fast and efficient scale-up key-value store, in: 2016 Annual Technical Conference (USENIXATC 16), 2016, pp. 537–550.
[45] J. Ren, YCSB-C, 2018, URL https://github.com/basicthinker/YCSB-C.
[46] SINA, Solid state storage (SSS) performance test specification (PTS) version 2.0.1, 2018, URL https://www.snia.org/sites/default/files/technical_work/PTS/SSS_PTS_2.0.1.pdf.
[47] M.Y.C. Wei, L.M. Grupp, F.E. Spada, S. Swanson, Reliably erasing data from flash-based solid state drives, in: FAST, vol. 11, 2011, pp. 8–8.
[48] A. Ku, Intel SSD 520 review: Sandforce's technology: Very low write, 2012, URL https://www.tomshardware.com/reviews/ssd-520-sandforce-review-benchmark,3124-11.html.
[49] Intel® SSD Technology Terminology Guide, URL https://www.intel.com/content/www/us/en/solid-state-drives/intel-ssd-technology-terminology-guide.html.



Hui Sun received the Ph.D. degree from Huazhong University Science and Technology, in 2012. He is an assistant Professor of computer science with Anhui University. His research interests include computer systems, edge computing, performance evaluation, Non-Volatile Memory-based storage systems, file systems,and I/O architectures.



Shangshang Dai born in 1995. He is currently pursuing the M.S. degree in Anhui University. His main research interests include computer systems, edge computing, and key–value storage systems.

Jianzhong Huang received the Ph.D. degree in computer architecture in 2005 and completed the Post-Doctoral research in information engineering in 2007 from Huazhong University of Science and Technology (HUST), Wuhan, China. He is currently an associate professor in the Wuhan National Laboratory for Optoelectronics at HUST. His research interests include computer architecture and dependable storage systems. Dr. Huang was a recipient of the National Science Foundation of China in Storage System Research Award in 2007. He is a member of China Computer Federation (CCF).