EDC: Improving the Performance and Space Efficiency of Flash-Based Storage Systems with Elastic Data Compression

Bo Mao, *Member, IEEE*, Suzhen Wu, *Member, IEEE*, Hong Jiang, *Fellow, IEEE*, Yaodong Yang, and Zaifa Xi

Abstract—By leveraging data reduction technologies, such as data compression, all flash-based storage systems can have the same total cost of ownership (TCO) as traditional HDD-based storage systems. Thus, data compression has become a commodity feature for space efficiency and reliability in flash-based storage systems by reducing write traffic and space capacity demand. However, it introduces noticeable processing overheads on the critical I/O path, which degrades the system performance significantly. Existing data compression schemes for flash-based storage systems use fixed compression algorithms for all the incoming write data, failing to recognize and exploit the significant diversity in compressibility and access patterns of data and missing an opportunity to improve the system performance, the space efficiency or both. To achieve a reasonable trade-off between these two important design objectives, in this paper we introduce an Elastic Data Compression scheme, called EDC, which exploits the data compressibility and access intensity characteristics by judiciously matching data of different compressibility with different compression algorithms while leveraging the access idleness. Specifically, for compressible data blocks EDC exploits the compression diversity of the workload, and employs algorithms of higher compression rate in periods of lower system utilization and algorithms of lower compression rate in periods of higher system utilization. For non-compressible (or very lowly compressible) data blocks, it will write them through to the flash storage directly without any compression. The experiments conducted on our lightweight prototype implementation of the EDC system show that EDC saves storage space by up to 38.7 percent, with an average of 33.7 percent . In addition, it significantly outperforms the fixed compression schemes in the I/O performance measure by up to 61.4 percent, with an average of 36.7 percent.

Index Terms—Elastic data compression, flash-based storage, data compressibility, I/O intensity

1 Introduction

Due to the slow mechanical positioning nature of Hard Disk Drives (HDDs), HDD-based storage devices have limited system performance. The I/O bottleneck has become an increasingly daunting challenge for big data analytics, along with the explosive growth in data volume [1]. Flash-based SSDs have the potential to replace HDDs and have consequently been extensively deployed in modern storage systems to satisfy the increasing demand of storage performance and energy efficiency [2], [3], [4]. At the same time, inline data compression techniques have been widely employed in flash-based storage products from leading companies, such as Nimble Storage [5], Pure Storage [6], and Tintri [7], for the purpose of enhancing system performance, reliability and space efficiency.

Flash-based storage systems and products have already deployed data compression as a standard commodity feature for two technological trends. The first trend is the need to increase the storage density of the NAND flash devices by increasing the number of bits per storage cell from a single bit (SLC) to two bits (MLC), and to multiple bits (TLC), which leads to significant deterioration of chip endurance (cell erase limit) while keeping the latency essentially constant. The second trend is the continuous improvement in the processing power of processors, such as GPU and multicore processors, which lowers the computation cost noticeably. The combined impact of these trends makes the data compression technique not only necessary but also affordable by trading compute overheads for several important benefits. First, it trades processing power for the improved space efficiency by storing much more user data than the physical capacity of a flash-based storage system [8], [9]. Second, it reduces the number of erase cycles for the NAND flash cells [10], thus increasing the lifetime of the flash-based storage systems. Third, it also reduces the I/O latency by shrinking the individual request size. Nevertheless, these benefits do come at the expenses of some system performance and extra system resources, which may cause the amount of the benefits to vary substantially depending on

- *B. Mao is with the Software School of Xiamen University, Xiamen 361005, Fujian, China. E-mail: maobo@xmu.edu.cn.*
- *S. Wu and Z. Xi are with Computer Science Department of Xiamen University, Xiamen 361005, Fujian, China. E-mail: suzhen@xmu.edu.cn,.*
- *H. Jiang is with the Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX 76019, USA. E-mail: hong.jiang@uta.edu.*
- *Y. Yang is now with Apple in San Francisco, CA 94105, USA. E-mail: yaodong.yangy@gmail.com.*

Manuscript received 7 Oct. 2017; revised 8 Dec. 2017; accepted 13 Jan. 2018. Date of publication 18 Jan. 2018; date of current version 11 May 2018. (Corresponding author: Suzhen Wu.)
Recommended for acceptance by M. Kandemir.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2794966

the data compressibility and access intensity of the workloads. Therefore, employing data compression should be done carefully in order to avoid potential pitfalls for flash-based storage systems [11], [12].

The existing studies have shown that the data compressibility distribution is skewed in that the main benefit of data compression comes from a subset of the data blocks [13], [14]. For example, based on the file data generated from 15 globally distributed file servers for over 2000 users in a large multinational corporation, researchers found that 50 percent of the data chunks are responsible for 86 percent of the compression savings and roughly 31 percent of the data chunks do not compress at all [14]. Our own analysis has been consistent with these published studies in showing that data blocks from different file types have different data compressibility characteristics. In addition, both previous studies and our own evaluations have demonstrated that different compression algorithms have different compression ratios and compute overheads [15], [16]. On the one hand, higher compression schemes require higher compute overheads in the compressing and decompressing processes, and vice versa, as shown in Section 2.2. On the other hand, the access patterns of real-world workloads exhibit a significant interspersed idleness and burstiness characteristics [17]: periods of high utilization alternate with periods of little or no external load, as shown in Section 2.3. Most existing data reduction schemes for flash-based storage systems use a fixed compression scheme in both high utilization and low utilization periods for all the incoming data. This approach has obvious drawbacks in that, for example, it will degrade the system performance if a scheme with a high compression ratio is used in high utilization periods, or diminish the space efficiency if an algorithm with a low compression ratio is used in low utilization periods. Moreover, applying data compression on non-compressible (or very lowly compressible) data chunks will both waste system resources and degrade the system performance.

To address these problems in current flash-based storage systems with the commodity data compression feature, we propose an Elastic Data Compression scheme, or EDC, to improve the system performance and space efficiency in such systems. EDC exploits the compression diversity of the workload characteristics, and for compressible data blocks employs algorithms with higher compression ratios in periods with lower system utilization and algorithms with lower compression ratios in periods with higher system utilization. For non-compressible (or very lowly compressible) data blocks, it will write them through to the flash storage directly without any compression. To validate EDC and evaluate its efficiency, we have conducted extensive trace-driven evaluations on a lightweight implementation of the EDC prototype. The performance results show that EDC achieves a much better trade-off between the performance and space efficiency than the existing schemes.

The rest of this paper is organized as follows. Background and motivation are presented in Section 2. We describe the design details of EDC in Section 3. The performance evaluation is presented in Section 4 and the related work is presented in Section 5. We conclude this paper in Section 6.

2 Background and Motivation

In this section, we first present the necessary background for the proposed solution, including a discussion on two unique characteristics of modern flash-based SSDs. Then we analyze the data compressibility of typical data formats and types and compression efficiency of representative compression algorithms, followed by a workload behavior characterization study that motivates our proposed elastic data compression for flash-based storage systems.

2.1 Flash-based SSDs

Different from HDDs that are consisting of mechanical positioning parts, flash-based SSDs are made of silicon memory chips and do not have moving parts [18], [19]. In addition to their high energy-efficiency and high random-read performance advantages, flash-based SSDs have the following two unique characteristics that distinguish them from HDDs.

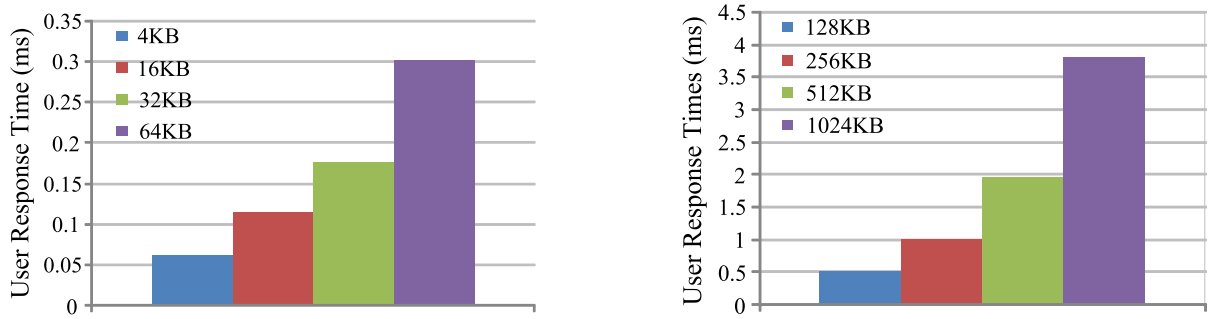
First, flash-based SSDs have asymmetric read-write performance, with the write performance lagging the read performance by an order of magnitude [20]. Moreover, the required Garbage Collection (GC) operations in SSD significantly affect the user I/O performance [21], [22]. That is, in the flash storage, each 64-128 KB flash block must be erased in advance before any part of it can be re-written. Due to the sheer size of a block, an erase operation typically takes milliseconds to complete. The GC operations are triggered if there are not sufficient free space available within flash-based SSDs. Thus, the total data written to the flash-based SSDs has a direct relationship to the GC frequency and impacts the system performance. Existing studies have extensively applied the data compression and data deduplication technologies to reduce the total written data on the flash-based SSDs [8], [23], [24].

Second, the response time of a flash-based storage system tends to increase linearly with the request size [25]. In order to understand the relationship between the response time and user request size for SSDs, we use the IOmeter tool to test an SSD device (Intel X25-E 64 GB), under different request sizes. Fig. 1 shows the normalized results, which tracks an approximately linear correlation between the average response time and the request size for the SSD. The reason for this correlation is that the read and write operations are implemented entirely through electronic circuitry for both control and data signal transmissions, which makes the data transmission time directly related to the request size and the dominant part of the user response time.

These two unique characteristics of flash-based SSDs imply that it is feasible and practical to improve the write performance by reducing the write request size with the data compression technique. With data compression, less data is written to the flash-based SSDs, resulting in better performance and better endurance. They are also the main reason why in-line data compression has become a commodity feature in flash-based storage products [5], [6], [7].

2.2 Data Compressibility and Compression Efficiency

Lossless data compression techniques have the potential to reduce the data size effectively. However, certain types of file formats, such as TIF, JPEG, video and sound files, etc.,



(a) Accesses of small request sizes (b) Accesses of large request sizes

Fig. 1. The impact of request size on user response time of an Intel SSD with random accesses.

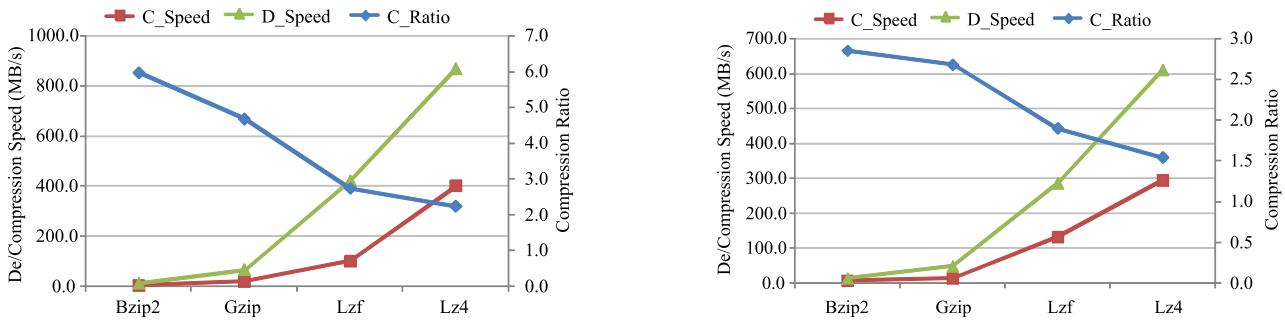
are non-compressible in practice because they are already in compressed formats. Applying data compression on these non-compressible files not only wastes system processing resources, but also significantly increases the I/O response time because the compression process sits on the IO critical path. On the other hand, previous studies also show that different compression algorithms have different compression ratios and compression/decompression speeds [13], [26]. To obtain a better understanding on compression efficiency, we conducted extensive experiments by using different compression algorithms on different data sets. Fig. 2 shows the compression efficiency of the different compression algorithms on two types of files: the Linux source files and the Mozilla Firefox files. The compression ratio is defined to be the size of the original data volume divided by the size of the compressed data, thus the higher the ratio the better (i.e., the higher the data reduction rate).

Clearly, different datasets have different compressibility and different algorithms achieve different compression ratios at different compression and decompression speeds. In general, an algorithm with a higher compression ratio is associated with a slower compression/decompression speed, and vise verse. Among the evaluated compression algorithms, the Bzip2 and Gzip algorithms achieve the best compression ratios but at the lowest compression/decompression speeds. Lz4 and Lzf achieve the lower compression ratios but at much higher compression/decompression speeds than Bzip2 and Gzip. The observed trade-offs between the compression ratio and speed among the different compression algorithms are the key to the design of our proposed elastic data compression for the flash-based storage systems.

2.3 Workload Characteristics and Motivation

Understanding workload characteristics is important for storage system design. Researchers have extensively collected and analyzed workloads on the storage I/O path, and found that burstiness and idleness are common among many applications [17], [27]. Fig. 3 plots the access patterns of two applications, i.e., the financial workload obtained from the Storage Performance Council [28] and the enterprise workload obtained from Microsoft Research Cambridge [29]. The figure shows that the accesses exhibit a mixed pattern of burstiness and idleness in terms of I/O intensity. With the help of the upper-layer optimizing techniques such as DRAM buffer and I/O scheduling, the I/Os seen at the lower level are usually bursty and clustered along the time dimension.

Flash-based storage devices are shrinking the latency between the CPU and HDD-based storage infrastructure layers. But the only way to make flash technology cost effective for the wide variety of enterprise application workloads is to use less physical flash capacity to store more actual data. Space efficiency techniques such as data compression accomplish this quite effectively. Data compression has been proved to be an effective way to reduce the data size and save storage space for flash-based storage systems. It is widely used to make flash-based products more affordable. However, not all data blocks are compressible. Good examples are compressed media formats that apply compression before data is written to the storage system. In these scenarios, employing data compression technologies likely provides little or no additional savings, as the efficiencies have already been attained at the application layer. In this case, it can be beneficial to disable data compression to avoid the



(a) Linux source code (b) Mozilla firefox files

Fig. 2. The compression efficiency of the different compression algorithms in terms of compression ratio and compression speed on two different data sets. Note: C_Speed denotes the compression speed, D_Speed denotes the decompression speed and C_Ratio denotes the compression ratio.

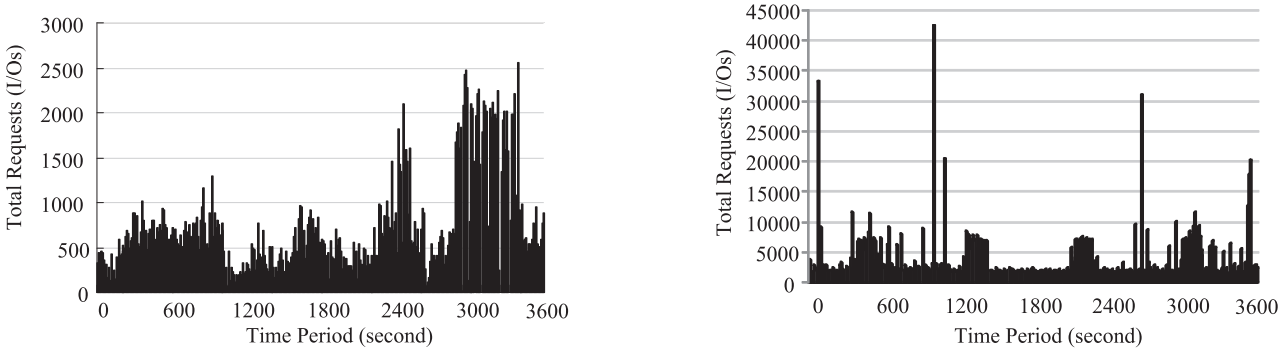


Fig. 3. The access patterns of the different applications exhibit clear burstiness and idleness for two applications: (a) OLTP application and (b) Enterprise workload.

latency impact. For the compressible data blocks, shrinking the overall size of the data set not only reduces the write traffic, but also improves the endurance of the flash-based storage systems.

However, this data reduction comes at the expense of additional compute overheads for compression and decompression. Recent studies have evaluated the data compression technology in the flash and NVM-based storage systems [8], [10], [11], [12] for the purpose of performance, space efficiency and reliability improvement. However, the diversity in compression algorithms and data compressibility associated with the bursty and clustering characteristics of the I/O workloads, while a potential opportunity for optimization of compression-based systems, has not been explored in previous studies and thus inspires us to rethink about the design of the compression-based flash storage systems. Applying data compression on non-compressible (or lowly compressible) data will directly degrade system performance, particularly in any bursty period when performance should be considered a first-priority design factor.

In other words, instead of using algorithms with higher compression ratios for all the requests all the time as is the case in existing compression-based flash storage products, including bursty periods where the I/O queue length will be increased to degrade the system performance, such compression algorithms are desirable only during an idle period to achieve higher space savings without noticeably affecting the system performance. Overall, we can see that the workload characteristics, including the data compressibility and I/O intensity have a significant impact on the data compression efficiency for flash-based storage systems. Based on these observations, an adaptive data compression scheme is preferred for flash-based storage systems to achieve a good balance between the performance and space efficiency, which motivates us to propose the elastic data compression scheme.

3 Elastic Data Compression

In this section, we first outline the main design objectives of the EDC system. Then we present the architecture and design details of EDC.

3.1 The Design Objectives of EDC

The design of EDC aims to achieve the following three objectives.

- *Improving the System Performance* - During the system's bursty periods, EDC will utilize data compression algorithms with lower overhead to reduce the I/O queue length. Moreover, for non-compressible data blocks, it will write them through to the flash storage directly without any compression, thus improving the system performance without noticeably sacrificing the space efficiency.
- *Improving the Space Efficiency* - By using data compression algorithms with higher compression ratios during the system's idle periods, the overall space efficiency is improved without degrading the system performance.
- *Improving the System Reliability* - The write traffic to and the amount of stored data on the flash-based storage system are significantly reduced by data compression. This leads to the number of block erase cycles to be significantly reduced, which improves the system reliability accordingly.

3.2 EDC Architecture

Fig. 4 shows an architectural overview of our proposed EDC within the storage subsystem and on the I/O path. EDC is located at the block device level that sits directly below the file system. This makes it possible for EDC to be incorporated into any existing file systems, such as Ext4 and F2FS [30]. Moreover, it directly controls the underlying flash-based storage system that can be either an Open-Channel SSD [31], a single SATA-based SSD, an SSD-based disk array or a set of pure flash chips.

As shown in Fig. 4, EDC has three main functional modules: Workload Monitor, Data Compression & Decompression Engine, and Request Distributer. The *Workload Monitor* module is responsible for monitoring the I/O accesses of applications, identifying the request type and data compressibility, and computing the I/O intensity. The value of I/O intensity is measured by the I/Os accessed Per Second (IOPS). The *Data Compression & Decompression Engine* module is responsible for compressing the incoming write data and decompressing the outgoing read data. Based on the I/O intensity value and the type of data determined by the *Workload Monitor* module, the *Data Compression & Decompression Engine* module will adaptively select the appropriate data compression algorithm or decide not to compress the data. The *Request Distributer* module is responsible for

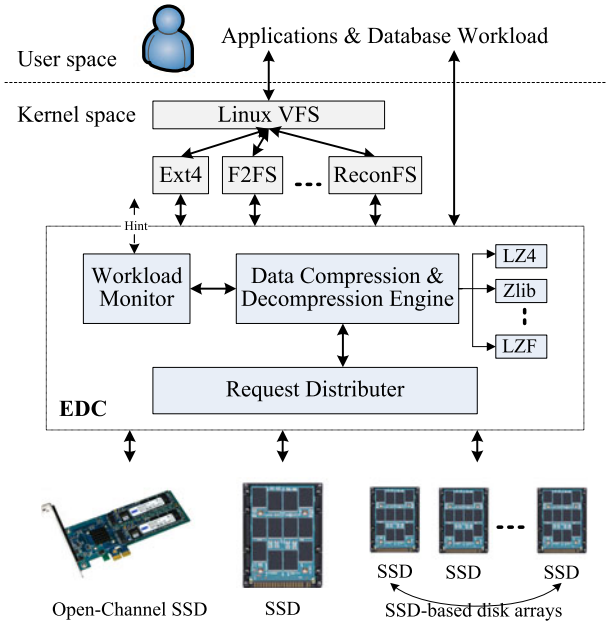


Fig. 4. Architecture overview of EDC.

issuing the processed data to or fetching the requested data from the flash-based storage subsystem, from or to the upper compression/decompression engine layer.

3.3 Data Structure

EDC is a block-level compression scheme that operates on fixed-size input data blocks from the upper layer applications. However, compression shrinks these data blocks variably due to the diversity in compressibility, generating data blocks of variable sizes. Therefore, there is a need to track the placement of the post-compression data blocks and the mapping between pre- and post-compression data blocks. Fig. 5 shows the data structure for the tracking mechanism. Since the flash translation layer (FTL) [2], at the heart of flash-based SSD control, uses an out-of-place update scheme, an overwrite or update request will only invalidate the old data block and the updated data is written to a new flash block, which further complicates things. For example, a 4096-Byte block is first compressed into a 1562-Byte block and written to the flash storage. After an update to this block, which entails a write to the uncompressed data block, the updated 4096-Byte block being compressed into a 2008-Byte block before writing to the flash, which means that the previously allocated space for this block is no longer sufficient to

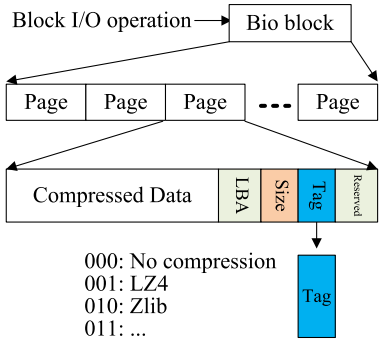


Fig. 5. Data layout for a compressed data block in EDC.

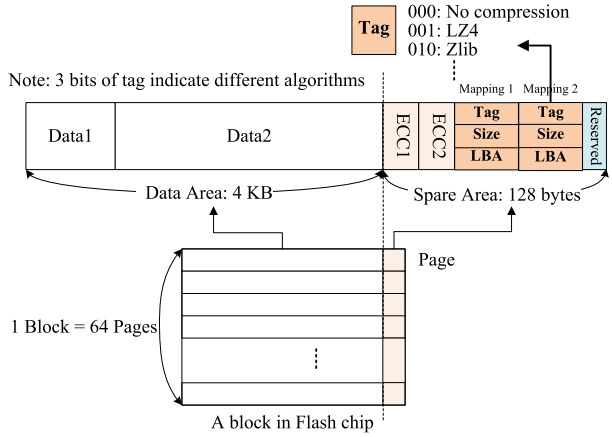


Fig. 6. The data layout for a compressed data page within FTL design in EDC.

store the newly compressed data. To overcome this problem, EDC allocates spaces to the compressed data blocks that are 75, 50 or 25 percent of their uncompressed (original) size, according to their compression ratios. If the compressed block is more than 75 percent of its original size, the data block is considered to be non-compressible and kept in its uncompressed form. Thus, the space can be well utilized and unnecessary fragmentation can be avoided [32].

Since EDC is designed for flash-based storage devices, it is also can be applied within flash-based SSDs, such as FTL layer. In such an environment, EDC can utilize the reserved space within each flash page to store the metadata of the compressed data pages. Fig. 6 shows the data layout for a compressed data page within FTL design in EDC. Section 4.3 also validates how this design by experimental results on an Open-Channel SSD [31]. The mapping information records how the compressed data and metadata is stored within a data block/page. It includes three important fields: *LBA*, *Size*, and *Tag*. The Tag field contains 3 bits that record the corresponding compression algorithm used for the given data block, where "000" indicates no compression is applied. The LBA field contains the logical block address of the beginning of the compressed data block and the Size field indicates the compressed data size.

3.4 Workload Monitor

The design of EDC is highly dependent on workload characteristics, especially the I/O intensity and the data compressibility. Thus, online detecting the workload characteristics is an important module in EDC. The I/O intensity measurement is an important factor for EDC in deciding the appropriate compression algorithm to use, as shown in Fig. 7. Besides the "raw" I/O request rate (raw IOPS) issued to the storage subsystem, the request size is also measured

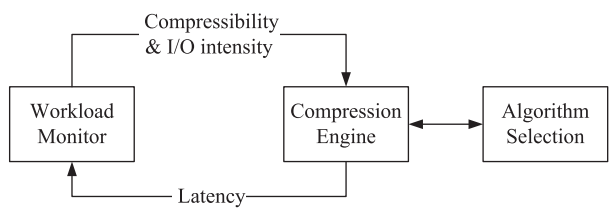


Fig. 7. The feedback mechanism in EDC for the selection of the appropriate compression algorithm.

in the monitoring scheme because it is the combination of IOPS and request size that determines the I/O intensity (I/O bandwidth requirement) for the flash-based storage system. In EDC, we quantify the I/O intensity by the number of 4 KB requests issued to the flash-based system per second, which we call *calculated IOPS*, where 4 KB is the default page size in Linux. In other words, when calculating the I/O intensity, EDC will convert a large request (of size greater than 4 KB) into multiple 4 KB requests. For example, one 8 KB request is traded as two 4 KB requests. By using the calculated IOPS, the latency involved in the data compression is also considered in the feedback mechanism.

The I/O intensity directly affects the user perceived responsiveness. During the I/O-intensive period, lengthening compression latency will further increase the queue length, thus significantly increasing the request response time. Under such circumstances, fast compression algorithms or temporally turning off the data compression will be preferred. Based on the calculated IOPS, EDC selects the most appropriate compression algorithm or does not apply data compression. It will set several calculated-IOPS thresholds for different compression algorithms. If the user workload I/O intensity falls in a specific I/O intensity range (i.e., between two neighboring thresholds), exceeds or is less than the specific calculated-IOPS threshold, the corresponding pre-selected compression algorithm will be applied to the incoming data blocks. Moreover, if the I/O intensity exceeds the highest calculated-IOPS threshold, EDC will skip the compression function to achieve the best performance.

On the other hand, data compressibility is also an important factor that affects the selection of data compression algorithms. In our current design, EDC only exploits the data compressibility in a simple way. Currently, EDC checks the data compressibility with a random sampling technique within SDGen [33]. Random sampling is a good indicator of many properties of the data content, particularly for properties that can be expressed by averages and sums such as compression ratio [33], [34]. Previous studies show that using random sampling on chunks is a good estimation for compression ratio within an additive percentage factor [33], [35]. For example, SDGen can identify the data compressibility and emulate compression ratios from 2.5 KB out of 64 KB [33]. Though the sampling's accuracy is not 100 percent, it is well within what is required for evaluations.

Fig. 8 shows the selection workflow of the compression algorithms in EDC based on the workload characteristics. When receiving the write requests, the data compressibility is first checked to determine whether the data compression should be applied. For the non-compressible data blocks, compressing them will introduce computing overhead with little benefit. EDC will write these data blocks through to the flash storage directly, skipping any compression. In other words, the data compressibility will determine whether or not EDC applies data compression on the data. For the compressible data blocks, EDC will apply data compression on it. The selection of compression algorithms is dependent on the I/O intensity characteristics. If the I/O intensity exceeds a high mark threshold, low compression algorithm is used. If the I/O intensity exceeds a low mark threshold, high compression algorithm is used. Otherwise, medium compression algorithm is used.

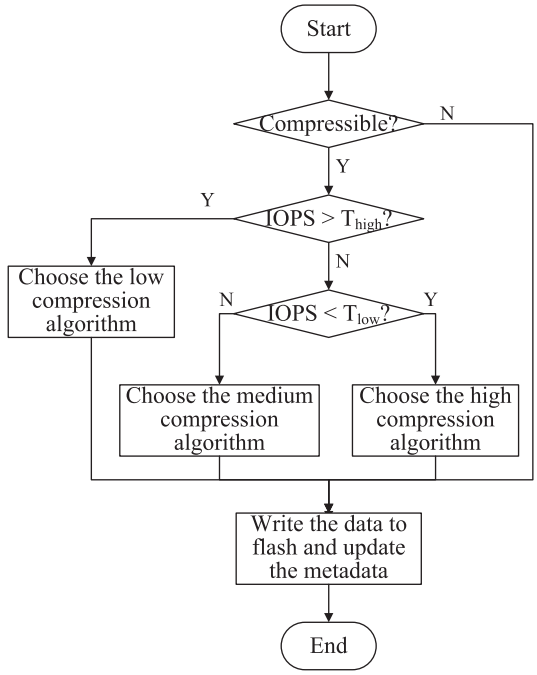


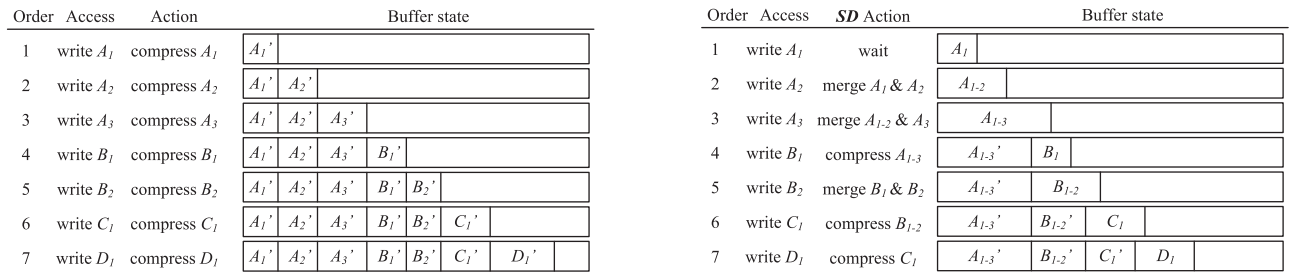
Fig. 8. The selection workflow of the compression algorithms in EDC based on the workload characteristics.

3.5 Data Compression and Decompression

Data compression works on the write path. Upon receiving a write steam, data compressibility of the data stream will be checked. If the compressibility is below a threshold, the data stream will be written without data compression. Otherwise, the written data will be stored in a compressed format. Based on the I/O intensity, data blocks may be compressed with different data compression algorithms. EDC uses the 3-bit Tag information to record the specific compression algorithm for the corresponding data blocks, as shown in Fig. 5. Previous studies have shown that the larger the data block, the higher the compression ratio can be achieved. Moreover, for the same total amount of data, a smaller number of larger data blocks are usually decompressed much faster than a larger number of smaller data blocks [8], [13]. Based on these studies and findings, EDC combines multiple sequential write data blocks into a single large block to improve the system compression efficiency.

Write requests rarely arrive sparsely, but in bursts most of the time, which is very common in real-world workloads [17], [27]. If a write data block is compressed immediately when it is written into the flash-based storage system, it is likely to miss the opportunity to combine with the subsequent, contiguous blocks into a larger block, resulting in reduced I/O efficiency. For example, suppose that the write requests arrive in the following order: A_1, A_2, A_3, B_1, B_2, C_1, and D_1. A_1, A_2 and A_3 are physically sequential, so they can be combined into a larger block, namely, A_{1-3}. If data is compressed before combining, the sequential data will not be combined, thus losing the opportunity of finding the same patterns for further compression among contiguous data blocks [13], as shown in Fig. 9(a).

To solve this problem, EDC uses a *Sequentiality Detector (SD)* to detect the sequential user accesses. The key here is to detect and merge as many contiguous write requests as possible to form a single larger one before compressing it.



(a) Data compressing without SD (b) Data compressing with SD

Fig. 9. A comparison of data block compressing workflow without and with SD. Data blocks are described as follows: $\mathbf{A}_1$ denotes the data block in the uncompressed form, $\mathbf{A}_1$' denotes the data block in the compressed form.

This write data contiguity (write sequentiality) is broken when a read request or non-contiguous write request arrives, at which point the currently detected and merged contiguous write requests are compressed in a single block. More specifically, when a request arrives, SD first checks whether it is a read request. If yes, its preceding write requests detected to be contiguous and merged are compressed in a single merged block. If it is a write request, SD checks whether it is sequential with its preceding write requests still waiting for more contiguous write requests to merge. If not, these preceding write requests are compressed in a single block. Otherwise, it is contiguous with these preceding write requests, and SD merges them together and continues to seek opportunity to merge with the subsequent requests. SD determines the sequential relationship between two requests based on their LBA and size values. If the LBA plus the size of a previous request is equal to the LBA of the current request, the two requests are considered sequential and merged. It must be noted that the requests with different arriving timestamps are not merged since they belong to different files. The data block compressing flow with SD is illustrated in Fig. 9b. Thus, with SD, as illustrated in Fig. 9b, all sequential write requests are combined before they are compressed.

Data decompression works on the read path. Upon receiving a read request, when the fetched data is read from the flash to the host memory, the data will be decompressed according to the *Tag* value. After the data is decompressed, it will be returned to the upper layer applications. For the uncompressed data blocks, they will be returned directly. Although the data decompression process will introduce extra computing overhead on the read path, the overall response time is not increased. The reason is that the stored data size is reduced by data compression, compared with the system without data compression. Thus the time spend on reading the data from the flash to the memory is reduced, which is elaborated in section 2.1. Furthermore, the decompression speed is significantly faster than the compression speed, as shown by our experimental results in Section 2.2 and the previous studies [8], [11]. The reduced read response time can offset the increased decompression overhead. Thus the overall read response times are not affected. It is also validated by our experimental results in Section 4.

4 Performance Evaluation

In this section, we first describe the evaluation setup and methodology. Then we evaluate the performance of the EDC prototype on different flash-based storage systems through trace-driven experiments.

4.1 Evaluation Setup and Methodology

Experimental platform: We have implemented an EDC prototype on top of the Linux operating system. The performance evaluation is conducted on a server with an Intel Xeon X5680 processor (3.33 GHz), 8 GB DDR memory and an attached SSD array. The array is composed of five SSDs of the Intel X25-E Extreme SATA SSD 64 GB (denoted as Intel X25-E SSD). We also use an Open-Channel SSD (CNEX Labs Westlake SDK [31]) with 2TB NAND MLC Flash in the experiments, denoted as OCSSD in the rest of this section. A separate HDD is used to house the operating system (Red Hat Enterprise Linux Server release 6.2) and other software. The experimental setup is outlined in Table 1.

Evaluation Baselines: In the experiments, we compare EDC with a system without any data compression, labeled *Native*, and a system with fixed compression algorithms, including Lzf, Gzip, and Bzip2, labeled *Lzf*, *Gzip* and *Bzip2*, that represent the latest flash-based storage products with always-on inline compression for all workloads [5], [6], [7]. For example, storage companies such as Nimble Storage and Pure Storage use the Lempel-Ziv style (LZ*) data compression algorithms [5], [6]. In the evaluations, we measure the space efficiency in terms of the compression ratio and measure the performance in terms of the average response time. Moreover, since EDC aims to achieve a balance between the space saving and the performance, we also use a composite metric of compression-ratio divided by response-time to quantify the overall benefit of EDC. Clearly, this metric attempts to assess a combined benefit of a scheme in terms of both compression ratio and performance, where the higher the value of this metric, the more beneficial this scheme is.

Workload and Compression Characteristics: The traces used in our experiments are obtained from the Storage

TABLE 1
Experimental Setup

Machine	Intel Xeon X5680, 8GB RAM
OS	Red Hat Enterprise Linux Server 6.2
Device adapter	PERC H710 SATA controller
Flash-based SSDs	Intel X25-E 64GB SATA SSD CNEX Labs 2TB Open-Channel SSD
Traces	OLTP [28] MSR Traces [29]
Trace generation	SDGen [33]
Compression algorithms	Lzf, Gzip, Bzip2

TABLE 2
The key Characteristics of Evaluation Workloads

Traces	Read Ratio	IOPS	Average Request Size (KB)
Fin1	32.8%	52	11.9
Fin2	82.4%	127	6.2
Usr_0	41.6%	4	20.9
Prxy_0	2.7%	19	2.5

Performance Council [28] and Microsoft Research Cambridge [29]. The two financial traces (Fin1 and Fin2) were collected from OLTP applications running at a large financial institution. The other two traces (Usr_0 and Prxy_0) were collected from storage volumes in an enterprise environment in Microsoft Research Cambridge. These traces represent different access patterns in terms of read/write ratio, raw IOPS and average request size, with their main workload parameters being summarized in Table 2. Since no real content is included in the traces, we use the SDGen scheme [33] to collect the data samples from real applications to emulate the compression characteristics. SDGen not only creates data with variable compression ratio, but also mimics the other properties and behaviors of data compression such as compression time and heterogeneity that are critical to system performance evaluation. More details about SDGen can be found in [33] and the GitHub website [36].

4.2 Performance Results

Fig. 10 shows the data compression ratios of different schemes normalized to that of the Native system (i.e., without any compression). The Bzip2 compression algorithm achieves the best data compression ratio, followed by the Gzip compression algorithm. The Lzf compression algorithm achieves the lowest data compression ratio. In contrast, EDC has an average compression ratio of 1.5, which is better than that of the Lzf algorithm and lower than that of both the Bzip2 and Gzip algorithms. The reason is that EDC uses both the Gzip and Lzf compression algorithms during different periods of workload intensity to achieve a balanced space saving between them. The data compression ratio is directly related to the space saving for the flash-based storage systems, the higher the better. Since the cost per GB of SSDs is much higher than HDDs, improving the flash storage efficiency is very important to make flash technology cost effective. However, improving storage efficiency cannot directly degrade the system performance. Algorithm with higher compression ratio is associated with higher compression/decompression latency, especially for the non-compressible data blocks and the I/O-intensive periods. Moreover, the space efficiency is not the sole objective for compression-based storage systems. As we will see from the following results, high compression ratio usually comes at the cost of high access latency.

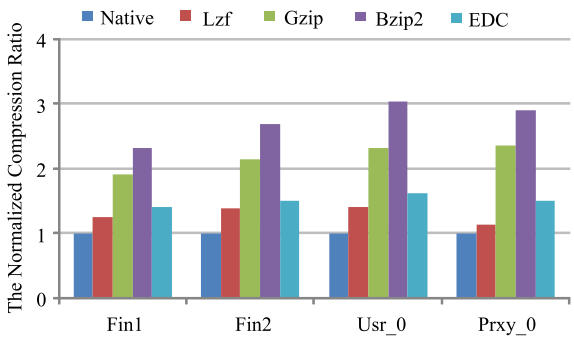


Fig. 10. A comparison of compression ratio, normalized to that of the Native scheme (without any compression), among different schemes under various workloads.

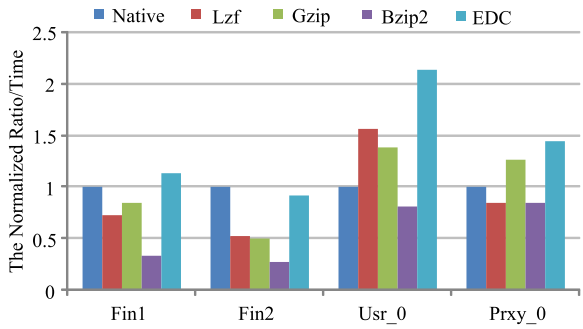


Fig. 11. A comparison of the Ratio/Time results, normalized to that of the Native scheme (without any compression), among different schemes under various workloads.

Fig. 11 shows the space-performance results in terms of a composite metric of compression-ratio divided by response-time, whose value is the larger the value. We can see that, when combining the two design objectives together, the fixed compression schemes are less beneficial than the Native system, especially for Fin1 and Fin2 traces. The reason is that these fixed compression schemes usually only consider one design objective of compression ratio while ignoring the other design objective of performance, resulting in an overall reduced composite measure. In contrast, EDC performs the best among all the compression schemes and even better than the Native system, except for the Fin2 trace. The reason is that the design objectives of EDC consider both the performance and the compression ratio, achieving a good balance between them.

Fig. 12 compares the response time, normalized to that of the Native scheme, on a single SSD among different schemes, driven by the four traces. We can see that EDC outperforms all the other compression schemes in the response time measure. For example, compared with the Lzf scheme, it reduces the average response time by up to 61.4 percent for the Fin1 trace, with an average of 36.7 percent. Compared with the Gzip and Bzip2 compression algorithms, EDC reduces the

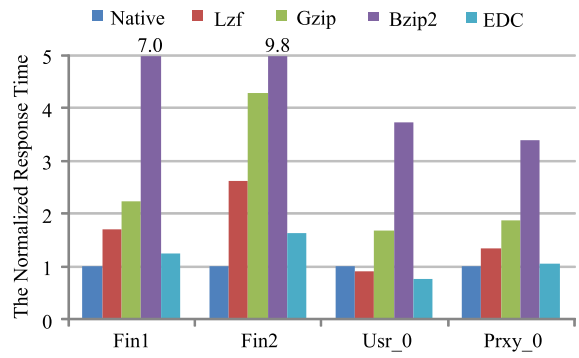


Fig. 12. A comparison of response time, normalized to that of the Native scheme, on a single SSD among different schemes under various workloads.

response time by an average of 2.1× and 4.9×, respectively. The reasons are two-fold. First, EDC does not apply data compression during periods when the I/O intensity is very high, which helps achieve the best system responsiveness. Even during the system idle periods, it does not apply data compression on non-compressible data blocks, which further eliminates the unnecessary computing overhead. Second, EDC exploits the I/O intensity characteristics to choose the most appropriate compression algorithm between Lzf and Gzip for the compressible data blocks. It reduces the average response time by reducing the long queuing latency during the I/O-intensive periods and achieves comparable space efficiency during the system idle periods. Thus EDC achieves a better balance between the system performance and the space efficiency than the other data compression schemes.

On the other hand, we see that the Bzip2 compression algorithm has the highest access latency, by up to 9.8 times more than the Native system, which is clearly unacceptable for the end users. Though data compression reduces the write request sizes, the write latency is not reduced accordantly. The reason is that the data compression and decompression speeds of Bzip2 are much lower than the bandwidth of the flash-based SSD, as shown in Fig. 2. Many user requests will be waiting in the I/O queue, which significantly degrades the system performance. Thus, the overhead introduced by directly using Bzip2 will offset the advantages of reducing the request sizes from the viewpoint of system performance. The Gzip compression algorithm shows a similar trend to that of the Bzip2 compression algorithm. In contrast, Lzf is shown to achieve much better average response time, even better than that of the Native system for the Usr_0 trace. The reason is that data compression technique reduces the request size, which in turn reduces the time spent on writing and reading the data to/from the flash accordingly. As a summary, EDC adaptively exploits the diversity of different data compression algorithms to fully adapt the workload characteristics, including data compressibility and I/O intensity, thus achieving the best system performance among all the schemes for flash-based storage systems.

A single SSD cannot satisfy the performance, capacity and reliability requirements in enterprise storage systems. Thus, applying the RAID (Redundant Array of Independent Disks) [37] algorithm to SSDs is a promising approach to building large-scale high performance and highly reliable SSD-based storage systems [18], [38]. In this paper, Redundant Array of Independent SSDs is abbreviated as RAIS. The different levels of RAIS are also abbreviated as RAIS0, RAIS5 and so on. To evaluate EDC's efficiency on multiple SSDs, we also build a software RAIS5 system consisting of five Intel X25-E SSDs. Fig. 13 shows the access latency, normalized to that of the Native scheme, on the RAIS5 system for different schemes. We can see that EDC's affect on system performance is much better than the other data compression schemes, even much better than the lower data compression scheme LZF. The reason is that EDC can reduce the request size much more than LZF scheme and does not affect system performance by considering the I/O intensity characteristics. On the other hand, we see that higher data compression schemes affect system performance significantly. The results on a RAIS5 system show a similar trend to that of a single SSD for the different schemes driven by the four traces. The results also validate EDC's applicability to and effectiveness for different flash-based storage systems.

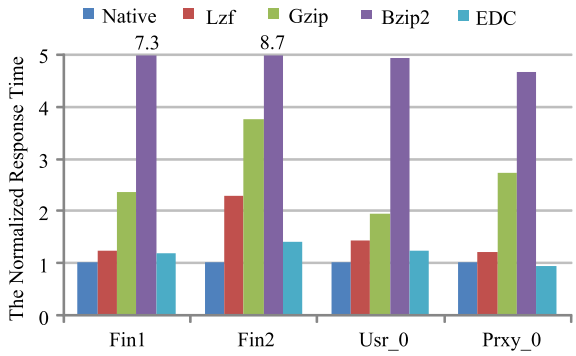


Fig. 13. A comparison of response time, normalized to that of the Native scheme, on a RAIS5 system consisting of five SSDs for different schemes, driven by the various workloads.

In modern large-scale storage systems such as Microsoft Bing, Facebook, and Amazon, the long tails of the service latency are of particular concerns. EDC's optimization is highly dependent on the I/O intensity, which is directly aware of the I/O latency. To see the EDC's efficiency on the tail latency, Fig. 14 shows the response time distributions for the different schemes driven by the various workloads. First, we can see that the response time distributions of EDC is similar to that of Native system. The reason is that EDC takes the workload characteristics into design considerations which does not affect system performance. The data compression's impact on the latency is alleviated for EDC. Second, we can see the medium and high compression schemes, Gzip and Bzip2, significantly affect system performance driven by the 4 workloads, especially increase the percentage of the long latency. The reason is obvious in that higher data compression schemes associated with higher compression and decompression overhead which increases the latency significantly during I/O intensity period. Moreover, the higher compression and decompression overhead also increases the queue length which further increases the I/O latency. Thus, we can see that Gzip and Bzip2 compression algorithms have much more higher latency requests than LZF and EDC schemes.

These results further validate that data compression is a double-edged sword for flash performance. On the one side, data compression reduces the request sizes thus reducing the I/O latency. On the other hand, data compression and decompression also consume system processing resources thus increasing the I/O latency, especially for the non-compressible data chunks and during the I/O intensity periods. Thus applying data compression on flash-based storage systems should be carefully designed. By exploiting the workload characteristics, including data compressibility and I/O intensity, EDC can achieve a much better trade-off between performance and space efficiency for flash-based storage systems.

4.3 Performance on an Open-Channel SSD

Open-Channel SSDs differ from traditional SSDs in that they expose the internal parallelism of the flash chips to the host and allows it directly manage the flash chips. By integrating the flash translation layer into the host, workload

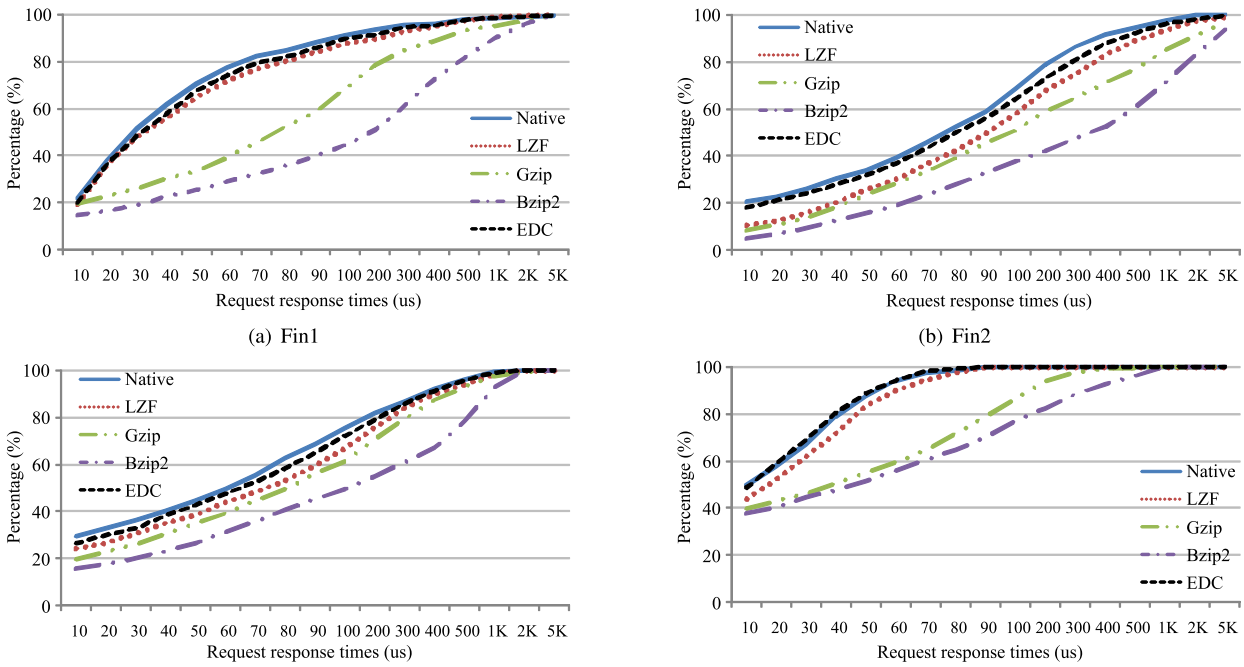


Fig. 14. Response time distributions for the different schemes driven by the various workloads, where the *X*-axis shows the request response times and the *Y*-axis indicates the fraction of requests with response times lower than the corresponding values on the *X*-axis.

optimizations can be applied either within a self-contained flash translation layer, file-system integration or applications themselves. Moreover, the Linux operating system kernel has already supports the Open-Channel SSDs which follow the NVM Express specification, by providing an abstraction layer called LightNVM and pblk [31].

Fig. 15 shows a comparison of response time on an open-channel SSD for different schemes, normalized to that of the Native scheme driven by the various workloads. We see that EDC performs the best among all the compression schemes and is the most approach the Native system. The reason is the intelligent choice of different compression strategies of EDC by exploiting the data compressibility and I/O intensity characteristics. Due to the high performance characteristics of NVMe-based SSDs, the process overhead is much more significant on the critical I/O path. Thus the selection of the suitable data compression algorithms is very important which implies that the fixed data compression schemes are not efficient. By skipping the non-compressible (or very lowly compressible) data blocks, the unnecessary compression overhead is avoid because applying data compression on these data blocks brings no advantages. Moreover, during system intensive period, using the lower data compression algorithm can significantly alleviate the request waiting overhead in the I/O queue, thus further improving system efficiency. The results on the Open-Channel SSD also further validates the applicability of EDC on different flash-based storage devices.

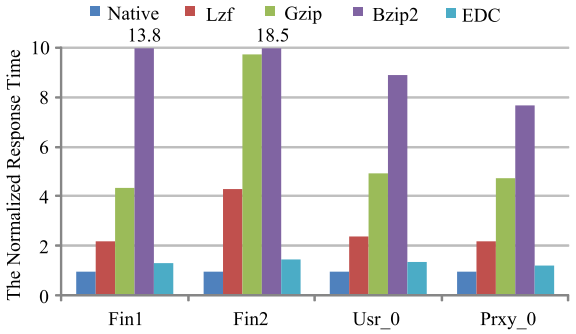


Fig. 15. A comparison of response time, normalized to that of the Native scheme, on an Open-channel SSD for different schemes, driven by the various workloads.

On the other hand, we see that medium and high compression schemes perform even worse than that on the SATA-based SSDs. The reason is that Open-Channel SSD has much lower read/write latency than that of the SATA-based SSDs [31]. The compression/decompression latency dominates the overall request response time. Moreover, the fixed compression schemes apply the data compression on all the data streams no matter they are compressible or not. Thus the incurred computing overhead is consistent on the critical I/O path. Accompanied with the I/O intensity, the queueing effect will make the I/O latency even worse in the medium and high compression schemes.

Though the data compression can reduce the request size and the total written data to the flash-based storage devices, the incurred processing overhead will significantly degrade system performance on modern flash-based storage devices. By sacrificing the flash performance for reliability and storage efficiency improvement is not acceptable for end users. Besides that, some studies and flash products use data compression to improve flash efficiency [6], [8]. With the improved performance of modern flash-based storage devices, this objective becomes nontrivial.

4.4 Sensitivity Study

One important design factor in EDC is the IOPS threshold that determines the selection of the most appropriate data compression algorithm for a given I/O intensity and

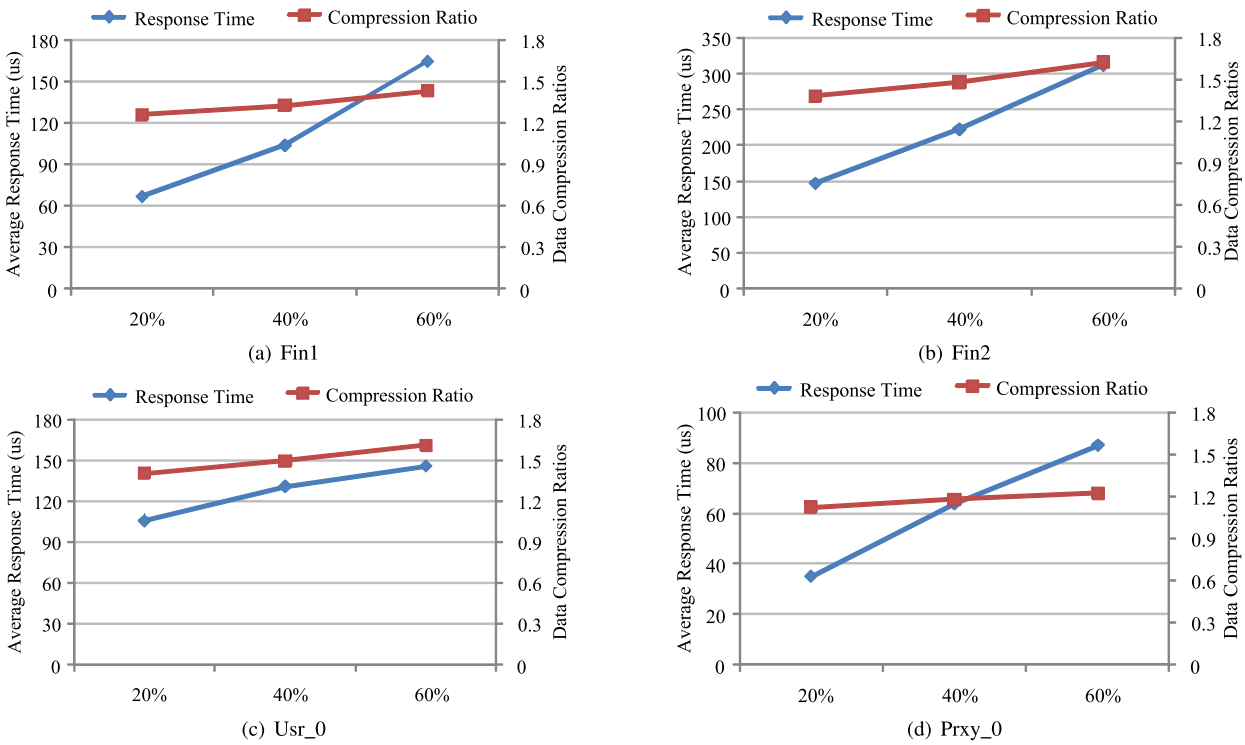


Fig. 16. The sensitivity of EDC's performance and compression ratio to the IOPS threshold, driven by the various workloads on a RAIS5 system consisting of five SSDs.

compressibility. Since we use the percentage of the calculated IOPS (see Section 3.4) as a metric for the I/O intensity threshold, we conduct sensitivity experiments on different threshold values (percentages). Moreover, we set the non-compression percentage unchanged and only change the calculated IOPS between the Lzf and Gzip compression algorithms.

Fig. 16 shows the sensitivity study results driven by the various workloads on a RAIS5 system consisting of five Intel X25-E SSDs. Take fin2 trace as an example, we can see that as the percentage of requests that use the Gzip compression algorithm increases, the data compression ratio is increased. However, the overall system response time is also increased significantly and rapidly. The reason is that as the requests compressed with the Gzip algorithm increase, the overall compression ratio and the system response times are both increased. Thus the appropriate percentage for the Gzip algorithm is 20 percent for a better balance between performance and space saving in our experiments for the Fin2 trace. The trend is also similar driven by the other workloads. However, the parameter is configurable to allow system administrators to achieve a much better balance between the performance and the space efficiency.

5 Related Work

Data compression and its effect on computer systems have been well studied in the literature. Recently, the data compression technology has been evaluated for its use in the flash and NVM-based storage systems [8], [10], [11], [12], [35], [40] for the purpose of performance, space efficiency and reliability improvement. Some studies have demonstrated how compression can be integrated into the FTL [40]. Their results show that data compression can reduce the write traffic to the storage medium and alleviate write amplification. However, applying data compression within the FTL will consume computing and memory resources in SSDs. To address this problem, Lee et. al [41] propose to use hardware assisted data compression to reduce the computing overhead. On the other hand, NVM-Compression [11] is designed to combine the best of application level compression and flash-aware integration by extending FTL capabilities.

Besides the studies on data compression integrated into SSDs, there are studies on the host-level data compression for SSD-based storage systems. Makatos et. al [8] propose to apply data compression to SSD to enlarge SSD-based cache for disk-based storage systems. Li et. al [10] propose and investigate an implicit data compression strategy to reduce cycling-induced flash memory cell physical damage and hence improve storage device lifetime. However, all these schemes use fixed compression algorithms for flash-based storage systems and ignore the performance and space impact of the user access intensity and data compressibility characteristics. Our proposed EDC scheme is orthogonal and complementary to these schemes and can be easily incorporated into these schemes to further improve system performance and space efficiency.

Data deduplication, another lossless data reduction technology, has been widely adopted for flash-based storage systems to improve their performance, reliability and space efficiency. CA-FTL [23] and CA-SSD [24] are two representative studies that apply the data deduplication technology to reduce write traffic to flash chips within SSDs. Delta-FTL [39] reduces write traffic to flash chips through extensive write buffering that is coupled with selective storing of compressed deltas for a small portion of the data. Flash-based storage companies, such as Nimble Storage [5] and Pure Storage [6], have incorporated both data compression

TABLE 3
Comparison of the Different Schemes Related to EDC

Schemes	Devices	Methods	Comment
FlaZ [8]	SSD	Fixed compression	Leveraging multicore CPUs to mitigate compression and decompression overheads
CA-FTL [23]	Flash	Deduplication	Careful design to reduce the computing and memory overhead within SSDs
Delta-FTL [39]	Flash	Delta compression	Selective storing of compressed deltas between the new and old data
EDC	Flash	Adaptive compression	Exploiting the workload characteristics to select different compression algorithms

and data deduplication in their products to improve the system performance and storage efficiency.

Table 3 shows the comparison of EDC with the related schemes based on several important characteristics for flash-based storage systems. From these studies, we see that compression/decompression overhead is indeed an important design issue for flash-based storage systems. Different from these studies, EDC takes the workload characteristics into the compression designs to achieve a better balance between performance and space efficiency.

While some adaptive data compression approaches have been proposed for network transmission on different types of data [15], [16], none of these studies has focused on flash-based storage systems. Our EDC study is inspired by these previous studies in the aspect of reducing write traffic to flash chips to improve system performance and reliability. However, EDC is different from the above studies in that it not only leverages data compression to reduce the write traffic, but also exploits workload characteristics and the diversity of compression algorithms to improve the system performance and reliability.

In addition to storage systems, memory/cache and network systems have also been targeted for performance, energy and space efficiency improvement by applying compression techniques [42], [43], [44], [45]. Alameldeen and Wood [42] propose to take advantage of small values to create a compression algorithm called Frequent Pattern Compression (FPC) to effectively increase CPU L2 cache capacity for program performance improvement. Ekman and Stenstrom [43] propose a main-memory compression scheme to practically eliminate performance losses by a highly-efficient structure for locating a compressed block in memory, and a hierarchical memory layout that allows compressibility of blocks to vary with a low fragmentation overhead. Tuduce and Gross [45] present an adaptive main memory compression system to improve the application performance when the main memory does not have sufficient capacity to satisfy the application's requirement. All these studies demonstrate that the application data are compressible and both the system performance and space efficiency can be improved by the data compression technology, which further validates the viability and feasibility of our EDC scheme for flash-based storage systems.

6 Conclusion

Data compression is an important technique to improve the performance and space efficiency for flash-based storage systems. However, employing fixed compression algorithms, as in most current flash-based storage products that incorporate data compression, fails to recognize and exploit the significant diversity in compressibility and access patterns of data and misses the opportunity to improve system performance, space efficiency or both. EDC is proposed in this paper to exploit the compression diversity of the workload characteristics. More specifically, for compressible data blocks EDC employs algorithms with higher compression ratios in time periods with lower system utilization and algorithms with lower compression ratios in time periods with higher system utilization. For non-compressible (or very lowly compressible) data blocks, it will write them through to the flash storage directly without any compression. Our extensive trace-driven evaluations on a lightweight implementation of the EDC prototype show that EDC achieves a much better trade-off between performance and space efficiency than the state-of-the-art schemes with fixed algorithms.

EDC is an ongoing research project that offers several directions for future research. First, we will further examine the data compressibility characteristic of data under different compression algorithms by exploiting semantic information about application and file type. For instance, the file type information can be incorporated into the EDC design, so that different compression algorithms are responsible for different data content in different file types. Second, we will conduct more experiments on other storage devices, such as HDD-based and NVM-based storage systems, to evaluate the efficiency of the EDC prototype. Third, we will investigate EDC's impact on system energy consumption, given its dichotomy of compression/decompression that consumes additional energy and data reduction that decreases data movement and thus energy consumption. Finally, we will conduct additional experiments to evaluate the EDC's efficiency on the reliability of the flash-based storage systems. Since data compression will reduce the request size and improve the space efficiency, it will also improve the endurance of the flash-based storage systems.

Acknowledgments

We thank the anonymous reviewers from IPDPS 2017 for constructive comments and feedback on the paper. We also thank the CNEX Labs for providing us the Open-Channel SSD and technique support. This work is supported by the National Natural Science Foundation of China under Grant No. 61772439, No. U1705261, No. 61472336 and No. 61402385, the US National Science Foundation under Grant No. CCF-1704504 and No. CCF-1629625. Suzhen Wu is the corresponding author. This is an extended version of our manuscript published in the Proceedings of the 31st IEEE International Parallel & Distributed Processing Symposium (IPDPS'17), Orlando, Florida USA, May 29-June 2, 2017.

REFERENCES

[1] Y. Deng, "What is the future of disk drives, death or rebirth?" *ACM Comput. Surveys*, vol. 43, no. 3, 2011, Art. no. 23.

[2] N. Agrawal, V. Prabhakaran, T. Wobber, D. J. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2008, pp. 57–70.

[3] B. Mao and S. Wu, "Exploiting request characteristics and internal parallelism to improve SSD performance," in *Proc. 33rd IEEE Int. Conf. Comput. Des.*, Oct. 2015, pp. 447–450.

[4] B. Mao, H. Jiang, S. Wu, Y. Yang, and Z. Xi, "Elastic Data Compression with Improved Performance and Space Efficiency for Flash-based Storage Systems," in *Proc. 31st IEEE Int. Parallel Distrib. Process. Symp.*, Jun. 2017, pp. 1109–1118.

[5] CASL Architecture in HPE Nimble Storage, Sep. 2016. [Online]. Available: http://www.nimblestorage.com/products/architecture.php

[6] Purity Reduce in Pure Storage, Sep. 2016. [Online]. Available: https://www.purestorage.com/products/purity/purity-reduce.html

[7] Tintri VMstore, Sep. 2016. [Online]. Available: http://info.tintri.com/vmstore-whitepaper

[8] T. Makatos, Y. Klonatos, M. Marazakis, M. D. Flouris, and A. Bilas, "Using transparent compression to improve SSD-based I/O caches," in *Proc. EuroSys Conf.*, Apr. 2010, pp. 1–14.

[9] X. Zhang, J. Li, H. Wang, K. Zhao, and T. Zhang, "Reducing Solid-State Storage Device Write Stress through Opportunistic In-place Delta Compression," in *Proc. 14th USENIX Conf. File Storage Technol.*, Feb. 2016, pp. 111–124.

[10] J. Li, K. Zhao, X. Zhang, J. Ma, M. Zhao, and T. Zhang, "How Much Can Data Compressibility Help to Improve NAND Flash Memory Lifetime?" in *Proc. 13th USENIX Conf. File Storage Technol.*, Feb. 2015, pp. 227–240.

[11] D. Das, D. Arteaga, N. Talagala, T. Mathiasen, and J. Lindström, "NVM Compression–Hybrid Flash-Aware Application Level Compression," in *Proc. 2nd Workshop Interactions NVM/Flash Operating Syst. Workloads*, Oct. 2014, pp. 1–10.

[12] A. Zuck, S. Toledo, D. Sotnikov, and D. Harnik, "Compression and SSDs: Where and how?" in *Proc. 2nd Workshop Interactions NVM/Flash Operating Syst. Workloads*, Oct. 2014, pp. 1–10.

[13] N. K. Edel, E. L. Miller, K. S. Brandt, and S. A. Brandt, "Measuring the compressibility of metadata and small files for disk/nvram hybrid storage systems," in *Proc. Int. Symp. Perform. Eval. Comput. Telecommun. Syst.*, Jul. 2004, pp. 1–10.

[14] A. El-Shimi, R. Kalach, A. Kumar, A. Oltean, J. Li and S. Sengupta, "Primary Data Deduplication - Large Scale Study and System Design," in *Proc. USENIX Conf. Annu. Tech. Conf.*, Jun. 2012, pp. 26–26.

[15] C. Pu and L. Singaravelu, "Fine-Grain Adaptive Compression in Dynamically Variable Networks," in *Proc. 25th Int. Conf. Distrib. Comput. Syst.*, Jun. 2005, pp. 685–694.

[16] E. Zohar and Y. Cassuto, "Automatic and Dynamic Configuration of Data Compression for Web Servers," in *Proc. USENIX 28th Large Installation Syst. Admin. Conf.*, Nov. 2004, pp. 97–108.

[17] A. Riska and E. Riedel, "Disk Drive Level Workload Characterization," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2006, pp. 9–9.

[18] B. Mao, H. Jiang, D. Feng, S. Wu, J. Chen, L. Zeng, and L. Tian, "HPDA: A hybrid parity-based disk array for enhanced performance and reliability," in *Proc. 24th Int. Parallel Distrib. Process. Symp.*, Apr. 2010, pp. 1–12.

[19] S. Wu, B. Mao, X. Chen, and H. Jiang, "LDM: Log disk mirroring with improved performance and reliability for SSD-based disk arrays," *ACM Trans. Storage*, vol. 12, no. 4, pp. 1–22, 2016.

[20] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Comput. Survey*, vol. 37, no. 2, pp. 138–163, 2005.

[21] J. Guo, Y. Hu, B. Mao, and S. Wu, "Parallelism and Garbage Collection aware I/O Scheduler with Improved SSD Performance," in *Proc. 31st IEEE Int. Parallel Distrib. Process. Symp.*, Jun. 2017, pp. 1184–1193.

[22] S. Wu, Y. Lin, B. Mao, and H. Jiang, "GCaR: Garbage collection aware cache management with improved performance for flash-based SSDs," in *Proc. 30th Int. Conf. Supercomputing*, Jun. 2016, Art. no. 28.

[23] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proc. 9th USENIX Conf. File Storage Technol.*, Feb. 2011, pp. 6–6.

[24] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging value locality in optimizing NAND flash-based SSDs," in *Proc. 9th USENIX Conf. File Storage Technol.*, Feb. 2011, pp. 7–7.

[25] B. Mao, S. Wu, and L. Duan, "Improving the SSD Performance by Exploiting Request Characteristics and Internal Parallelism," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 2, pp. 472–484, Feb. 2017.

[26] E. Jeannot, B. Knutsson and M. Bjorkmann, "Adaptive online data compression," in *Proc. 11th IEEE Int. Symp. High Perform. Distrib. Comput.*, Jul. 2002, pp. 379–388.

[27] R. Golding, P. Bosch, and C. Staelin, "Idleness is not sloth," in *Proc. Winter USENIX Conf.*, Jan. 1995, pp. 17–17.

[28] OLTP Trace from UMass trace repository, Mar. 2016. [Online]. Available: http://traces.cs.umass.edu.

[29] MSR Cambridge Traces, Mar. 2016. [Online]. Available: http://iotta.snia.org/tracetypes/3.

[30] C. Lee, D. Sim, J. Hwang, and S. Cho, "F2FS: A new file system for flash storage," in *Proc. 13th USENIX Conf. File Storage Technol.*, Feb. 2015, pp. 273–286.

[31] M. Bjørling, J. Gonzalez, and P. Bonnet, "LightNVM: The Linux open-channel SSD subsystem," presented at the *15th USENIX Conf. File Storage Technol.*, Santa Clara, CA, USA, Feb. 2017.

[32] Linux Main Memory Compression, Sep. 2016. [Online]. Available: http://linux-mm.org/compressedcaching

[33] R. Gracia-Tinedo, D. Harnik, D. Naor, D. Sotnikov, S. Toledo, and A. Zuck, "SDGen: Mimicking datasets for content generation in storage benchmarks," in *Proc. 13th USENIX Conf. File Storage Technol.*, Feb. 2015.

[34] F. Xie, M. Condict, and S. Shete, "Estimating Duplication by Content-based Sampling," in *Proc. USENIX Conf. Annu. Tech. Conf.*, Jun. 2013, pp. 181–186.

[35] D. Harnik, R. Kat, O. Margalit, D. Sotnikov, and A. Traeger, "To zip or not to zip: Effective resource usage for real-time compression," in *Proc. 11th USENIX Conf. File Storage Technol.*, Feb. 2013, pp. 229–242.

[36] Synthetic data generator for storage benchmarks, May 2016. [Online]. Available: https://github.com/iostackproject/sdgen.

[37] D. Patterson, G. Gibson, and R. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proc. Conf. Manage. Data*, Jun. 1988, pp. 109–116.

[38] S. Wu, W. Yang, B. Mao, and Y. Lin, "MC-RAIS: Multi-chunk redundant array of independent SSDs with improved performance," in *Proc. 15th Int. Conf. Algorithms Archit. Parall. Process.*, Nov. 2015.

[39] G. Wu and X. He, "Delta-ftl: Improving SSD lifetime via exploiting content locality," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, Apr. 2012, pp. 253–266.

[40] Y. Park and J. Kim, "zFTL: Power-efficient data compression support for NAND flash-based consumer electronics devices," *IEEE Trans Consumer Electron.*, vol. 57, no. 3, pp. 1148–1156, Aug. 2011.

[41] S. Lee, J. Park, K. Arvind, and J. Kim, "Improving performance and lifetime of solid-state drives using hardware-accelerated compression," *IEEE Trans. Consumer Electron.*, vol. 57, no. 4, pp. 1732–1739, Nov. 2011.

[42] Alameldeen and D. Wood, "Adaptive cache compression for high-performance processors," in *Proc. Int. Conf. Comput. Archit.*, Jun. 2004, pp. 212–223.

[43] M. Ekman and P. Stenstrom, "A robust main memory compression scheme," in *Proc. Int. Conf. Comput. Archit.*, Jun. 2005, pp. 74–85.

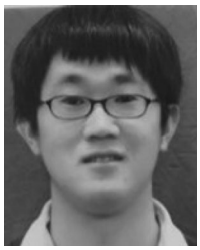
[44] J.-S. Lee, W. Hong, and S. Kim, "Design and evaluation of a selective compressed memory system," in *Proc. Int. Conf. Comput. Des.*, Oct. 1999, pp. 184–191.

[45] I. C. Tuduce and T. Gross, "Adaptive Main Memory Compression," in *Proc. USENIX Annu. Tech. Conf.*, Apr. 2005, pp. 29–29.

Bo Mao received the BSc degree in computer science and technology from Northeast University, in 2005 and the PhD degree in computer architecture from the Huazhong University of Science and Technology, in 2010. His research interests include storage system, flash-based SSDs and disk arrays, data deduplication, cloud storage and storage reliability. He is a postdoc researcher with the University of Nebraska-Lincoln between 2010 and 2013. After that he joined in the Software School of Xiamen University and becomes an associate professor since August 2015. He has more than 40 publications in international journals and conferences including the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, the *ACM Transactions on Storage*, USENIX FAST, IPDPS, ICS, Cluster, USENIX LISA, MASCOTS, ICCD, and ICPADS. His research has been supported by NSFC, Huawei, Intel, and Inspur. He is a member of the IEEE, ACM, and USENIX.

Suzhen Wu received the BSc and PhD degrees in computer science and technology and computer architecture from Huazhong University of Science and Technology, in 2005 and 2010, respectively. She is an associate professor in the Computer Science Department of Xiamen University since August 2014. Her research interests include computer architecture and storage system. She has more than 40 publications in journal and international conferences including the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, the *ACM Transactions on Storage*, USENIX FAST, USENIX LISA, IPDPS, ICS, ICCD, MASCOTS, and ICPADS. Her research has been supported by NSFC, Huawei, Intel and Inspur. She is a member of the IEEE and member of ACM.

Hong Jiang received the BSc degree in computer engineering from Huazhong University of Science and Technology, China, in 1982, the MASc. degree in computer engineering from the University of Toronto, Canada, in 1987, and the PhD degree in computer science from the Texas A&M University, USA, in 1991. He is currently chair and Wendell H. Nedderman endowed professor of Computer Science and Engineering Department, University of Texas at Arlington. Prior to joining UTA, he served as a program director in National Science Foundation (2013.1-2015.8) and he was with the University of Nebraska-Lincoln since 1991, where he was Willa Cather professor of the Computer Science and Engineering. His present research interests include computer architecture, computer storage systems and parallel I/O, high performance computing, big data computing, cloud computing, performance evaluation. He recently served as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than publications in major journals and international Conferences in these areas, including the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, *the Proceedings of the IEEE*, the *ACM Transactions on Architecture and Code Optimization*, the *Journal of Parallel and Distributed Computing*, ISCA, MICRO, USENIX ATC, FAST, EUROSYS, LISA, SIGMETRICS, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, ICPP, etc., and his research has been supported by NSF, DOD, the State of Texas and the State of Nebraska. He is a fellow of the IEEE, member of ACM, and USENIX.



Yaodong Yang received the BSc degree in computer engineering from Tianjin University, China, in 2008, the MASc degree in computer engineering from the Huazhong University of Science and Technology, China, in 2011, and the PhD degree in Computer Science and Engineering Department, University of Nebraska-Lincoln, USA, in 2016. He is an intern in Tintri Storage and familiar with their flash products. He is previously a software engineer with Microsoft, Redmond and currently with Apple in San Francisco, CA. His research interests include flash-based storage systems, VM storage migration and cloud storage.

Zaifa Xi received the BSc degree in computer engineering from Huazhong University of Science and Technology, China, in 2011 and the MASc degree in Computer Science Department, Xiamen University, China, in 2015. He is currently a software engineer with Citigroup Shanghai, China. His research interests include PCRAM, flash-based SSDs, and data reduction technology.