Multimedia Systems (2019) 25:695–708
https://doi.org/10.1007/s00530-019-00620-0

REGULAR PAPER

Lifetime-aware solid-state disk (SSD) cache management for video servers

Jungwoo Lee[1] · Hwangje Han[2] · Sungjin Lee[3] · Minseok Song[2]

Received: 21 December 2017 / Accepted: 20 May 2019 / Published online: 27 May 2019

Abstract
Solid-state disks (SSDs) are now being used as enterprise storage servers owing to their technical merits such as low power consumption, shock resistance, and excellent random read performance. To handle the large storage requirements for video data, they can be used as a cache for hard disk drives (HDDs) in video servers, but this poses several questions such as (1) which video segments can be cached on SSD, (2) how to guarantee the lifetime of SSD, and (3) how to make combined use of dynamic random-access memory (DRAM) and SSD for caching. We start by introducing the concept of caching gain to express the amount of disk bandwidth saved by caching, and go on to propose three algorithms: (1) a dynamic programming algorithm that allows for segment popularity in determining which videos should have initial segments (prefixes) stored on the SSD; (2) a throttling algorithm, which limits the number of cache replacements to guarantee the specified lifetime while maximizing caching gain using a parametric search technique; (3) an algorithm that determines the intervals between pairs of consecutive requests to be stored on the DRAM. We quantitatively explore the effect of this caching scheme through simulations, which show that: (1) prefix caching is quite effective for disk bandwidth saving, (2) our throttling algorithm guarantees the lifetime of the SSD, and (3) DRAM caching can be effectively combined with SSD caching with the aim of maximizing overall caching gain.

Keywords Video storage · Solid-state disks · Video caching

1 Introduction

It is now popular to access a variety of video applications such as digital libraries, education-on-demand, distance learning and UCC (user created content), through the wide range of Internet-connectable devices such as smartphones and tablet PCs. In particular, thanks to the growth of public Internet infrastructure, over-the-top (OTT) video streaming services such as YouTube, Netflix and Hulu have become increasingly popular [1]. For example, it has been estimated that YouTube delivers 40 million videos every day, which amounts to 200 TB of data [2]. To make things more challenging, ultra-high-definition (UHD) video services, which require even more computing resources, are becoming popular. For example, it is projected that UHD will reach 20.7% of Internet-based video-on-demand (VoD) traffic by 2020, compared with 1.6% in 2015 (see [2]).

Delivering substantial quantities of video to users requires a large server structure due to the high demand for storage and bandwidth requirements [3–5]. To support these increasing demands, a video server usually consists of hundreds of hard disk drives (HDDs), making the management of storage a major problem. In particular, although the storage capacity of HDDs continues to increase steadily, their bandwidth remains limited by seek time and rotational latency. For example, it was reported that disk access latency has improved by only a factor of four over the last two decades [6]. Disk bandwidth is thus one of the most important determinants for the performance of video servers [6]. What is worse, HDDs consume a lot of power, having a negative effect on the reliability of the servers [7].

Communicated by P. Shenoy.

✉ Minseok Song
mssong@inha.ac.kr

[1] TmaxSoft, Seoul, South Korea

[2] Department of Computer Engineering, Inha University, Incheon, South Korea

[3] Department of Information and Communication Engineering, DGIST, Daegu, South Korea

Solid-state disks (SSDs) have many technical advantages, such as non-volatility, low power consumption, shock resistance, and excellent random read performance, and thus they have begun to be introduced into enterprise storage systems [8, 9]. Reasons which have recently been suggested [6] for using SSDs in video servers include: (1) low latency in reading data, which is important in video servers; (2) good throughput for large random read operations, which is important to support large numbers of simultaneous video streams; and (3) good sequential write performance, which is essential to mitigate side effects of video replacement.

The high cost of SSDs makes it prohibitive to use them solely as the form of storage in video servers because of the amount of data in videos. For example, consider a video server that stores 10,000 2-h movies with 15 Mb/s, which amounts to 130 TB. Considering recent storage prices, we assume that a 10 TB disk is $419 [10] and a 4TB SSD is $1500 [11]. The storage cost required to configure a 130TB video server is $5447 for HDD, but $49,500 for SSD. In addition, Salkhordeh et al. [12] showed that the price gap between HDDs and SSDs was 8.5x as of 2018, and this gap is expected to persist in the near future. Therefore, to construct a cost-effective video server, it is necessary to use SSD as cache of HDD [13].

The number of program/erase (P/E) cycles in NAND flash determines its lifetime; beyond a certain number of P/E cycles, NAND flash is completely worn out and data can no longer be written [14–16]. Therefore, endurance is the main barrier to using an SSD as a cache in video servers due to frequent cache replacement operations. Although file-level least-frequently-used (LFU) caching was introduced to allow a video server to serve more clients from the SSD [6], it took no account of the popularity of different video segments and SSD lifetime.

A dynamic random-access memory (DRAM) cache can be used to reduce the number of disk requests. For example, an interval caching scheme defines the concept of interval, which is the distance between the offsets of two consecutive requests for the same video object, and caches as many of the shortest intervals as the cache can accommodate [17]. This allows the later request to be served by the data in the cache without accessing disk to reduce the number of disk I/O requests, but it was mainly developed for disk-based video server, and took no account of SSD caching issues.

We propose a new SSD caching scheme for video servers, which is composed of three algorithms: SSD allocation, DRAM caching and SSD lifetime management algorithms. Based on the concept of caching gain which expresses the benefit of SSD caching, we propose an SSD allocation algorithm to determine which videos will have their initial segments (called prefixes) cached on SSD, with the aim of minimizing the amount of disk bandwidth required, and an algorithm that limits the number of cache replacements based on the past observation to guarantee the SSD lifetime while minimizing a negative effect on caching gain. We finally present a DRAM caching algorithm that determines the intervals between pairs of consecutive requests to be stored on DRAM to make effective use of SSD bandwidth.

The rest of this paper is organized as follows. We review related work in Sect. 2 and introduce the basic idea in Sect. 3. We propose an SSD allocation algorithm in Sect. 4, a throttling algorithm in Sect. 5, and a DRAM caching algorithm in Sect. 6. We assess the effectiveness of these schemes through simulations in Sect. 7 and draw conclusions in Sect. 8.

2 Related work

2.1 SSD storage servers

As the cost of SSDs has dropped, their use in servers has been widely considered across several application areas. For example, Arteaga et al. [18] developed schemes for allocating flash memory across virtual machines to balance cache loads in cloud computing systems. Kang et al. [19] used flash memory to add capacity to a DRAM cache to maximize the throughput of transactional workloads. Meng et al. [20] developed schemes for partitioning flash memory to optimize the allocation of flash memory to virtual machines.

It is infeasible to build storage servers using SSDs only because servers may store millions of files requiring a tremendous amount of storage space [13]. For example, Niu et al. [13] have analyzed the efficacy of using SSDs for servers and found that a hybrid storage architecture based on combined use of SSDs and HDDs is a viable solution for server workloads to provide both large storage capacity and fast access speed with the minimal cost. Kim et al. [21] have proposed using an SSD as a complementary device for high-performance storage systems. They presented a capacity planning technique for the cost-effective storage hierarchy and a technique to enhance SSD performance and lifetime for random write workloads. Boukhelef et al. [22] analyzed the cost of storage devices and present two algorithms to minimize cost used for cloud construction by taking account of various factors such as multitenancy and service level agreement (SLA). Wang et al. [23] designed and implemented a new cluster architecture specifically for MapReduce environments which dynamically allocate the SSD cache space between the input data and localized data to overcome low-bandwidth problem of HDDs. Lin et al. [24] developed a new data migration scheme between HDD and SSD, by eliminating unnecessary movements of dirty data between two devices. Xu et al. [25] proposed a

new hybrid architecture specifically for scientific applications in which analysis data are typically written once onto a storage system. None of these methods, however, takes multimedia workloads into account.

To guarantee the lifetime of SSDs, many techniques have been proposed. Among them, performance throttling has recently received much attention from academia and industry because of its aggressive approach to guarantee SSD lifetime [14, 15, 26, 27]. Static throttling is the simplest form of the performance throttling for SSDs [26, 27]. By limiting the maximum write throughput of SSDs, it guarantees SSD lifetimes specified by enterprise customers, e.g., 3–5 years. Lee et al. [14, 15] further improved it by adaptively changing the degree of throttling levels according to the write intensiveness of workloads and the effective wear out of the underlying NAND devices. Liu et al. [28] developed an SSD cache scheme that identifies long-term popular data and caches them on SSD for a quite long period to improve SSD lifetime specifically for a deduplication system. However, none of these works were developed for video servers.

2.2 Use of SSDs for video servers

Ryu et al. [6] analyzed the pros and cons of using SSD in video servers. This study claimed that low-end SSDs could provide essentially constant throughput, while supporting a combination of large random read and sequential write operations, which is typical in video servers. This suggests that SSD is a good storage medium to cache video files.

Recently, several techniques have been developed to make use of SSDs for the storage of video servers. Manjunath and Xie [29] proposed a dynamic data replication scheme in which popular videos can be dynamically replicated on SSD to improve SSD caching hit ratio. Ryu et al. [30] analyzed the performance of two SSD-based multi-tiered storage systems (i.e., ZFS and flashcache) for adaptive streaming and presented design guidelines for building HTTP video streaming servers.

Jeong et al. [31] presented a data placement scheme for SSD-based video servers to provide interactive VCR media operations such as fast-forward and fast-rewind effectively. It separately encodes video frames depending on their frame type and stores them onto different blocks. This, however, requires additional file organization process for frame classification. Ryu et al. [32] developed three schemes to make SSDs suitable for video servers: SSD block size determination, SSD admission control and request scheduling schemes. These schemes were implemented on a Flashstream web server to examine the effect of SSD operations on video streaming performance.

Ho et al. [33] developed a data placement scheme for video servers, in which SSDs are used to store some initial portion of video data with the aim of reducing disk energy consumption. To allow more disks to be spun down, this scheme mainly focuses on the placement of video files onto disks, which leads to effective use of patching and batching techniques. Zhang et al. [34] developed a flash memory error tolerance technique to enable low-cost flash memory to be used for video storage with the aim of building cost-effective SSD-based video servers. It integrates on-line transcoding techniques with SSD storage management to minimize transcoding complexity at a reduced storage overhead.

Song [35] developed an SSD caching algorithm for a video server with multi-speed disks, in which the speed of each disk is adjusted with the aim of minimizing overall disk power consumption. Lee et al. [36] proposed a cost-effective SSD cache architecture for video servers, in which cold data are stored on tape. DT storage significantly reduces the storage costs by keeping latency-insensitive cold data in cost-effective tape storages, and achieves high throughput by adaptively balancing the data availability with its contention-aware replica management policy. Kim et al. [37] presented the hybrid caching scheme combining HDDs and SSDs for content delivery network services to deliver multimedia contents to end-users quickly. Ling et al. [38] developed a VoD caching system that utilized the real-time feedback of the write load of SSDs to decide which data can be moved between HDDs and SSDs.

Lee and Song [39] were the first to address video cache management by taking segment popularity into account, but this previously reported project did not take SSD lifetime into account. To the best of the authors' knowledge, this present paper reports the first approach that includes optimal caching of video segments by taking SSD lifetime into account.

3 Basic idea

Video servers require significant bandwidth to support large numbers of users requesting high bit-rate video streams, but due to disk bandwidth limitation, they typically built on an array of disks which can be composed of thousands of disks, having a negative impact on scalability and reliability of the system [40]. Therefore, disk is known to be one of the most important determinants to control the number streams served by the server. In contrast, SSDs support high bandwidth; for example, recently released NVMe-based SSD is known to support 4 GB/s [41]. Therefore, using SSD as a cache, the disk bandwidth limitation problem can be overcome, but this requires an effective caching strategy due to the limited SSD capacity.

The focus of this paper is to minimize the total stream bandwidth to the disk array, which allows the system to be configured with fewer disks, regardless of the disk specification. For this purpose, we make use of an SSD cache,

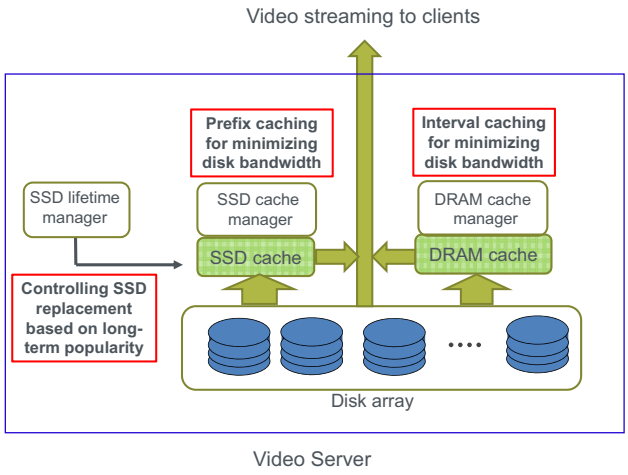


Fig. 1 An architecture of the SSD-based video storage server

which allows many requests to be served by the SSD, minimizing disk bandwidth consumption. Figure 1 shows the architecture of our SSD-based video storage server. It is mainly composed of four components: a DRAM cache manager, an SSD cache manager, an SSD lifetime manager and a collection of hard disk drives. We introduce the concept of caching gain, to express the amount of disk bandwidth saving for each video effectively. Then, the SSD cache and lifetime managers aim to maximize the overall caching gain.

The popularity of each video tends to go through three stages: a hot stage when it is first released; an intermediate warm stage; and a cold stage when the videos are hardly viewed [42]. At each stage, popularity data can be stable for quite a long period but short-term fluctuation of the popularity may be possible [43]. To reflect such popularity behavior effectively, we summarize the functionality of each component in Fig. 1 as follows:

1. SSD cache manager: It determines the number of initial segments of each video to be cached on the SSD with the aim of minimizing disk bandwidth consumption. This algorithm is invoked when video popularity changes are detected.
2. SSD lifetime manager: The lifetime of the SSD is usually set to be 3–5 years [14]. To guarantee this SSD lifetime, it monitors popularity trends over the relatively long term; based on this, it determines the threshold value of caching gain with the aim of limiting number of write operations while maximizing overall caching gain. It then delivers this value to the SSD cache manager to allow cache replacement only if a new caching gain value exceeds the threshold.
3. DRAM cache manager: It is invoked when a client requests or closes a video stream. It determines which intervals should be stored on the DRAM to reflect prompt changes of interval sizes of each video.

4 SSD caching

Let λ be the aggregate request arrival rate for all the segments in a video server. (Table 1 summarizes the notation used in this paper.) Let b_i be the bit-rate of video i, $(i = 1, \ldots, N^v)$, where N^v is the total number of videos requested by clients. Let $\lambda_{i,j}$ be the request arrival rate for the jth segment of video i. The popularity of a segment of a video is assumed to follow a Zipf distribution, giving the first segment the highest popularity [44], so that $\lambda_{i,1} > \cdots > \lambda_{i,N_i^s}$, where N_i^s is the number of segments for video i. Let $p_{i,j}$ be the probability that the jth segment of video i is requested, where $\lambda_{i,j} = p_{i,j}\lambda$. We define p_i to be the aggregate probability for video i, which is calculated as $p_i = \sum_{j=1}^{j=N_i^s} p_{i,j}$. Let u be the size of a segment, so the length of a segment of video i, which is L_i, can be expressed as $\frac{u}{b_i}$ seconds.

Let $G_i(k)$ be the caching gain, which is the reduction of disk bandwidth when the first k segments of video i are on the SSD, $(k = 0, \ldots, N_i^s)$. By Little's law, the number of concurrent requests expected during the L_i seconds for which the jth segment of video i is served can be expressed as $\lambda_{i,j}L_i$ (see [45]). Since $\lambda_{i,j} = p_{i,j}\lambda$, the disk bandwidth saved over the duration of the entire segment, which is $G_i(k)$, depends on the sum of popularities for the first k segments, $\sum_{j=1}^{k} p_{i,j}$, and expressed as follows:

$$G_i(k) = \lambda b_i L_i \sum_{j=1}^{k} p_{i,j}.$$

Let S^{ssd} be the size of the SSD in units of u MB. The storage required for the first k segments of video i is $S_i(k) = kb_iL_i$. We can now formulate the prefix selection problem ($\mathcal{PSP}$) that determines the first F_i segments $(F_i = 0, \ldots, N_i^s)$ to be cached on SSD for each video i, as follows:

$$\text{Maximize} \sum_{i=1}^{N^v} G_i(F_i)$$

$$\text{subject to} \sum_{i=1}^{N^v} S_i(F_i) \le S^{\text{ssd}},$$
$$\forall i, F_i = 0, \ldots, N_i^s.$$

The $\mathcal{PSP}$ is a variant of the multiple-choice knapsack problem. We propose a dynamic programming algorithm to find

Table 1 Notation

Symbols	Meaning
λ	Aggregate request arrival rate for all the segments in a video server
$\lambda_{i,j}$	Request arrival rate for the jth segment of video i
N_i^s	Number of segments for video i
$p_{i,j}$	Probability that the jth segment of video i is requested
p_i	Probability that video i is requested
b_i	Bit-rate of video i
N^v	Number of videos
N_i^s	Number of segments in video i
u	Size of a segment in megabytes
L_i	Length of a segment of video i in seconds
$G_i(k)$	Reduction in disk bandwidth when the initial k segments of video i are cached on the SSD
S^{ssd}	Size of the SSD in u MB
$S_i(k)$	Storage requirement for the first k segments of video i
F_i	Number of initial segments for video i stored on the SSD
$C_{i,m}^{\text{gain}}$, $C_{i,m}^{\text{prefix}}$	Parameters in the SSD allocation algorithm
T^{period}	Period of the throttling algorithm executions
N^{life}	SSD lifetime as a unit of the period length T^{period}
N^{limit}	Number of replacements for N^{life} periods
S_i^{spc}	Set of segments provided by the algorithm for the ith period
S_i^{seg}	Set of segments actually cached for the ith period
S_i^{old}	$S_i^{\text{old}} = S_i^{\text{seg}} - (S_i^{\text{seg}} \cap S_{i+1}^{\text{spc}})$
S_i^{new}	$S_i^{\text{new}} = S_{i+1}^{\text{spc}} - (S_i^{\text{seg}} \cap S_{i+1}^{\text{spc}})$
N_i^d	Number of elements in S_i^{old} which satisfy that $D_i^j > 0$
N^{monitor}	Number of past periods monitored
N^{remain}	Number of remaining periods
D_i^j	Difference in caching gain between the jth highest segment in S_i^{new} and the jth lowest segment in S_i^{old}
$D^{\text{threshold}}$	Threshold value of the caching gain difference
$N^{\text{rep}}(D^{\text{threshold}})$	Number of replacements against the value of $D^{\text{threshold}}$
N^{replace}	Number of replacements allowed for N^{remain} periods
$r(R)$	Bit-rate of request R
$I(R)$	Interval size of request R
S^{dram}	Size of the DRAM cache
B^{ssd}	SSD bandwidth limit
A^{ssd}, A^{disk}	Arrays of requests for segments stored on SSD and disk
A^{select}	Array of requests served by the SSD
N^{change}	Number of videos whose popularities are changed
T^{change}	Period of popularity change
$t_{i,j}$	Probability that the playback is terminated while segment j of video i is playing

the value of F_i for each video i. Let $C_{i,m}^{\text{gain}}$ be the maximum caching gain when m is the size of the SSD, in units of u MB, required to cache all the videos between 1 and i ($i = 1, \ldots, N^v$ and $m = 1, \ldots, S^{\text{ssd}}$). The main idea of the algorithm is to construct a table of the optimal caching gain $C_{i,m}^{\text{gain}}$ for each video i when the size of the SSD is m as shown in Fig. 2a, in which the value of $C_{N^v,S^{\text{ssd}}}^{\text{gain}}$ corresponds to the maximum caching gain. Our algorithm is composed of three steps: (1) an initialization step which fills in the first row of the table, (2) a recurrence establishment step which constructs the entire table, and (3) a backtracking step which finds the value of F_i for each video i. These three steps can be summarized as follows:

1. Initialization: the variables for all i, m, $C_{i,m}^{\text{gain}}$, are all initialized to zero.

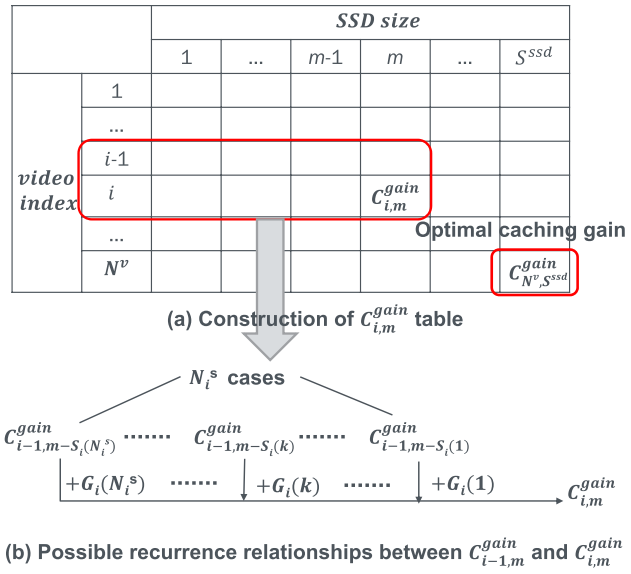


Fig. 2 Basic concept of the SSD caching algorithm

2. Recurrence establishment: $C^{\text{gain}}_{i,m}$ is calculated as follows:

$$C^{\text{gain}}_{1,m} = \max_{k\in\{k|S_1(k)\le m\}} G_i(k).$$

The maximum caching gain for video i and SSD size m, $C^{\text{gain}}_{i,m}$ can be derived from one of the caching gains for the previous video $i-1$, $C^{\text{gain}}_{i-1,m-S_i(k)}$, $(k=1,\ldots,N^s_i)$, by adding $G_i(k)$ to $C^{\text{gain}}_{i-1,m-S_i(k)}$ as shown in Fig. 2b. In addition, the following inequality should hold for all the values of m, $(m=1,\ldots,S^{\text{ssd}})$: $C^{\text{gain}}_{i,m} \ge C^{\text{gain}}_{i,m-1}$. Therefore, to find the maximum caching gain, if $C^{\text{gain}}_{i-1,m-S_i(k)} > 0$, then $C^{\text{gain}}_{i,m}$ $(i=2,\ldots,N^v)$ is updated using the following recurrence:

$$C^{\text{gain}}_{i,m} = \max(C^{\text{gain}}_{i,m-1}, \max_{k=1,\ldots,N^s_i}\{C^{\text{gain}}_{i-1,m-S_i(k)} + G_i(k)\}).$$

3. Backtracking: $C^{\text{gain}}_{N^v,S^{\text{ssd}}}$ represents the maximum caching gain. Starting from video N^v and the SSD capacity S^{ssd}, the backtracking phase finds the number of segments cached, F_i, of each video i which are cached. To do this, we introduce the variables $C^{\text{prefix}}_{i,k}$ $(i=1,\ldots,N^v$ and $m=1,\ldots,S^{\text{ssd}})$, which are the number of prefix segments of each video i that should be cached to achieve the greatest caching gain, $C^{\text{gain}}_{i,m}$, when the capacity of the SSD is k.

Based on the steps above, Fig. 3 shows the SSD prefix caching (SPC) algorithm composed of three steps: an initialization step (lines 2–13), a recurrence step (lines 14–26), and a backtracking step (lines 27–32). Figure 3 clearly shows that SPC runs in $O(N^v S^{\text{ssd}} \frac{\sum_{i=1}^{N^v} N_i^s}{N^v})$ time.

```
1:  Temporary variables: i, m and k;
2:  for i = 1 to N^v do
3:      for m = 1 to S^ssd do
4:          C^gain_{i,m} ← −∞;
5:          C^prefix_{i,m} ← 0;
6:      end for
7:  end for
8:  for m = 1 to S^ssd do
9:      if m ≥ S_1(1) then
10:         C^gain_{1,m} ← max_{k∈{k|S_1(k)≤m}} G_1(k);
11:         C^prefix_{1,m} ← arg max_{k∈{k|S_1(k)≤m}} G_1(k);
12:     end if
13: end for
14: for i = 2 to N^v do
15:     for m = 1 to S^ssd do
16:         C^gain_{i,m} ← C^gain_{i,m−1};
17:         for k = 1 to N^s_i do
18:             if m − S_i(k) > 0 then
19:                 if C^gain_{i,m} > C^gain_{i−1,m−S_i(k)} + G_i(k) then
20:                     C^gain_{i,m} ← C^gain_{i−1,m−S_i(k)} + G_i(k);
21:                     C^prefix_{1,m} ← k;
22:                 end if
23:             end if
24:         end for
25:     end for
26: end for
27: i ← N^v, m ← S^ssd;
28: while i > 0 do
29:     F_i ← C^prefix_{i,m};
30:     m ← m − S_i(F_i);
31:     i ← i − 1;
32: end while
```

Fig. 3 SSD prefix caching algorithm

The SPC algorithm can be easily modified to determine what should be stored on the SSD when the problem is modified so that videos must be cached in their entirety or not at all by removing the "for" loop in line 17 and setting the value of k in lines 18–23 to N^s_i. We call this scheme SPC with whole-video caching.

Over long periods, such as a day, the probability $p_{i,j}$ that segment j of video i is requested can be predicted by observation and analysis of the popularity of each video [43]. In a VoD service, popularity data can be stable for quite a long period [43]. For example, measurements have shown that the top 10 videos only change an average of 10–20% for most hours of the day and the top 100 videos are stable with variation between 12 and 20% in a week [43]. As a consequence, a cache configuration determined by SPC could be expected to remain valid. When changes in video popularity are detected, SPC is run to determine which videos should have their prefixes on the SSD and the appropriate replacement of segments on the SSD is then undertaken.

5 Throttling mechanism

To guarantee the lifetime of SSD, we limit the number of new segments promoted to the SSD cache from the disk, allowing less amount of data to be written to the SSD, to prevent an SSD from being quickly worn out due to excessive segment replacements. However, some of popular segments would remain in the disk, making undesirable effect on an overall cache hit ratio.

Suppose that the SSD lifetime must be guaranteed for N^{life} periods where the length of a period is T^{period} hours, so that the expected lifetime is calculated as $N^{\text{life}}T^{\text{period}}$ hours. To calculate the actual number of write operations, the write amplification factor caused by internal activities inside an SSD, such as garbage collection and wear-leveling, needs to be considered, so that the actual number of write operations can be lower than the value provided by the SSD manufacturers. Let N^{limit} be the number of allowable replacements by reflecting such effect, assigned to each segment for N^{life} periods.

At the beginning of each period, the SPC algorithm is executed to find the best cache configuration; if its configuration is different from the previous one, replacements of segments are required. Instead of replacing all the segments for the new configuration, we find the threshold value of caching gain so that segments above the threshold are only allowed to be promoted to the SSD from the disk with the aim of guaranteeing N^{limit} replacements of each segment for N^{life} periods. For this purpose, past popularity change is monitored, and the threshold value of caching gain is calculated assuming that the popularity change will continue to exhibit the same behavior.

Let S_i^{seg} be the set of segments actually cached in the ith period. Let S_i^{spc} be the set of segments from the execution of the SPC algorithm for the ith period. Note that S_i^{seg} may be different from S_i^{spc}. Let S_i^{old} and S_i^{new} be the set of segments that are deleted and added to S_{i+1}^{spc}, respectively, compared with S_i^{seg}; they can be calculated as follows:

$$S_i^{\text{old}} = S_i^{\text{seg}} - (S_i^{\text{seg}} \cap S_{i+1}^{\text{spc}}),$$
$$S_i^{\text{new}} = S_{i+1}^{\text{spc}} - (S_i^{\text{seg}} \cap S_{i+1}^{\text{spc}}).$$

To maximize the sum of caching gain, our scheme replaces a segment with the lowest caching gain in S_i^{old} by a segment with the highest caching gain in S_i^{new}, one by one. Let D_i^j be the difference in caching gain between two segments of the jth highest segment in S_i^{new} and the jth lowest segment in S_i^{old}. If $D_i^j \le 0$, then replacement is not allowed, because this decreases the sum of the caching gain. Let N_i^d be the number of elements in S_i^{old} which satisfy that $D_i^j > 0$. The number of replacements is limited by N^{limit} for N^{life} periods, so all the replacements may not be allowed although $D_i^j > 0$. To allow for this, we define $D^{\text{threshold}}$ to be the threshold value of the caching gain difference; so, if $D_i^j \le D^{\text{threshold}}$, then replacement is not permitted; otherwise, it is allowed.

It is impossible to predict future video popularity behavior exactly but video access patterns usually follow typical behaviors; for example, the popularity of each video follows lifecycles of hot, warm, and cold stages [42], and the content popularity fluctuation is known to be predictable and stable in standard VoD systems [46]. Therefore, the value of $D^{\text{threshold}}$ is searched based on the assumption that future video access pattern exhibits the same behavior of the long past N^{monitor} periods. This statistically guarantees that the number of replacements can be limited by N^{limit} for the entire N^{life} periods.

Let N^{replace} be the number of replacements per each segment permitted for the remaining period, N^{remain}, to guarantee N^{limit} replacements for N^{life} periods. The value of $D^{\text{threshold}}$ is searched so that the number of replacements can be bounded by N^{replace} for N^{remain} periods. We define a function, $N^{\text{rep}}(D^{\text{threshold}})$ to represent how the number of replacements varies with the value of $D^{\text{threshold}}$ for N^{remain} periods, which monotonically decreases with the values of $D^{\text{threshold}}$. The number of replacements for the past N^{monitor} periods is calculated, which is then multiplied by $\frac{N^{\text{remain}}}{N^{\text{monitor}}}$ to derive the value of $N^{\text{rep}}(D^{\text{threshold}})$ for N^{remain} periods.

The number of replacements may not be exactly equal to N^{replace}, so a constant value of ϵ is introduced to express such a margin. Based on this, we define the caching gain determination problem ($\mathcal{CGDP}$) that determines the value of $D^{\text{threshold}}$ as follows:

Definition 1 The caching gain determination problem ($\mathcal{CGDP}$)

The $\mathcal{CGDP}$ determines the value of $D^{\text{threshold}}$ to guarantee that $N^{\text{replace}} - \epsilon \le N^{\text{rep}}(D^{\text{threshold}}) \le N^{\text{replace}}$.

To solve the $\mathcal{CGDP}$, we use a parametric search technique [47]. Let V^{low} and V^{high} be the lowest and highest bounds for the value of $D^{\text{threshold}}$, respectively. The basic idea is to simulate a series of test algorithms by taking a median value of $D^{\text{threshold}}$, which is t^{crt}, between the specified range of caching gains, t^{low}, ($t^{\text{low}} \ge V^{\text{low}}$) and t^{high}, ($t^{\text{high}} \le V^{\text{high}}$), to examine the number of replacements, $N^{\text{rep}}(t^{\text{crt}})$ when $D^{\text{threshold}} = t^{\text{crt}}$. If $N^{\text{rep}}(t^{\text{crt}})$ is lower than $N^{\text{replace}} - \epsilon$, then the value of t^{crt} is decreased to $\frac{t^{\text{crt}}+t^{\text{low}}}{2}$, so as to increase overall caching gain; otherwise, $N^{\text{rep}}(t^{\text{crt}})$ is expected to exceed N^{replace}, which requires the algorithm to increase the value of t^{crt} to $\frac{t^{\text{crt}}+t^{\text{high}}}{2}$. This process is then repeated until the optimal value of $D^{\text{threshold}}$ can be found.

The details of the algorithm can be found in Fig. 4, which is composed of two phases: examination and

1: Temporary variables for the bound: t^{low} and t^{high};
2: Temporary variable for the current threshold: t^{crt};
3: Temporary variable: T;
4: Boolean parameter: $B \leftarrow TRUE$;
5: $t^{\text{low}} \leftarrow V^{\text{low}}$;
6: $t^{\text{high}} \leftarrow V^{\text{high}}$;
7: **while** $B = TRUE$ **do**
8: $t^{\text{crt}} \leftarrow \frac{t^{\text{low}}+t^{\text{high}}}{2}$;
9: $T \leftarrow 0$;
10: **for** $i = 1$ to N^{monitor} **do**
11: **for** $j = 1$ to N_i^d **do**
12: **if** $D_i^j \geq t^{\text{crt}}$ **then**
13: $T \leftarrow T + 1$;
14: **else**
15: Break the loop;
16: **end if**
17: **end for**
18: **end for**
19: $N^{\text{rep}}(t^{\text{crt}}) \leftarrow \frac{N^{\text{remain}}}{N^{\text{monitor}}} T$;
20: **if** $N^{\text{replace}} - \epsilon \leq N^{\text{rep}}(t^{\text{crt}}) \leq N^{\text{replace}}$ **then**
21: $B \leftarrow FALSE$;
22: **else**
23: **if** $N^{\text{rep}}(t^{\text{crt}}) < N^{\text{replace}} - \epsilon$ **then**
24: $t^{\text{high}} \leftarrow t^{\text{crt}}$;
25: **else if** $N^{\text{rep}}(t^{\text{crt}}) > N^{\text{replace}}$ **then**
26: $t^{\text{low}} \leftarrow t^{\text{crt}}$;
27: **end if**
28: **end if**
29: **end while**
30: $D^{\text{threshold}} \leftarrow t^{\text{crt}}$;

Fig. 4 An algorithm that finds $D^{\text{threshold}}$

adjustment. In the examination phase, t^{crt} is set to $\frac{t^{\text{low}}+t^{\text{high}}}{2}$, and the number of replacements, $N^{\text{rep}}(t^{\text{crt}})$ is calculated for the threshold value of t^{crt}, (lines 8–19). In the adjustment phase, the bound values (t^{low} and t^{high}) are adjusted so that the value of $N^{\text{rep}}(t^{\text{crt}})$ gets closer to N^{replace}, (lines 20–29). These two phases are repeated until $N^{\text{replace}} - \epsilon \leq N^{\text{rep}}(t^{\text{crt}}) \leq N^{\text{replace}}$, (lines 7–29). This algorithm requires $O[N^{\text{monitor}} \log(V^{\text{high}} - V^{\text{low}})]$ time complexity.

6 DRAM caching

An interval represents the time period between the playback positions of two consecutive requests (preceding and following requests) for the same video object, and the data during this interval can be stored in the DRAM cache to allow the following request to be served from the cache without accessing the disk [17].

Let $I(R)$ be the amount of DRAM size required to cache data during an interval for the following request R. If there exists no preceding request to the same video, then $I(R)$ is assumed to be set to ∞, so that no caching is allowed for the following request R. Let S^{dram} be the size of DRAM cache. Interval caching basically sorts the intervals by their space requirements and caches as many of the shortest intervals as the cache can accommodate. It therefore sorts the intervals

1: Temporary variable for storing bandwidth: $t^{\text{bw}} \leftarrow 0$;
2: Temporary variable for storing bandwidth: $t^{\text{dram}} \leftarrow 0$;
3: Array of requests for videos stored on the SSD: A^{ssd};
4: Array of requests for videos stored only on disk: A^{disk};
5: **while** $A^{\text{ssd}} \neq \phi$ **do**
6: Find a request, R which has the largest interval;
7: $A^{\text{ssd}} \leftarrow A^{\text{ssd}} - \{R\}$;
8: **if** $t^{\text{bw}} + r(R) \leq B^{\text{ssd}}$ **then**
9: $t^{\text{bw}} \leftarrow t^{\text{bw}} + r(R)$;
10: **else**
11: $A^{\text{disk}} \leftarrow A^{\text{disk}} \cup \{R\}$;
12: **end if**
13: **end while**
14: **while** $A^{\text{disk}} \neq \phi$ **do**
15: Find the request R which has the smallest interval;
16: $A^{\text{disk}} \leftarrow A^{\text{disk}} - \{R\}$;
17: **if** $t^{\text{dram}} + I(R) \leq S^{\text{dram}}$ **then**
18: $t^{\text{dram}} \leftarrow t^{\text{dram}} + I(R)$;
19: **end if**
20: **end while**

Fig. 5 DRAM caching algorithm

in order of $I(R)$ and caches the intervals with small values of $I(R)$ first subject to $\sum I(R) \leq S^{\text{dram}}$, so as to maximize the number of requests served from the cache, using a small amount of memory.

Unlike interval caching, our scheme takes SSD bandwidth into account. Let B^{ssd} be the maximum bandwidth of the SSD. Let A^{ssd} and A^{disk} be the arrays of requests for video segments stored on SSD and disk, respectively. If the SSD bandwidth is sufficient, then every request R ($R \in A^{\text{ssd}}$) can be served by the SSD; otherwise, some requests must be handled by disk. Let A^{select} be the subset of A^{ssd}, where requests $R \in A^{\text{select}}$ are served by the SSD. Extracting a subset A^{select} from A^{ssd} should maximize $\sum_{R \in A^{\text{select}}} I(R)$, which allows segments with shorter intervals to be served by the DRAM. Therefore, a DRAM caching algorithm in Fig. 5 sorts the intervals into a non-ascending order of intervals and serves the requests with the largest intervals first from the SSD, subject to its bandwidth limitation. It runs whenever a client requests or closes a segment, to determine which requests can be cached on DRAM.

7 Experimental results

7.1 Simulation setup

We performed simulations to examine the effect of our scheme with SSD capacity, bandwidth and Zipf parameters. Videos are assumed to last between 1 and 3 h, and the length of each video is selected randomly in this range. A server is assumed to serve 1000 video files with randomly chosen bit-rates between 10.4 and 20.8 Mb/s, which is typical of HD-quality videos, so the average bit-rate was set to 15.6

Mb/s. The size of each video file is assumed to be a multiple of the segment, which is set to 32 MB.

The access probability follows a Zipf distribution, such that the probability of requiring video i, p_i, is calculated as follows:

$$p_i = \frac{1}{\sum_{m=1}^{N^v} m^{\frac{1}{1-\theta}}} N^v \frac{1}{i^{1-\theta}}, \tag{1}$$

so that video 1 has the highest popularity but video N^v has the lowest. Here θ was set to 0.271 as profiled in a real VoD application [48].

Users can terminate playback of a video before completion [43, 44] so that the actual length of playback may be modeled by a Zipf distribution [44]. For example, let $t_{i,j}$ be the probability that the playback is terminated just after segment j of video i is played. Then, $\forall\, i$, $(i = 1, \ldots, N^v)$, $\sum_{j=1}^{N_i^s} t_{i,j} = 1$, and $t_{i,j}$ can be modeled using Zipf distribution with $\theta = 0.2$ [44] as follows:

$$t_{i,j} = \frac{1}{\sum_{m=1}^{N_i^s} m^{\frac{1}{1-\theta}}} N_i^s \frac{1}{j^{1-\theta}}. \tag{2}$$

The arrival rate of client requests is modeled as a modified version of the Poisson distribution, in which the independent variables of the distribution are shifted for accurate modeling of the tail part of the distribution [43]. The request rate is also assumed to vary during a day to model diurnal access pattern of VoD service [42, 43]; simulations were then carried out with alternating arrival rates of 1.25/s, 1.75/s, 2.25/s and 2.75/s every 6 h so that the average arrival rate was set to 2/s. We profiled the percentage saving in disk bandwidth over 20 h.

This arrival rate requires a significant disk array structure. To illustrate this, in the above simulation environment where the peak arrival rate is 2.75/s, we calculated the number of concurrent requests, which was found to be 4544. Assume that a disk array is composed of Seagate Baraccuda disks with the bandwidth of 133 MB/s [49]. To support video requests with an average of 15.6 Mb/s, a single disk can process 68 requests simultaneously, which means that at least 67 disks are needed to support 4544 requests.

The purpose of the simulation is to assess the total stream bandwidth supported by the SSD. This allows the system to be configured with fewer disks, regardless of the disk specification. For example, in a disk array with Seagate disks above [49], if the SSD can support 2 GB/s of stream bandwidth, SSD can be estimated to support a total of 1050 requests with 15.6 Mb/s, which corresponds to the number of requests that 15 disks can support.

The popularity of each video typically goes through three stages: a hot stage, an intermediate warm stage and a cold stage [42]. As described in Eq. (2), the video index represents the popularity; video i has higher popularity than video j if $i < j$. To reflect such a life cycle of video popularity, we demote video popularity periodically at the beginning of each period (the period length is set to T^{change}) as follows [50].

1. Indices between 1 and N^{change} are assigned to newly generated N^{change} videos to give the highest priorities to them.
2. Popularities of existing videos are demoted by adding N^{change} to their previous indices.
3. Videos with previous indices between $N^v - N^{\text{change}} + 1$ and N^v are no longer watched.

Every T^{change} hours, when video popularity changes, the SPC algorithm is executed to find the optimal SSD cache configuration. T^{change} is then set to 6 h, which requires each algorithm to be executed every 6 h.

7.2 Efficacy of SSD caching

We compared our algorithm with two popular caching schemes: least recently used (LRU) and least frequently used with dynamic aging (LFUDA) [51, 52]. The LRU scheme keeps track of the most recent access time for each segment, and evicts least recently used segments subject to SSD storage limit. The LFUDA scheme, a variant of LFU, keeps a key value for each segment, and evicts a segment with the lowest key value when the cache is full. The key value is updated when the segment is requested by adding the key value of the evicted segment to its reference count. It is noteworthy that the original LFU scheme is hardly applicable to the video cache because of its cache pollution problem: previously popular segments may be cached instead of currently popular ones, resulting in inefficient cache utilization [52]. LFUDA resolves this problem by taking an aging factor into account [52].

Figure 6 shows the percentage of disk bandwidth saved by each algorithm, for different SSD cache sizes, when DRAM caching is not used and $B^{\text{ssd}} = 2$ and 4 GB/s [41, 53]. We make the following observations:

1. There is a greater saving in bandwidth with a larger SSD, which can store more segments so as to serve more requests.
2. The effect of SSD capacity on bandwidth saving is attenuated especially when the SSD bandwidth is limited.
3. For a given SSD capacity, the saving in disk bandwidth increases with SSD bandwidth, and this effect is pronounced when the SSD is large.

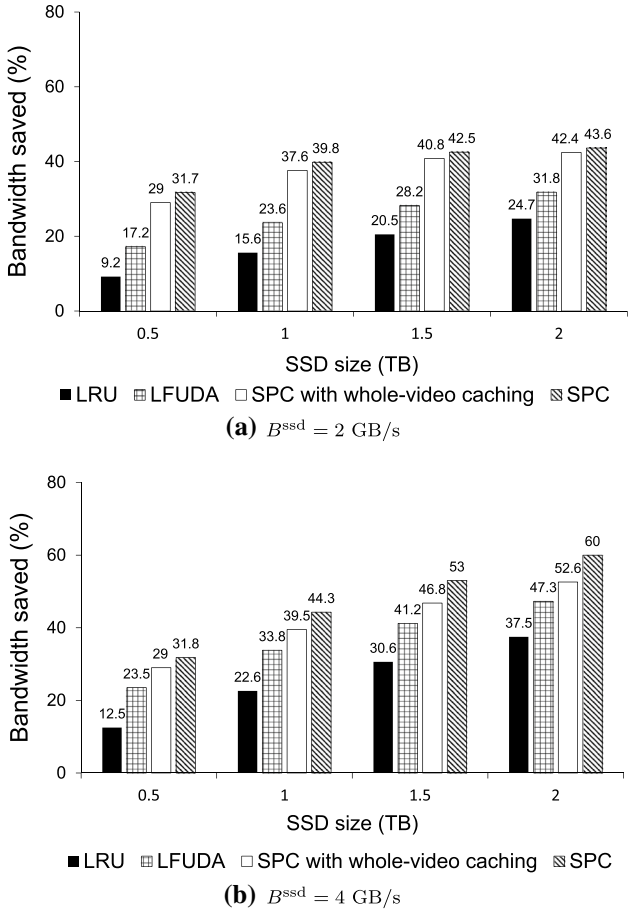


Fig. 6 Percentage of bandwidth saved against SSD size

4. SPC always shows the best performance, allowing between 31.7 and 60% of stream bandwidth to be supported by the SSD.
5. SPC shows better performance than SPC with whole-video caching, because SPC takes segment popularity into account. This gap is especially pronounced when the SSD bandwidth is 4 GB/s.
6. LRU always shows the worst performance. The difference between SPC and LRU is more pronounced when the SSD is small.
7. SPC can support between 8.3 and 14.5% more stream bandwidth on SSD than LFUDA, and the gap is especially pronounced when the SSD bandwidth is 2 GB/s.

7.3 Effect of the throttling algorithm on SSD lifetime and cache hit ratio

Increasing the value of N^{change} intensifies the degree of popularity change. We thus examined how the value of N^{change} affects (1) SSD lifetime and (2) the amount of bandwidth saved, when $S^{ssd} = 2$ TB [54] and $B^{ssd} = 4$ GB/s [41]. Here, $N^{monitor}$ is set to 120, and T^{period} is set to 6 h, so that the past 30 days are monitored to determine the threshold value of the caching gain, $D^{threshold}$.

Although the number of P/E cycles may vary depending on device characteristics, we assume that it is limited by 1000 for triple-level cell (TLC)-based SSDs and 3,000 for multiple-level cell (MLC)-based SSDs [55]. However, actual value of N^{limit} is lower than these ideal values because of additional write operations required for SSD internal operations. Thus, N^{limit} can be calculated by dividing the ideal limit value by the write amplification factor (WAF) value.

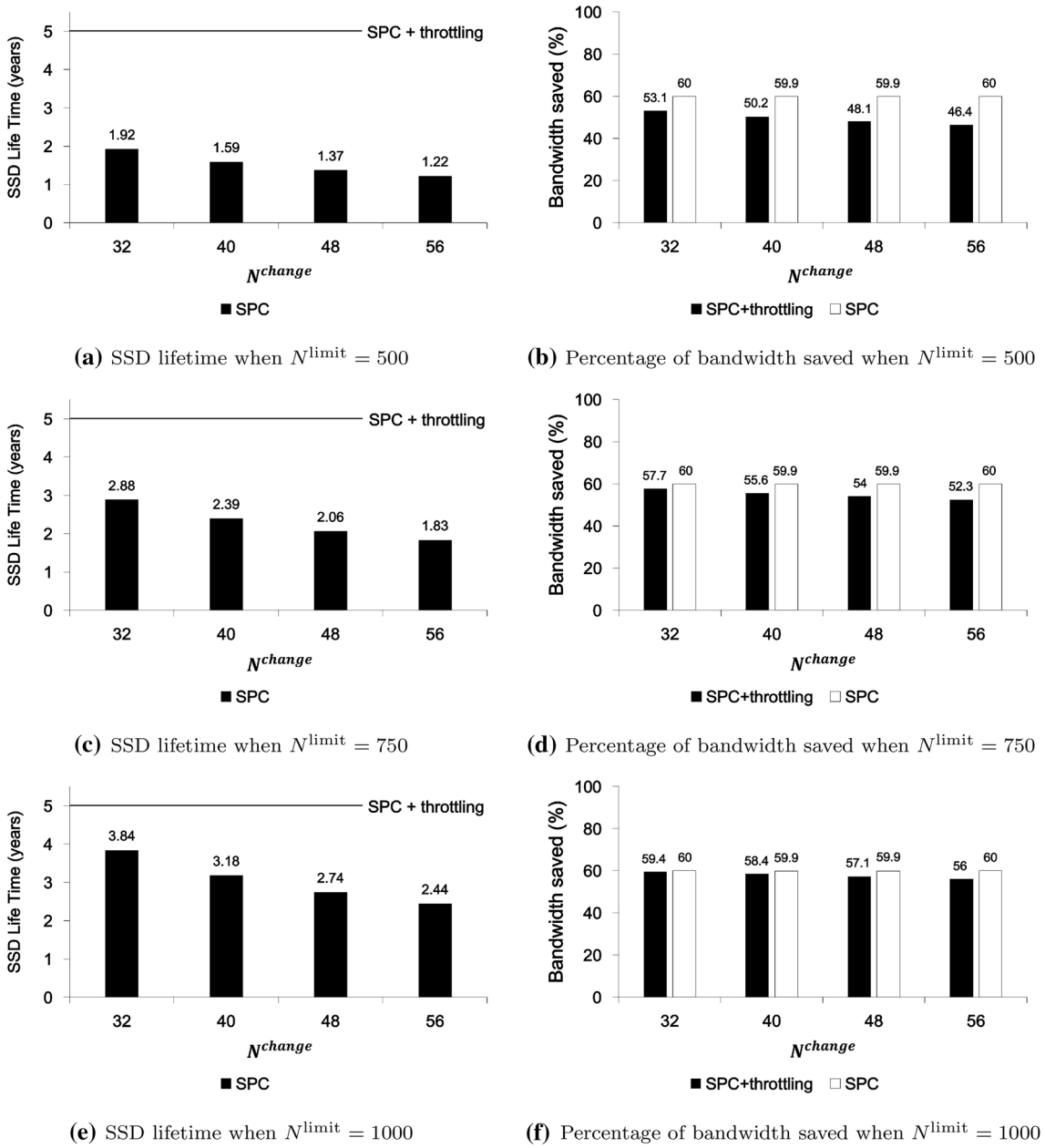
The value of WAF is highly dependent upon several internal factors and workloads [56]. For example, the write amplification factor can be as high as 4 when dealing with random short writes [56]. It is noteworthy that even when sequential writes are dominant in input workloads, WAF can be larger than 1.0. For example, in experiments with RocksDB that issue sequential writes all the times, WAF values higher than 1.0 were observed [57]. By reflecting this, the WAF values were set to 1, 1.5 and 2, so N^{limit} were set to 500, 750 and 1000 for TLC-based SSDs. Assume that the lifetime of the SSD is set to 5 years [14].

Figure 7 shows how the expected lifetime and percentage of disk bandwidth saved vary with different values of N^{limit} and N^{change}, for two schemes, SPC-only and SPC + throttling, and we make the following observations.

1. SPC + throttling always guarantees the SSD lifetime of 5 years, because it only allows segments above the threshold value of the caching gain to be promoted to SSD from disk. However, SPC-only cannot guarantee the SSD lifetime; the lifetime decreases dramatically for lower values of N^{limit} and higher values of N^{change}, because lower values of N^{limit} means shorter SSD endurance and higher values of N^{change} change the video popularity more often, requiring more replacements.
2. SPC + throttling performs worse than SPC-only in terms of disk bandwidth saved because it limits the number of segment replacements to guarantee SSD lifetime. However, SPC-only allows more segments to be replaced, which greatly shortens SSD life. For example, when $N^{limit} = 500$, SPC + throttling improves the SSD lifetime from 2.6 to 4 times compared with SPC-only at the cost of the modest reduction in bandwidth savings.

7.4 Effect of additional DRAM caching

Figure 8 shows how the percentage of disk bandwidth saved varies against SSD bandwidth when S^{ssd} is 2 TB [54]. As expected, the additional saving compared with SPC increases with DRAM size: for example, when the bandwidth is 1



(a) SSD lifetime when $N^{\text{limit}} = 500$

(b) Percentage of bandwidth saved when $N^{\text{limit}} = 500$

(c) SSD lifetime when $N^{\text{limit}} = 750$

(d) Percentage of bandwidth saved when $N^{\text{limit}} = 750$

(e) SSD lifetime when $N^{\text{limit}} = 1000$

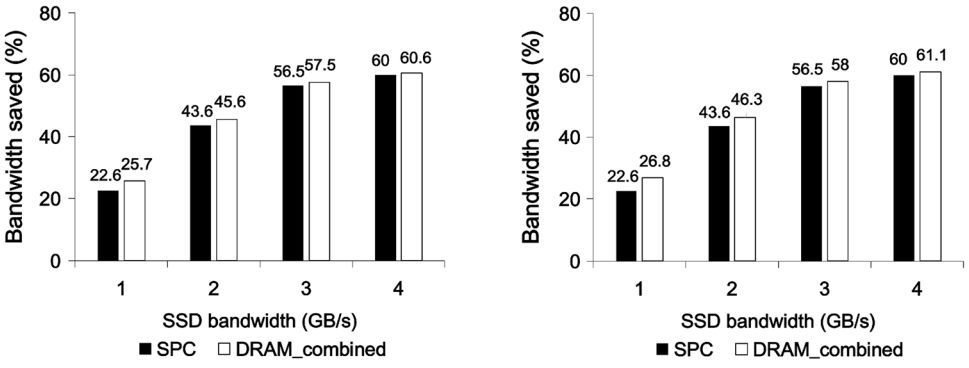
(f) Percentage of bandwidth saved when $N^{\text{limit}} = 1000$

Fig. 7 Expected SSD lifetime and percentage of bandwidth saved against N^{change}

GB/s, the improvement on SPC without DRAM caching is 3.21% when $S^{\text{dram}} = 16$ GB, but reaches 4.2% when $S^{\text{dram}} = 32$ GB. However, this gain diminishes with the SSD bandwidth: for example, when the SSD bandwidth is 4 GB/s, it ranges between 0.6 and 1.1%. These results suggest that DRAM caching can be effectively combined with SSD caching especially when the SSD bandwidth is limited.

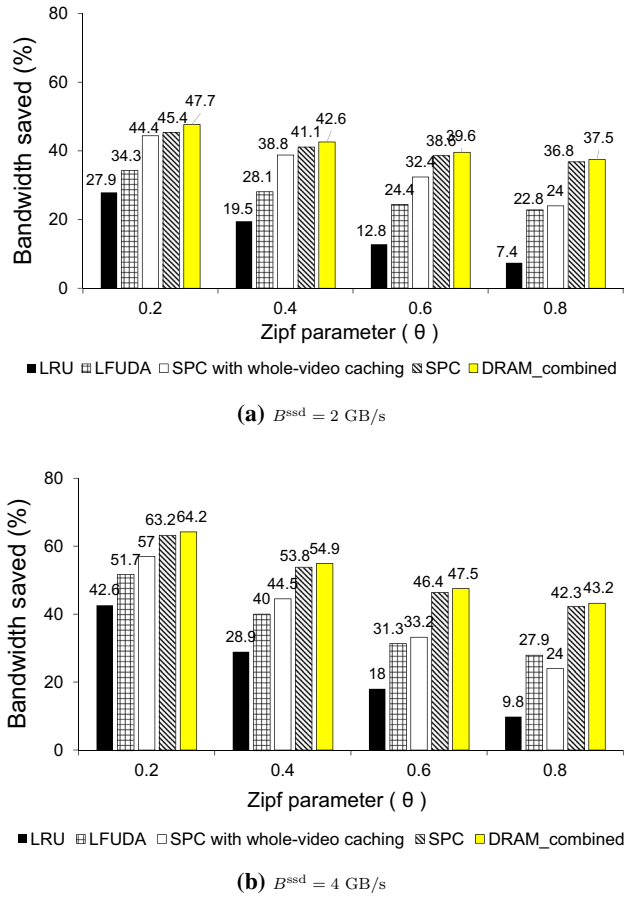
7.5 Effect of Zipf parameters

The Zipf parameters model the popularity of videos; skewness decreases with θ. Figure 9 shows how the values of the Zipf parameters affect the saving in disk bandwidth when $S^{\text{ssd}} = 2$ TB and $B^{\text{ssd}} = 2$ and 4 GB/s. Increasing the value of the Zipf parameter evens out the popularity of videos; then, SPC becomes relatively more effective compared with SPC with whole-video caching, supporting between 12.8 and 18.3% more stream bandwidth on the SSD when $\theta = 0.8$.



(a) $S^{\text{dram}} = 16$ GB **(b)** $S^{\text{dram}} = 32$ GB

Fig. 8 Percentage of disk bandwidth saved compared with SPC with DRAM caching



(a) $B^{\text{ssd}} = 2$ GB/s

(b) $B^{\text{ssd}} = 4$ GB/s

Fig. 9 Percentage of bandwidth saved against the Zipf parameters

It seems that SPC with whole-video caching allows more unpopular segments to be stored on the SSD, so that it is not used as efficiently as possible. Conversely, low values of the Zipf parameter make a few videos very popular, and this emphasizes the benefit of DRAM caching, because short intervals can be cached, making effective use of the DRAM cache.

8 Conclusions

We have presented a new SSD management scheme for video servers. We introduced the concept of caching gain to express the benefit of caching by taking video segment popularity into account. We then proposed three algorithms that make effective use of the limited SSD capacity, bandwidth, and lifetime; (1) a dynamic programming algorithm that determines which segments are cached on the SSD, with the aim of maximizing the overall caching gain; (2) a DRAM caching algorithm which serves the requests with the largest interval from the SSD first to make effective use of SSD bandwidth; (3) a throttling algorithm which limits the number of segment replacements to guarantee SSD lifetime, by deriving an appropriate caching gain value that minimizes the decrease in cache hit ratio.

Experimental results showed that the proposed scheme can process streams from 31% to 60% on the SSD which is highly dependent on the SSD size, bandwidth, and Zipf parameters. They also demonstrated that: (1) video segment popularity needs to be considered in caching decisions; (2) DRAM caching can be combined effectively with SSD caching, especially when SSD bandwidth is limited and a few videos are very popular; (3) evening out the popularity of videos makes segment caching more effective than entire video caching; (4) our throttling algorithm guarantees SSD lifetime while minimizing a decrease in overall caching gain.

Although SSDs have many technical advantages, their high cost and limited lifetime impede their use for video servers. Our results show that our schemes can make a contribution to the construction of cost-effective SSD-based video servers by exploiting the skewness pattern of video requests.

As our future works, we plan to implement our schemes in the video server test bed. We are also considering diverse disk storage architectures such as hot and cold storage farms for the extension of the proposed schemes.

Acknowledgements This work was supported by Next-Generation Information Computing Development Program through National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (2017M3C4A7080248) and the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology under Grant NRF-2015R1D1A1A01058646. This work was also supported by INHA UNIVERSITY Research Grant. Minseok Song is a corresponding author.

References

1. Gao, G., Hu, H., Wen, Y., Westphal, C.: Resource provisioning and profit maximization for transcoding in clouds: a two-timescale approach. IEEE Trans. Multimed. **19**(4), 836–848 (2017)
2. YouTube bandwdith analysis article (2006). http://blog.forret.com/2006/05/youtube-bandwidth-terabytes-per-day/. Accessed 24 May 2019
3. Zhang, G., Liu, W., Hei, X., Cheng, W.: Unreeling xunlei kankan: understanding hybrid CDN-P2P video-on-demand streaming. IEEE Trans. Multimed. **17**(2), 229–242 (2015)
4. Li, Z., Wu, Q., Salamatian, K., Xie, G.: Video delivery performance of a large-scale VoD systems and its implication on content delivery. IEEE Trans. Multimed. **17**(6), 880–892 (2015)
5. Hu, Y., Niu, D., Li, Z.: A geometric approach to server selection for interactive video streaming. IEEE Trans. Multimed. **18**(5), 840–851 (2016)
6. Ryu, M., Kim, H., Ramachandran, U.: Impact of flash memory on video-on-demand storage: analysis and tradeoffs. In: Proceedings of the ACM Multimedia Systems, pp. 175–186 (2011)
7. Khatib, M., Bandic, Z.: PCAP: performance-aware power capping for the disk drive in the cloud. In: Proceedings of the USENIX File and Storage Technologies, pp. 227–240 (2016)
8. Ryu, Y.: A flash translation layer for NAND flash-based multimedia storage devices. IEEE Trans. Multimed. **13**(3), 563–572 (2011)
9. Chen, X., Chen, W., Member, Z. Lu, Long, P., Yang, S., Wang, Z.: A duplication-aware SSD-based cache architecture for primary storage in virtualization environment. IEEE Syst. J. **11**(4), 2578–2589 (2017)
10. 10TB Western Digital HDD specification. https://www.amazon.com/Red-10TB-Hard-Disk-Drive/dp/B0719498XY/. Accessed 24 May 2019
11. 4TB Samsung SSD specification. https://www.amazon.com/Samsung-2-5-Inch-Internal-MZ-75E4T0B-AM/dp/B01G844OOO/. Accessed 24 May 2019
12. Salkhordeh, R., Ebrahimi, S., Asadi, H.: ReCA: an efficient reconfigurable cache architecture for storage systems with online workload characterization. IEEE Trans. Parallel Distrib. Syst. **29**(7), 1605–1620 (2018)
13. Niu, J., Xu, J., Xie, L.: Hybrid storage systems: a survey of architectures and algorithms. IEEE Access **6**(1), 13385–14406 (2018)
14. Lee, S., Kim, T., Kim, K., Kim, J.: Lifetime management of flash-based SSDs using recovery-aware dynamic throttling. In: Proceedings of the USENIX conference on File and Storage Technologies, pp. 537–540 (2012)
15. Lee, S., Kim, J.: Effective lifetime-aware dynamic throttling for NAND flash-based SSDs. IEEE Trans. Comput. **65**(4), 1331–1334 (2016)
16. Grupp, L., Davis, J., Swanson, S.: The bleak future of NAND flash memory. In: Proceedings of the USENIX Conference on File and Storage Technologies, pp. 2–2 (2012)
17. Dan, A., Sitaram, D.: Multimedia caching strategies for heterogeneous application and server environments. Multimed. Tools Appl. **4**(1), 279–312 (1997)
18. Arteaga, D., Cabrera, J., Xu, J., Sundararaman, S., Machines, P., Zhao, M.: Cloudcache: on-demand flash cache management for cloud computing. In: Proceedings of the USENIX Conference on File and Storage Technologies, pp. 355–369 (2016)
19. Kang, W., Lee, S., Moon, B.: Flash-based extended cache for higher throughput and faster recovery. In: Proceedings of the ACM VLDB Conference, pp. 1615–1626 (2012)
20. Meng, F., Zhou, L., Ma, X., Uttamchandani, S., Liu, D.: Vcacheshare: automated server flash cache space management in a virtualization environment. In: Proceedings of the USENIX Annual Technical Conference, pp. 133–144 (2014)
21. Kim, Y., Gupta, A., Urgaonkar, B., Berman, P., Sivasubramaniam, A.: Hybridstore: a cost-efficient, high-performance storage system combining SSDs and HDDs. In: Proceedings of the IEEE MASCOTS, pp. 227–236 (2011)
22. Boukhelef, D., Boukhalfa, K., Boukhobza, J., Ouarnoughi, H., Lemarchand, L.: COPS: cost based object placement strategies on hybrid storage system for DBaaS Cloud. In: Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 659–664 (2017)
23. Wang, B., Jiang, J., Wu, Y., Yang, G., Li, K.: Accelerating MapReduce on commodity clusters: an SSD-empowered approach. IEEE Trans.Big Data **4**(3), 396–407 (2018)
24. Lin, M., Chen, R., Xiong, J., Li, X., Yao, Z.: Efficient sequential data migration scheme considering dying data for HDD/SSD hybrid storage systems. IEEE Access **5**(1), 23366–23373 (2017)
25. Xu, C., Wang, W., Zhou, D., Xie, T.: An SSD-HDD integrated storage architecture for write-once-read-once applications on clusters. In: Proceedings of the IEEE International Conference on Cluster Computing, pp. 74–77 (2015)
26. Yoon, H., Woo, Y., Lee, D., Moon, Y., Kwon, H.: Semiconductor storage device and method of throttling performance of the same. US patent, US20120047317 (2012)
27. Tressler, G., Walls, A.: Enabling throttling on average write throughput for solid state storage devices. US patent, US20130086302 (2013)
28. Liu, J., Chai, Y., Qin, X., Liu, Y.: Endurable SSD-based read cache for improving the performance of selective restore from deduplication systems. J. Comput. Sci. Technol. **33**(2), 58–78 (2018)
29. Manjunath, R., Xie, T.: Dynamic data replication on flash SSD assisted video-on-demand servers. In: Proceedings of the International Conference on Computing, Networking and Communications, pp. 502–506 (2012)
30. Ryu, M., Kim, H., Ramachandran, U.: Why are state-of-the-art flash-based multi-tiered storage systems performing poorly for http video streaming? In: Proceedings of the ACM NOSSDAV Conference, pp. 3–8 (2012)
31. Jeong, Y., Park, Y., Seo, K., Yoo, J., Park, K.: An SSD-based storage system for an interactive media server using video frame grouping. ETRI J. **35**(1), 69–79 (2013)
32. Ryu, M. Ramachandran, U.: Flashstream: a multi-tiered storage architecture for adaptive HTTP streaming. In Proceedings of the ACM Multimedia Conference, pp. 313–322 (2013)
33. Ho, C., Chen, H., Chang, Y., Chang, Y.: Energy-aware data placement strategy for SSD-assisted streaming video servers. In: Proceedings of the IEEE Non-Volatile Memory Systems and Applications Symposium (2014)
34. Zhang, X., Xiong, D., Zhao, K., Chen, C., Zhang, T.: Realizing low-cost flash memory based video caching in content delivery systems. IEEE Trans. Circuits Syst. Video Technol. **28**(4), 984–996 (2018)

35. Song, M.: Minimizing power consumption in video servers by the combined use of solid-state disks and multi-speed disks. IEEE Access **6**(1), 25737–25746 (2018)
36. Lee, J., Ahn, J., Park, C., Kim, J.: DTStorage: dynamic tape-based storage for cost-effective and highly-available streaming service. In: Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 376–387 (2016)
37. Kim, T., Kim, E.: Hybrid storage-based caching strategy for content delivery network services. Multimed. Tools Appl. **74**(5), 1697–1709 (2015)
38. Ling, Q., Xu, L., Yan, J., Zhang, Y., Li, F.: A feedback-based adaptive data migration method for hybrid storage VOD caching systems. Multimed. Tools Appl. **75**(1), 165–180 (2016)
39. Lee, J., Song, M.: Cache management for video servers by the combined use of DRAM and SSD. In: Proceedings of the IEEE International Symposium on Multimedia, pp. 537–540 (2016)
40. He, S., Wang, Y., Sun, X.: Improving performance of parallel I/O systems through selective and layout-aware SSD cache. IEEE Trans. Parallel Distrib. Syst. **27**(10), 2940–2952 (2016)
41. Marvell NVMe SSD specification. https://www.marvell.com/storage/ssd/. Accessed 24 May 2019
42. Zhou, Y., Chen, L., Yang, C., Chiu, D.: Video popularity dynamics and its implications for replication. IEEE Trans. Multimed. **17**(8), 1273–1285 (2015)
43. Yu, H., Zheng, D., Zhao, B. Y. , Zheng, W.: Understanding user behavior in large-scale video-on-demand systems. In Proceedings of the ACM Eurosys, pp. 333–344 (2006)
44. Lim, S., Ko, Y., Jung, G., Kim, J., Suei, P.L., Yeh, M.Y., Kuo, T.W.: Inter-chunk popularity-based edge-first caching in content-centric networking. IEEE Commun. Lett. **18**(8), 1331–1334 (2014)
45. Kleinrock, L.: Theory, Volume 1, Queueing Systems. Wiley-Interscience, New York (1975)
46. Cha, M., Kwak, H., Rodriguez, P., Ahn, Y., Moon, S.: Analyzing the video popularity characteristics of large-scale user generated content systems. IEEE/ACM Trans. Netw. **17**(5), 1357–1370 (2009)
47. Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms. J. ACM **30**(4), 852–865 (1983)
48. Dan, A., Sitaram, D., Shahabuddin, P.: Dynamic batching policies for an on-demand video server. ACM/Springer Multimed. Syst. J. **4**(3), 112–121 (1996)
49. Seagate HDD benchmark results. https://hdd.userbenchmark.com/Seagate-Barracuda-Pro-10TB-2016/Rating/3901/. Accessed 24 May 2019
50. Guo, Y., Ge, Z., Dan, A., Urgaonkar, B., Shenoy, P., Towsley, D.: Dynamic cache reconiguration strategies for a cluster-based streaming proxy. In: Proceedings of the International Workshop on Web Content Caching and Distribution, pp. 139–157 (2004)
51. Wu, J., Li, B.: Keep cache replacement simple in peer-assisted VoD systems. In: Proceedings of the IEEE International Conference on Computer Communications, pp. 2591–2595 (2009)
52. Li, S., Xuy, J., Schaarz, M., Lix, W.: Popularity-driven content caching. In: Proceedings of the IEEE International Conference on Computer Communications (2016)
53. Marvell PCIe 2.0 SATA controllers specification. https://www.marvell.com/storage/system-solutions/sata-controllers/pcie-2-sata-controllers/. Accessed 24 May 2019
54. Samsung SSD 960 PRO NVMe M.2 specification. https://www.samsung.com/us/computing/memory-storage/solid-state-drives/ssd-960-pro-m-2-2tb-mz-v6p2t0bw/. Accessed 24 May 2019
55. Samsung SSD 840 benchmark test results. https://www.anandtech.com/show/6459/samsung-ssd-840-testing-the-endurance-of-tlc-nand/. Accessed 24 May 2019
56. Hu, X., Eleftheriou, E., Haas, R., Iliadis, I., Pletka, R.: Write amplification analysis in flash-based solid state drives. In Proceedings of the ACM International Systems and Storage Conference, pp. 74–77 (2009)
57. Kim, T., Hahn, S., Lee, S., Hwang, J., Lee, J., Kim, J.: PCStream: automatic stream allocation using program contexts. In: Proceedings of the USENIX HotStorage Workshop (2018)