Information Systems 58 (2016) 24–42

Contents lists available at ScienceDirect

Information Systems

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

A fully persistent and consistent read/write cache using flash-based general SSDs for desktop workloads

Sung Hoon Baek [a], Ki-Woong Park [b,*]

[a] Department of Computer System Engineering, Jungwon University, Republic of Korea
[b] Department of Computer Hacking and Information Security, Daejeon University, Republic of Korea

ARTICLE INFO

Article history:
Received 6 April 2015
Received in revised form
10 December 2015
Accepted 4 February 2016
Recommended by: B. Kemme
Available online 12 February 2016

Keywords:
Secondary storage

ABSTRACT

The flash-based SSD is used as a tiered cache between RAM and HDD. Conventional schemes do not utilize the nonvolatile feature of SSD and cannot cache write requests. Writes are a significant, or often dominant, fraction of storage workloads. To cache write requests, the SSD cache should persistently and consistently manage its data and metadata, and guarantee no data loss even after a crash. Persistent cache management may require frequent metadata changes and causes high overhead. Some researchers insist that a nonvolatile persistent cache requires new additional primitives that are not supported by general SSDs in the market. We proposed a fully persistent read/write cache, which improves both read and write performance, does not require any special primitive, has a low overhead, guarantees the integrity of the cache metadata and the consistency of the cached data, even during a crash or power failure, and is able to recover the flash cache quickly without any data loss. We implemented the persistent read/write cache as a block device driver in Linux. Our scheme aims at virtual desktop infra servers. So the evaluation was performed with massive, real desktop traces of five users for ten days. The evaluation shows that our scheme outperforms an LRU version of SSD cache by 50% and the read-only version of our scheme by 37%, on average, for all experiments. This paper describes most of the parts of our scheme in detail. Detailed pseudo-codes are included in the Appendix.

1. Introduction

The advent of flash-based solid-state drive (SSD) has spurred a proliferation of studies on new storage architectures. Flash memory is widely used in mobile phones and embedded systems because it is smaller and more resistant to shock than mechanical storage devices such as hard disk drives (HDD). In addition, SSD that utilizes tens or hundreds of independent flash chips is superior to HDD in terms of bandwidth and response time. Thus, SSD has been replacing HDD in devices ranging from desktop computers to enterprise servers.

The capacity per price of SSD is greater than that of RAM and lower than that of HDD. In the aspect of performance, SSD is slower than RAM but faster than HDD. Especially, HDD exhibits much longer latency for non-sequential requests than SSD due to its mechanical components, while SSD provides a short constant response time regardless of request patterns.

In terms of the performance, the capacity, and the price, SSD is in between RAM and HDD. Hence, various tiered architectures where SSD is used as a second level cache between RAM and HDD have been studied [1,2].

Many researchers have studied the nonvolatile memory (NVRAM) such as phase-change memory and magnetoresistive

* Corresponding author.
Tel.: +82 10 9165 1624; fax: +82 42 280 2404.
E-mail addresses: shbaek@jwu.ac.kr (S.H. Baek), woongbak@dju.kr (K.-W. Park).

http://dx.doi.org/10.1016/j.is.2016.02.002
0306-4379/

RAM as an intermediate tier between RAM and HDD [3–6]. The byte-addressing feature of NVRAM makes persistent cache management easy but SSD is not byte-addressable. SSD, however, is superior to NVRAM in terms of performance per price and has been used as a cache below RAM in commercial storage systems [7–9].

Approaches to use the raw flash memory as a tiered cache have been introduced [10,11]. Consequences of the drive to lower the price and increase the capacity of flash memory, have been that the reliability of flash cells continues to diminish and a variety of error patterns has arisen. The flash architecture turned into 3D NAND from planar NAND [12], and its error types greatly changed. The manufacturers have investigated various solutions for these errors and keep them secret for market advantage. It is difficult for a third party company to manage the low-level flash memory. However, SSD provides us an error-free interface; thus, it can easily be managed as a tiered cache.

Many studies on databases that use flash SSDs as a mid-tier cache have been carried out [13–15]. Canim et al. introduced a read cache replacement algorithm that estimates I/O costs between SSD and HDD for each region to determine which region is best to be promoted to SSD [14]. Do et al. showed that a write cache can give a significant gain to database systems [15]. However, a cache metadata management scheme that has no data staleness is needed for database systems to adopt a write cache.

1.1. Persistency and consistency

The conventional second-level cache schemes for flash-based SSDs are variations of traditional RAM-based cache policies. Hence, they do not utilize the nonvolatile feature of SSD. Cached data in SSD cannot be used at the next restart because they store metadata in RAM for fast processing. Data stored in SSD does not volatilize but the data cannot be used after a restart because the metadata in RAM is lost and we cannot know which sector the cached data is related to [1,2,8,16–18].

The capacity of SSD as the second level cache is much bigger than the main memory by two or three orders of magnitude. The conventional second cache schemes ignore all data stored in SSD at a restart and require too long time until the SSD cache is filled with data at the rate of application I/O. This process takes several hours or even days to fill the SSD.

A warm-start scheme was proposed [19] that would shorten the restart time of the storage-class second level cache. It makes a log of warmup data to fill the second level cache at the next boot time. However, this scheme still requires at least several hours to fill the storage-class cache.

A cache is persistent if its cache data is immediately reusable after a power failure. The proposed system is persistent, does not need a warming process, and has a low overhead. The cache metadata and data are stored in a nonvolatile device such as SSD and is consistent without data loss even after a crash or a power failure.

1.2. Write cache and overhead

“Writes are significant, or often dominant, fraction of storage workloads” [20–23], thus, a write cache can greatly improve the write performance. The traditional caches do not safely manage their cache metadata during a crash, so they cache only read requests and cannot retain dirty data, even though the cache device is nonvolatile.

Most tiered cache technologies use a write-through policy that prohibits dirty data in SSD. In other words, a write request invalidates a block that is cached in SSD and is directly delivered to HDD, thus all the newest data are stored in HDD. This means that conventional technologies utilize SSD as a read-only cache that cannot improve the write performance.

Even though SSD is a nonvolatile device, it is very difficult to apply the write-back policy to the tiered cache. To persistently maintain dirty data in SSD, cache metadata must be stored in a nonvolatile device and consistently updated whenever a block is evicted or cached. In addition, cached data and cache metadata must not be lost and be consistent even at a crash. A persistent cache that employs dirty data may require a high real-time overhead for consistent metadata management.

A persistent read/write cache improves both read and write performance but it must also guarantee the integrity of the cache metadata and the consistency of the cached data even during a crash or a power failure.

Saxena et al. [24] proposed a durable (same as persistent) write cache with a small overhead. They insisted that storage provide new primitives (write-dirty, write-clean, evict, clean, exists) to consistently manage a write cache with a low overhead. However, general SSDs, though they support only the basic primitives, read and write; have the benefits of fast development time, low cost, and popularity.

An approach using general SSDs was introduced [25]. It can cache write data and does not return stale data but may restart with an empty cache if a crash occurred while updating its cache metadata.

The system proposed herein is a fully persistent read/write cache with low overhead, it never return stale data, and it utilizes general SSDs that are now available in the retail market.

1.3. Lossless recovery and no-write-back

Koller et al. [20] introduced a write-back policy for an SSD cache, which is journaled write-back that makes a block-level journal that aggregates multiple write requests with a header block and a commit block, while limiting the amount of data loss. The journaling cache quickly recovers by replaying the last completed transaction to its home location. However, it cannot provide a recovery point objective (RPO) of zero (i.e., no data loss). It loses the last journaled data if a crash occurs in the middle of journaling.

Holland et al. [26] investigated flash write policies, which are write-through, asynchronous write-through, and periodic write-back. The write-through policy does not permit dirty data on flash. The other write policies may return stale data after a crash or a power failure. Some SSD caches use the write-back policy for the write performance by sacrificing a risk of data loss [8,23,27]. However, our solution never loses dirty data for highly available systems.