Leveraging Value Locality in Optimizing NAND Flash-based SSDs

Aayush Gupta, Raghav Pisolkar, Bhuvan Urgaonkar, and Anand Sivasubramaniam
{axg354,rvp116,bhuvan,anand}@cse.psu.edu
Department of Computer Science and Engineering
The Pennsylvania State University, University Park 16802, PA

Abstract: NAND flash-based solid-state drives (SSDs) are increasingly being deployed in storage systems at different levels such as buffer-caches and even secondary storage. However, the poor reliability and performance offered by these SSDs for write-intensive workloads continues to be their key shortcoming. Several solutions based on traditionally popular notions of temporal and spatial locality help reduce write traffic for SSDs. However, another form of locality - *value locality* - has remained completely unexplored. Value locality implies that certain data items (i.e., "values," not just logical addresses) are likely to be accessed preferentially. Given evidence for the presence of significant value locality in real-world workloads, we design CA-SSD which employs content-addressable storage (CAS) to exploit such locality. Our CA-SSD design employs enhancements primarily in the flash translation layer (FTL) with minimal additional hardware, suggesting its feasibility. Using three real-world workloads with content information, we devise statistical characterizations of two aspects of value locality - value popularity and temporal value locality - that form the foundation of CA-SSD. We observe that CA-SSD is able to reduce average response times by about 59-84% compared to traditional SSDs. Even for workloads with little or no value locality, CA-SSD continues to offer comparable performance to a traditional SSD. Our findings advocate adoption of CAS in SSDs, paving the way for a new generation of these devices.

1 Introduction and Motivation

NAND flash-based SSDs offer several advantages over magnetic hard disks: lower access latencies, lower power consumption, lack of noise, and higher robustness to vibrations and temperature. Several researchers have explored the performance benefits of employing these SSDs, either as complete replacements for magnetic drives or in supplementary roles (e.g., caches) [23]. Whereas a number of other non-volatile memory technologies - phase-change, ferroelectric, and magnetic RAM - exist at different levels of maturity and offer similar benefits, cost/feasibility projections suggest that NAND flash (simply flash, henceforth) is likely to be at the forefront of these significant changes in storage for the next decade [17]. Another trend from EMC suggests that SSD prices will continue to fall to the extent of becoming cheaper than 15K RPM HDDs by 2017 [7]. Thus, exploring ways to further improve flash technology and its use in designing better storage systems will continue to be worthwhile pursuits in the coming years.

Flash is a unique memory technology due to the sensitivity of its reliability and performance to write traffic. A flash page (the granularity of reads/writes) must be erased before it may be written. Erases occur at the granularity of blocks which contain multiple pages. Furthermore, blocks become unreliable after 5K-100K erase operations [38, 39, 37]. This erase-before-write property of flash necessitates out-of-place updates to prevent the relatively high latency of erases from affecting the performance of writes. These out-of-place updates create invalid pages that contain older versions of data requiring garbage collection. This further exacerbates the reliability/performance concerns by introducing additional write operations. Techniques that reduce the number of writes to SSDs are, therefore, desirable and have received a lot of attention. Existing approaches for write reduction have relied on exploiting the presence of (i) temporal locality (e.g., buffering writes within file system/SSD/other media to eliminate duplicate writes to flash [24, 46, 45]), and/or (ii) spatial locality (e.g., coalescing multiple sub-page writes into fewer page writes [30]) within workloads. However, there is yet another dimension of locality - *value locality* - that has remained unexplored for flash SSDs. The presence of value locality in a workload means that it preferentially accesses certain content (i.e., values) over others. This property facilitates data de-duplication (storing only one copy of each unique value), which is especially attractive for SSDs as it nat-

urally offers the write reduction that these devices can benefit from: a SSD employing such data de-duplication need not do an additional write of a value that it has already stored. This benefit applies even if the two writes belong to entirely different logical addresses and even in the absence of any temporal/spatial correlation between these two writes. Data de-duplication can also reduce read traffic, with additional performance benefits.

Content addressable storage (CAS) is a popular de-duplication technique which operates on data by dividing it into non-intersecting *chunks*, and employing a cryptographic hash to represent each chunk. By storing only unique hashes (and their corresponding data chunks), duplicate chunks in data are removed. Hashing can result in collisions where different data blocks can be mapped to the same value. However, it has been shown that such collisions are practically unlikely, with probabilities in the range $10^{-9} - 10^{-17}$ [40, 42] for MD5 and SHA-1. Additionally, techniques to further reduce this probability to as low as 10^{-46} have been shown to be feasible [40, 43]. Thus, consistent with most CAS research [12, 42, 35], we also assume hash functions to be collision-resistant. CAS has been extensively used in archival and backup systems [42, 43, 14], but its benefits specific to SSDs have not been explored. Whereas SSDs could benefit from existing host-level (e.g., file system [47]) implementations of CAS, thereby reducing I/O traffic, there is significant motivation to realize this functionality within the device itself. It allows incorporation of value locality without requiring any modifications to the upper layers (filesystem, block layer etc.), thus allowing quick adoption in existing systems. Several SSD optimizations that rely upon information about flash data layout are better implemented within the SSD. For example, garbage collection efficiency can be improved by using data placement policies which reduce overheads of copying valid pages. Also, scalability of a CAS-based scheme crucially depends on its ability to carry out fast calculations/look-ups of hashes. This can be achieved by using dedicated hardware such as that increasingly available in SSDs (e.g., those with Full Disk Encryption capabilities [5, 44, 41]), relieving the host of these computational overheads.

Key Choices and Challenges: A number of interesting design choices and challenges arise when designing a SSD that employs CAS for its internal data management. First, in order to maintain compatibility with existing storage software, we choose that our SSD continue to expose its existing block interface. Modifications to the SSD interface such as nameless writes [11] can potentially benefit CA-SSD but require changes to the upper layers. Second, employing CAS necessitates several enhancements to the data structures maintained by our SSD's flash translation layer (FTL). This increased "meta-data" puts additional pressure on the scarce on-SSD RAM and must be managed carefully. Third, data de-duplication renders ineffective existing mechanisms employed by the FTL to recover its meta-data after power failures. Existing FTLs store information about the logical address (LPN) stored on a flash page in a special region called the out-of-band area (OOB) within the page itself. Due to de-duplication with CAS, a given page may correspond to multiple LPNs (different LPNs may contain the same content) , and thus, its OOB area cannot be used as before. Fourth, with CAS the notion of when a page becomes invalid changes - a page should now be invalidated only when all the LPNs having that content have written a "different content" - implying a re-consideration of the design of the garbage collector. Finally, whereas we design our SSD to exploit value locality whenever present, we would like it not to exhibit degraded performance or reliability than a state-of-the-art SSD in the absence of such locality.

Research Contributions: We make the following contributions in this paper.

- We propose CA-SSD, a flash solid-state drive that employs CAS for internal data management and addresses all the concerns outlined above. We demonstrate how CA-SSD functionality can be achieved mostly by modifying the FTL and with minimal support in the form of additional hardware compared to traditional SSDs. This additional hardware is similar to that already present in many state-of-the-art SSDs.
- We identify and characterize salient aspects of value locality- value popularity and temporal value locality and design CA-SSD algorithms to exploit them.
- Using three real-world workloads with content information, we evaluate the efficacy of CA-SSD by simulations. We observe that CA-SSD is able to reduce the average response times by about 59–84% for these workloads. Additionally, from 10 real-world traces, we synthesize workloads with different degrees of value locality. We find that CA-SSD consistently outperforms traditional SSD with even small degrees of locality and offers comparable performance when there is little or no value locality.

The rest of this paper is organized as follows. In Section 2 we provide an overview of the design of our CA-SSD comparing it to traditional SSDs. We discuss key aspects of value locality that affect CA-SSD design in Section 3. We design CA-SSD using insights gained in Section 4 and evaluate it in Section 5. Finally, we present related work in Section 6 and conclude in Section 7.

2 Overview of Our CA-SSD

Type	Data Unit			Access Time			Lifetime
	Page (Bytes)		Block	Read	Write	Erase	Write/Erase
	Data	OOB	(Bytes)	(us)	(us)	(ms)	(cycles)
SLC1	2048	64	128K+4K	25	200	1.5	100K
SLC2	4096	128	256K+8K	25	500	1.5	100K
MLC	4096	224	512K+28K	60	800	2.5	5K

Table 1: SLC & MLC NAND Flash characteristics [38, 39, 37]. SLC1/SLC2 represent SLC SSDs with different page sizes. Read/write latencies are at the granularity of pages while erase latencies are for blocks.

In this section, we describe how a flash-based SSD works and provide an overview of the changes to implement our CA-SSD.

2.1 Flash Solid-State Drives: A Primer

Figure 1(a) presents the key components of a traditional NAND flash-based SSD. In addition to the read and write operations which are performed at the granularity of a *page*, flash also provides an *erase* operation which is performed at the granularity of a *block* (composed of 64-128 pages). The coarser spatial granularity of erases makes them significantly slower than reads/writes. Furthermore, there is an asymmetry in read and write latencies, with writes being slower than reads. Blocks are further arranged in *planes* which can allow simultaneous operations through multi-plane commands thus improving performance [10]. In this paper, we only consider a single plane and our ideas and results apply readily to multiple planes. A page must first be erased before it can be written. The erase-before-write property of flash memory necessitates out-of-place updates to prevent the relatively high latency of erases from affecting the performance of updates. These out-of-place updates result in invalidation of older versions of pages requiring Garbage Collection (GC) to reclaim certain invalid pages in order to create room for newer writes. At a high level, GC operates by erasing certain blocks after relocating any valid pages within them to new pages. A final characteristic concerns the lifetime of flash memory, which is limited by the number of erase operations on its cells. Each block typically has a lifetime of 5K(MLC) or 100K(SLC) erase operations. Wear leveling (WL) techniques [20, 22, 32] are employed by the FTL to maintain similar lifetime for all the blocks. Table 1 presents representative values for the operational latencies, page/block sizes, and lifetime for two main flash technologies (SLC and MLC) [38, 39, 37]. We consider SLC-based flash in this work, although our ideas apply equally to MLC.

The Flash Translation Layer (FTL) is a software layer that helps in emulating an SSD as a block device by hiding the erase-before-write characteristics of flash memory. The FTL consists of three main logical components: (i) a Mapping Unit that performs data placement and translation of logical-physical addresses, (ii) the GC, and (iii) the WL. A key data structure maintained by the FTL is a *Mapping Table* which stores address translations. Upon receiving a write/update request for a logical page the FTL: (i) chooses an erased physical page where it writes this data, (ii) invalidates the previous version (if any) of the page in question, and (iii) updates its mapping table to reflect this change. The Mapping Table is typically stored on SSD's RAM to allow fast translation[1].

2.2 SSD Enhancements for CAS

In Figure 1(b), we present the additional components/functionality (compared to a traditional drive) required by CA-SSD. For both devices, we also show the steps involved in processing requests coming from the block device driver to help understand the difference in their operation. We refer to the FTL in CA-SSD as *CA-FTL*. Read requests are handled identically in both the SSDs and so we only focus on write requests. Whereas a traditional SSD requires all writes to be sent to physical pages, CA-SSD returns a write request without requiring flash page writes if hashes, representing their content, are found in RAM. We require four key enhancements to a traditional SSD to achieve this functionality.

(i) *Hashing Unit*: CA-FTL requires the ability to compute/compare content hashes such that these operations only degrade the CA-SSD performance to a negligible (or tolerable) extent. To ensure this, we propose to employ a dedicated co-processor to implement our hashing unit. Recently, manufacturers like OCZ [5], Samsung [44] and pureSilicon [41] have developed high performance SSDs with on-board cryptographic processors, suggesting that the desired fast hashing is feasible.

(ii) *Additional Meta-data*: Mapping Unit must maintain additional data structures for CAS that puts additional pressure on the *on-SSD RAM*. These structures represent CA-FTL's *meta-data* (to be distinguished from the meta-data for software such as the file system) and the portion of on-SSD RAM used for storing it is referred as the *meta-data cache*. We describe these data structures and space-efficient ways of managing them in Section 4.1.

(iii) *Persistent Meta-data Store*: Our CA-SSD design necessitates a re-consideration of the mechanism for recovering the contents of the meta-data cache after a power failure. When writing a physical page (PPN), a traditional FTL also stores the logical page number (LPN) in a special-purpose part of the PPN called the

[1] An SSD typically has a small SRAM and a larger DRAM cache whose size is in the range 64-512 MB for an SSD with capacity 256-1024 GB [1, 6]. We ignore this distinction in our discussion.

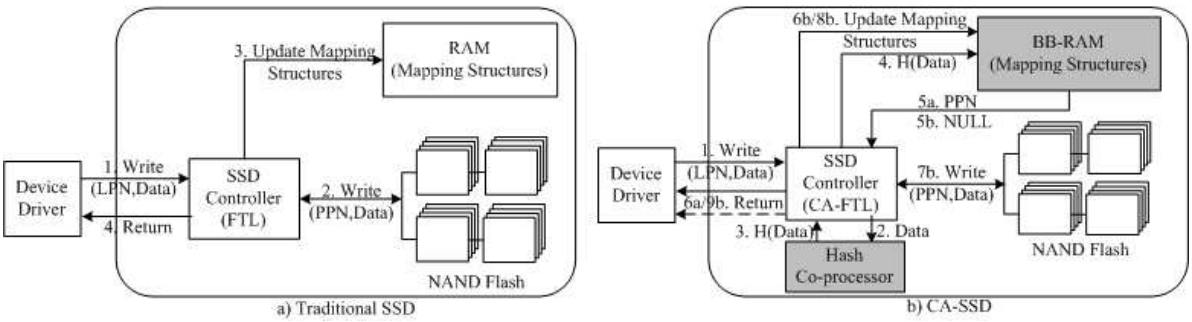


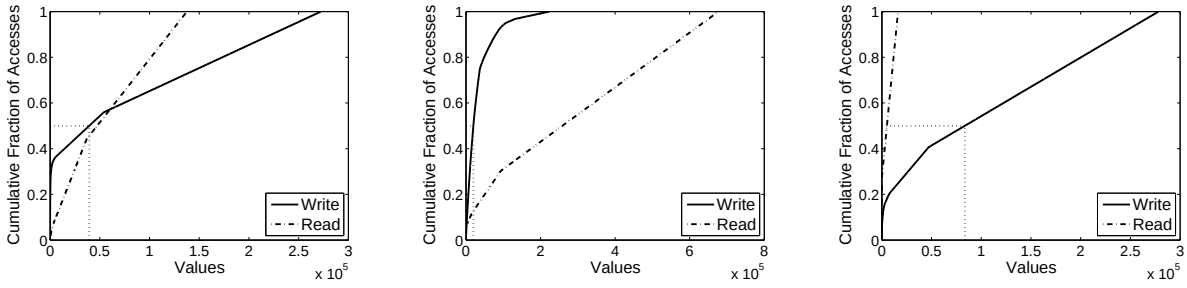
Figure 1: Components of a CA-SSD compared to traditional SSD. CA-SSD has two new hardware elements: (i) a hashing co-processor and (ii) a battery-backed RAM (BB-RAM). Furthermore, CA-SSD stores hashes instead of LPN in the page OOB area. Also shown is a comparison of how writes are handled in the two devices. (a) Traditional SSD: (1-2) On receiving a write request from device driver, SSD controller issues a flash page write. (3-4) On completion, the Mapping Table in the volatile RAM is updated and driver is notified of request completion. (b) CA-SSD: (1-2) On receiving a write request, the SSD controller sends the content to the hash co-processor for hash computation. (3-4) The returned hash is then looked up in the Mapping Table in the BB-RAM. (5-6(a)) On a hit, the mapping structures are updated and the request completes. (5-9(b)) On a miss, a flash page write is performed, mapping structures are updated and the request is completed.

out-of-band (OOB) area, which is typically 64-224 B in size. After a power failure, these entries in the OOB are used to reconstruct the LPN-to-PPN mappings. In CA-FTL, multiple LPNs may contain the same value and hence correspond to the same PPN. The OOB area may not have enough room for all these LPNs. Furthermore, a value can be associated with a changing set of LPNs over its lifetime, requiring multiple writes to the same OOB area, with corresponding erase/copying operations. We address this difficulty by requiring that CA-FTL's Mapping Table be kept in a fast persistent storage in the first place, without any need to store a copy on flash. Storing a copy on flash would result in large number of meta-data writes on flash increasing the number of flash page writes. An alternative approach could be to perform periodic check-pointing of Mapping Table instead of immediate writes on flash to reduce the number of meta-data writes, thereby providing weaker guarantees on meta-data consistency. In order to provide consistency guarantees similar to existing SSDs without impacting the overall performance, we employ persistent battery-backed RAM. We indicate this as BB-RAM in Figure 1(b). Write caches based on such battery-backed DRAM are commonly used in RAID controllers [3]. Even SSD manufacturers have started providing battery-backed DRAM as a standard feature to deal with power failures [5, 4]. Such SSDs with both battery-backed DRAM as well as on-board cryptographic processors have similar performance and costs as compared to traditional SSDs [5] mitigating performance and cost concerns for CA-SSD. Recent work has considered employing other persistent media (e.g., PCM [45] and even hard disk [46]) for SSD write optimizations, and exploring such alternatives for CA-SSD meta-data cache is part of future work.

(iv) *Re-design of GC*: CAS results in a change to GC. In conventional FTLs, each update results in the invalidation of a page requiring an eventual erase operation. But *CA-FTL only needs to invalidate a page when no LPN points to the value in that page*. This redefines the way garbage is created and distributed in blocks impacting the efficiency of GC. We study these issues in Section 4.2. We do not modify WL policy in this work and assume CA-SSD continues to employ the default WL.

3 Value Locality Characterization

We describe two aspects of value locality (VL) that have performance/lifetime implications for a CA-SSD. We propose ways to express these aspects statistically and discuss their implications for possible improvements in CA-SSD. Throughout our discussion, we employ three workload traces [26] described in Table 2 to present examples of our VL characterization. *homes* represents a file server of the home directories of a research group in FIU's CIS department. A major source of content similarity in this workload can be attributed to work done by different members of the group on copies of same software codes, technical documents etc. present in their directories. *mail* has been collected from the e-mail server of the same department containing similar mailing-list emails and circulated attachments resulting in content similarity across user INBOXes. Finally, *web* is their Web server workload consisting of virtual machines hosting an online course management system



(a) web (b) mail (c) homes

Figure 2: Value popularity in real-world workloads (1 day traces). The x-axis consists of unique values sorted according to their read or write popularity. That is, a given point on the x-axis might correspond to different values for reads and writes. We also show the number of unique values that correspond to 50% of all write requests.

Workload	Size (GB)	% Writes	Req. (mill.)	Unique Request (%)		Seq. %
				Write	Read	
web	1.95	77.01	3.8	42.35	32.05	83.8
mail	4.22	77.32	3.6	7.83	80.85	94.7
homes	3.02	96.76	4.4	66.37	80.75	70.8

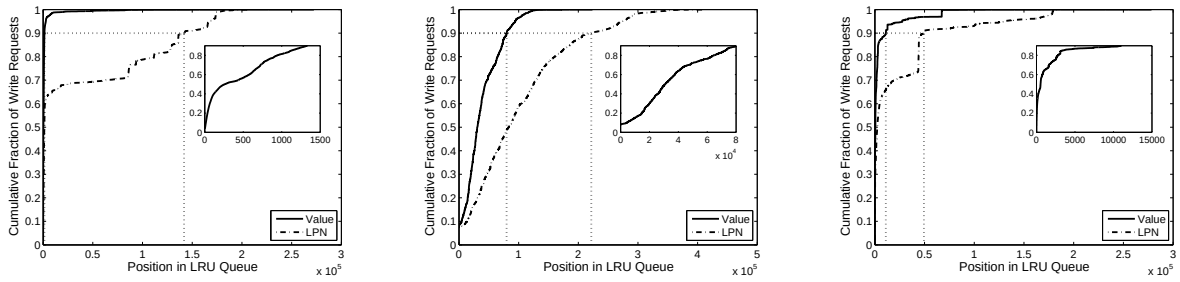
Table 2: Workload statistics. Workload duration varies from 1 day (*mail*) to 7 days (*web*,*homes*). Size represents the total number of unique LPNs accessed in the trace over the mentioned duration and hence represents a compacted trace without any intermediate non-accessed LPNs (The SSD size chosen for evaluation is 4GB for *homes* & *web* and 6GB for *mail*). The logical address space exposed to the file-system is much larger [26]. Unique Request denotes the fraction of write(read) requests which write(read) unique 4KB chunks. Requests are deemed sequential(seq.) if they access consecutive LPNs.

and email access portal. These workloads are primarily write-dominant, especially *homes*, which has about 97% write requests. Individual requests in these workloads are of size 4KB, along with a 16B hash(MD5) of the contents.

Value Popularity (VP): The most straightforward characterization of VL represents the popularity (number of occurrences) of each unique value, for both reads and writes separately. The VL for writes and reads have different implications for CA-SSD: whereas the former captures reduction in write traffic offered by caching the corresponding (value, LPN, physical page) information in the meta-data cache, the latter captures reduction in reads due to caching the corresponding content in the content cache. Table 2 shows the high VP exhibited by real-world workloads. For instance, *mail* has only 7.83% unique write requests, representing a huge potential for de-duplicating the remaining 2.63 million writes. Similarly, *web* and *homes* can provide 57.65% and 33.63% write reductions respectively, improving the performance and lifetime of SSDs substantially. Furthermore, only a small fraction of writes in these workloads are due to same values being written at the same locations. For example, about 8% overall writes in *mail* and *homes* are due to same LPN writing the same content successively. A majority of duplicate writes are attributed to same content being written to different locations requiring sophisticated CAS-based scheme for de-duplication. In Figure 2, we present VP (as CDFs) for reads and writes for the three workloads. A given point on the x-axis can correspond to different values for reads/writes.

The following insights and observations emerge from our definition and these statistics. *First*, all these workloads exhibit significant skewness in VP, i.e, a small fraction of total values account for large number of accesses. For example, the fraction of total unique values that account for 50% of the overall writes are 14.44%, 8.84%, and 29.99% for *homes*, *mail* and *web* respectively(shown by dotted lines). Therefore, pinning these values in the meta-data cache can offer write traffic reduction of 35.56%, 41.16%, and 20.01%, respectively. Similar benefits apply for reads upon caching the most popular (value, content) pairs in the content cache. *Second*, we find that these workloads exhibit different degrees of value popularity (e.g., *homes* has higher VP for reads than *mail*, while *mail* has higher VP for writes) implying different degrees of potential benefits for reads/writes.

Temporal Value Locality (TVL): The presence of TVL in a workload implies that if a certain value (as opposed to LPN) is accessed now, it is likely to be accessed again in the near future, not necessarily by the same LPN. We distinguish TVL for writes and reads to be able to differentiate benefits that could be obtained



(a) web (b) mail (c) homes

Figure 3: Temporal value locality and temporal locality(labeled LPN) for writes in real-world workloads (1 day traces). We show the meta-data cache size that can contribute to 90% of the total writes.

from the use of meta-data vs. content caches. We modify a standard way of characterizing LPN-based temporal locality for representing TVL [21]. For each workload, assuming the meta-data cache to be managed as a queue with a least-recently-used (LRU) eviction policy for values, we present CDFs of number of writes of the value at the $(i+1)^{st}(i \geq 0)$ location within the LRU queue in Figure 3.

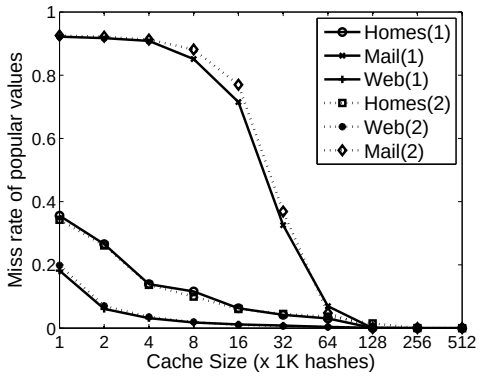


Figure 4: Cache miss rate for popular values. The number in brackets represent the length of the trace in number of days. Note that popular values denotes the minimum number of values which account for 50% of accesses in the workload. The cache size on the X-axis (logscale) is in terms of 1K hashes.

Implications for writes: The presence of TVL for writes implies that even a small meta-data cache could achieve high hit rates to provide write reduction. For example, the maximum size of the meta-data cache required for storing all the values in the 1 day trace of *homes* is around 7.5MB (each entry in this cache requires 28B for storing the hashing structures as we explain in Section 4.1). However, 90% of writes for *homes* are satisfied within 11046 positions in the LRU queue requiring only about 600KB in the meta-data cache, thus reducing the space requirements by about 96%. Even *mail* which shows lesser TVL provides savings of approximately 65% for achieving 90% hit rate.

Clearly the size of meta-data cache affects these gains. Figure 4 shows the miss rate for popular value lookups done for writes as a function of different sizes of this cache. Additionally, we use portions of the workloads over 1-day and 2-day periods and find that TVL sustains over this duration. We find that for our workloads, a LRU cache based on TVL is able to hold popular values, thus offering an easy way to implement a technique that can recognize VP. Whereas in our workloads, TVL and skewness in VP occur together, generally speaking, these could be mutually exclusive. For example, it may be the case that for a workload with high TVL, all values are equally popular, i.e., have comparable number of write accesses, thus displaying low skewness in VP. Alternately, a workload with high skewness(in VP) can exhibit low TVL if the popular values have a long time gap between successive accesses. We design CA-SSD so that it can exploit these properties whenever present, but not experience degraded performance (compared to a regular SSD) when these are absent.

Implications for reads: We observe higher TVL than temporal locality even for reads suggesting that, for these workloads, a value-based content cache is likely to outperform one using LPNs and offer reduction in read traffic to the SSD. Similar observation was made in [26] for developing a content based cache for improving I/O performance in the context of HDD-based storage.

Finally, one could also consider a notion of *spatial value locality* (SVL). The principal of spatial locality, as used conventionally, can be stated as follows: if (content corresponding to) a logical address X is accessed now, addresses in the neighborhood of X are likely to be accessed in the near future. SVL emerges from a generalized take on what the neighborhood or proximity of a data item means. It posits that a given value, even when part of different logical data items, is likely

to see similarities among the values in its neighborhood. Stated another way, spatial value locality hypothesizes that there might exist positive correlations among certain values in terms of their closeness with respect to their addresses within (possibly multiple) logical data objects. SVL has been used for handling disk bottleneck for meta-data management in CAS systems for backup applications by prefetching key-value pairs which are accessed together [48]. For SSDs, it can provide additional benefits for reads when sub-page level chunks are used. We do not explore SVL or other optimizations for reads in this work and leave it as part of our future work.

4 Design of CA-FTL

We develop the CA-FTL mapping unit and GC based on the issues discussed in Section 2. We assume a CAS chunk unit to be equal to the flash page size.

4.1 The CA-FTL Mapping Unit

Address Translation and Meta-data Management: As discussed in Section 2, CA-FTL requires additional data structures for maintaining information about hashes and their relationship with LPNs. Figure 5(b) shows the data structures we employ to realize CA-FTL's Mapping Unit. We assume address translations to be kept at the granularity of a page. Such page-level mappings have been shown to be desirable and scalable in recent research [16, 25]. *First*, similar to existing FTLs, we have a table called LPT which stores translations between LPNs to PPNs. Each entry requires 4B for storing the LPN and another 4B for PPN. Thus, the maximum space needed for LPT in a 4GB SSD is 8MB (for 100% flash utilization). *Second*, an inverted LPT (iLPT) stores the list of LPNs that correspond to the same value and thus the same PPN. The iLPT is used to keep track of valid values. If the LPN list for a PPN is empty, it signifies that no LPN stores the value present in that PPN and the page should be invalidated. The iLPT is queried during GC and WL for updating the LPT whenever the PPN storing a value changes. *Third*, we use a hash-to-PPN table (HPT) to store hash to PPN mappings that is looked up on a write request to decide whether the write is for an existing value (no flash write needed) or a new value (requires a flash write). Entries are inserted or updated in the HPT upon (i) a write request with a new value or (ii) a page write due to GC/WL, respectively. Page invalidations result in removal of entries. Each hash is 16-20B long depending on the hashing algorithm used (16B for MD5 and 20B for SHA1) whereas a PPN is 4B long. For a 4GB SSD, the maximum space needed for the HPT is 20-24MB since the maximum number of PPNs it can store is 1M. All further discussion is in context with MD5 hashes present in the available real-world workloads but our ideas apply readily for SHA1 hashes also. *Fourth*, we employ an inverted HPT (iHPT) which maps PPNs to hashes by storing the addresses of the corresponding HPT entries. It stores the same number of valid entries as HPT. When a flash page is invalidated, iHPT provides the address of the corresponding HPT entry to be removed without incurring an OOB read.

Let us now understand how to deal with space overheads of these data structures. (i) Gupta et al. [16] have proposed page based FTL which exploits temporal locality in workloads to reduce the LPT space requirements. As shown in Figure 3, real-world workloads also demonstrate significant temporal locality apart from TVL. Thus, we can utilize variants of page-based FTLs to reduce the space requirements by only storing a subset of the LPT/iLPT in our BB-RAM. (ii) Since the HPT/iHPT's space needs can be prohibitively large (recall that on a 4GB flash, they require up to 28MB of RAM), we are forced to only store a subset. Given our findings about the presence of TVL in workloads, we implement the HPT as a cache of hash-to-PPN mappings employing a LRU eviction policy for writes of values. The size of this cache could be chosen by CA-FTL based on how much RAM it can afford to use for meta-data storage. When all of this cache is occupied, to insert a new entry we discard the least-recently-used entry from the HPT and the iHPT. A salient aspect of our strategy is that, unlike a traditional LRU-based queue, we do not maintain the remainder of the HPT/iHPT (which does not fit in RAM) on another storage medium (e.g., the flash medium itself). On an entry's eviction from the meta-data cache, we simply discard it. This saves us potential flash page writes (write-back of evicted dirty entries) and reads (mapping entry lookup on a HPT/iHPT miss). This scheme trades off reduction in RAM occupied for meta-data for a reduction in the degree of data de-duplication achieved, since some values may be re-written upon HPT misses. Our findings on TVL in real workloads in Section 3 suggest that such misses are likely to be rare even for nominal cache sizes. This is shown in Figure 4 where a cache size of 1.75MB (for storing 64K hashes) yields miss rates less than 7% for *mail*. For *web* and *homes*, these miss rates are even smaller, being 0.4% and 4%, respectively. Thus, most of the discarded entries correspond to less popular values which have low write frequency and less impact on de-duplication efficiency. In Section 5, we evaluate the performance of CA-SSD with different meta-data cache sizes. Data/meta-data consistency is not impacted due to this scheme since the LPT which stores the LPN-to-PPN mappings required for managing consistency (explained earlier in Section 2.2) is managed independent of this strategy. Furthermore, BB-RAM is only needed for persistent storage of

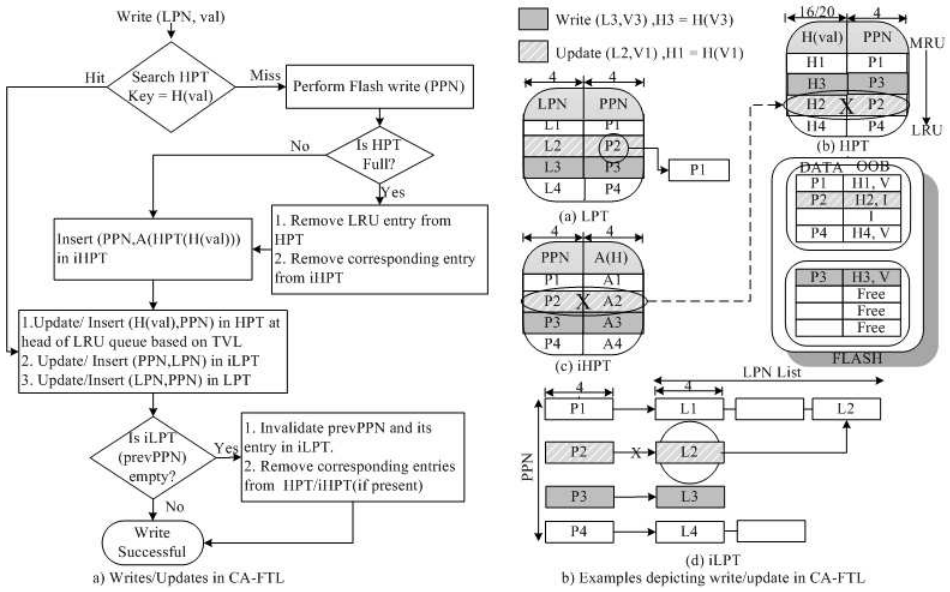


Figure 5: (a) Flowchart depicting how writes are handled by CA-FTL. *val* represents the content to be written. (b) Example of write requests: (1) Write request (L3,V3) to a new LPN L3 with a new value V3 results in a flash page write (P3). Entries are added to all four data structures. (2) Update (L2,V1) results in a HPT hit for H1, the entry is moved to the head of LRU queue(based on TVL) in HPT. L2 is then added to the the LPN list for P1 in the iLPT and removed from P2's list. Since P2's list (in the iLPT) is now empty, the flash page (P2) is invalidated and the corresponding entries in HPT and iHPT are removed. (Note that iHPT only stores the address of the corresponding HPT entry and not the complete hash.)

LPT whereas other mapping structures can be stored on volatile RAM without impacting consistency.

Handling Read/Write Requests: Read requests in CA-FTL are handled similar to traditional FTLs. LPT is looked up to locate the PPN storing the value and its contents are returned to the upper layers. The flowchart in Figure 5(a) describes the handling the write/update requests in CA-FTL. On receiving a write request, the hash of the value for each LPN comprising the request is calculated and the HPT is then looked up with this hash. A miss is deemed to indicate request for a new value and a flash page write is issued. If the HPT is fully occupied, the least recently used entry is discarded and the new (hash, PPN) entry is inserted at the head of the LRU queue based on TVL. Corresponding updates are made in the iHPT also. Finally, the LPT and the iLPT are updated. On a hit in the HPT, the entry is moved to the head of the LRU queue and LPT/iLPT are updated. Furthermore, update requests may result in LPN storing a different value, requiring modifications to the mapping entries for the LPN's earlier value. If the LPN list in iLPT for the PPN corresponding to the LPN's earlier value is empty, the entry and the physical page on flash are invalidated. Finally, the HPT/iHPT entries for this PPN are also removed (Note that the eviction strategy may have already discarded the HPT/iHPT entries, hence not requiring an explicit removal). Figure 5(b) gives examples describing the handling of writes in CA-FTL including relevant meta-data cache management.

4.2 Garbage Collection in CA-FTL

Unlike conventional SSDs where all writes are propagated to flash, CA-SSD only requires one write per unique value (W_{unique}) except in the case of meta-data cache misses (due to limited cache size) where some duplicate values (W_{dup}) may also be written. Writes may also be needed for values which have been invalidated/erased (when no LPN points to them) and are reborn (W_{reborn}) due to subsequent write requests. Similar to conventional SSDs, the final component is GC writes (W_{gc}) which depends on the number of GC invocations as well as the number of valid pages copied upon each such invocation. Therefore, the total writes for a CA-SSD can be expressed as a sum of these components: $W_{total} = W_{unique} + W_{dup} + W_{reborn} + W_{gc}$.

In traditional SSDs, every LPN update results in invalidation of the PPN containing the previous LPN version. CA-SSD only invalidates pages when the value in them becomes *dead* in the sense of no LPN being associated with it any longer. Thus, garbage is likely to be gener-

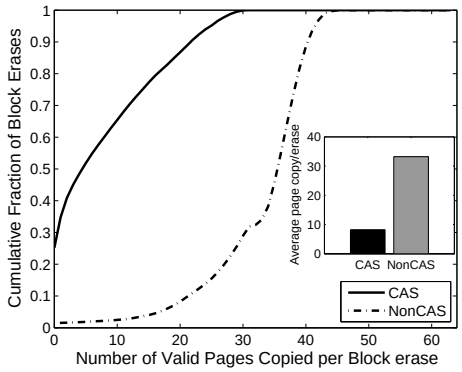


Figure 6: Cumulative distribution of valid pages in blocks erased during GC in *web* workload.

ated at a slower rate in CA-SSD. This coupled with the reduction in write traffic to flash due to de-duplication decreases the number of GC invocations for the same GC policy as in a traditional SSD. The other aspect is the number of pages copied during GC. As shown in Figure 6 for *web*, the valid content in the victim blocks is much lower in CA-SSD as compared to that in traditional SSD. The average number of pages copied per block decreases from 33.20 to 8.21, a reduction of about 75.27% with CA-SSD. This is primarily due to data de-duplication which reduces the amount of total valid content stored on flash, in turn increasing the fraction of invalid pages in victims. These observations lead us to conclude that existing GC mechanisms should work well even in a CA-SSD. We evaluate the impact of our choice in Section 5.

5 Experimental Evaluation

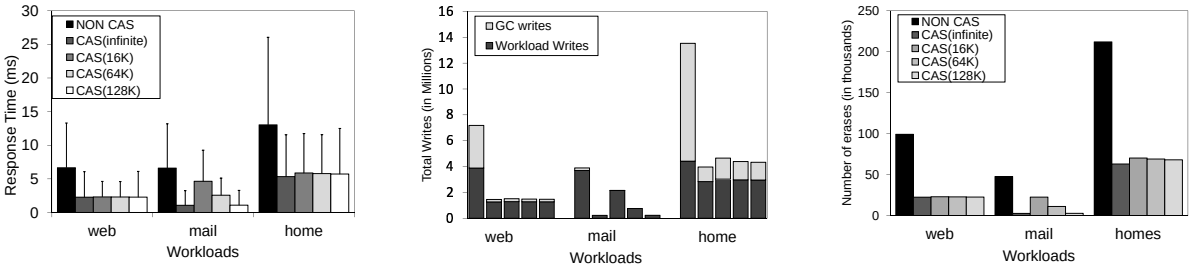
5.1 Experimental Setup

We simulate both traditional and CA-SSDs using SSD simulator [10] which has been integrated into Disksim-4.0 [19]. The SSD simulator is capable of simulating both SLC and MLC SSDs with multiple planes and dies. As described in Section 2, we use SLC SSDs with extra large pages(SLC2) and single plane in this study (refer to Table 1 for SSD properties). We have modified the Disksim interface to use block-based traces with content hashes. We have implemented the FTL for our CA-SSD (CA-FTL) with the meta-data cache manintained using LRU eviction based on TVL. We simulate the hashing unit in CA-SSD by modeling the overheads (32μs [18]) of performing hash calculation along with their impact on the queueing delays at the SSD controller. Note that this is a conservative estimate and the hash calculation overheads are likely to be much lower in CA-SSD (As discussed in Section 2.2, SSDs with crypto-units have reported similar performance to traditional SSDs [5]). As explained earlier, we do not simulate read caching in either traditional or CA-SSD.

5.2 Real-world Traces

We first focus on the three real workload traces that were found to exhibit high VL in Section 3. Figure 7(a) shows the mean response time comparing the standard SSD with two CA-SSD configurations: (i) sufficient capacity in its RAM to store HPT/iHPT and (ii) capacity to store only a fixed number of hashes in RAM. For example, storing 128K hashes in HPT/iHPT requires 3.5MB. We also present mean response times for other meta-data cache configurations. We note the tremendous performance benefits obtained with our CA-SSD compared to the traditional SSD and the benefits directly correlate to the value locality/popularity in writes. For instance, the *mail* workload, which in Figure 2(b) demonstrates the highest VP of the three for writes, shows a 84% reduction in response time with CA-SSD compared to the traditional SSD. The reductions are substantial for *homes* and *web* as well, which show 59% and 65% improvements in response times.

In order to understand these benefits further, we break down the write traffic into those that are (a) directly imposed by the workload and (b) additional writes imposed due to GC when valid pages need to be copied across blocks. The number of writes in each category is shown in Figure 7(b) for the traditional SSD and our CA-SSD. Overall, the reductions in write traffic for CA-SSD are 77%, 93% and 70% for *web*,*mail* and *homes*, respectively over a traditional SSD. We see significant reductions in writes of both categories. The drop in category (a) is intuititve to follow given the value popularity in the workloads. Additionally, there is significant reduction in category (b) writes as well - 94%, 100% and 87% for *web*, *mail* , and *homes*, respectively. In fact, in percentage terms, these GC write reductions overshadow the category (a) reductions. Note that GC overhead is a function of the amount of garbage in the flash, and the distribution of this garbage across the blocks. Since a page in CA-SSD is treated as invalid only when all the LPNs having that content have written a "different value," it is less likely to be marked as garbage compared to a traditional SSD where "any" (including the prior identical) LPN write necessitates a page invalidation. Furthermore, the decrease in the amount of valid content on the SSD due to de-duplication directly reduces pages copied during GC. All these reasons contribute to the substantial benefits that CA-SSD experiences in lower induced writes/copies compared to a traditional SSD. In fact, for the *mail* workload we observe no GC writes since the total number of unique values seen for this workload fits



(a) Response Time (b) Total Writes (c) Block Erases

Figure 7: Performance of CA-SSD vs traditional SSDs. (b)The reborn writes fraction is extremely low and hence not shown. The bars for each workload should be read in the following order: NonCAS, CAS(infinite), CAS(16K), CAS(128K), CAS(256K). Note that CAS(x) represents the meta-data cache size in terms of number of hashes(x) it can store. For response times, we also present the standard deviation, and observe that CA-SSD offers reduction in the variance in addition to the average.

within the chosen SSD size without triggering GC.

Another important characteristic is the lifetime of SSD which depends on the write-erase cycles of blocks. Higher incoming write traffic results in higher block erases, reducing the useful lifetime of SSD. Write reduction benefits from CA-SSD on both workload and GC writes directly translate into reduced block erases. As shown in Figure 7(c), the number of block erases in *mail* reduces from 47819 to 2876, more than 15-fold decrease. Similarly, *homes* and *web* experience 70% and 77% reductions in block erases, respectively.

In Figure 7, we showed results for CA-SSD with both unlimited RAM capacity to store the HPT/iHPT, as well as finite capacities of 16K, 128K, and 256K entries that require about 450KB, 3.5MB and 7MB of space respectively. Even for meta-data cache capacities less than 1MB, CA-SSD shows significant improvements over traditional SSD. For example, the mean response time for *homes* decreases by about 7ms (for 16K hashes) in CA-SSD as compared to traditional SSD whereas the block erases reduce by 65%. As we had seen in Section 3, *mail* shows lower TVL for writes and hence requires a larger meta-data cache to exploit CA-SSD benefits. However, we note that beyond 128K entries, we observe close to the infinite CA-SSD behavior for all workloads, reiterating the observations made in Section 3 regarding the ability to hold a substantial portion of the working set of the meta-data in these workloads within a relatively small space because of presence of TVL. 3.5MB of RAM is a relatively small amount of space to support in today's SSDs - for instance, a 1TB [6] SSD has 512MB of DRAM which can be used for storing the meta-data. Regardless of the actual amount of available space to store this meta-data, CA-SSD can avail of whatever space is allocated to it, and as we will show in the next subsection, even "complete absence of value locality" makes CA-SSD only slightly worse than a traditional SSD.

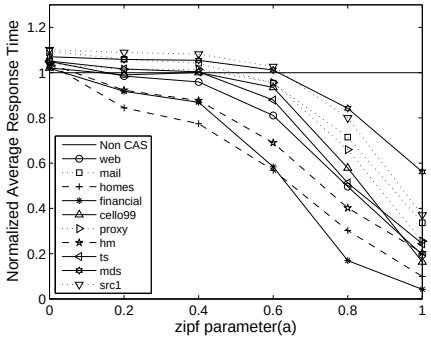


Figure 8: Impact of VL. zipf parameter on X-axis represents the extent of VP skewness in the workload. Higher zipf paramter indicates higher skewness in VP. The average response times on Y-axis are normalized with respect to average response times for traditional SSDs. Note that these response times are for unlimited cache-space. We observe similar response times for meta-data cache which can store 128K hashes.

5.3 Impact of Value Locality on CA-SSD

We next conduct a more extensive analysis of the impact of value locality on CA-SSD performance to demonstrate that it is beneficial across a broad spectrum of workload behaviors and not just for the three real workloads used above which exhibit good value locality. One difficulty in considering a wide range of workloads is the lack of real workload traces for which content of each write is made available in the trace (most traces contain just the timestamp, address and size fields). On the other hand, considering a purely synthetic workload,

Workload	Description	Size (GB)	Requests (in mill.)	% Writes
financial	OLTP	0.50	6.50	79.60
cello99	HP-UX OS	0.46	0.44	70.79
proxy	Proxy server	0.33	2.44	95.64
hm	H/W Monitor	2.43	11.11	54.74
ts	Terminal Server	0.91	4.17	74.06
mds	Media Server	3.09	2.89	70.46
src1	Source Control	1.47	5.00	93.73

Table 3: Workload description. Apart from the above 7 workloads, we use *mail*, *web*, and *homes* that were described in Section 3. The workload size represents the total number of unique logical addresses(LPNs) accessed in the trace. The logical address space exposed to the file system can be much larger.

may mandate assumptions on parameters - such as arrival rate, sequentiality, temporal locality, etc. - over and beyond those pertaining to value locality. Instead, we pick a set of 10 real workload traces(refer to Table 2 and Table 3) that have been studied in prior literature - *financial* from UMass [8], *cello99* from HP Labs [2], *proxy,hm,ts,mds* and *src1* from MSR [9] including the three workloads(*homes,web* and *mail*) from FIU [26] . We use the arrival times, block addresses and sizes from these traces, and only synthesize the "content"(v) for the blocks using a *zipf* distribution, given as: $P(v_i) = Cv_i{}^{-a}$, where, $C = 1/\sum_{i=1}^{N} v_i^{-a}$, N is the total unique values in the workload and a is the zipf parameter representing the skewness in value popularity. Many prior studies [13] have shown content popularity can be characterized by this distribution. Furthermore, we vary the exponent(a) characterizing the distribution from 0 (which corresponds to no VP) to 1.0 (which corresponds to a very highly skewed VP behavior). In the experiments, we use this zipf probability distribution to pick a value for each incoming request. This exercises only the popularity of values and ignores the spatial and temporal dimensions of value locality, and can thus be viewed as a pessimistic evaluation of CA-SSD since any spatial/temporal VL will only benefit it further (and not affect the performance of a traditional SSD which only relies on LPN-based spatial/temporal locality). Figure 8 shows mean response times for these workloads on CA-SSD normalized with respect to traditional SSD response times. Similar to results in Section 5.2, as VP increases, the response times for these workloads decreases. Furthermore, even when VP is low, the response times for CA-SSD and traditional SSDs are comparable. We observe that when the workloads show no VP (a=0.0), the average response time of CA-SSD only increases by atmost 10% (for src1). This is primarily due to the overheads of the hashing unit for write requests which we have chosen conservatively. Thus, we expect the average response time to be lower with a more aggressive estimate (If needed, one could even explore the possibility of dynamically turning off CAS in CA-SSD in complete absence of VL). On the other hand, for high VP ($a = 1.0$), we see tremendous benefits with CA-SSD. We observe around 25 times reduction in average response times for financial trace and on average all workloads show an improvement of about 74%. Furthermore, the number of values which account for 50% of the write requests in hm workload decreases from 4.5M for no VP ($a = 0.0$) to 1.3M for moderate VP ($a = 0.4$), a reduction of approximately 71%. This clearly illustrates that the benefits accrued through VP specifically and value locality in general, strengthen the case for adoption of content addressability in SSDs, paving the way for a new generation of SSDs.

6 Related Work

Value Locality/Content Addressability: CAS has been extensively used in archival and backup systems such as Venti [42], Foundation [43], Pastiche [14] etc for space savings , Internet suspend/resume [27], LBFS [35] for saving network bandwidth file system and buffer cache design [35, 47, 34], etc. Some recent work has evaluated real-world workloads and demonstrated significant value locality which bodes well for CA-SSD [26, 36]. However, to the best of our knowledge, this paper is the first to focus on issues that arise when designing an SSD that uses CAS internally.

Meta-data Management for CAS: The scalability of a system employing CAS depends on careful management of CAS related meta-data. Larger-sized chunks help in reducing the amount of meta-data to be stored while smaller chunks provide good duplicate elimination. Pasta [33], Pastiche [14], REBL [29] and Foundation [43] have explored more complex chunking methods. Bimodal chunking attempts to combine the benefits of two different chunk sizes [28] CA-SSD could benefit from all of these techniques and evaluating the benefits of different/variable chunk sizes is part of our future work. Sparse indexing divides the incoming data stream into large segments which are then de-duplicated against a few similar segments found using sampling [31]. Like sparse indexing, the degree of de-duplication in CA-SSD depends on the available meta-data cache space. Researchers have developed CAS meta-data management techniques which utilize HDD/SSDs for storing chunk indexes [48, 15]. These techniques utilize spatial locality in data segments for reducing index lookups by

pre-fetching meta-data in RAM. Unlike these techniques, CA-SSD does away with index lookups on HDD/SSD and utilizes TVL for reducing meta-data misses.

7 Conclusion

Given evidence for the presence of significant VL in real-world workloads, we designed CA-SSD which employed CAS for its internal data management. Using three real-world workloads with content information, we devised statistical characterizations of two aspects of VL - value popularity and temporal VL - that formed the foundation of CA-SSD. The design of CA-SSD presented us with interesting choices and challenges related to exploiting VL for write reduction and maintaining meta-data consistency under constrained cache space. Using several real-world workloads, we conducted an extensive evaluation of CA-SSD. We found significant improvements (59-84%) in average response times. Even for workloads with little or no value locality, we observed that CA-SSD continued to offer comparable performance to a traditional SSD.

Acknowledgments

We would like to express our gratitude to our shepherd Ohad Rodeh of IBM Almaden Research Center and the anonymous reviewers for their detailed comments that helped us improve the quality of our paper. This research was supported in part by NSF grants CCF-0811670, CNS-0720456, CNS-0615097, and CAREER award CNS-0953541.

References

[1] 64MB Cache on SSD. `http://www.tomshardware.com/news/A-DATA-OCZ-64MB-Cache,7263.html`.

[2] HP Labs. Tools and Traces. `http://tesla.hpl.hp.com/public_software`.

[3] HP Memory Smart Array Controller. `http://www1.hp.com`.

[4] Intel's 3rd Generation X25-M SSD Specs Revealed. `http://www.anandtech.com/show/3965/intels-3rd-generation-x25m-ssd-specs%-revealed`.

[5] OCZ Vertex 2 EX Series SATA II 2.5" SSD. `http://www.ocztechnology.com`.

[6] OCZ Z-Drive R2 e88 PCI-Express SSD. `http://www.ocztechnology.com`.

[7] Raw Drive Capacity Cost Trends. `http://wikibon.org/w/images/a/a4/EMCRawDriveCapacityCostTrends.jpg`.

[8] UMass Trace Repository, 2007. `http://traces.cs.umass.edu`.

[9] SNIA. IOTTA repository, January 2009. `http://iotta.snia.org/`.

[10] AGRAWAL, N., PRABHAKARAN, V., WOBBER, T., DAVIS, J. D., MANASSE, M. S., AND PANIGRAHY, R. Design Tradeoffs for SSD Performance. In *ATC 08: Proceedings of the USENIX Annual Technical Conference* (2008).

[11] ARPACI-DUSSEAU, A., ARPACI-DUSSEAU, R. H., AND PRABHAKARAN, V. Removing The Costs Of Indirection in Flash-based SSDs with Nameless Writes. In *HotStorage 10: Proceedings of the 2nd Workshop on Hot Topics in Storage and File Systems* (2010).

[12] BLACK, J. Compare-by-hash: a reasoned analysis. In *ATC '06: Proceedings of the USENIX '06 Annual Technical Conference* (2006).

[13] CHERVENAK, A. L. Challenges for tertiary storage in multimedia servers. *Parallel Computing* (1998).

[14] COX, L. P., MURRAY, C. D., AND NOBLE, B. D. Pastiche: Making Backup Cheap and Easy. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation* (2002).

[15] DEBNATH, B., SENGUPTA, S., AND LI, J. ChunkStash: Speeding up Inline Storage Deduplication using Flash Memory. In *ATC'10: Proceedings of the USENIX 2010 Annual Technical Conference* (2010).

[16] GUPTA, A., KIM, Y., AND URGAONKAR, B. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems* (2009).

[17] HANDY, J. PCM becomes a reality, 2009. `http://www.objective-analysis.com`.

[18] HELION. Fast hashing cores. `http://www.heliontech.com/fast_hash.htm`.

[19] JOHN S. BUCY, J. S., SCHLOSSER, S. W., AND GANGER, G. R. *The DiskSim Simulation Environment Version 4.0 Reference Manual*. `http://www.pdl.cmu.edu/DiskSim/`.

[20] JUNG, D., CHAE, Y.-H., JO, H., KIM, J.-S., AND LEE, J. A group-based wear-leveling algorithm for large-capacity flash memory storage systems. In *CASES '07: Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded systems* (2007).

[21] KAREDLA, R., LOVE, J. S., AND WHERRY, B. G. Caching strategies to improve disk system performance. *Computer 27*, 3 (1994), 38–46.

[22] KAWAGUCHI, A., NISHIOKA, S., AND MOTODA, H. A flash-memory based file system. In *TCON'95: Proceedings of the USENIX 1995 Technical Conference* (1995).

[23] KGIL, T., AND MUDGE, T. N. FlashCache: a NAND flash memory file cache for low power web servers. In *CASES 06: Proceedings of the 2006 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems* (2006).

[24] KIM, H., AND AHN, S. BPLRU: A buffer management scheme for improving random writes in flash storage. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies* (2008).

[25] KIM, J. K., LEE, H. G., CHOI, S., AND BAHNG, K. I. A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems. In *EMSOFT 2008: Proceedings of the 8th ACM & IEEE International conference on Embedded software* (2008).

[26] KOLLER, R., AND RANGASWAMI, R. I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance. In *FAST'10: Proceedings of the 8th USENIX Conference on File and Storage Technologies* (2010).

[27] KOZUCH, M., AND SATYANARAYANAN, M. Internet suspend/resume. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications* (2002).

[28] KRUUS, E., UNGUREANU, C., AND DUBNICKI, C. Bimodal Content Defined Chunking for Backup Streams. In *FAST'10: Proceedings of the 8th USENIX Conference on File and Storage Technologies* (2010).

[29] KULKARNI, P., DOUGLIS, F., LAVOIE, J., AND TRACEY, J. M. Redundancy elimination within large collections of files. In *ATC '04: Proceedings of the USENIX Annual Technical Conference* (2004).

[30] LEE, S.-W., AND MOON, B. Design of flash-based DBMS: an in-page logging approach. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (2007).

[31] LILLIBRIDGE, M., ESHGHI, K., BHAGWAT, D., DEOLALIKAR, V., TREZISE, G., AND CAMBLE, P. Sparse indexing: large scale, inline deduplication using sampling and locality. In *FAST '09: Proccedings of the 7th USENIX conference on File and storage technologies* (2009).

[32] LOFGREN, K. M. J., NORMAN, R. D., THELIN, G. B., AND GUPTA, A. Wear leveling techniques for flash EEPROM systems. In *United States Patent, No 6850443* (2005).

[33] MORETON, T. D., PRATT, I. A., AND HARRIS, T. L. Storage, Mutability and Naming in Pasta. In *Revised Papers from the NETWORKING 2002 Workshops on Web Engineering and Peer-to-Peer Computing* (2002).

[34] MORREY, C. B., AND GRUNWALD, D. Content-Based Block Caching. In *MSST 06: 23rd IEEE, 14th NASA Goddard Conference on Mass Storage Systems and Technologies* (2006).

[35] MUTHITACHAROEN, A., CHEN, B., AND MAZIÈRES, D. A low-bandwidth network file system. In *SOSP '01: Proceedings of the 18th ACM Symposium on Operating systems principles* (2001).

[36] NATH, P., URGAONKAR, B., AND SIVASUBRAMANIAM, A. Evaluating the Usefulness of Content-Addressable Storage for High-Performance Data-Intensive Applications. In *HPDC 08: Proceedings of the ACM/IEEE International Symposium on High Performance Distributed Computing* (Jun 2008).

[37] NUMONYX MEMORY SOLUTIONS. *16-Gbit MLC NAND flash memories*. `http://numonyx.com/Documents/Datasheets/NAND16GW3D2B.pdf`.

[38] NUMONYX MEMORY SOLUTIONS. *2-Gbit SLC NAND flash memories*. `http://numonyx.com/Documents/Datasheets/NAND02G-BxD.pdf`.

[39] NUMONYX MEMORY SOLUTIONS. *64-Gbit SLC NAND flash memories*. `http://numonyx.com/Documents/Datasheets/NAND64GW3FGA.pdf`.

[40] PRIMMER, R., AND HALLUIN, C. D. Collision and preimage resistance of the centera content address. Tech. rep., 2005.

[41] PURESILICON. Puresi 1TB SSD with hardware based encryption. `http://www.marketwire.com`.

[42] QUINLAN, S., AND DORWARD, S. Venti: A new approach to archival data storage. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies* (2002).

[43] RHEA, S., COX, R., AND PESTEREV, A. Fast, inexpensive content-addressed storage in foundation. In *ATC'08: Proceedings of the USENIX 2008 Annual Technical Conference* (2008).

[44] SAMSUNG. Samsung self encrypting ssd. `"http://www.engadget.com/2009/04/16/samsung-comes-clean-with-self-encry%pting-ssds`.

[45] SMULLEN, C. W., COFFMAN, J., AND GURUMURTHI, S. Accelerating enterprise solid-state disks with non-volatile merge caching. In *IGCC'10: Proceedings of the 1st International Conference on Green Computin* (2010).

[46] SOUNDARARAJAN, G., PRABHAKARAN, V., BALAKRISHNAN, M., AND WOBBER, T. Extending SSD Lifetimes with Disk-Based Write Caches. In *FAST 10: Proceedings of the 8th USENIX Conference on File and Storage Technologies, 2010* (2010).

[47] VILAYANNUR, M., NATH, P., AND SIVASUBRAMANIAM, A. Providing Tunable Consistency For a Parallel File Store. In *FAST05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies* (2005).

[48] ZHU, B., LI, K., AND PATTERSON, H. Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies* (2008).