Content Popularity-Based Selective Replication for Read Redirection in SSDs

Nima Elyasi† Mohammad Arjomand‡ Anand Sivasubramaniam†
Mahmut T. Kandemir† Chita R. Das†
†*The Pennsylvania State University* ‡*Georgia Institute of Technology*
{*nxe125, anand, kandemir, das*}*@cse.psu.edu* *marjomand3@gatech.edu*

***Abstract*—Despite high degrees of parallelism in terms of the number of chips and channels on state-of-the-art SSDs, resource contention continues to be a big impediment to boosting their performance for both read and write requests. This is particularly significant in the delays due to queueing for service from individual NAND-flash chips that can take dozens/hundreds of microseconds to perform the read/write operations. Owing to the no-write-in-place policy that is employed in flash chips, writes are inherently suited to be redirected to chips with lower load, in case their original destination chip is overloaded. However, to date, there has been *no* work to redirect read requests, since they cannot be serviced by other chips, which do not have the data. While blindly replicating all the data everywhere seems very promising from a read redirection perspective, doing so results in high space overheads, high write/replication overheads and lower endurance. This paper presents a novel approach to selective replication, wherein the popularity of data is used to figure out the "what", "how much", "where" and "when" questions for replication. Leveraging value locality/popularity, that is often observed in practice, popular data is replicated across multiple chips to provide more opportunities for dynamic read redirection to less loaded flash chips. Using extensive workload traces running over weeks from real systems, we show that our Read Redirected SSD (RR-SSD) can provide up to 45% improvement in read performance, with average improvement of 23.9%, and up to 40% improvement when considering both read and write requests, with 16% improvement on average.**

***Keywords*-SSD; Content Popularity; Replication; Read Redirection;**

I. Introduction

NAND flash-based SSDs are being employed as a replacement to hard disk drives or as a tiering layer in storage hierarchies of a wide range of platforms. An important concern with SSDs in any of these configurations is that their performance can be influenced by various complexities posed by their internal architecture leading to high resource contention. Redirecting read and write requests to flash chips that have the least contention at any time can alleviate the resource (chips) contention problem. However, while there have been previous efforts to redirect writes leveraging the no-write-in-place property of flash, there has been *no* prior effort at redirecting reads. This paper fills that gap, and presents a novel approach to redirect reads, leveraging the content popularity that has often been observed in data storage.

A modern SSD is typically composed of multiple flash chips and the *Flash Translation Layer* (FTL), which is a software layer responsible for processing and scheduling the requests. The scheduler unit in FTL usually maintains separate read and write queues for each chip. In such architectures, the service time of a request consists of three components: (i) the *waiting time at chip-level queue* (a.k.a. *queuing time*), (ii) the *transfer time on shared channels*, and (iii) the *actual latencies of read or write operation*. The first two components are due to conflicts for shared resources and contribute to the latency overhead in the request service time. To quantify this latency overhead, Figure 1 presents the breakdown of read and write service time for six workloads. We make two observations from this figure. *First*, contention is large for both read and writes. This overhead is very high for writes (80% to 90%). The reason is that, as writes are typically about 10–20 times slower than reads, the FTL's schedulers prioritize reads over writes which in turn increases waiting time of writes in the queues [1]. The contention overhead for reads is also significant and exhibits high variability (between 25% and 77% of total read service time). Note that reads are normally more latency sensitive, being on the critical path of the computation. *Second*, the queuing time is much more dominant than the channel transfer time. In fact, with current ONFi signaling used by modern flash memories, transferring one page to a chip takes at most a few microseconds. As such, *the queuing time is the main contributor to the service time*, and by reducing or removing it, we can expect significant benefits.

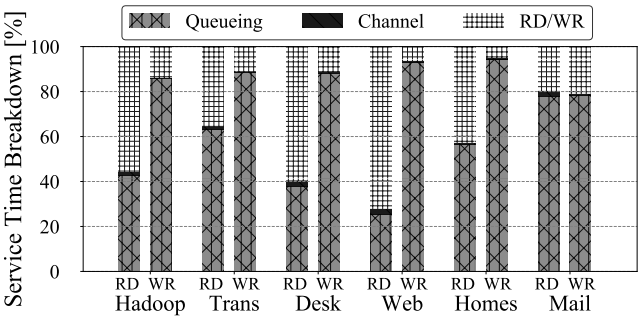


Figure 1: Breakdown of read and write service time in a 64-chip SSD. Section IV describes the evaluated SSD configuration and workloads' characteristics.

The most straightforward way to reduce the queuing time is to map each incoming request to the least-loaded flash

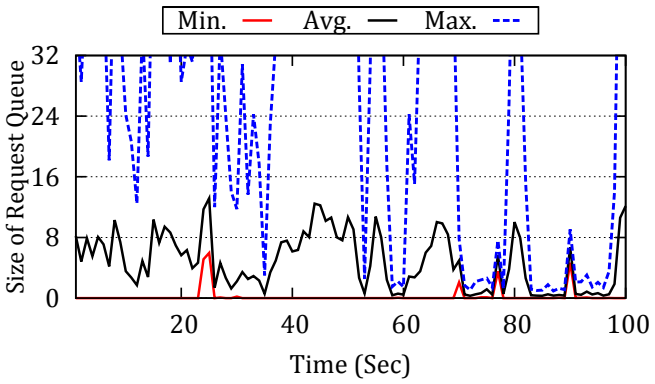


Figure 2: The maximum, minimum and average request queue sizes in an SSD, over 100 seconds of the *webvm* workload, with no redirection of incoming traffic.

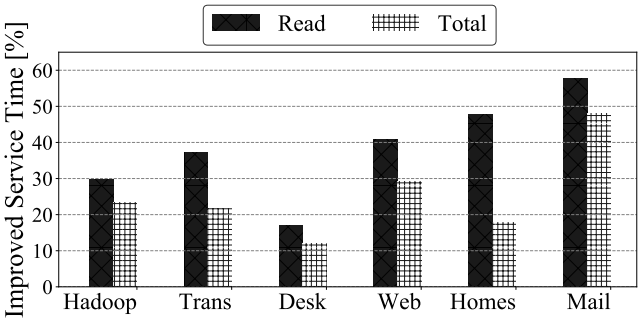


Figure 3: The percentage of read and total service time improvement given by the *Ideal-All-Chip Replication* scheme. Section IV describes the system configuration.

chip at that moment, making chip-level traffic balanced (i.e., at any given time, not all flash chips are load balanced). Figure 2 demonstrates the load imbalance across the chips in an SSD with *no* redirection of incoming traffic [2], [3], by showing the maximum, minimum, and average for the number of requests in the request queues of each chip waiting to be serviced. We assume *page-level mapping* [4], [5]. At any instant, the load varies significantly – sometimes the maximum queue length is 32 times larger than the minimum. As a consequence, a typical request does not necessarily get serviced from the chip with the least load. This motivates the need for a load-aware dynamic mapping/redirection of logical pages to physical pages, so that any incoming request is allocated to the least-loaded chip. Write redirection is easy to implement in SSDs, since updates in flash memories are always out-of-place – i.e., when updating a page, FTL invalidates the current physical page and writes it in another physical page which may be on another chip. Prior works [2], [6], [7] have explored the potential performance benefits of write redirection and discussed the corresponding overheads.

Read redirection is particularly challenging in today's SSDs since a page is read from the location, where it was written. Thus, at the time of reading a page, it is unlikely that the target page is on the currently least-loaded chip. The simplest approach to enable read redirection is to replicate each page (when it is written) on all chips, i.e., keeping N copies of each logical page in an N-chip SSD. To see how much this *Ideal-All-Chip Replication* scheme can improve performance, we conduct a simple experiment: With all the data blocks replicated on all the flash chips without worrying about storage space or the replication costs (i.e., no write costs for replication), the FTL chooses the least-loaded chip for serving each read request. Figure 3 shows the read and total (reads+writes) service time improvements brought by this scheme over a conventional SSD. We can see that read redirection (enabled by the Ideal-All-Chip Replication) can improve read service time by 17% to 57% (38.37% on average) and total service time by 12% to 48% (25.32% on average).

Despite its performance benefits, this Ideal-All-Chip Replication design is very costly to implement in terms of (write) performance, lifetime and space overheads. It amplifies write traffic by N times (in an SSD with N chips) which can become overwhelming since writes are 10–20 times slower than reads, and decreases the device lifetime by almost N times. Furthermore, it also increases design complexity in two ways: (i) it needs N-times more capacity, and (ii) FTL has to keep N *Physical Page Numbers* (PPNs) for each *Logical Page Number* (LPN) in the mapping table.

Prior work [8], [9] have attempted to enable read redirection in SSDs by exploiting temporal locality in the accesses. PBLRU [9] proposes a dynamic page replication mechanism to exploit the multi-chip parallelism in SSDs. However, their physical page replication is not able to reap most of the benefits achieved by the Ideal-All-Chip-Replication and its efficacy is very limited by the amount of space reserved for the replication. Later in this paper, we will quantitatively compare our proposed approach with this work. [8] on the other hand, explores the opportunity of re-directing read requests in an array of SSDs by performing inter-SSD replication taking into account deduplication of data. However, their proposed approach does not consider the internal architecture and resource contention of SSDs which play a significant role in determining the request's response time.

To overcome these challenges, while still enabling read redirection, we propose *content popularity-based selective replication* and introduce a novel SSD design, called *Read-Redirected SSD (RR-SSD)*. The key insight behind RR-SSD is that "*value popularity*" (or *"value locality"*), which has often been observed in I/O datasets, can be exploited so that a small portion of all possible values can be selectively replicated so as to enable redirection of a large fraction of read requests. As such, if the FTL maintains multiple copies of every popular value on multiple chips, at the time of reading a page with the same content, it can redirect the

request to any one of the chips containing the target value. This builds upon the use of SSD as a *Content-Addressable Storage* device[1] as previously proposed [10], [11].

An efficient and low overhead implementation of RR-SSD requires us to answer five important questions: (1) *which data has to be chosen for replication?*, (2) *how much space can be tolerated for holding replicated data?*, (3) *where should each copy of a value be placed?*, (4) *when should the replication process be invoked?*, and (5) *what are the required changes on the SSD board and FTL to support replication/redirection?* In the following sections, we answer these questions separately and describe the design choices and trade-offs that should be taken into account when employing RR-SSD for a storage system.

Evaluation of RR-SSD on a 64-chip SSD with six real-workload disk traces indicates that it improves read service time by up to 45%, with 23.9% improvement on average (by up to 40% improvement across both read and write requests, with average improvement of 16%).

II. Selective Replication

This section systematically answers the five questions raised in the introduction for supporting a read redirection mechanism in SSDs.

A. What to Replicate?

To avoid the high cost of *Ideal-All-Chip Replication* while providing read redirection for a great amount of read traffic, we leverage the phenomenon of *Value Popularity (VP)* (number of occurrences) of each unique value for reads/writes. In Figure 4, we present VP (as CDFs) for reads and writes for three of our workloads (Section IV gives detailed characterization of the studied workloads). In each sub-figure, the points on X-axis are sorted based on their popularity in reads. The following insights emerge from analyzing the results in this figure.

- We observe high *Read Value Popularity* (RVP) for each workload. For instance, the fraction of total unique values that accounts for 80% of overall reads are 14%, 7%, and 30% for `homes`, `web`, and `mail`, respectively (shown by dotted lines). Therefore, in each workload, *only a small set of values are heavily popular*, and by replicating only these values, we can reduce the high costs of replication while providing performance benefits of redirecting a majority of read requests.
- We find that these workloads exhibit different behaviors in the popularity of a specific value for reads versus writes. For example, in `web`, 7% most popular read values (which are 80% of overall read values) contribute to less than 8% of total written content. The case for `mail`, however, is different: most popular values in reads are still the most popular values in writes. Since we target redirection for read requests, we need a mechanism that can selectively capture the popular values in reads, rather than focusing on universal popularity across both reads and writes, as in the prior works.

The presence of high RVP in our workloads has an important implication on our SSD design. By capturing heavily popular values in reads and replicating them over multiple chips we increase the chances for redirecting a majority of read requests to a chip with low load, while we can limit the replication costs. However, the performance improvement and replication costs are related to three other design factors, discussed in the next three subsections.

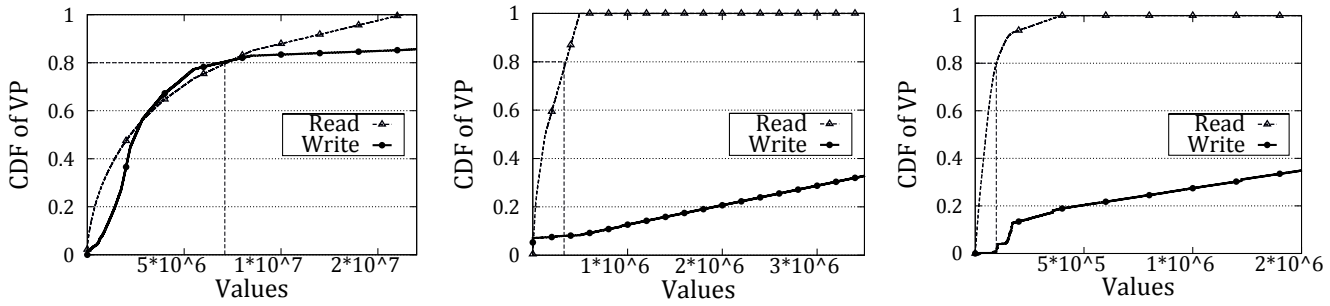
B. How Much Space for Replication?

The space provided for replicated data determines both the performance achieved by read redirection and the replication costs (performance and lifetime). Specifically, by allocating a larger space to the replicated data, the write traffic for replication exacerbates (which also increases the corresponding performance and lifetime costs), while increasing the chances for read redirection (probably reducing the average read access latency). Suppose that the maximum allocated space for replication is $S\%$ of the workload's size (i.e., the total number of unique LPNs accessed in the trace over its entire duration), which has to be determined by the system administrator or the user at the system set-up time. Note that this space overhead ($S\%$) can be zero in our design, while a portion of popular values can be still replicated (with respect to a non-deduplicated store). The reason is that storing dataset in a CA-SSD (which is our baseline system) usually needs less space compared to a traditional SSD which does not remove replicas. Indeed, as prior works [10]–[12] show, most real-world workloads have the same content being written into the same or different addresses which are de-duplicated in a CA-SSD type of architecture. Based on our workload characterization presented in Section IV, the space required to store our workload set in a CA-SSD varies between 15% and 70% of the original size, providing a considerable room for replication.

C. When to Replicate?

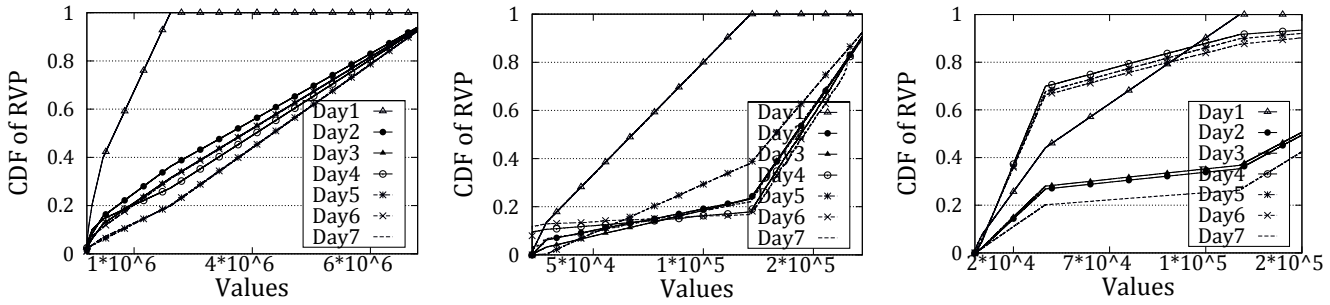
The replication process is expensive as it incurs high write traffic in SSD. Since write operations are very slow in flash memories, this is a serious concern. As a result, the replication process should be *infrequent*. To this end, we leverage another characteristic of value popularity, called *Temporal Value Popularity* (TVP). The presence of TVP in a workload implies that, if a certain value is a popular one

[1]Content-Addressable SSD (CA-SSD) is a type of de-duplication mechanism, which stores only one copy of each value in flash memories and modifies FTL to relate multiple LPNs with same content to one PPN. There are multiple implementations of CA-SSD available in literature [10], [11] which usually employ a cryptographic hash to represent each chunk (usually with size of a page). Similar to these studies, we assume hashed values are collision-resistant (the probability of collision in range of 10^{-9}–10^{-17} [12] for MD5 and SHA-1).



(a) *mail* **(b) *homes*** **(c) *webvm***

Figure 4: VP for read and write requests for the entire duration of three workloads. The points (values) on X-axis are sorted based on their popularity in reads.



(a) *mail* **(b) *homes*** **(c) *webvm***

Figure 5: Temporal VP for reads for the first 7 consecutive days of three workloads. The points on X-axis are sorted based on their read value popularity in the first day.

now, it is likely to remain popular in the near future, even if the LPNs are different. In this work, we specifically employ TVP for reads and discuss its implications on our design. To give a better insight of TVP, Figure 5 presents the value popularity of reads (as CDFs) for the first 7 consecutive days of three workloads. The points on X-axis are sorted based on their popularity in the first day of execution. We observe high TVP for reads in a workload such as `webvm` over a long time[2], implying the set of most read popular values does not change in great extent from day to day[3]. This suggests a simple mechanism that captures the RVPs during one day and replicating them at the end of day, in order to use them for read redirection in the next day.

One may consider read TVP for periods larger than one day (e.g., one week), in order to further reduce the replication costs. However, if the replication is performed very infrequently, the potential gains of read redirection might be affected, as we may fail to capture part of dynamism in VPs of a workload, which could be captured had we used a shorter period. Figure 6 shows the number of distinct values accessed for the first time during each day of the entire execution of three of our workloads. As can be seen, the number of new values accessed each day is variable in a workload, and it is quite considerable on some days (around 10^6). Thus, if we perform replication very infrequently (weekly as an example), we may lose a great amount of benefits gained by the daily read redirection. Table I reports the replication cost (in terms of the number of page copies as well as the time duration) for the three workloads, when the replication is performed daily or weekly. The results are reported for a typical day and week. In this experiment, we set the space overhead of replicated data to 0% (with respect to the size of the non de-duplicated dataset), the read/write latencies to 75us/400us, and the channel transfer time of one (4KB) page to 10us. Two important insights emerge from the results in this table. *First*, even if we perform replication daily, its overhead is not significant in real workloads – it only takes a few minutes (a maximum of 8 minutes in `mail`). *Second*, the replication latency increases almost linearly as its frequency decreases. In other words, weekly-replication generally takes (almost) seven times longer time than daily-replication (a maximum of 62 minutes per week compared to a maximum of 8 minutes per day). Thus, by performing replication daily, we are able to capture dynamism of VP with almost proportionally-same

[2]This pattern does not seem dominant for homes workload. However, as we show later in the results section, other parameters affect the performance benefits and even for this workload, read service time can be improved by employing our approach.

[3]Similar observation was made in prior works developing a content-based cache for I/O performance improvement in content addressable storage based schemes [12].

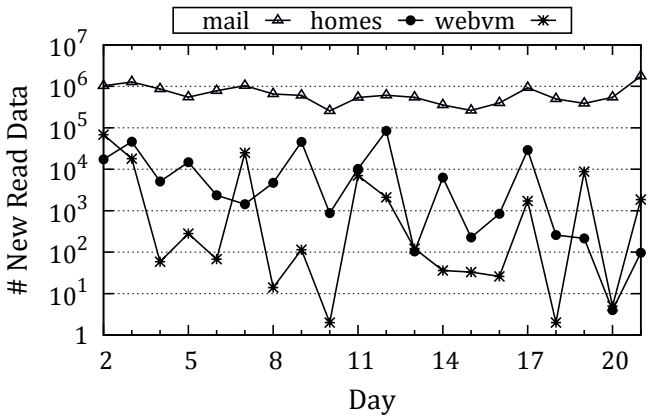


Figure 6: The number of distinct values accessed for the first time during each day in each of the three workloads.

Table I: Comparative analysis of daily-replication versus weekly-replication.

Freq.	mail		homes		webvm	
	Copies (Kilo)	Time (Min)	Copies (Kilo)	Time (Min)	Copies (Kilo)	Time (Min)
Daily	160.9	7.96	5.6	0.28	2.2	0.11
Weekly	1253.2	62.01	39.4	1.95	17.2	0.85

cost of the weekly-replication.

Figure 7 depicts a high-level view of the proposed replication process. The daily operation has (i) the normal operation phase which includes read redirection and collecting information for the replication process at the end of the day; and (ii) the Replication phase at the end of the day (for 8 minutes) when replicas are created based on the VP of that day.

Profiling for VP – Each day, we first need to collect information that helps determine the value popularity of the dataset. A typical way of profiling requires counters for every hash value and incrementing the counter for the corresponding data on a read. The counters can be implemented in hardware or software. To capture all read activities at high resolution, we generally need large counters which are costly (especially hardware counters). Instead we employ N bit saturating counters –with N referring to the $Log(Numberofflashchips)$ – since the most popular data will be, at most, replicated on all flash chips, with each flash chip only storing one copy of that data. Thus, these counters suffice to distinguish very popular values from the rest. As we will show, this counter can be maintained in the flash controller or in the FTL itself which is anyway looked upon each access to this value for a normal operation.

Replication – At the end of each day, the profiled data is used to perform replication for the next day. This can be expensive since it involves counter sorting, mapping each replicated data, and updating the mapping tables at

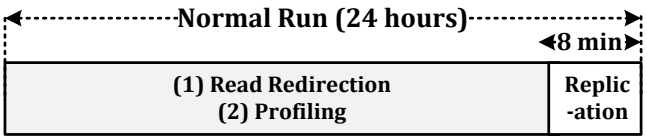


Figure 7: A high-level view of one period of our replication scheme.

the FTL in addition to *actual* data replication (i.e., reading and writing each data to be copied). Section III describes how the execution phase can be realized through a series of administrative actions, which have no impact on the system availability and minimal impact on the system performance, even though it takes at most 8 minutes for every 24 hours. Although this replication interval is very short, there might be opportunities of spreading the work during the day and performing the replication in an on-line fashion. As mentioned above, due to high cost of replication, frequently replicating data can impose many temporary episodes of high latency, especially for reads, thereby diminishing the effectiveness of our mechanism. However, later in this paper, we will study the on-line replication mechanism and compare it with the end-of-the-day replication.

D. How Many and Where to Replicate?

Due to limited space for replicated data, the read performance efficiency of the proposed mechanism heavily depends on *(i) the number of copies kept for each data*, and *(ii) the place (or chip) that each copy is mapped to*. We describe the impact of each of these factors, separately.

- *Number of copies:* Assuming a fixed space for replication, one can take three actions: (i) keeping lots of copies of a few very popular data, (ii) keeping a few copies of a lot of data, and (iii) following VP proportionally (i.e., the more popular, the more copies). The two former decisions are static and may not be effective since different workloads and different phases of one workload exhibit different behaviors. The third decision, on the other hand, is efficient to capture a workload's dynamism and use the collected information to determine the replication degree of the each value.
- *Placement of copies:* The mapping decisions made for data copies at the end of each day determines the chip-level loads in the following day. One may use static or random mapping strategies, but as our experimental results also confirm (Section IV), such strategies usually are not able to capture most of the performance benefits resulting from Ideal-All-Chip Replication (Figure 3) because they are generally load-unaware. On the other hand, load-aware mapping is challenging since the VP information is the only available information we have at the end of a day. Thus, we need richer profiling

information (such as popularity counts and the amount of available space on each chip) to use for mapping.

To address the above issues, we introduce a novel and low-overhead mechanism which collects extra information during the normal execution to help find the number of copies and their placement. When replicating each data on single/multiple chip(s), we attempt to evenly distribute the replicas/load on different flash chips in order not to overwhelm some flash chips while some others experience much lower loads. To this end, we develop our replica placement algorithm considering the following parameters for each flash chip:

Available Space (*S*): One of the main parameters based on which the destination chip for a replica is determined is the available space on that chip. By keeping track of the number of occupied pages on each flash chip, we can select the one with the least number of occupied pages to place a replica.

Erase Counts (*E*): Updating a page in NAND flash SSD requires an *erase* operation beforehand which takes much longer time than read and write operations. Thus, SSDs perform out-of-place-update to overcome this issue. This property necessitates FTL to periodically run Garbage Collection (GC), in order to free up space by reading out the valid data, and a long latency erase operation. Apart from performance issues of GC, each NAND flash cell can bear only a limited number of erases before it wears out. We take the number of erases occurred in each flash chip into account when making replica placement decisions, thereby attempting to evenly distribute the wear on different flash chips.

Popularity Count (*Pop*): As already discussed, the existing value popularity results in more accesses to popular values while non-popular values are not accessed very frequently. If we blindly select the target chip for replication, without considering popularity degrees, there can be situation where a chip stores a larger number of popular values and gets more congested as a consequence. Thus, we need to take the popularity degree of values already stored on a flash chip into account when determining the target flash chip(s) for replication. By storing the popularity information of different flash chips, we attempt to evenly distribute the popular values on different flash chips in order not to swamp a flash chip with majority of popular values being stored on it.

We take these three parameters into account when making data placement decisions for selective replication: Available Space, S, Mean Erase Count, E, and Mean Popularity Count, Pop. We select a chip to place the replica for which the $\frac{S}{E \times Pop}$ is higher. To determine the values of S, E, and Pop, we maintain several counters in the device controller, having the FTL provide such information. Note than, when studying on-line replication later in this paper, we will add another parameter which takes the instant chip-level loads when placing replicas into account.

Moreover, we maintain information about popularity of each unique value and which flash chips it has already been stored on. The details of our implementation is described in the next section. With this information, at the end of the day, one can figure out where the popular values have to be placed based on their read requests for those data values during the day. Such statistics collection will take significantly lower times than the latencies of actual read/write operations to the flash chips. Note the replication can be spread out during the day and execute more frequently on shorter intervals (e.g., on hourly basis). However, as explained before, doing replication more frequently does not reduce relative replication costs (due to linear behaviour of replication costs), neither helps with the performance improvements as our results show, rather results in more frequent SSD slowdown due to replication.

III. Implementation

This section describes how a flash-based Content-Addressable SSD works and then provides the details of the modifications to implement our *Read-Redirected SSD* (RR-SSD) along with details of the changes required in the FTL.

A. Content-Addressable SSD: The Baseline

A traditional NAND flash-based SSD has four key components: *an array of NAND flash chips*, *an embedded core*, *a DRAM cache*, and *an interconnection network between flash chips and the core*. Each flash chip comprises thousands of *pages*; a page is the smallest unit of read and write operations. Writing into a page requires us to *erase* it, which is performed at the granularity of a *block*. There is an asymmetry in latencies of flash operations, with write being much slower than reads, and erase being significantly slower than both reads and writes. The erase-before-write property necessitates out-of-place updates in order to prevent the high latency of erase deteriorating the write performance. The embedded core runs the *Flash Translation Layer (FTL)* software, which helps emulating SSD as a block-level device similar to hard disk drives. The FTL has three main components: (i) a Mapping Unit that performs page-level mapping and stores LPN-to-PPN address mapping table in the on-SSD RAM; (ii) the Garbage Collection (GC); and (iii) the Wear Leveling (WL). Beneath FTL, there is a software module called *Flash Interface Logic (FIL)* with two main responsibilities: (i) it resolves resource contention and schedules the requests, and (ii) it behaves as a mediator between the FTL and the flash chips by issuing memory transactions, while obeying timing of the flash chips and channels.

CA-SSD typically requires additional components (compared to a traditional SSD) for implementation [10], [11]. We use a recent CA-SSD design proposed by Chen el al. [10] as the baseline in this paper. They refer to FTL in CA-SSD as *CA-FTL* which needs three key enhancements to a traditional SSD to achieve the CA-SSD functionality. *First*, CA-FTL employs a dedicated on-board processor (called *Hashing*

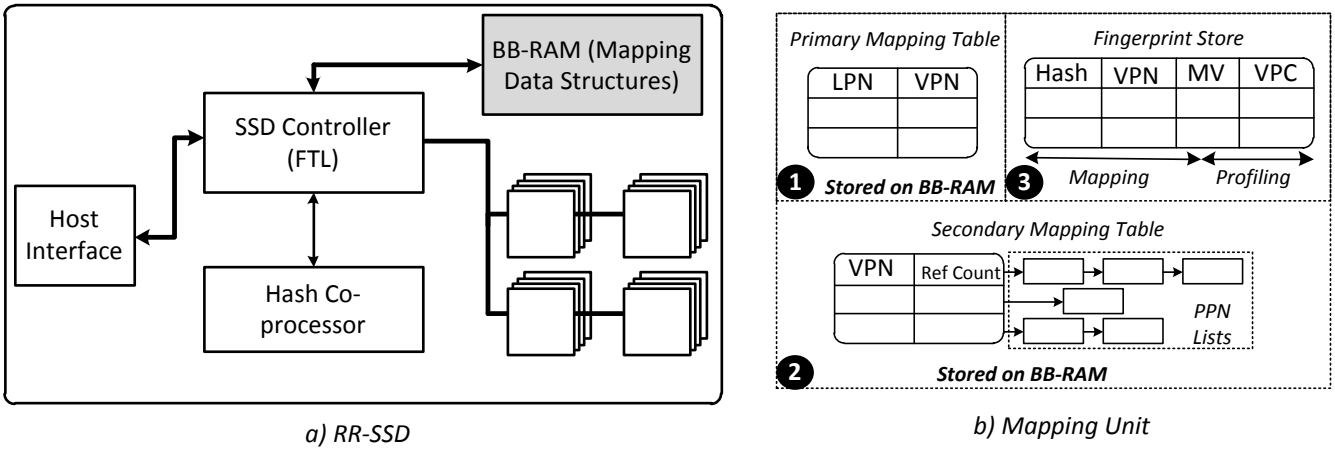


Figure 8: (a) Components of the proposed RR-SSD. (b) Tables used in the Mapping Unit.

Unit) to compute/compare hashes very fast, which is proposed to use the cryptographic processor found in modern SSD devices (e.g., SSD products by Intel [13] and Samsung [14] implement real-time AES encryption in hardware). *Second*, the Mapping Unit should maintain additional data structures for CA-SSD (called CA-FTL's *meta-data*) stored in the on-SSD RAM. The GC also needs to be changed, since compared to conventional SSDs (where each update results in page invalidation requiring an eventual erase operation), CA-FTL only needs to invalidate a page when no LPN points to a value in that page. *Third*, since the per-page meta-data information in this scheme is large, CA-FTL employs a *Battery-Backed RAM* (BB-RAM) [11], [15] to guarantee loss-less write-back of the CA-FTL's meta-data in flash chips on power failure. Next, we describe the required data structures in FTL of the proposed Read-Redirected SSD (RR-SSD) for supporting read redirection.

B. SSD Enhancements for RR-SSD

In Figure 8.a, we present a high-level view of the the proposed RR-SSD which requires two key enhancements to the baseline CA-SSD. These components are shown in gray in this figure and described below.

1) Mapping Unit for RR-SSD's FTL: The one-to-one mapping (from LPN to PPN) in conventional SSD is not applicable to our design as in our design, a PPN can be pointed to by many other LPNs thus, we require a many-to-one mapping structure to begin with – else the relocation and updates due to GC mandate many updates to the Mapping Unit as each page movement will need updates to all LPNs. Mapping Unit must have additional data structures for RR-SSD to maintain the relationship among hashes, LPNs and PPNs. Here, we assume a page-level address mapping. Figure 8.b shows the data structures employed by the Mapping Unit for RR-SSD that is composed of three tables. Despite traditional FTL which maintains a mapping table for mapping each LPN to the corresponding PPN, similar to [10], we employ two tables, a *primary mapping table* and a *secondary mapping table*, along with an auxiliary table, called *fingerprint store*, to maintain the hash of the contents (data chunks) and the mapping information required. Below we scrutinize each table. Circled numbers in the text below refer to table numbers in Figure 8.b.

1) To avoid storing too many redundant mapping information and also minimize the searches in the Mapping Unit, the primary mapping table ❶ maps each LPN to a Virtual Page Number (VPN). A VPN is a virtual address assigned to a set of LPNs mapped to the same PPN. This table is also used in CA-SSD, hence no modification is required to employ it in our design.
2) In CA-SSD, the secondary mapping table ❷ maps a VPN to a PPN, with each entry of this table being indexed by the VPN. Thus, a mapping from each LPN to its respective PPN is done in an indirect fashion: first, from LPN to VPN, then from VPN to PPN. In RR-SSD, however, a data can be replicated and reside in multiple physical pages, rather than only one page, thereby, requiring to maintain a list of PPNs for each entry of this table. Lastly, like CA-SSD, we store the reference count for each VPN entry, denoting the number of LPNs mapped to a VPN. This reference count is used as a means to identify the VPNs which are not valid any more, hence, their associated PPNs can be reclaimed by GC.
3) In order to find the information required for de-duplication and replication, we maintain a fingerprint store ❸ which maps the hashed value to a VPN. This table has also been used in CA-SSD. However, to find replicas of a hashed value and also collect profiling data, we add the following items to each entry of the fingerprint store: (i) an N-bit vector (called *Mapping Vector (MV)*), with N referring to the number of flash chips, which keeps track of the list of flash chips that have same

content corresponding to each unique hash of value (i.e., a replica), (ii) and a 6-bit saturating counter (VPC). The last attribute is used for the profiling logic which is later discussed in this section. Entries are inserted in the fingerprint store upon writing a new data to SSD, which is mapped to one of the flash chips and the corresponding bit in MV vector is set to '1'. The MV attribute gets updated when data is either replicated or invalidated (setting it to '1' or '0').

Storing Mapping Unit in RR-SSD: The primary and secondary mapping tables are maintained in the BB-RAM so that on a power failure, the critical mapping information does not get lost. Moreover, the fingerprint store is maintained on the RAM space in the SSD and in case of power failure, a capacitor (such as SuperCap [16]) will provide enough current to flush the mapping information to the persistent storage. The space required to store the aforementioned tables exceeds the available RAM/BB-RAM space in SSDs. However, observing significant temporal locality and temporal value locality, similar to [11] and [5], we employ a simple LRU caching mechanism to selectively store a subset of these tables with the most frequently accessed entries. As discussed by Gupta et al. [11], the miss rate of this cache is around 5% and does not impact the performance benefits. We set the size of this cache (after an extensive sensitivity analysis which is skipped in the interest of space) to accommodate this Mapping Unit as 256MB which is simply affordable in today's SSDs with GBs of RAM space. To provide the consistency of tables and not miss any profiling information, we propose to periodically checkpoint the Mapping Unit tables in flash chips. Thus, the in-memory tables are periodically synced into the flash.

Handling Write Requests: On receiving a write request, the hash of the value for each LPN comprising the request is calculated and the fingerprint store ❸ is then looked up with this hash. On a miss in this table, a new page is allocated in one of the chips to store the new data and a write operation is issued. Meantime, a new entry is allocated in each of the mapping tables – it stores (LPN, VPN) in primary table ❶, (VPN, PPN) in secondary table ❷, and (hash, MV vector) in the fingerprint store ❸. On a hit, on the other hand, after updating MV in fingerprint store ❸ and (LPN, VPN) in the primary mapping table ❶, SSD returns a write request without requiring flash chip writes, indicating that its content already exists in SSD.

Handling Read Requests: On a read, the primary mapping table ❶ is first looked up to get the VPN of data that is going to be read. Then this VPN is used to determine the PPNs storing the replicas for this VPN[4]. The FTL chooses the chip with the lowest load at that time for servicing the read request and the read operation is then issued.

Finally, note that we do not modify GC or WL policies in this work and assume that RR-SSD continues to employ the GC and WL policies used in the CA-SSD with the only difference that, once a VPN is invalidated, all the respective PPNs are marked invalid, waiting to be reclaimed by GC.

2) Profiling Logic: The profiling logic (for what to replicate and where to replicate) has two parts (Figure 8). *First*, in order to reduce the space needed for profiling tables, we use the fingerprint store in Mapping Unit to keep the profiling information. Each entry of this table stores VPC and MV, as already explained. *Second*, we collect profiling information for each chip as discussed in the previous section. For each chip we maintain the information about the available space, erase counts and popularity counts. To this end, we employ several counters and increase or decrease them with each read, write, and erase event.

Normal Execution and Profiling: At the beginning of profiling, all VPC counters are zero. On reading a value, after finding its hash entry in the fingerprint store, its VPC counter increments (and saturated at 63). On inserting a new entry in the fingerprint store by a write request, the VPC counters have initial values of zero. Also, these counters are not subsequently touched by any writes.

Replication: Data centers usually employ redundancy techniques such as mirroring to achieve high levels of reliability [17], [18]. Some prior work [19], [20] have also burrowed this assumption to accomplish their maintenance time in the SSD. Thus, we assume having a Mirror SSD (probably cheaper and with lower performance) which is a conventional SSD that works in parallel with RR-SSD, receiving the same read/write requests (maintaining the same content). While RR-SSD is busy at the end of the day (for maximum of 8 minutes) performing the data replication, the mirror SSD keeps servicing the incoming requests before the RR-SSD comes back online.

The replication process involves *counter sorting*, *actual data copying*, and *updating tables in the Mapping Unit*. Since counter sorting is costly, we traverse the fingerprint table several times, and during each pass, we decide whether or not to replicate each data. More precisely, in the first pass, we select the hash values with VPC of 63, if there is any, and replicate them in the chip(s) suggested by our data placement metric. This requires (1) reading the actual data from one of the previously-copied places, and (2) updating the MV in the fingerprint store table, to represent where data should be copied in order to be used in the next day for read redirection. In the next passes, we repeat the same process, this time for hashes with lower VPCs. The replication process terminates when we reach the space limit for replication. Note that during replication, we may need to invalidate one (or more) copies of a data, if they are no longer popular for the next day. To reflect this, FTL has to update all the tables in Mapping Unit. However, this process not need being done every day, and as prior work [21] suggests, this adjustment process can be done with less frequency than the replication.

[4]Only first few bits of each PPN can be examined to determine the flash chip that PPN belongs to.

Table II: Main characteristics of simulated SSD.

Evaluated SSD Configuration
8 Channels and 8 Flash Chips per Channel, Channel Width = 8 bits, Chip Interface Speed = 533 MT/s (ONFI 4.0), Hashing Latency = 12us [22]
NAND flash
Page Size = 4KB, Metadata Size = 448B, Block Size = 256 pages, Planes per Die = 2, Dies per Chip = 4, Flash Chip Capacity = 16GB, Read Latency = 75μs, Typical Program Latency = 400μs, Erase Latency = 3.8ms

For instance, we can do it on a weekly basis to reduce the overheads associated with it.

IV. Evaluation

This section describes our evaluation methodology and the experimental results using a diverse set of six workloads.

A. Methodology

Evaluation Platform: We use SSDSim [2] for simulating both a CA-SSD [10] and our RR-SSD. SSDsim is a trace-driven simulator which has a detailed implementation of FTL algorithms and request schedulers. We have modified SSDSim to model the CA-SSD (which is used as "baseline" of our design) and augmented it with additional components and functionalities required by our RR-SSD. We assume that the overhead of hash calculation is 12us [22] and modeled its impact on the queuing latency of the incoming read and write requests.

Studied Configurations: The baseline SSD consists of eight channels, each of which is connected to eight NAND flash chips. Each channel works on ONFi 4.0 [23]. Table II provides specifications of the modeled SSD (which is very similar to [24]) along with parameters of the baseline configuration. Also we equip our model with NVMe [25] standard interface at HIL. We compare the performances of three systems:

1) ***RR-SSD*** employs the proposed data replication and read redirection schemes. RR-SSD reserves a space (overhead) of $\alpha\%$ of the workload's size (original non-deduplicated dataset) for replicated data. The α parameter is specified by the user or system administrator).
2) ***CA-SSD*** is a content addressable SSD based on [10].
3) ***Oracle*** uses *Ideal-All-Chip Replication* and redirects all reads to the least-loaded chip at any moment. This system in a sense represents the maximum achievable performance gain by read redirection without considering associated replication costs.

For each of the evaluated systems, we report the amount of reduction in read response time as well as total (read+write) response time as our performance metrics.

Workloads: We use a set of six disk traces [10], [12] which contain the values read or written for each request. These traces have been extensively used by previous related studies [10]–[12]. They are collected from different kinds of big data applications and servers (including mail, web, database servers) and systems (an experimental system and an office system) at FIU and OSU universities. Individual requests in these workloads are of size 4KB, along with hash (SHA-1 or MD5) of the contents. Table III summarizes the main characteristics of the disk traces. This table also reports the read and write performance improvements achieved by CA-SSD compared to conventional SSD (with same internal organization). We see that although the primary goal of CA-SSD is to improve write performance, it also results in some read response time improvement, which is mainly due to huge reduction in write traffic (queue lengths for reads also goes down). We use CA-SSD as our baseline in this paper and *normalize* the results of both RR-SSD and the Oracle system to that of CA-SSD.

Table III: Workload Characteristics. Unique values represent the percentage of read (write) requests which read (write) unique 4KB chunks. Improvement by CA-SSD denotes the percentage of reduction in read (write) response time by [10] compared to the conventional SSDs.

Trace Name	WR %	Unique Value [%]		Imp. by CA-SSD [%]	
		WR	RD	WR	RD
webvm	77	42	32	75	49
homes	96	66	80	89	71
mail	77	8	80	61	53
hadoop	30	63.9	17.5	150	100
trans	55	77.4	13.8	100	82
desktop	42	74.7	49.7	200	300

B. Experimental Results

In the next three subsections, we analyze the performance of RR-SSD. We first present the results for a system with no space overhead for replication, i.e., ISO-capacity analysis compared to original non-deduplicated dataset. We then analyze the performance and overhead trade-off as we increase the replication space overhead. Then, we study the need for a sophisticated way of figuring out where to create the replicas compared to more naive strategies (such as random and/or static approaches). Lastly, we discuss the on-line replication and compare our approach with prior work.

1) ISO-Capacity Performance Analysis: RR-SSD is targeted to improve the read request service times with low overhead in terms of performance and storage. Figure 9 compares the read service time of different systems in an ISO-capacity design (space overhead is 0% of the non-deduplicated dataset). The results are given for 6 consecutive days of each workload starting from the second day[5]. The upper chart for each workload shows the read service time reduction for RR-SSD and Oracle system (shown by a solid line), compared to the CA-SSD.

[5]As replication starts at the end of the first day, the performance results for RR-SSD are the same as CA-SSD in the first day.

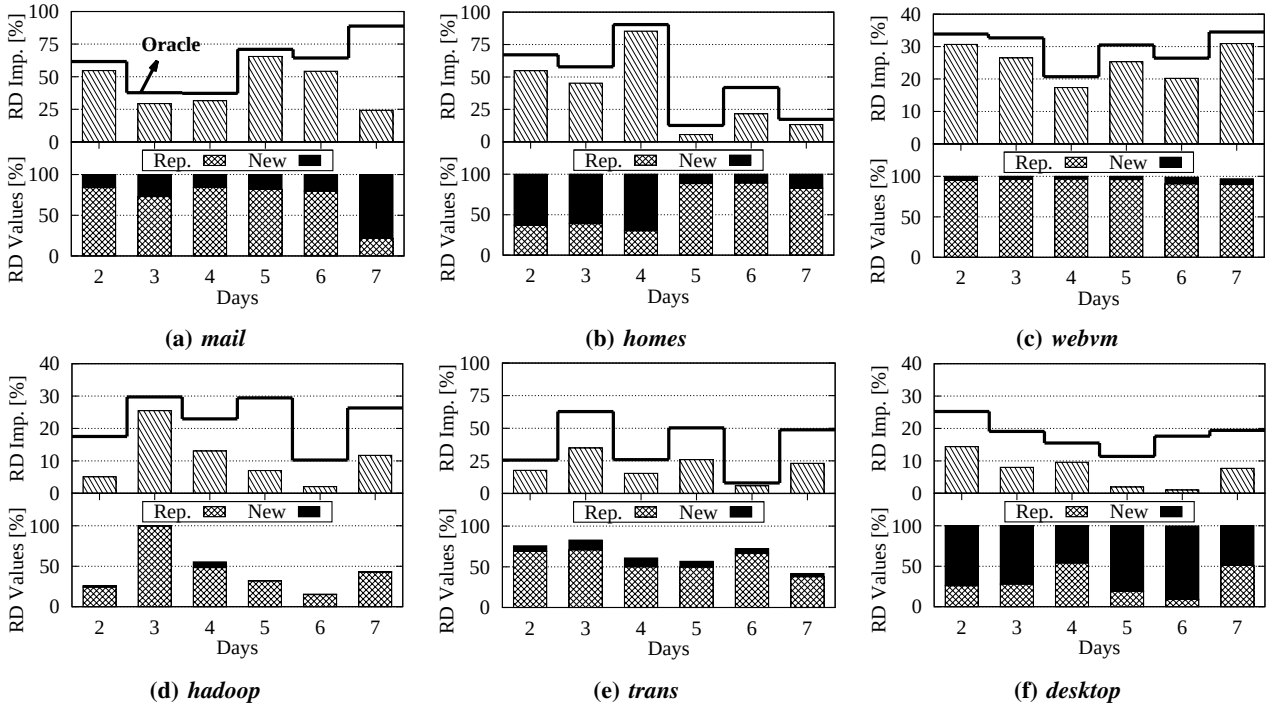


Figure 9: Read performance improvement brought by RR-SSD and Oracle system compared to CA-SSD. The solid line in the top graph of each workload is the improvement achieved by Oracle. The bottom graph of each workload plots the fraction of read requests that had opportunity to read the replicated data from previous day (Rep) and data that was newly read (New) for the day under study.

We observe different behaviors for different workloads/days. For three workloads (`mail`, `homes` and `webvm`) which are less read-intensive, RR-SSD's read improvement is very close to that of the Oracle system for most of the time – it is around 90% of the maximum improvement for all days of these workloads, except for one day of `homes` and `mail`. RR-SSD is able to improve read response time by 24%–65% for `mail`, 5%–80% for `homes`, and 17%–32% for `webvm`. For the other workloads (`hadoop`, `trans` and `desktop`) which are either highly read-intensive or have balanced number of reads and writes, RR-SSD is able to improve the read service time by up to 25% for `hadoop`, 35% for `trans`, and 14% for `desktop`. However, the gap between RR-SSD and the Oracle system is variable and significant in some days for these three.

To better explain the performance gap between RR-SSD and the Oracle system, Figure 9 (lower chart of each workload) also shows the workload's dynamism during each day by reporting (i) the percentage of values read for the first time (beginning from the first day) – called *New*, and (ii) the percentage of read values in each day that have been replicated from the prior day – called *Rep*. As expected, if we see high percentage of Replicated values and low percentage of New values during a day, this usually (except for `homes`) corresponds to a lower gap between RR-SSD and the Oracle system performance. In fact, we observe three behaviors. *First*, in case of `webvm` with very high contribution of Replicated values, FTL has a reasonable chance to redirect reads to a lower-loaded chip, which results in reducing response time of many read requests. Similarly, the huge gap between RR-SSD and Oracle in `desktop` is because the fraction of New values introduced every day is very high. *Second*, in case of `hadoop` and `trans`, although the percentage of New values is small, there is still a considerable gap between the two systems. The reason is that, as Table III indicates, the amount of space we save in these two workloads and use for replication (at the end of day) and redirection (in the next day) is small (i.e., 17.5% for `hadoop` and 14% for `trans`), resulting in low contribution of Replicated values in the next day. *Third*, in case of the other three workloads (`mail`, `homes`, `desktop`), the small performance gap between RR-SSD and Oracle system is due to both *huge space saved by de-duplication* and *low fraction of New values* (except for the first few days of `homes` execution).

Finally, Table IV reports read and total (read+write) performance improvement given by RR-SSD and the Oracle compared to the CA-SSD.

2) Impact of Replication Space Overhead on Read Performance: The RR-SSD's performance efficiency largely depends on the amount of space used for replicated data. This is specially important for workloads like `hadoop` and

Table IV: Read and total (read+write) service time improvement of RR-SSD and Oracle system over the baseline CA-SSD for entire duration of each workload.

Trace Name	RR-SSD's Imp. [%]		Oracle's Imp. [%]	
	RD	Total	RD	Total
webvm	23.43	15.71	26.60	19.07
homes	35.11	16.89	45.21	17.71
mail	45.37	40.26	57.61	48.05
hadoop	11.70	9.55	22.73	19.38
trans	19.97	11.01	37.23	21.59
desktop	7.83	5.11	17.05	11.95

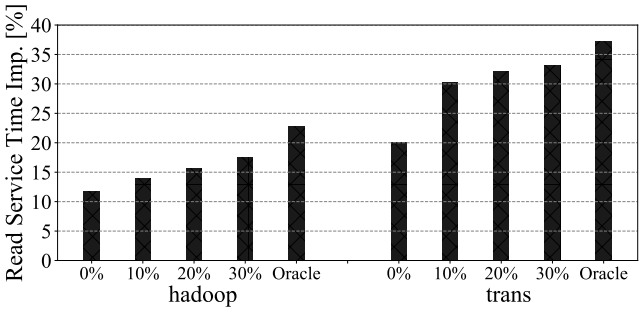


Figure 10: Read service time improvement as a function of allowed space overhead for replicating data. Results of the Oracle system is shown for reference.

`trans` where we do not save significant space with de-duplication. Figure 10 shows the sensitivity of RR-SSD's read performance improvement to the replication space overhead. We only show the results for `hadoop` and `trans`, since the other workloads generally save large space with de-duplication and already achieve near-ideal (Oracle) read improvement. We can see that, as a larger replication space is used, a higher read performance improvement is achieved (around 25% and 33% read improvement for `hadoop` and `trans`, respectively), but it saturates after a point (i.e., around 20%). Consequently, by employing RR-SSD with almost 20% space overhead for replication, one can achieve a performance improvement which is quite close to that of Oracle (last column for each application in Figure 10).

3) Efficiency of Mapping Strategy in RR-SSD: Instead of our proposed replica placement mechanism in RR-SSD, one may employ simpler strategies for mapping the replicated data. Figure 11 compares our RR-SSD mechanisms with two other naive strategies: *Random* and *Static*. In the Random scheme, data copies are randomly allocated over all flash chips. In the Static scheme, on the other hand, data copies are located on flash chips in a round robin fashion, starting from the chip that the original data was mapped to, i.e., if it was originally in chip i, and j copies have to be created, replicas are made at chips $(i+1)\%N, \ldots, (i+j)\%N$, with N representing the number of chips. Note that the number of copies for each data is equal in all these three schemes, and is determined by the same mechanism that we used

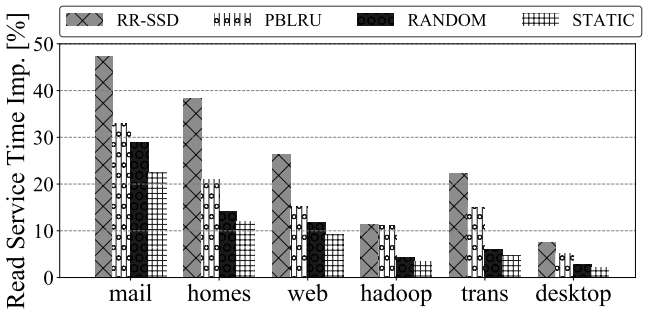


Figure 11: Read service time improvement with different mapping strategies (normalized to the CA-SSD).

in our approach (only chip-level mappings are different). The results plotted in Figure 11 are averages over all read requests for the entire workload's run. We can see that our history-based mapping (shown as RR-SSD in the figure) significantly increases the chance of finding a low-loaded chip when servicing each read request. More accurately, the read improvement achieved by employing RR-SSD mapping is generally twice that for the random and static mappings.

We also show the improvements gained by prior work [9] (**PBLRU** in this Figure 11) for comparison. Note that, prior work does not consider the content-addressable design and cannot exploit the content popularity to selective replication. They however, consider only the temporal locality among the physical pages and employ a priority-based least recently used (PBLRU) mechanism to dynamically replicate the frequently accessed physical pages. To conduct a fair comparison, we implemented this approach in CA-SSD – else the significant write reduction due to deduplication would blur the benefits of PBLRU. Also, like Random and Static, we fix the space available for replication to be same in all experiments. As shown in the figure, although PBLRU outperforms Random and Static replica placement and yields superior benefits in terms of read service time improvement, it falls behind RR-SSD as it is not able to capture the content popularity of data and use it towards a more efficient replication. More specifically, two main reasons for our approach to outperform PBLRU are: (i) replicating a set of very popular values leads to more efficient use of the allocated space for replication, rather than only considering the temporal locality in the accesses to each physical page, and (ii) replicating values with higher popularity on more number of flash chips can help provide re-direction opportunity for larger number of read requests as the popular values account for a larger number of accesses.

4) On-line Replication: As mentioned in section II-C, one may attempt to spread the replication during the day. We conduct an experiment in which replication is done on-line instead instead of doing it at the end of the day. Note that in order to determine which flash chip to place the replica, we slightly need to modify our replica selection algorithm

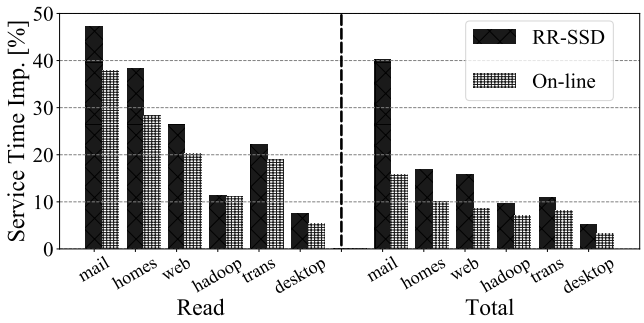


Figure 12: Read/Total service time improvement for end-of-the-day replication and on-line replication.

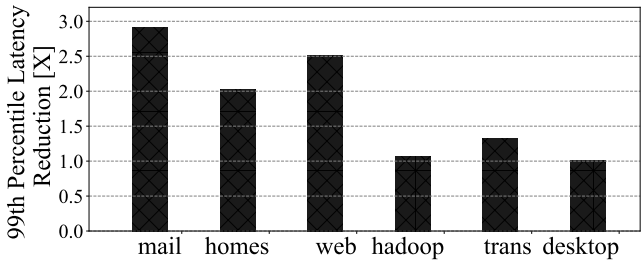


Figure 13: Reduction in tail latency.

to take the instant load of each flash chip into account. We define the parameter QL to represent the chip-level load and select the flash chip for which $\frac{S}{E \times Pop \times QL}$ is higher. Also, to minimize the performance degradation due to replication, we can exploit the idle time of flash chips as discussed in the prior work [9]. We report the read and total service time improvements in Figure 12. As illustrated in this figure, on-line replication also improves both read and total service time with average of 20.4% and 8.9% improvements, respectively. However, due to high cost of replication, spreading the work during the day may impose some temporary episodes of high latency. This is evident in the total service time improvements, which is much less than read service time improvement in this case. This is mainly due to slowing down writes to perform data copies, thus diminishing the effectiveness of read re-direction.

5) Tail Latency Analysis: The importance of minimizing the tail has drawn a lot of recent attention from an SSD latency viewpoint [20], [26]–[28] since storage is a dominant component of many client facing applications. RR-SSD optimizations, in addition to improving the mean latency, result in a reduction in the tail latency. We report the reduction in tail latency ($99^{th} percentile$) for all requests, in Figure 13. Our proposed mechanism, results in around $2X$ improvement in the tail latency of the studied workloads, where the majority of benefits is achieved by enabling read-redirection to provide flexibility for a read request to circumvent a highly loaded flash chip.

Table V: Ratio of Replication Writes to Total Writes

Trace	web	mail	homes	hadoop	trans	desktop
Ratio	9.27%	12.39%	11.41%	12.16%	9.87%	6.19%

6) Overhead Analysis: Although our proposed RR-SSD design aims at reducing the overheads associated with the Ideal-All-Chip Replication, it still incurs some overheads in terms of performance, lifetime and on-SSD RAM space. However, these overheads are not significant, considering current SSD architectures. More specifically, the overheads of our RR-SSD design can be classified into three categories:

- **Memory Overheads**. The memory space required to store the mapping data structures, as discussed in Section III, is similar to that in CA-SSD. The only additional information are those required for the profiling logic, which add only several bits (with regards to the number of chips) to each entry already storing a 20B hashed value. On the other hand, with increasing the number of flash chips, the amount of RAM space on SSD will also scale, providing sufficient space to accommodate the mapping tables of RR-SSD.
- **Replication Overheads**. Replicating a set of very popular values helps reducing the overheads associated with replication while providing high performance benefits. We report the ratio of replication writes to all writes in table V. As reported in this table, our proposed mechanism incurs around 10% more writes compared to CA-SSD, on average. However, compared to a conventional SSD which does not employ content deduplication, we do not incur extra writes as we use up the write traffic saved by deduplicatoin.
- **Table Look-up and Updates**. Modern SSDs have multiple cores and accommodate complicated scheduling and mapping mechanisms. Equipping SSDs with such strong computation power (e.g., with clock speeds over 2GHz), will limit the table look-ups in our design to few micro seconds, which is negligible compared to the actual read and program latencies.

V. Related Work

Resource contention problems in modern NAND flash SSDs has been a hurdle towards boosting their performance. A variety of solutions have been proposed to tackle this problem and provide better system-level performance. Previous solutions attack this issue from four angles:

Replication and Read Re-direction: Prior work have addressed replication either as a means to improve reliability [21] or to boost performance [8], [9]. The former discusses object request popularity in distributed storage systems equipped with HDDs, and attempts to customize replication degree to enhance the system availability. These optimizations are not directly applicable to a system with SSDs. PBLRU [9] as discussed in preliminary section, does

not consider the value popularity when making replication decisions, resulting in lower improvements compared to our proposed RR-SSD mechanisms. Moreover, [8] operates on an array of SSDs and proposes inter-SSD replication. This study does not take the internal architecture of SSD and chip-level parallelism into account. Also their replication decision is not based on value popularity. [29] on the other hand, explores the opportunity of re-directing read requests by exploiting the existing redundancy in the SSD (provided for the reliability purposes). This work is completely orthogonal to our proposed approach and can be combined with our proposal to provide higher improvements.

Scheduling Efforts: New schedulers order incoming requests to take advantage of high storage parallelism and at the same time resolve the resource contention problem. A number of these schedulers are implemented at the host side (such as [30]–[32]) that sends the requests to the SSD. Others [33]–[39] operate inside the SSD and distribute requests across various internal resources (channels, chips, etc). Jung et al [37] proposed a QoS-aware and GC-aware scheduler, considering the low-level contention in SSD, which strives to reduce the resource contention by redistributing the overheads of GC across non-critical I/O requests. In comparison, to eliminate the contention problem for read requests, [40], [41] proposed SSD caches and pre-fetching mechanisms.

Dynamic Resource Allocation: Some prior works proposed dynamism for freely allocating internal resources regardless of request address to help write requests get faster service. Among these, load-aware scheduling techniques, [2], [6], [7], [42] use dynamic write mapping and write order base mapping, respectively, to re-direct write requests to the resources for which fewer requests are waiting to get serviced. It is important to note that these techniques only improve the write requests' service and in some cases, as prior works show [7], may even hinder service of reads. Despite write redirection being straightforward to implement due to out-of-place updates in NAND flash, read redirection is not feasible in the current SSD setting. Our work in this paper presents the first solution to facilitate read redirection.

Reducing Write Traffic: Prior studies concentrate mostly on reducing the write traffic to enhance lifetime and performance. Apart from studies on spatial- and temporal-locality [43]–[46], some previous papers [10], [11] provide several methodologies to help reduce write traffic to the SSD by exploiting *value locality* and *data de-duplication* [12], [47]–[49]. In the context of SSDs, existing works propose several enhancements, primarily to the FTL, to avoid redundant writes on the SSD. Gupta et al [11], study several workload characteristics from the value popularity point of view. They attempt to remove the redundant writes by augmenting the FTL with required mapping tables. CAFTL [10], on the other hand, focuses on employing different sampling and mapping techniques to reduce overheads associated with mapping tables and the hashing mechanism. We want to emphasize that, all these techniques, have focused on optimization for writes, while reads have not received such attention. Thus, we fill this void by proposing a read-redirection mechanism through which reads can enjoy flexibility of selecting the target flash chip to read from.

VI. Conclusions

Modern SSDs suffer from resource contentions when servicing the I/O requests. A straightforward way to address this problem is to evenly distribute the incoming traffic across all chips, making them load-balanced. This needs to redirect both reads and writes. While implementing write redirection is easy (due to no-write-in-place property of flash memories), read redirection is not feasible in conventional SSDs. In this paper, we show that, although Ideal-All-Chip Replication (i.e., replicating each data everywhere) gives an impressive performance improvement, it is very costly in terms of performance, lifetime and storage space. Alternatively, this paper proposes to use "content popularity-based selective replication" to reduce the costs associated with replication, while still providing read redirection for a majority of read requests. The proposed design is called *Read-Replicated SSD (RR-SSD)*, which tries to answer five complementary questions: (1) *which data to replicate?*, (2) *how much space for replication?*, (3) *where to replicate?*, (4) *when to replicate?*, (5) *how to replicate?* Experimental evaluations show that our RR-SSD design improves read service time by up to 45%, with 23.9% improvement on average (by up to 40% improvement across both read and write requests, with an average improvement of 16%).

Acknowledgements

This research has been funded in part by NSF grants 1629129, 1439021, 1302225, 1714389, 1302557, 1526750, 1763681, 1439057, 1409095, 1626251 and 1629915, a DARPA/SRC JUMP grant, and a grant from Intel.

References

[1] Q. Xu, H. Siyamwala, M. Ghosh, M. Awasthi, T. Suri, Z. Guz, A. Shayesteh, and V. Balakrishnan, "Performance characterization of hyperscale applicationson on nvme ssds," in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '15, (New York, NY, USA), pp. 473–474, ACM, 2015.

[2] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," in *International Conference on Supercomputing*, pp. 96–107, 2011.

[3] M. Jung and M. T. Kandemir, "An evaluation of different page allocation strategies on high-speed SSDs," in *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2012.

[4] J. K. Kim, H. G. Lee, S. Choi, and K. I. Bahng, "A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems," in *International Conference on Embedded Software (EMSOFT)*, pp. 31–40, 2008.

[5] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 229–240, 2009.

[6] C. Park, E. Seo, J.-Y. Shin, S. Maeng, and J. Lee, "Exploiting internal parallelism of flash-based SSDs," *IEEE Computer Architecture Letters*, vol. 9, pp. 9–12, Jan 2010.

[7] A. Tavakkol, M. Arjomand, and H. Sarbazi-Azad, "Unleashing the potentials of dynamism for page allocation strategies in SSDs," in *ACM International Conference on Measurement and Modeling of Computer Systems*, pp. 551–552, 2014.

[8] Y. Du, Y. Zhang, and N. Xiao, "R-dedup: Content aware redundancy management for ssd-based raid systems," in *2014 43rd International Conference on Parallel Processing*, pp. 111–120, Sept 2014.

[9] B. He, J. X. Yu, and A. C. Zhou, "Improving update-intensive workloads on flash disks through exploiting multi-chip parallelism," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, pp. 152–162, Jan 2015.

[10] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *USENIX Conference on File and Storage Technologies (FAST)*, pp. 77–90, 2011.

[11] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging value locality in optimizing NAND flash-based ssds," in *USENIX Conference on File and Storage Technologies (FAST)*, pp. 91–103, 2011.

[12] R. Koller and R. Rangaswami, "I/o deduplication: Utilizing content similarity to improve i/o performance," *Trans. Storage*, vol. 6, pp. 13:1–13:26, Sept. 2010.

[13] O. Bar-El and S. Rand, "Deploying Intel Solid-State Drives with managed hardware-based encryption," 2015.

[14] B. Bosen, "Full drive encryption with Samsung Solid State Drives." http://www.samsung.com/global/business/semiconductor/file/product/ssd/SamsungSSD_Encryption_Benchmarks_201011.pdf, 2010.

[15] R. Kateja, A. Badam, S. Govindan, B. Sharma, and G. Ganger, "Viyojit: Decoupling battery and dram capacities for battery-backed dram," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, (New York, NY, USA), pp. 613–626, ACM, 2017.

[16] HGST, "Ssds with super capacitors." https://www.hgst.com/sites/default/files/resources/[FAQ]_ZeusRAM_FQ008-EN-US.pdf.

[17] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "Raid: High-performance, reliable secondary storage," *ACM Comput. Surv.*, vol. 26, pp. 145–185, June 1994.

[18] Q. Xin, E. L. Miller, T. Schwarz, D. D. E. Long, S. A. Brandt, and W. Litwin, "Reliability mechanisms for very large storage systems," in *Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)*, MSS '03, (Washington, DC, USA), pp. 146–, IEEE Computer Society, 2003.

[19] D. Skourtis, D. Achlioptas, N. Watkins, C. Maltzahn, and S. Brandt, "Flash on rails: Consistent flash performance through redundancy," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, (Philadelphia, PA), pp. 463–474, USENIX Association, 2014.

[20] J. Huang, A. Badam, L. Caulfield, S. Nath, S. Sengupta, B. Sharma, and M. K. Qureshi, "Flashblox: Achieving both performance isolation and uniform lifetime for virtualized ssds," in *15th USENIX Conference on File and Storage Technologies (FAST 17)*, (Santa Clara, CA), pp. 375–390, USENIX Association, 2017.

[21] M. Zhong, K. Shen, and J. Seiferas, "Replication degree customization for high availability," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, (New York, NY, USA), pp. 55–68, ACM, 2008.

[22] "SHA-1, SHA-2 and MD5 hashing cores." http://www.heliontech.com/hash.htm.

[23] "Open NAND flash interface specification 4.0." http://www.onfi.org/specifications, 2014.

[24] "Samsung 960 evo ssd." http://www.samsung.com/us/business/products/computing/ssd/s/_/n-10+11+15pexq/.

[25] A. Huffman, "NVM express 1.1a specifications." http://www.nvmexpress.org, May 2017.

[26] S. Yan, H. Li, M. Hao, M. H. Tong, S. Sundararaman, A. A. Chien, and H. S. Gunawi, "Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND ssds," in *15th USENIX Conference on File and Storage Technologies (FAST 17)*, (Santa Clara, CA), pp. 15–28, USENIX Association, 2017.

[27] "Google: Taming the long latency tail - when more machines equals worse results." http://highscalability.com/blog/2012/3/12/google-taming-the-long-latency-tail-when-more-machines-equal.html.

[28] J. Kim, D. Lee, and S. H. Noh, "Towards SLO complying ssds through OPS isolation," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, (Santa Clara, CA), pp. 183–189, USENIX Association, 2015.

[29] J. Liang, Y. Xu, D. Sun, and S. Wu, "Improving read performance of ssds via balanced redirected read," in *2016 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 1–10, Aug 2016.

[30] M. Huang, Y. Wang, Z. Liu, L. Qiao, and Z. Shao, "A garbage collection aware stripping method for solid-state drives," in *Asia and South Pacific Design Automation Conference*, pp. 334–339, 2015.

[31] R. Love, "Kernel korner – I/O schedulers," *Linux Journal*, vol. 2004, no. 118, pp. 10–, 2004.

[32] Q. Zhang *et al.*, "An efficient, QoS-aware I/O scheduler for solid state drive," in *IEEE EUC*, pp. 1408–1415, 2013.

[33] M. Jung and M. Kandemir, "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *International Symposium on High Performance Computer Architecture*, pp. 524–535, 2014.

[34] M. Jung, E. H. Wilson, III, and M. Kandemir, "Physically addressed queueing (PAQ): improving parallelism in solid state disks," in *International Symposium on Computer Architecture*, pp. 404–415, Jun 2012.

[35] J. Kim, Y. Oh, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Disk schedulers for solid state drivers," in *International Conference on Embedded Software*, pp. 295–304, 2009.

[36] E. H. Nam, B. Kim, H. Eom, and S. L. Min, "Ozone (O3): an out-of-order flash memory controller architecture," *IEEE Transactions on Computers*, vol. 60, no. 5, pp. 653–666, 2011.

[37] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir, "HIOS: a host interface IO scheduler for solid state disks," in *International Symposium on Computer Architecuture*, pp. 289–300, 2014.

[38] N. Elyasi, M. Arjomand, A. Sivasubramaniam, M. T. Kandemir, C. R. Das, and M. Jung, "Exploiting intra-request slack to improve ssd performance," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, (New York, NY, USA), pp. 375–388, ACM, 2017.

[39] A. Tavakkol, M. Sadrosadati, S. Ghose, J. Kim, Y. Luo, Y. Wang, N. M. Ghiasi, L. Orosa, J. G. Luna, and O. Mutlu, "Flin: Enabling fairness and enhancing performance in modern nvme solid state drives," in *ISCA*, vol. 2, pp. 10–3, 2018.

[40] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," in *SIGMETRICS*, pp. 181–192, June 2009.

[41] M. Jung and M. Kandemir, "Revisiting widely held SSD expectations and rethinking system-level implications," in *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, pp. 203–216, 2013.

[42] F. Chen, R. Lee, and X. Zhang, "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing," in *Proceedings of the IEEE 17th International Symposium on High Performance Computer Architecture*, pp. 266–277, Feb 2011.

[43] S.-W. Lee and B. Moon, "Design of flash-based dbms: An in-page logging approach," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, (New York, NY, USA), pp. 55–66, ACM, 2007.

[44] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending ssd lifetimes with disk-based write caches," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, (Berkeley, CA, USA), pp. 8–8, USENIX Association, 2010.

[45] M. Vilayannur, P. Nath, and A. Sivasubramaniam, "Providing tunable consistency for a parallel file store," in *Proceedings of the 4th Conference on USENIX Conference on File and Storage Technologies - Volume 4*, FAST'05, (Berkeley, CA, USA), pp. 2–2, USENIX Association, 2005.

[46] D. H. Kang, C. Min, and Y. I. Eom, "Ts-clock: Temporal and spatial locality aware buffer replacement algorithm for nand flash storages," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, pp. 581–582, June 2014.

[47] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, Y. Zhang, and Y. Tan, "Design tradeoffs for data deduplication performance in backup workloads," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, (Santa Clara, CA), pp. 331–344, USENIX Association, Feb. 2015.

[48] J. a. Paulo and J. Pereira, "A survey and classification of storage deduplication systems," *ACM Comput. Surv.*, vol. 47, pp. 11:1–11:30, June 2014.

[49] Z. Chen and K. Shen, "Ordermergededup: Efficient, failure-consistent deduplication on flash," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*, (Santa Clara, CA), pp. 291–299, USENIX Association, Feb. 2016.