NAND Flash Architectures Reducing Write Amplification Through Multi-Write Codes

Saher Odeh and Yuval Cassuto
Technion EE Department
{sahero@tx,ycassuto@ee}.technion.ac.il

***Abstract*—Multi-write codes hold great promise to reduce write amplification in flash-based storage devices. In this work we propose two novel mapping architectures that show clear advantage over known schemes using multi-write codes, and over schemes not using such codes. We demonstrate the advantage of the proposed architectures by evaluating them with industry-accepted benchmark traces. The results show write amplification savings of double-digit percentages, for as low as 10% over-provisioning. In addition to showing the superiority of the new architectures on real-world workloads, the paper includes a study of the write-amplification performance on synthetically-generated workloads with time locality. In addition, some analytical insight is provided to assist the deployment of the architectures in real storage devices with varying device parameters.**

I. Introduction

NAND flash memories have become the most attractive option for solid-state drive (SSD) deployment, owing to their superior performance over prior mass-storage technologies. The main caveat of flash storage is the inability to write data in-place, which requires frequent internal copying of data. This leads to significant *write amplification* (WA), i.e., performing many physical writes on average per user write. High write amplification causes performance degradation, reduced data reliability, and accelerated wear. Hence reducing write amplification is a primary objective for the solid-state storage industry. A common remedy to high write amplification is to increase the storage over-provisioning, which does reduce write amplification but at the same time adds to the cost per GB of the device.

A promising approach to mitigating write amplification is to use *multi-write codes*. Multi-write codes allow to re-write a data page in-place t times without erase operations, thereby consuming less physical storage space for multiple copies of the same logical-page address. The promise of this approach stems from the high availability of effective multi-write codes, which constantly improve as a result of very active research by the information theory community. In particular, a code that supports t writes requires a storage expansion factor r, where r is much smaller than t. In addition to attractive storage efficiency, multi-write codes also improve in encoding and decoding complexities, which makes it feasible to deploy them in SSD controllers.

This work is not the first to identify the promise of multi-write codes toward reducing the write amplification. A study by Jagmohan et. al [8] developed an SSD architecture that reduces write amplification with a combination of multi-write coding and compression. A more recent work by Luojie et. al [1] studied the effect of multi-write codes on write amplification through analytic and experimental treatment. The unique novelty of this present paper is in offering simple-to-implement mapping architectures that show significant write-amplification reduction on traces of industry-accepted benchmarks, with no assumption on the compressibility of the data, and at very low amounts of over-provisioned storage. This is in contrast to previously proposed mapping architectures that offer multi-write benefits only with over-provisioning amounts that are much higher than a typical storage product can afford [1].

The key idea driving the proposed mapping architectures to effectiveness is in fact quite simple. It starts with the observation that due to the $r > 1$ expansion factor required for in-place re-writes, multi-write capabilities should only be endowed where the potential benefits are high, i.e., to pages with high likelihood to be re-written in the near future. Based on that idea we present two mapping architectures that use $t = 2$ multi-write codes for *part* of the stored data, which are able to cut the write amplification by double-digit percentages for as low as 10% over-provisioning (or higher). The performance of the architectures is extensively studied using industry-accepted benchmarks. This study shows conclusive advantage for the new architectures over known schemes using multi-write codes, and over schemes not using such codes. Because the expansion factors of multi-write codes depends on the cell-level technology, i.e., the number of levels a cell can be programmed to, our study examines the write amplification for all such technologies in common deployment: SLC, MLC, and TLC. An additional study of the write-amplification performance using synthetic workloads explores how the multi-write benefits of the new architectures grow with the time-locality of the workload.

The results of the paper are organized as follows. In Section II we discuss previous work on write amplification, multi-write codes, and their combination. In Section III, after some background on write amplification in flash storage and on multi-write codes, we formally state the problem of combining the two in a mapping architecture. The new mapping architectures are detailed in Section IV. Then in Section V the write-amplification performance of the new architectures is studied in comparison with known architectures, both using traces of industry-accepted benchmarks and by evaluation on synthetically generated workloads. Section VI gives some ana-

978-1-4799-5671-5/14/$31.00©2014 IEEE

lytical insights on the inner workings of the new architectures, which can assist their practical deployment with the correct parameters. Finally, we conclude the paper in Section VII and VIII with forward-looking ideas toward successful adoption of our architectures by real-world SSDs.

II. Previous Work

The severeness and importance of the write amplification problem has triggered a large body of work aiming at mitigating and analyzing it, e.g., [2], [3], [4], [5]. All of these works (and others) explore and improve the tradeoff between the write amplification and the storage over-provisioning, hence offering device architectures with maximal performance and lifetime per unit of cost. Several of these works, such as [2], exploit the access non-uniformity in the write workload partitioning the storage space into hot and cold logical pages. The outcome from this line of work is that write amplification in NAND flash is fairly well understood, and a significant set of algorithmic and analytical tools is available for deployment in SSD products.

It has also been recognized in prior work that multi-write codes (also known as re-write codes, WOM codes, and similar names) offer the most promising tool to improve the write amplification beyond the current techniques. That is because the ability to re-write data in place can reduce the number of invalid pages and the resulting amount of garbage-collection copying. This high promise of re-write codes has motivated a number of studies that explore their benefits. In [8], a NAND device architecture is proposed with multi-write codes, where the overhead cost of the code redundancy is offset by compression. In [1], the write-amplification is studied analytically and experimentally when capacity-achieving WOM codes are employed in the system. This study shows that as the multi-level order of the NAND increases (i.e., from SLC to MLC to TLC), the write amplification can be reduced more for a given amount of storage over-provisioning. This is due to the fact that WOM codes become more efficient as the code alphabet size increases. The quantitative outcome from [1] is that using multi-write codes may improve write amplification, but only when the amount of over-provisioning is as high as 75% (for SLC), 55% (for MLC), or 40% (for TLC). With lower over-provisioning, the redundancy of the WOM codes reduces the effective physical storage space so much that their benefits succumb to the resulting high frequency of garbage collections.

Our work presented in the sequel solves the above shortcoming of multi-write codes by endowing multi-write capabilities selectively to only a part of the writes, such that the utility vs. cost of the coding is maximized. Similarly to previous works on write-amplification reduction (without multi-writes), the time-locality of the workload and the access non-uniformity will play an important role within our proposed architectures and their performance.

III. Background and Problem Statement

The main contribution of this paper is a scheme to reduce *write amplification* using *multi-write codes*. In this section we will first give some background on each of write amplification and multi-write codes separately. Then we will state the problem of combining the two in the flash mapping layer.

A. Write Amplification in NAND Flash

A NAND flash memory device is composed of write units called *pages* and erase units called *blocks*. Each block consists of N_p pages. The usable storage capacity of the device is $U \cdot N_p$ pages, where U denotes the number of *logical* blocks. A well-known limitation of the NAND flash technology is that erasure , i.e., charge removal from cells, is possible only at the block granularity, and hence it is in general not possible to re-write a data page in-place. This limitation has significant ramifications on the performance of flash storage devices and their lifetimes, as block-erasure operations are costly in terms of time and cell wear. Therefore, NAND flash storage devices employ elaborate mapping layers that optimize the physical placement of data pages to minimize internal copying and block erasures.

The main tool used by the flash mapping layer for reducing write costs is over-provisioned storage blocks not allocated for logical pages. The total number of physical blocks (allocated plus over-provisioned) in the device is denoted by T. The over-provisioning factor ρ is defined as $\rho = \frac{T-U}{U}$. Note that $\rho \geq 0$ because by definition $T \geq U$. Using the T physical blocks, the mapping layer implements an out-of-place write mechanism as follows. Upon a write request of a logical page from the user (the host), the previous version of the page (if exists) is marked *invalid* in the meta-data, and the page is written to the *frontier* of the device, which is a block that is currently allocated for incoming writes. When the frontier block becomes full and there are few available blocks left, the mapping layer invokes a *garbage collection* (GC) operation, to clean up used blocks by copying their valid pages to the frontier and then erasing them. The number of free blocks below which GC is invoked is called the *watermark*. The natural and accepted criterion to choose a block for clean up is *min-valids*: choosing the block with the smallest number of valid pages. This criterion, also called the *greedy* GC policy, was widely used and studied in the literature [2], [3], [5], [19]. The resulting efficiency of the mapping layer is quantitatively evaluated using the measure of *write amplification* (WA), which is defined as the ratio between the number of physical writes performed by the device and the number of logical writes requested by the user

$$WA = \frac{Physical\,Writes}{Logical\,Writes}. \tag{1}$$

Since every logical write needs to be written to the media, it is clear that $WA \geq 1$. It is also clear that WA values strictly greater than 1 imply inefficiency in access-time and wear, and hence minimizing the write amplification is the premier objective of the NAND flash mapping layer.

B. Multi-Write Codes

The asymmetry between program and erase in flash storage has motivated the study of *multi-write codes* as a potential tool for improving the flash access efficiency. Multi-write codes allow unrestricted in-place updates with no need to erase cells to lower charge levels. A typical way to specify a multi-write code is by the integer number of user writes t it guarantees before an erase operation is required. Without multi-write coding, the device supports only $t = 1$ write before erase, and multi-write codes give t values larger than 1. The larger t gets, the more in-place writes the device supports. Thus the added in-place writing capabilities can improve the write efficiency over the standard flash write interface. The main caveat of multi-write codes as a remedy to write-amplification is that growing t comes at the cost of adding overhead for the code redundancy. Because of that, we have to consider the multi-write overhead requirement when we evaluate the write amplification in comparison with a mapping layer that does not use multi-write codes. For this purpose, we briefly discuss the relevant theory of multi-write coding, first proposed by Rivest and Shamir under the framework of *write-once memory* (WOM) coding [6].

1) Multi-write / WOM theory: The starting point of the theory of multi-write codes is the observation that t writes can be trivially supported using a factor t larger storage space, by writing each generation of the data in a distinct physical location. Starting from there, Rivest and Shamir have proposed the write-once memory (WOM) coding model [6], including a simple binary code that gives $t = 2$ writes using only factor $r = 1.5$ larger storage space than a single write. Since then, a large number of multi-write code constructions were reported, with smaller overheads and richer parameters than this initial example. A sample from these contributions can be found in [6], [9], [10], [11], [12], [13], [14], [15], [16], [17]. The large body of available constructions offers a comprehensive menu of codes to choose from, varying in the parameters that they support and the encoding and decoding complexities that they require. To aid the deployment of multi-write codes in a real storage device, we give a brief survey of the available code families and types. In the WOM model we use the following parameters to characterize the multi-write capabilities. q is the alphabet size of the code, which equals the number of levels supported by the technology. Note that for $q > 2$ levels the term write-once memory (WOM) is a misnomer because the physical cells are not limited to be written only once as in the original binary ($q = 2$) case. n is the number of physical cells used in a single code block, which determines the minimal unit of physical read and write that the code can support (longer than n accesses are possible by encoding and decoding multiple code blocks in parallel). The parameter t is the number of write generations supported by the code. Accompanying t is the vector parameter $(M_1, \ldots, M_t)$, which denotes the number of possible values the information input can take in each of the t write generations. Equivalently, we may use the alternative vector parameter $(k_1, \ldots, k_t)$ to designate the number of input *bits* the encoder takes in each write generation. Hence $k_i = \log_2 M_i$, but note that in many known constructions the k_is are not necessarily integers. In a practical realization of a WOM code we will only use codes with integer numbers of input bits k_i, and in addition we will restrict ourselves to codes with a fixed input size $k_1 = k_2 = \cdots = k_t = k$. That is because the access between the storage device and its hosting system is specified as an interface with a fixed number of bits across the entire sequence of write generations. Therefore, the most natural WOM codes for implementation are characterized by the four parameters (q, n, k, t). Usually WOM codes are constructed as *families* of codes that fix all except one of the parameters, leaving one parameter free to be choosen from a broad set of options to fit the exact implementation setup. The most common type of family proposed in the information theory literature fixes the alphabet size q and the number of writes t, while varying the block size n with k growing in proportion to n [22], [24], [23]. Another useful type of family fixes the input/output parameters n and k, while increasing the number of writes t as the number of levels q grows [9], [11]. The advantage of this type is that it allows dynamic tradeoffs between storage efficiency and multi-write capability without changing the data layout in the device. An important distinction between WOM constructions is whether they *guarantee* the multi-write capability t, or only provide it *with high probability* assuming a uniform distribution on the input bits and long code blocks (as the codes in [12]). If one wishes to deploy codes with non-guaranteed multi-write capability, then there is need to implement a feedback and recovery mechanism in case the write fails. For codes with multi-write guarantees, a write counter is sufficient. After choosing a particular WOM code with desired parameters, it is implemented by adding an encoder function to the write path and a decoder function to the read path. In WOM terminology the encoder function is called the *update* function. The decoder function is simply a map from the current physical levels of the n cells to the k information bits, denoted by $\psi : \{0, \ldots, q-1\}^n \to \{0, \ldots, 2^k - 1\}$. The update function specifies how the n cell levels need to change as a function of the current cell levels and the new k information bits at the input, and is denoted by $\mu : \{0, \ldots, q-1\}^n \times \{0, \ldots, 2^k - 1\} \to \{0, \ldots, q-1\}^n$.

For the sake of evaluating our schemes in this paper we do not need to discuss specific multi-write constructions, but only need a bound or estimate on the redundancy overhead that they will require. To remain agnostic to the particular code chosen for deployment, we will use known information theoretic bounds on the redundancy given the code parameters. In many cases these bounds are achievable by known code constructions [17]. While the constructions that achieve capacity are not necessarily practical for implementation, in many cases there are known constructions with low implementation complexity that are quite close to optimal.

Let a t-write multi-write code be implemented over cells with q levels. q reflects the cell-level technology of the device, i.e., $q = 2$ for single-level cell (SLC), $q = 4$ for multi-level cell (MLC), and $q = 8$ for triple-level cell (TLC). Then the expansion factor r of the code, defined as the ratio between the multi-write codeword size n and the data input size (in units of q-ary symbols, $== k/\log_2 q$), is known to be bounded by

$$r \geq \frac{t \cdot \log_2 q}{\log_2 \left(\begin{array}{c} q+t-1 \\ t \end{array} \right)}. \tag{2}$$

In the sequel when we plot write amplification results of our multi-write architectures we assume for simplicity that there exist codes that meet the bound (2) with equality. We later validate that our architectures offer significant benefits even with known practical codes that do not necessarily meet this bound.

C. Problem Statement

Now that we have reviewed the write-amplification problem and the multi-write coding tool, we can formally state the problem of combining the two. Given a q-level flash storage media that supports physical writes that only increment the cell levels, implement an architecture, i.e., a combination of data structures and algorithms, that maps user writes of logical pages to physical writes, where the mapping may include encoding of a multi-write code at write time, and decoding at read time. The primary objective of this mapping architecture is to minimize the write amplification.

IV. Proposed Mapping Architectures

The principal fact underlying this paper is that when using multi-write codes, many user writes are fulfilled as in-place updates, thus reducing the rate at which free pages are consumed at the write frontier. As a result, the frequency of garbage collection operations is reduced, and this lowers the write amplification. Keeping in mind the multi-write redundancy overhead, we need to make sure that the benefit of in-place writes is not canceled by the reduction in the storage space available for writes to the frontier. In fact, we find that consistent reduction in write amplification is only possible in conjunction with a mapping architecture that makes clever (and parsimonious) use of multi-write capabilities. Such mapping architectures are detailed in this section, and analyzed in the subsequent sections.

A. Partial Multi-Write Architectures

The main idea of our proposed architectures is that multi-write capabilities are only used for writes with a higher likelihood of benefiting from them. For other writes, we do not use multi-write capabilities, and thus save the overhead that these writes would have consumed. If we succeed in allocating multi-write capabilities to the "correct" writes, then we enjoy both multi-write benefits and low total overhead. As it turns out, a simple and robust multi-write allocation policy delivers excellent write-amplification performance. This policy is specified as follows:

1) Allocate pages with $t = 2$ multi-write to all writes coming from the user.
2) Allocate pages with $t = 1$ (no multi-write) to all internal writes performed during garbage collection.

First notice that the allocation policy we use does not differentiate between writes based on their logical-page address, access history, hot/cold classification, or any other discriminatory criterion. This uniformity saves the need of maintaining complex access-history databases, or implementing involved access-prediction algorithms. Only the *type* of write – user or internal – determines its multi-write capabilities. The rationale behind this policy is that due to time-locality in the workload, logical pages written by the user are more likely to be re-written in the near future, while logical pages copied as internal writes do not have a similar likelihood. In case a logical page is indeed re-written close in time to its initial write to the frontier (before its block is erased), an in-place update is performed, and free storage space is saved.

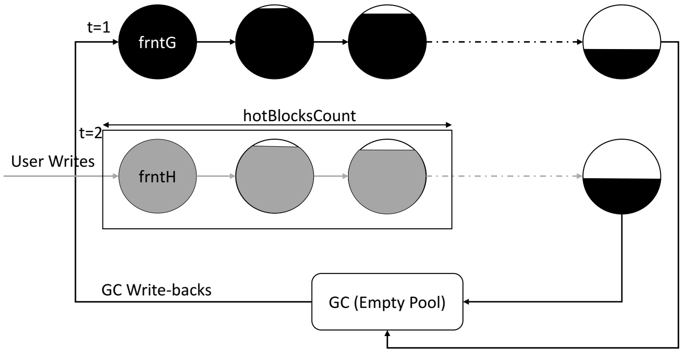

Figure 1. Double-Fronted architecture employes two frontiers, the first is for accommodating GC write-backs (dark "cold" pages using the standard no multi-write code - $t = 1$), whereas the second is for accommodating user write requests (light "hot" pages using a multi-write code - $t = 2$. The $t = 2$ blocks enter a queue of size hotBlocksCount and are only available for GC after exiting.

We propose two architectures employing the above partial multi-write policy. The first is called the *double-fronted* architecture, owing its name to the two separate write frontiers it employs: one for user writes (with $t = 2$ multi-write) and one for internal writes (with $t = 1$). The second is called the *selective* architecture, in which a single frontier accommodates both $t = 2$ user writes and $t = 1$ internal writes within the same block.

B. Double-Fronted Architecture

The double-fronted architecture uses two frontiers, one to write logical pages incoming from the user with $t = 2$, and another with $t = 1$ to store logical pages that need to be rewritten to free up a block during a GC operation. The blocks storing $t = 2$ writes are entered to a queue of size $hotBlocksCount$, where the frontier is the tail of the queue. Blocks in the queue are not considered for clean-up in GC events. Each time the queue exceeds its allocated size, the head block is popped from the queue back to general pool of blocks, where it becomes a candidate for clean-up in GC events together with all the blocks storing $t = 1$ writes (selection is done according to the usual min-valids criterion). The number of pages that can be written to a block serving $t = 1$ writes is denoted N_p. Due to code redundancy, the same physical block used for storing $t = 2$ writes can store only $\lfloor N_p/r \rfloor$ pages. See Figure 1 for a pictorial description of the double-fronted architecture. The light shading represents pages that are likely to be re-written soon due to time locality. The dark shading represents pages with no special likelihood to be rewritten. In each block the white part represents invalid pages.

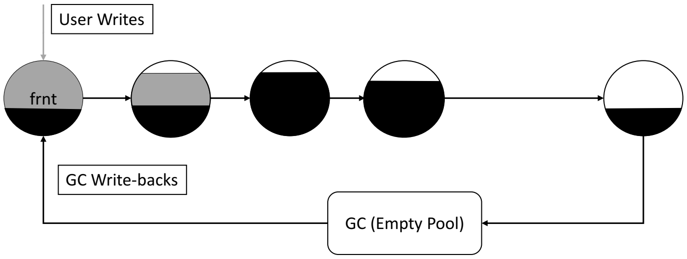

Figure 2. Selective architecture employs a single frontier to which both user writes (light "hot" pages using a multi-write code - $t = 2$) as well as GC write-backs (dark "cold" pages using the standard no multi-write code - $t = 1$) are performed.

C. Selective Multi-Write Architecture

The selective multi-write scheme uses a single frontier to which user writes are written with $t = 2$, and internal GC writes are written with $t = 1$. When a block is set as the frontier, the first writes into it are internal GC writes from the block being cleaned-up. Then the remainder of the block is used for $t = 2$ writes from the user. This continues until the block is full and then another frontier is chosen. The main advantage of the selective architecture is its simplicity, in that there is no need to manage two separate frontiers and to take care to level the wear between $t = 2$ and $t = 1$ blocks. The number of pages that fit in a physical block may vary between N_p and N_p/r, depending on the fractions of $t = 2$ and $t = 1$ writes. See Figure 2 for a pictorial description of the selective architecture.

V. Results

To evaluate the write amplification of the proposed mapping architectures, we implemented a C++ code for a discrete-event simulation of the double-fronted and selective architectures. To compare with known schemes, we also implemented a standard mapping architecture with no multi-write capabilities (see for example [2], [3], [4], [5]), and a full $t = 2$ multi-write architecture as reported in [1]. In the subsequent write-amplification result plots we assign tag names to the above four architectures: *D_Front* for double-fronted, *Selective* for selective, *REG_RW1* for no multi-write, and *REG_RW2* for full multi-write. The simulator can be invoked with different device settings such as the device logical and physical capacities, the cell-level technology, the mapping architecture, workload source, and others. We discuss these parameters in the next sub-section. In the first part of our evaluation in Section V-B, we use traces of industry-accepted benchmarks taken from the Microsoft Research (MSR) Cambridge repository recommended by the storage networking industry association (SNIA) [18]. In the second part of our evaluation in Section V-C, we use synthetic workloads generated from a simple access model that considers the time-locality of the user write requests. In both workload types our proposed architectures show clear advantage over the previously known ones.

A. Simulation Settings and Ranges

Using two server machines each with four Quad-Core AMD Opteron(tm) Processors 8356 and 128GB memory, we ran 32 simulation jobs in parallel, each with different device parameters and write workloads. Running this many jobs in parallel was necessary to obtain accurate results for a large number of setups in a reasonable time. We now detail the ranges of the parameters we used.

1) The Workload Length L: The workload length L is the number of write requests issued to the device during the test. For trace workloads L is simply the trace length; for synthetic workloads we fixed it to 15M.

2) Page Size: The page size was fixed to 4KB

3) Pages per Block N_p: N_p is the number of pages that can fit in one block. We used $N_p = 128$ throughout the evaluation.

4) Logical and Physical Block Counts U,T: The number U of logical blocks in the device sets the external capacity of the device to $UN_p \cdot 4KB$. For the benchmark trace workloads the device size was chosen in a way that on average a block will be cycled about 100 times during a single run through the trace file. For tests with synthetic workloads we used $U = 2^{11}$, which means that for the aforementioned values of N_p and page size, the device can store ~256K pages, or 1GB. A much larger device size would have required very long test runs to overcome warm-up effects. The number T of physical blocks in the device sets the internal capacity of the device to $TN_p \cdot 4KB$. This storage space is used both to store logical pages and as a resource for the different architectures to reduce the write amplification. Given U and T the over-provisioning factor of the device is set as $\rho = \frac{T-U}{U}$.

5) The Mapping Architecture: This parameter sets the device architecture to one of the four aforementioned architectures: 1) D_Front, 2) Selective, 3) REG_RW1, 4) REG_RW2.

6) The Watermark: Sets the minimal number of empty blocks for the frontier allocation. GC is invoked when the number of empty blocks drops below the watermark. We used watermark=2.

7) (RW, RW expansion factor): Sets the t and r parameters, respectively, to describe the number of writes and the expansion factor of the used multi-write code. We fixed $t = 2$ throughout the evaluation, reflecting a modest multi-write capability of one extra write beyond the standard case. The resulting expansion factor of the $t = 2$ code was calculated from (2) for three different cell technologies: $q = 2$ (SLC), $q = 4$ (MLC), and $q = 8$ (TLC). We also include in the sequel results where the expansion factor is set based on a very simple to implement multi-write code given in [10].

8) hotBlocksCount: This parameter applies only to the double-fronted architecture, and determines the size of the $t = 2$ block pool as described in Section IV-B. It is chosen so that the $t = 2$ block pool will be large enough that many logical-page re-writes will find their previous versions still in the $t = 2$ pool to reap the multi-write benefits. A more detailed discussion on how to choose this parameter is deferred to SectionVI-C.

B. Real-Benchmark Results

We now plot the write-amplification results obtained by the four different architectures on the MSR traces. Each trace represents the real storage access recorded on a server hosting a different application. Each trace file is tagged with a tag name that describes the application that generated it. We aggregated all the files with the same tag name into one large file, which we then used as input to our simulator. We repeated this process for all tag names.

We first examine the results for the PRXY trace run on a $q = 8$ (TLC) device, shown in Figure 3. The x-axis of the plot is ρ, the percentage of over-provisioned storage beyond the device logical capacity. We emphasize that the storage expansion caused by the multi-write codes is counted toward this over-provisioning allocation. The y-axis is the resulting write amplification. Figure 3 shows that the double-fronted architecture gives the best (lowest) write-amplification for all over-provisioning amounts. For example, for 10% over-provisioning double-front reduces the write amplification from 3.45 to 1.9. The second best throughout this range is the selective architecture. Note that the full multi-write (REG_RW2) architecture fails to improve write amplification over the standard mapping architecture (REG_RW1) in that range. We now show in Figure 4 the results for the same trace, but now over a $q = 2$ (SLC) device. For lower q values the efficiency of the multi-write code decreases according to (2), so we may expect worse performance for the schemes using $t = 2$ writes. While this is indeed the case for the full multi-write architecture (REG_RW2), we see that this hardly affects the performance of the double-fronted architecture. This can be attributed to the fact that only part of the writes use $t = 2$, and thus the sensitivity to the multi-write efficiency is lower. Similarly to the results obtained for PRXY, all other traces showed significant advantage for the double-fronted architecture, and a slightly lower advantage for the selective architecture. The only exceptions to that are the traces SRC1 and SRC2, where no such decisive improvement was observed (some q and ρ values show improvement and some do not).

Next we show another sample of the trace results. In Figure 5 we plot the results for the WEB trace run on a $q = 4$ (MLC) device. Here we also include a plot of the percentage of the physical writes saved by the D_Front and Selective architectures compared to the no multi-write (REG_RW1) architecture. This gives another measure of the efficiency offered by the proposed architectures - more than 60% of the excess internal writes are saved by D_Front throughout the over-provisioning range, and between 20-30% are saved by Selective. Finally, in Figure 6 we show the results for the PRXY trace where the expansion factor of the code is taken as $r = 1.5$, corresponding to a very simple to implement known multi-write code. We see that even for this pessimistic assumption on the expansion factor, D_Front offers significant savings.

C. Synthetic-Workload Results

To gain a better insight on the performance of the proposed architectures, we complement our experiments on trace workloads with synthetically generated ones. Synthetic workloads allow us to evaluate the architectures' performance as a function of the time-locality of the workload. It is clear that workloads with strong locality in time benefit the most from the partial multi-write capabilities of double-fronted and selective, as locality induces in-place updates of logical pages while they are still in $t = 2$ storage. When there is weak or no time locality, a re-write of a logical page may come when the $t = 2$ physical page holding the previous write is already erased and its block is returned to the general pool. Our objective now is to quantify this dependence of performance on time locality using synthetically generated workloads. We clarify that *no* spatial locality is assumed throughout this study, i.e., a write to a logical page does not make proximate (or any other) logical pages more likely to be written.

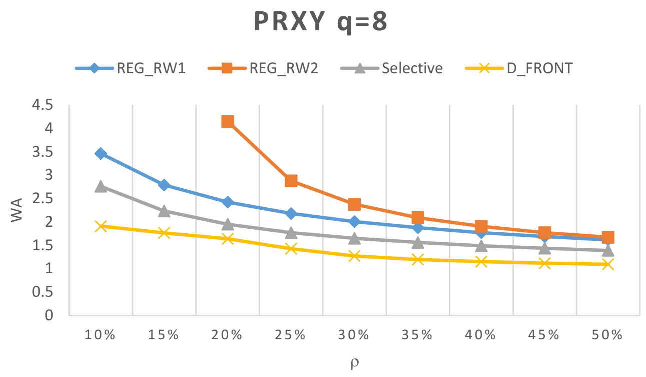

Figure 3. Write-amplification as a function of over-provisioning for the PRXY workload over a TLC ($q = 8$) device.

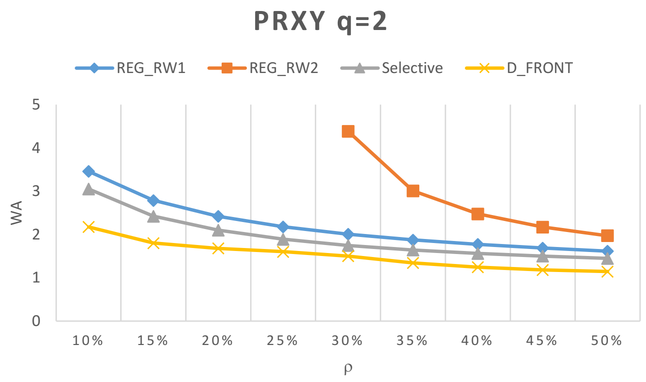

Figure 4. Write-amplification as a function of over-provisioning for the PRXY workload over a SLC ($q = 2$) device.

1) The locality parameters p, h: We choose a simple model to generate workloads with time locality. The parameter p determines the probability of the drawn logical-page write to be taken from a set of recently written logical addresses. With probability p the logical address is chosen uniformly from this set. With probability $1-p$ the write is selected uniformly from the remaining logical addresses (not in this recent-write set). Hence $p = 0$ represents the standard uniform access model with no locality. As we grow p, the workload become more local – with a higher probability to re-write a recently written logical page. The second parameter of the model is h, the size of the above set of recently written logical pages. The size-h set is managed as a FIFO queue, i.e., any page selected

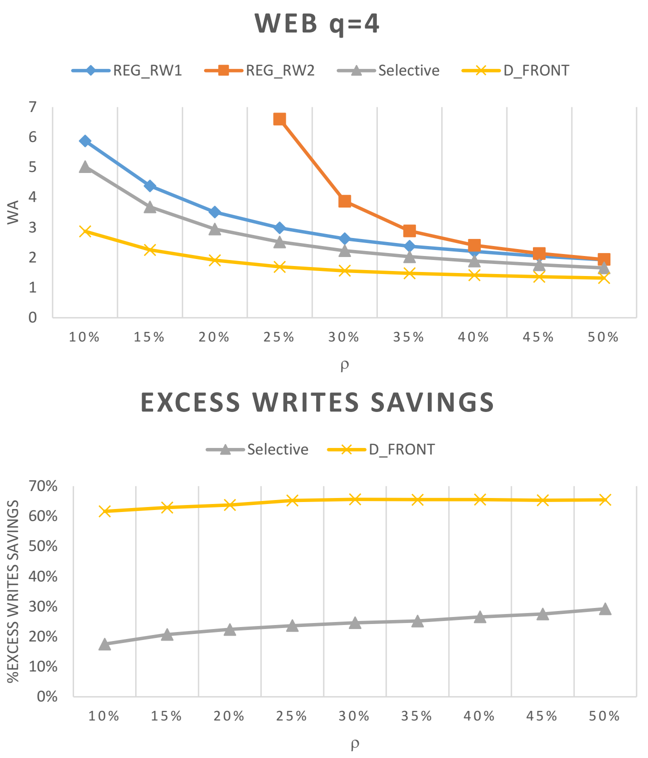

Figure 5. Write-amplification as a function of over-provisioning for the WEB workload over a MLC ($q = 4$) device. The lower plot shows the percentage of excess internal writes that are saved by the D_Front and Selective architectures over REG_RW1.

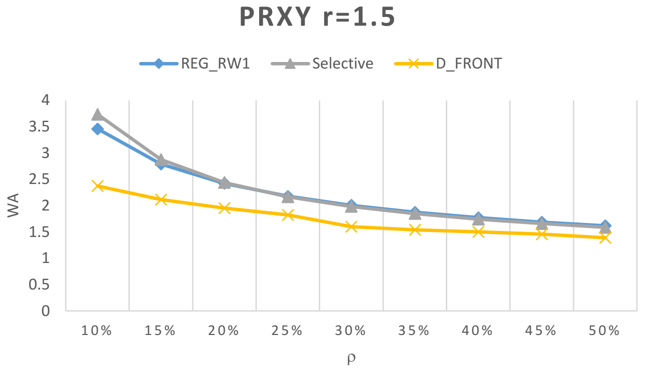

Figure 6. Write-amplification as a function of over-provisioning for the PRXY workload and devices using a simple multi-write code with a "pessimistic" expansion factor of 1.5.

for writing is entered at the tail, and popped at the head when there are h other logical pages that were written more recently. If a page existing in the queue is written, the page is popped and reentered at the tail. See Figure 7 for a schematic of the locality model.

2) *Results:* In the following we show results on synthetic workloads with p varying between 0 and 1 in jumps of 0.05. h is fixed to $h = 2 \cdot N_p$. The length of the generated workloads was taken to be $L = 15M$ writes. In Figure 8 we show the write-amplification results for an over-provisioning

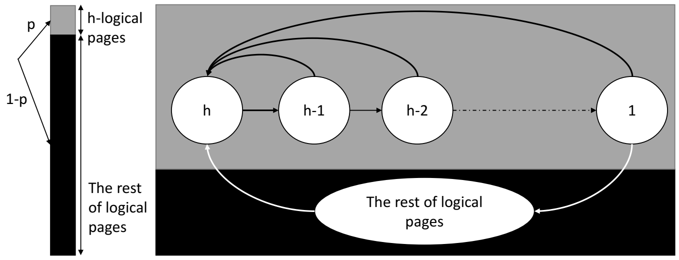

Figure 7. Schematic description of synthetic workloads with locality.

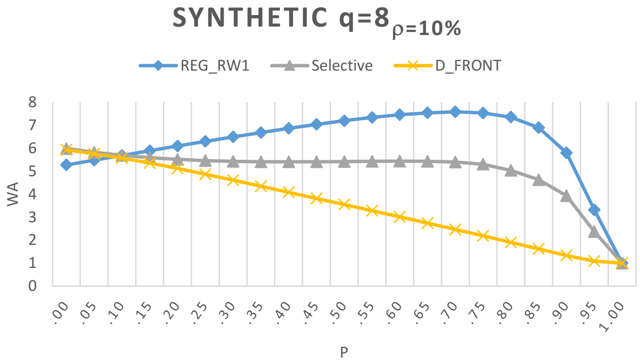

Figure 8. Synthetic workload with $\rho = 10\%$ and $q = 8$. Write amplification is plotted as a function of the locality parameter p of the workload.

factor of $\rho = 10\%$ over a $q = 8$ (TLC) device. It can be observed that when the locality parameter p is 0.1 or higher, the D_Front and Selective architectures outperform the no multi-write REG_RW1. As p grows larger, the improvement becomes more significant, until p approaches 1, when all the architectures approach the same write amplification value of 1 (The $p \to 1$ case is rather pathological and not very interesting, since only a small number of logical pages are actually written). The all multi-write REG_RW2 is not included in the plot, since $\rho = 10\%$ is not sufficient to accommodate the multi-write expansion factor throughout the storage space. Another example shown in Figure 9 uses $\rho = 20\%$ over a $q = 4$ (MLC) device. Here on the one hand the over-provisioning is higher, but on the other hand the multi-write efficiency is lower because of the smaller q. The sum effect of the two changes in parameters shows an advantage for D_Front and Selective over REG_RW1, where now a smaller p value is required for them to improve write amplification.

VI. Analytical Insights

In this section we move from a mode of evaluation of the proposed architectures toward a design oriented discussion. Our objective here is to provide analytical insight on the inner workings of the architectures, such that an SSD developer will be able to choose well-performing parameters for a variety of real-product circumstances. To keep the presentation clean and digestible, we only sketch the high level analytic considerations, and omit the unwieldy complete derivations.

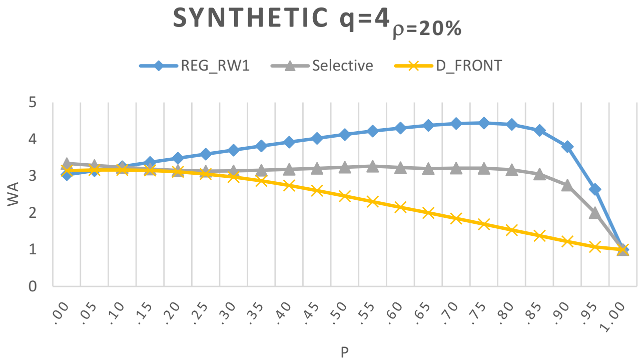

Figure 9. Synthetic workload with $\rho = 20\%$ and $q = 4$.

A. Workloads

In the standard uniform access model, the probability for a page i to be chosen at a specific write is the reciprocal of the total number of logical pages: $\Pr(i) = \frac{1}{U \cdot N_p}$. In a workload with locality, the probability to choose a specific page i is dependent on whether the page exists in the recent-page set or not. In the sequel we use the terms recent-page *set* and recent-page *queue* alternately, noting that these are different views of the same object.

1) p_{out}: the probability to choose page i when it is *not* in the recent-page set. In this case i can be chosen only when the recent set is not selected, and because $h \ll U \cdot N_p$ we have

$$p_{out} \approx \frac{1-p}{U \cdot N_p}. \tag{3}$$

2) p_{in}: the probability to choose page i when it is in the recent-page set. In this case, because of the uniform selection within the recent-page set we have

$$p_{in} = p \cdot \frac{1}{h}. \tag{4}$$

An important expression is for the mean time for a page to stay in the recent-page queue. Consider the rate of the new pages entering the recent-page queue (which is also equal to to the rate of pages exiting the queue):

$$p_{exit} = 1 - p. \tag{5}$$

The above is because a new page enters the queue only when the recently-written set is not selected. The expected time (time == number of user writes) T_q a page stays in the queue is the length of the queue divided by the arrival rate

$$T_q = \frac{h}{p_{exit}} = \frac{h}{1-p}. \tag{6}$$

As one can deduce from equation (6), the larger h is and the higher the locality p gets, the more time a page will stay in the recent-page queue.

Another important value for multi-write analysis is the mean number of user writes N_w for a page *after* it enters the recent-page queue:

$$N_w = p_{in} \cdot T_q = p \cdot \frac{1}{h} \cdot T_q = \frac{p}{1-p}. \tag{7}$$

The value of N_w is very important for the design of a storage device using the double-fronted architecture. To obtain good performance we need to have N_w away from zero, which implies a high likelihood for logical re-write while the logical page is still in the recent-page queue. Now we would like to state that $N_w \underset{p \to 1}{\to} \infty$. Moreover, N_w is independent from h, and that starting from $p = 0.2$ there is on average 0.25 extra writes induced by the locality.

B. The Block Life Cycle

If one considers the life cycle of a block from the time it is selected as the write frontier for user writes until it is selected for garbage collection, then it can be seen that it moves through 4 phases:

1) The frontier phase.
This is the beginning of the block's life cycle. It becomes the write frontier starting to accept user writes.
2) The recent-page phase.
After the block is filled as the frontier, it is replaced by a new block. At this phase many of the pages written to it are still in the recent-page queue.
3) The out phase.
In this phase all the pages in the block are out of the recent-page queue. The block stays in this phase during most of its life time.
4) The GC phase.
In this phase the block is chosen for GC and becomes the new frontier, ending its life cycle.

C. Choosing hotBlocksCount

The objective in this sub-section is to provide guidance on how to choose hotBlocksCount – the size of the $t = 2$ frontier in the double-fronted architecture.

Definition 1. Define x as the expected number of pages being copied in a garbage collection of a single block.

x is a decreasing function of ρ (the more over-provisioning, the fewer pages are copied in a GC event). The write-amplification is an increasing function of x (the more copies in a GC event, the higher the write amplification).

Definition 2. Define n as the expected number of user logical-page writes to a new write frontier before it is exhausted and a new write frontier is chosen.

In all the architectures except double-fronted, the frontier firstly captures write-backs due to the last GC event; consequently the space left for user writes n decreases with x. In the double-fronted architecture, we have $n = \lfloor N_p / r \rfloor$ irrespective of x,because the user-write frontier is not used for GC writes.

For fixed Np, t a good choice of $hotBlocksCount\,(h, p)$ in the double-fronted architecture is taken to be the minimum between two values specified next.

1) Time to invalidation (page written many times while still in the recent-page queue): Which is the mean time from the time that the page is written to the frontier until the time it becomes invalid. This equals the time until the page is written

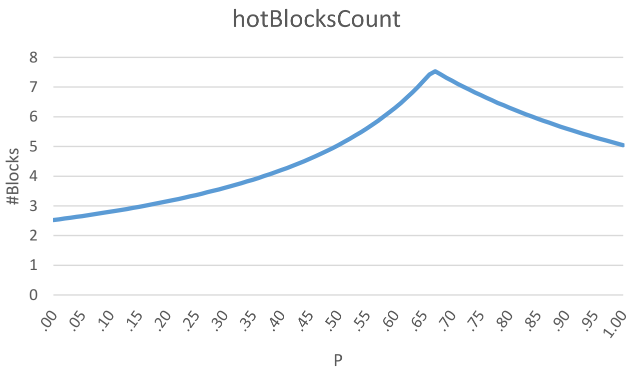

Figure 10. The right-hand side of equation (9) under the simulation parameters ($q = 2$) as a function of p.

t times after being added to the frontier. On each logical write, the probability for the write to fall on the page in question is p_{in} from equation (4), and the mean time to invalidation is thus

$$T_{inv} = \frac{t}{p_{in}} = t \cdot \frac{h}{p}. \tag{8}$$

2) Time to exit queue (page leaves recent-page queue while still valid): Which is the mean time for the page to leave the recent-page queue, denoted above as T_q in equation (6).

Note that both T_{inv} and T_q depend upon p and h. Now we write the value to be taken for $hotBlocksCount\left(h,p\right)$:

$$hotBlocksCount\left(h,p\right) \geq \frac{\min\left\{T_{inv}, T_q\right\}}{n}, \tag{9}$$

where the division by n follows from the need of n times $hotBlocksCount\left(h,p\right)$ to be at least the smaller between the time to invalidation and the time to exiting the recent-page queue. This is because until either of invalidation or queue-exit happens, the logical page is likely to be re-written in-place, and for good performance we want to keep it "protected" from GC in the $t = 2$ block queue.

To accommodate the worst-case workload scenario, the hotBlocksCount is chosen to be the maximum over all the workloads:

$$hotBlocksCount = \max_{h,p}\left\{hotBlocksCount\left(h,p\right)\right\}. \tag{10}$$

For example, taking the simulation parameters, one can plot the right-hand side of equation (9) as a function of p and then take the maximum value plus some margin. See Figure 10. Here hotBlocksCount was chosen to be 10.

D. The Effect of Locality

In the known architectures REG_RW1 and REG_RW2, increasing the locality of the workload has a negative effect on the write amplification. When a new block is selected as the write frontier, it will accommodate new logical writes whose addresses will enter the recent-page queue. Because they reside in the recent queue, these pages will be repeatedly re-written, and as a result will decrease the invalidation rate in the general population of blocks, including in the "oldest" ones that are the best candidates for GC. As a result GC events will see fewer invalid pages and high write amplification. In the double-fronted mapping architecture there is no such problem, because the re-writes due to locality hit a separate (small) set of blocks, with little effect on the general population of blocks.

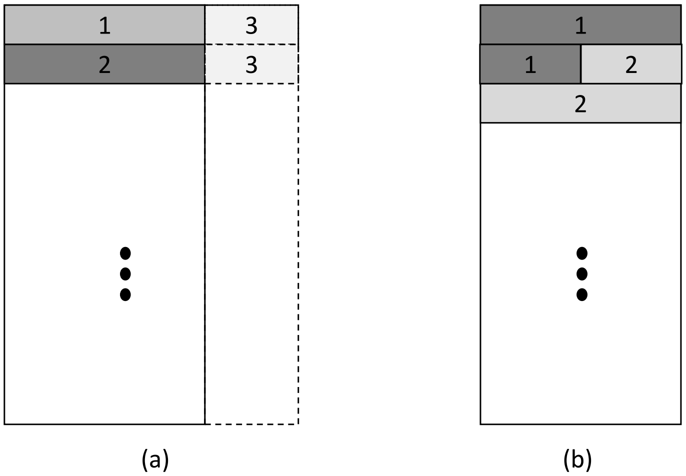

Figure 11. Variable-fixed page mapping shown for expansion factor $r = 1.5$. a) Expanded page allocation. b) Normal page allocation.

VII. Toward Real Implementation

The main complication introduced by the proposed architectures toward implementation is the mix between coded $t = 2$ pages and standard $t = 1$ pages. In the double-fronted architecture each block holds one of the two types of pages, while the selective architecture mixes the two types even in the same block. Due to the $r > 1$ expansion factor of the $t = 2$ pages, each of the two types of pages has a different size given a logical-page size. In case the physical media does not support variable-size pages, it is up to the storage device to implement a mapping between variable-size pages and the fixed-size physical pages exposed by the media. This mapping can be done in one of two ways:

(a) Expanded page allocation: The physical page size is chosen to fit a page written with expansion r. In physical pages where a $t = 1$ page is written the residual space is used to accommodate portions of additional $t = 1$ pages. This option can be seen in Figure 11 (a) for the case $r = 1.5$.

(b) Normal page allocation: The physical page size is chosen to fit a page written without expansion. In physical pages written with $t = 2$ the extra bits are written on a page allocated for the overhead portions of $t = 2$ pages. This option can be seen in Figure 11 (b) for the case $r = 1.5$.

To avoid redundant write accesses to physical pages hosting multiple logical pages, the device may need to aggregate multiple incoming logical pages before writing them to the physical pages. This will happen for $t = 1$ writes in option (a) and for $t = 2$ writes in option (b). For read access the device may need to access multiple physical pages per logical page.

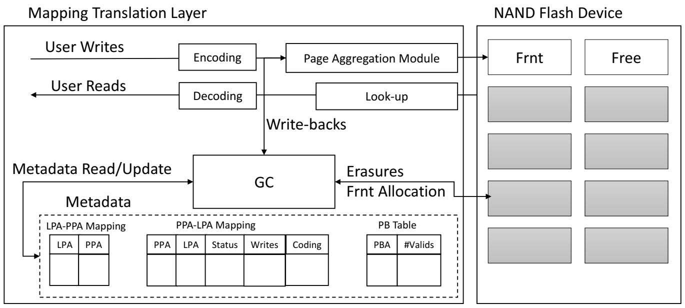

Figure 12. Diagram of a storage device using a multi-write mapping architecture.

This will happen for some of the $t = 1$ reads in option (a) and for all $t = 2$ reads in option (b). Consequently, assuming that most of the writes and reads will be served from $t = 2$ pages, option (a) is a preferable choice.

A block diagram of a storage device using a multi-write architecture is depicted in Figure 12. The right part consists of the physical storage blocks, including the frontier, used blocks and free blocks. The left part contains the functions comprising the mapping architecture. The encoding and decoding blocks are added to support the multi-write coding functionality, on top of other coding schemes used for error correction and detection. A page aggregation module is added before dispatching writes to the physical media. At the bottom of the left part we include the meta-data used by the architecture. Here we add (on top of a standard-device meta-data) fields describing the coding type of the physical page ($t = 1$ or $t = 2$), and its status (how many times it has been written since erasure).

VIII. Conclusion

The significant write-amplification savings demonstrated here by the new proposed architectures attest to their attractiveness for real SSD implementation. We believe that the insights provided in this and previous papers, together with the maturity of the multi-write coding theory can lead in the near future to SSD controllers with low-cost implementations of redundancy-efficient multi-write codes. The main barrier we see to wide adoption of our architectures is the partial support flash vendors currently offer for re-programming of physical pages to higher charge levels. To the best of our knowledge, there is no serious technology limitation to support such re-programming (in fact, some vendors do support it). However, it is clear that a fully supported re-programmable media will require addressing potential data integrity issues, such as increased inter-cell disturbs. We are convinced that the decisive results of this paper will give compelling motivation for flash vendors to address these issues.

IX. Acknowledgment

This work was supported in part by the Israel Ministry of Science and Technology, by a European Union CIG grant, and by a Viterbi faculty fellowship

References

[1] Luojie, Xiang, Kurkoski, Brian M and Yaakobi, Eitan. WOM codes reduce write amplification in NAND flash memory, Global Communications Conference (GLOBECOM), 2012 IEEE, pages 3225-3230, 2012. IEEE.

[2] Desnoyers, Peter. Analytic modeling of SSD write performance, Proceedings of the 5th Annual International Systems and Storage Conference, pages 12, 2012. ACM.

[3] Agarwal, Rajiv and Marrow, Marcus. A closed-form expression for write amplification in NAND flash, GLOBECOM Workshops (GC Wkshps), 2010 IEEE, pages 1846-1850, 2010. IEEE.

[4] Hu, Xiao-Yu, Eleftheriou, Evangelos, Haas, Robert, Iliadis, Ilias and Pletka, Roman. Write amplification analysis in flash-based solid state drives, Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference, pages 10, 2009. ACM.

[5] Luojie, Xiang and Kurkoski, Brian M. An improved analytic expression for write amplification in NAND flash, Computing, Networking and Communications (ICNC), 2012 International Conference on, pages 497-501, 2012. IEEE.

[6] Rivest, Ronald L and Shamir, Adi. How to reuse a "write-once" memory. Information and control, 55(1):1-19, 1982.

[7] Haas, XYHR and Hu, X. The fundamental limit of flash random write performance: Understanding, analysis and performance modelling. Technical report. IBM Research Report, 2010/3/31, 2010.

[8] Jagmohan, Ashish, Franceschini, Michele and Lastras, Luis. Write amplification reduction in NAND flash through multi-write coding, Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, pages 1-6, 2010. IEEE.

[9] Cassuto, Yuval and Yaakobi, Eitan. Short Q-ary Fixed-Rate WOM Codes for Guaranteed Re-writes and with Hot/Cold Write Differentiation. IEEE Transactions on Information Theory, to appear.

[10] Rivest, Ronald L and Shamir, Adi. How to reuse a "write-once" memory. Information and control, 55(1):1-19, 1982.

[11] Bhatia, Aman, Iyengar, Aravind R and Siegel, Paul H. Multilevel 2-cell t-write codes, Information Theory Workshop (ITW), 2012 IEEE, pages 247-251, 2012. IEEE.

[12] Burshtein, David and Strugatski, Alona. Polar write once memory codes, Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on, pages 1972-1976, 2012. IEEE.

[13] Kurkoski, Brian M. Lattice-based WOM codebooks that allow two writes, Information Theory and its Applications (ISITA), 2012 International Symposium on, pages 101-105, 2012. IEEE.

[14] Kurkoski, Brian M. Rewriting codes for flash memories based upon lattices, and an example using the E8 lattice, GLOBECOM Workshops (GC Wkshps), 2010 IEEE, pages 1861-1865, 2010. IEEE.

[15] Shpilka, Amir. Capacity achieving multiwrite WOM codes. 2012.

[16] Shpilka, Amir. New constructions of WOM codes using the Wozencraft ensemble. 2011.

[17] Buzaglo, Sarit and Etzion, Tuvi. Tilings with $ n $-Dimensional Chairs and their Applications to WOM Codes. Technical report. 2012.

[18] MSR Cambridge traces http://iotta.snia.org/traces/388

[19] Bux, Werner. Performance evaluation of the write operation in flash-based solid-state drives. IBM Research, Zurich, Rschlikon, Rep. RZ3757, 2009.

[20] Bhatia, Aman, Iyengar, Aravind R and Siegel, Paul H. Multilevel 2-cell t-write codes, Information Theory Workshop (ITW), 2012 IEEE, pages 247-251, 2012. IEEE.

[21] Burshtein, David and Strugatski, Alona. Polar write once memory codes, Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on, pages 1972-1976, 2012. IEEE.

[22] Cohen, G, Godlewski, Philippe and Merkx, Frans. Linear binary code for write-once memories (corresp.). Information Theory, IEEE Transactions on, 32(5):697-700, 1986.

[23] Haymaker, Kathryn and Kelley, Christine A. Geometric WOM codes and coding strategies for multilevel flash memories. Designs, Codes and Cryptography, 70(1-2):91-104, 2014.

[24] Yaakobi, Eitan, Kayser, Scott, Siegel, Paul H, Vardy, Alexander and Wolf, Jack Keil. Codes for write-once memories. Information Theory, IEEE Transactions on, 58(9):5985-5999, 2012.