Performance Evaluation of Dynamic Page Allocation Strategies in SSDs

ARASH TAVAKKOL, Sharif University of Technology and Institute for Research in Fundamental Sciences (IPM)
POOYAN MEHRVARZY, Institute for Research in Fundamental Sciences (IPM)
MOHAMMAD ARJOMAND, Sharif University of Technology
HAMID SARBAZI-AZAD, Sharif University of Technology and Institute for Research in Fundamental Sciences (IPM)

Solid-state drives (SSDs) with tens of NAND flash chips and highly parallel architectures are widely used in enterprise and client storage systems. As any write operation in NAND flash is preceded by a slow erase operation, an out-of-place update mechanism is used to distribute writes through SSD storage space to postpone erase operations as far as possible. SSD controllers use a mapping table along with a specific allocation strategy to map logical host addresses to physical page addresses within storage space. The allocation strategy is further responsible for accelerating I/O operations through better striping of physical addresses over SSD parallel resources. Proposals already exist for using static logical-to-physical address mapping that does not balance the I/O traffic load within the SSD, and its efficiency highly depends on access patterns. A more balanced distribution of I/O operations is to alternate resource allocation in a round-robin manner irrespective of logical addresses. The number of resources that can be dynamically allocated in this fashion is defined as the degree of freedom, and to the best of our knowledge, there has been no research thus far to show what happens if different degrees of freedom are used in allocation strategy. This article explores the possibility of using dynamic resource allocation and identifies key design opportunities that it presents to improve SSD performance. Specifically, using steady-state analysis of SSDs, we show that dynamism helps to mitigate performance and endurance overheads of garbage collection. Our steady-state experiments indicate that midrange/high-end SSDs with dynamic allocation can provide I/O operations per second (IOPS) improvement of up to 3.3x/9.6x, response time improvement of up to 56%/32%, and about 88%/96% average reduction in the standard deviation of erase counts of NAND flash blocks.

Categories and Subject Descriptors: B.3.2 [**Memory Structures**]: Design Styles—*Mass storage*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids—*Simulation*; D.4.2 [**Operating Systems**]: Storage Management—*Secondary storage*

General Terms: Design, Algorithm, Performance

Additional Key Words and Phrases: Dynamic page allocation, garbage collection, performance evaluation, solid-state drive

ACM Reference Format:
Arash Tavakkol, Pooyan Mehrvarzy, Mohammad Arjomand, and Hamid Sarbazi-Azad. 2016. Performance evaluation of dynamic page allocation strategies in SSDs. ACM Trans. Model. Perform. Eval. Comput. Syst. 1, 2, Article 7 (June 2016), 33 pages.
DOI: http://dx.doi.org/10.1145/2829974

Authors' addresses: A. Tavakkol, P. Mehrvarzy, M. Arjomand, and H. Sarbazi-Azad, School of Computer Science, Institute for Research in Fundamental Sciences (IPM), No. 70, Lavasani Ave., Tehran, Iran; emails: tavakkol@ce.sharif.edu, p.mehrvarzy@ipm.ir, arjomand@ce.sharif.edu, azad@sharif.edu.

DOI: http://dx.doi.org/10.1145/2829974

1. INTRODUCTION

Due to ever-increasing demand for high-performance and reliable storage systems, the critical role of solid-state drives (SSDs) is increasingly highlighted in both enterprise and client systems. The SSD is a promising design paradigm that uses a number of NAND flash memory chips as storage media. The flash chips are arranged in a multi-channel multiway bus architecture to achieve a high level of internal parallelism. In addition, the architecture of state-of-the-art flash chips employs a hierarchy of multiple dies and planes to enhance SSD parallelism. Each of the mentioned parallelism levels (i.e., channel-, chip-, die-, and plane level) has its own limitations and opportunities, making SSD performance highly dependent on the strategy used for conducting flash operations in parallel. This is known as allocation strategy, and its role is to map logical host addresses to physical page addresses (PPAs) on the storage space of the SSD. This strategy determines how to effectively exploit different resource parallelism levels and avoid their inherent limitations to get better performance.

The main focus of past studies was on allocation strategies where the target of a flash operation is *statically* determined using simple mathematical equations. For example, the target channel, chip, die, and plane are determined based on the quotient and reminder of the successive divisions of the logical address in a predefined order [Shin et al. 2009; Hu et al. 2011; Jung and Kandemir 2012; Hu et al. 2013]. The write efficiency of these approaches strongly depends on the address patterns of I/O requests, and their static nature may lead to *conflicts* on shared resources. For example, the logical address of two consecutive program (write) operations may be mapped on the same channel while other channels are idle. To address this issue, one can relax deterministic resource assignment in one or more dimensions of parallelism and use *dynamism* instead. Dynamism suggests allocating resources in a circular order (e.g., round-robin) and helps to *reduce* resource conflicts for simultaneous I/O operations. According to this, we define the *degree of freedom* as the number of parallelism levels whose resources are dynamically allocated, and we introduce new allocation strategies with various freedom degrees of one, two, three, or four. We then comprehensively analyze dynamic strategies under real and synthetic disk traffics and highlight their performance benefits with respect to those of conventional static allocation strategies. Although several prior works adopted fully dynamic allocation (with a freedom degree of four) as a possible allocation strategy [Shin et al. 2009; Park et al. 2010b; Hu et al. 2013], our detailed simulation of two SSD configurations under 32 I/O benchmarks confirm that it cannot efficiently utilize available parallelism opportunities. More precisely, allocations with freedom degrees of two and three better exploit intraflash chip parallelisms (die level or plane level) and provide higher performance than a fully dynamic allocation strategy.

When evaluating an SSD, *garbage collection* (GC) plays an important role. In fact, due to the out-of-place update property of NAND flash memories, free (clean) physical pages are exhausted and further program transactions need reclamation (erase) of some pages containing invalid data. This procedure is named *GC* and is very slow because of time-consuming erase operations. The SSD works with no concern of GC overheads only at early stages of its lifetime, after which GC plays an important role and noticeably interferes with normal flash operation [Jung and Kandemir 2013]. Therefore, there is no guarantee that an allocation strategy will have acceptable performance throughout the entire lifetime of the SSD by just considering its early stage behavior, especially for workloads with high write traffic rates. In contrast to all previous proposals, we further focus on steady-state analysis of allocation strategies by considering the effects of GC. To this end, we extend our evaluation methodology to simulate steady-state behavior of the SSD and then uncover that some degrees of freedom in the allocation strategy can help to tolerate GC effects on overall performance and lifetime. To summarize, the contributions of this work are as follows:

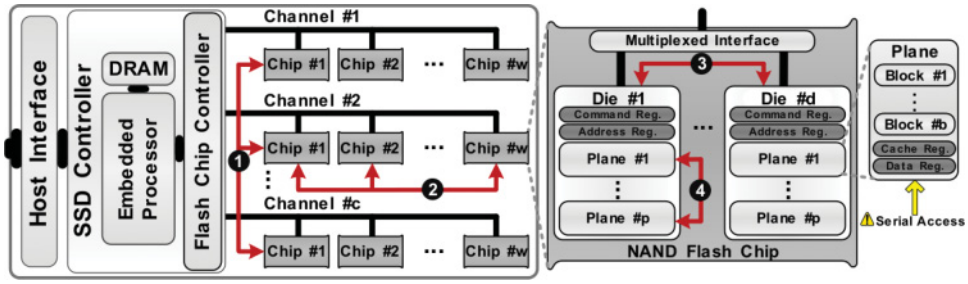


Fig. 1. Internal architecture of a NAND flash-based SSD. Red arrows highlight four levels of parallelism. The plane provides the lowest level of parallelism, and simultaneous accesses to multiple blocks within it are not permitted.

—We introduce dynamic resource allocation as a solution to mitigate the dependency of SSD performance to the access patterns of I/O requests. We theoretically and empirically explore the effects of dynamism on response time (RT) and maximum I/O operations per second (IOPS) of midrange and high-end SSDs under a large set of 32 different workloads.
—We study the mutual effects of GC and allocation strategies. To this end, we continue replaying a workload up to a point that GC is regularly executed with a rate converging to some fixed value. We observe that adoption of dynamism in page allocation helps the SSD tolerate most GC-induced performance fluctuations and keep the performance almost stable during its lifetime.
—We explore the benefits of freedom in allocation strategy with respect to fair distribution of operations over flash planes. Our findings confirm that dynamism leads to uniform distribution of program/erase (P/E) operations and thus improves flash endurance.

The rest of the article is organized as follows. Section 2 provides background on SSD internal architecture and basics of allocation strategy in current SSDs. Section 3 presents the evaluation platform. Section 4 gives details of dynamic allocation strategies that are comprehensively evaluated and compared to the conventional static allocations in Section 5. Section 6 studies the steady-state behavior of our selected SSD configurations under dynamic and static page allocation strategies. Section 7 provides a summary of previous sections and suggests the best choices within dynamic strategies. Section 8 discusses related work, and Section 9 concludes the article.

2. BACKGROUND

2.1. SSD Internals

Figure 1 illustrates the internal architecture of a modern SSD based on NAND flash memory [Jung et al. 2012; Hu et al. 2013; Hsieh et al. 2014]. This architecture is composed of four components. First, the *host interface* supports communication between the host system and SSD controller and is responsible for queuing host I/O requests as well as returning responses back to the host. Second, the *SSD controller* is responsible for processing I/O requests and managing SSD resources. Using a microprocessor along with a DRAM memory, the controller tasks are conducted through executing *flash translation layer* (FTL) firmware. Third, the *flash chip controller* (FCC) logic acts as an interface between the controller and NAND flash chips. Fourth, a set of NAND flash memory chips provide the actual storage volume and are connected to the FCC through a multichannel multiway bus.

A NAND flash memory supports three main operations: *read*, *program* (*write*), and *erase*. A read or program is performed in a unit of page that is typically of size 2KiB,[1] 4KiB, or higher. However, updating the data of a previously programmed physical page should be preceded by a very slow erase operation. To hide the negative impacts of erase operations, physical pages are organized in groups of 128, 256, or more pages, called *block*, and erase is performed at block level. In addition, to update previously written data, the physical page containing old data is marked as *invalid* and new data is written to a *free* (clean) physical page turning its state to *valid*. This is known as an *out-of-place* update and helps to postpone erases. Another obstacle issue with flash memories is the limited number of reliable P/E cycles per block. Today, FTLs adopt write caching and wear-leveling policies to guarantee long-term endurance for NAND flash SSDs [Micron Technology Inc. 2010a; Cintra and Linkewitsch 2013].

As of 2015, flash memory products are provided in three main categories based on the number of bits stored in each memory cell (i.e., SLC (1 bit/cell), MLC (2 bit/cell), and TLC (3 bit/cell)). As cell density (bit/cell value) increases, the price per gigabyte of storage noticeably reduces for the cost of performance reduction. In addition, a typical SLC product provides endurance of up to 100k P/E cycles, whereas MLC and TLC products may provide up to 10k and 1k P/E cycles [Grupp et al. 2012], respectively.

The SSD resources are organized in a highly parallel architecture, and a key responsibility of the FTL is to fully exploit such a rich parallelism to concurrently conduct multiple flash operations and meet the desired performance goals [Chen et al. 2011; Hu et al. 2013]. Figure 1 illustrates the four levels of parallelism [Chen et al. 2011; Jung and Kandemir 2012; Hu et al. 2013], each highlighted with red arrows and marked with a circled number:

—*Channel level* (❶): Flash operations can be striped over channels, and each channel can independently transmit data and commands.
—*Way level* (❷): A channel is shared between multiple flash memory chips (each connected to one way), and while a chip is transmitting data or command, others may concurrently execute commands to improve resource utilization.
—*Die level* (❸): A flash chip consists of a number of *dies*, each having its own *command* and *address* registers, and can independently execute a command (i.e., read, program, and erase). Thus, the SSD can interleave command execution between dies in the same chip. We refer to this type of execution as *interleaved* operations.
—*Plane level* (❹): At the lowest level, a die is composed of a few number of *planes,* each including a large set of blocks. The plane is the smallest unit serving a flash operation within which simultaneous accesses to multiple pages or blocks are not allowed. All planes of a die share the corresponding command and address registers. Thus, for parallel execution of read/program operations on two different planes within a die, the target physical pages must have the same block and page addresses. Additionally, block addresses must be the same for parallel execution of erase operations on two different planes within a die. We refer to this type of execution as *multiplane* operations.

2.2. Flash Translation Layer

The FTL implements core algorithms and functions required for normal SSD operation and emulates the block-device interface provided in conventional HDDs. Here, we go over major FTL functionalities.

I/O request management and address mapping. Figure 2 illustrates the steps of I/O request handling in a contemporary SSD that supports physically addressed queueing

[1]KiB, MiB, and GiB are used for binary units, whereas KB, MB, and GB are used for decimal units.

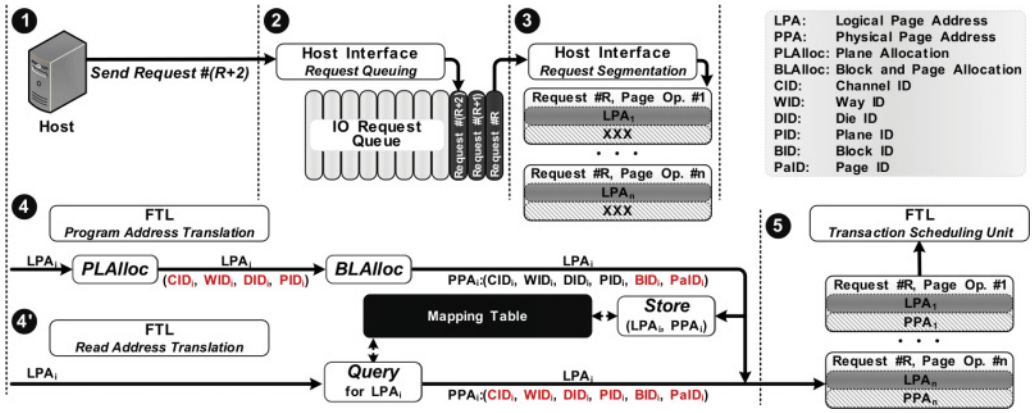


Fig. 2. Steps of handling I/O requests in contemporary SSDs.

for both read and program operations [Jung et al. 2012; Jung and Kandemir 2014]. We refer to each step with a circled number. First, the host system issues a read or write request (❶) with a specific transfer size and a logical file system address. After receiving this request, the host interface inserts it into a request queue to be processed (❷). The host interface further parses this request into several page-sized transactions (❸), each with a specific *logical page address* (LPA). Due to out-of-place update property, the FTL should translate these LPAs PPAs. The translation procedure can be performed at either page or block level. The focus of this work is on page-level translation that follows two different paths for program (❹) and read (❹') operations:

—*Program address translation* (❹): To translate the LPA of a program operation, the FTL uses two primitives. The *PLAlloc* primitive determines the target channel ID (CID), way ID (WID), die ID (DID), and plane ID (PID) according to a predefined allocation strategy. Then, a *BLAlloc* primitive assigns a block ID (BID) within the target plane. Inside a block, pages are programmed sequentially, and hence the page ID (PaID) can be deterministically assigned. Finally, the PPA is determined based on the outcomes of *PLAlloc* and *BLAlloc*. The corresponding (LPA, PPA) pair is also stored in a *mapping table* for future read translations. In this work, we concentrate on *PLAlloc* design, which plays a key role in optimizing SSD performance and maximizing striping likelihood. There are well-known choices for *BLAlloc*, such as *wear-aware* [Micron Technology Inc. 2008] or *first-fit* [Shore 1975] approaches. In this article, we choose first-fit *BLAlloc*, which is explained in Section 3.

—*Read address translation* (❹'): The translation is simply performed by searching the mapping table for the LPA entry that has been stored during program address translation.

With address translation accomplished, the page-sized transactions are delivered to the transaction scheduling unit (TSU) to be scheduled for execution on their target dies (❺). The TSU is a portion of FTL firmware. It is responsible for resolving resource contentions among flash transactions and tries to schedule a higher number of transactions for parallel execution. Moreover, the TSU searches for read/program operations that conform to die-interleaved and multiplane requirements and groups them for parallel execution [Jung and Kandemir 2014].

Garbage collection. The out-of-place update policy consumes free physical pages and generates a large number of invalid pages. When the number of free pages inside a

Table I. Enterprise Products from Well-Known SSD Vendors

Vendor	A	A	B	B	C	C	D	D	H	H	I	I
Capacity (GB)	800	700	400	1,600	480	800	480	400	400	400	2,400	1,200
Spare Factor	27%	36%	27%	27%	17%	58%	13%	27%	35%	49%	27%	27%
Number of Chips	8	64	32	128	16	56	4	16	18	40	48	48
Chip Capacity (GiB)	128	16	16	16	32[a]	32	128	32	16	16	64	32
Flash Technology	MLC	SLC	MLC	MLC	MLC	MLC	TLC	MLC	MLC	SLC	MLC	SLC
Category	MID[b]	HIG[c]	MID	HIG	MID	HIG	MID	MID	MID	MID	HIG	HIG

[a]Almost 85% of the flash chips have this capacity. [b]Midrange. [c]High end.

plane falls below a specific threshold, the FTL triggers a GC procedure for that plane to reclaim invalid pages. Having GC launched, the *GCSelect* primitive determines the candidate blocks for erase, their valid pages are written into a different block, and finally erase operations are initiated. Due to high latency of page movement and erase operation, this procedure takes a long time, which motivates SSD vendors to provide extra storage space than the user-visible capacity, known as *overprovisioning*. Overprovisioning helps to relieve GC side effects and is usually measured in terms of *spare factor* (S_f), which is defined as the ratio of added space to total storage capacity [Van Houdt 2013a].

Wear leveling. Due to nonuniform distribution of program operations, the frequency of updating physical pages is not identical and blocks containing hot data will have more invalid pages. Further, most *GCSelect* strategies are designed to minimize GC cost by selecting blocks with higher numbers of invalid pages. This causes the erase count of blocks with hot data pages to drastically increase, hence resulting in fast wear-out. To mitigate this and prolong the SSD life span, the FTL uses a set of wear-leveling policies, such as executing a dedicated primitive that swaps data between hot and cold blocks [Chang et al. 2004; Micron Technology Inc. 2010a] or utilizing an endurance-aware *BLAlloc* [Micron Technology Inc. 2010a].

3. EXPERIMENTAL SETUP

Simulator. Throughout this article, we perform discrete-event and trace-driven simulations using SSDSim [Hu et al. 2011]. SSDSim supports a detailed SSD model given in Section 2, including a standard multichannel communication protocol and flash command execution conditions/restrictions, along with request buffering and *PLAlloc*, *BLAlloc*, and *GCSelect* primitives. More importantly, the aforementioned levels of parallelism and advanced operation scheduling policies similar to Sprinkler [Jung and Kandemir 2014] are provided in SSDSim to fairly exploit channel-, way-, die-, and plane-level resources. We disable the wear-leveling primitive in SSDSim to study the interplay of GC and allocation strategy on erase distribution of blocks. The *greedy* algorithm is our choice for *GCSelect*, which searches the whole set of blocks in a plane and selects the block with the maximum number of invalid pages [Bux and Iliadis 2010]. In addition, our *BLAlloc* strategy is *first-fit*; it allocates the first block containing free pages (within a plane) without considering performance or endurance issues. Thus, the selected *BLAlloc* is similar to round-robin, and we can fairly decide on the efficiency of *PLAlloc* strategies with respect to performance and endurance metrics.

Configuration. There are various SSD products on the market, each offering different performance characteristics and operational conditions. Most SSD vendors classify their products into two main categories: *enterprise* and *client*. Tables I and II present an overview of the structural parameters of some SSD products from well-known vendors in enterprise and client markets, respectively. The SSDs in the enterprise

Table II. Client Products from Well-Known SSD Vendors

Vendor	A	A	B	B	C	C	D	D	E	E	F	G
Capacity (GB)	960	256	240	512	240	480	500	250	240	256	512	240
Spare Factor	13%	7%	13%	7%	13%	13%	10%	10%	13%	7%	7%	13%
Number of Chips	16	16	16	16	16	16	4	8	8	4	8	8
Chip Capacity (GiB)	64	16	16	32	16	32	128	32	32	64	64	32
Flash Technology	MLC	MLC	MLC	MLC	MLC	MLC	TLC	TLC	MLC	MLC	MLC	MLC

Table III. Details of SSD-MLC and SSD-SLC Configurations Used in Simulation Experiments

	Structural Parameters	NAND Flash Parameters
SSD-MLC	User Capacity = 480GB	Micron Technology Inc. [2010b]
	Spare-factor = 13%	Number of Dies per Flash Chip = 4
	Host Interface = SATA 3, 6Gbit/s	Number of Planes per Die = 2
	Number of Bus Channels = 4	Block Size = 2MiB, Page Size = 8KiB
	Bus Channel I/O Transfer Rate = 200MT/s	Page Read Latency = 75μs
	Flash Chip per Bus Channel = 4	Page Program Latency = 1600μs
	Flash Chip Capacity = 32GiB	Block Erase Latency = 5ms
SSD-SLC	User Capacity = 700GB	Micron Technology Inc. [2010c]
	Spare Factor = 36%	Number of Dies per Flash Chip = 4
	Host Interface = PCIe 2.0 x8, 3.2GB/s	Number of Planes per Die = 2
	Number of Bus Channels = 8	Block Size = 1MiB, Page Size = 8KiB
	Bus Channel I/O Transfer Rate = 333MT/s	Page Read Latency = 35μs
	Flash Chip per Bus Channel = 8	Page Program Latency = 350μs
	Flash Chip Capacity = 16GiB	Block Erase Latency = 1.5ms

Note: The SSD-MLC represents client and midrange enterprise SSDs, whereas SSD-SLC conforms to the properties of high-end enterprise SSDs.

category are further divided into two subcategories: *midrange* and *high end*. The former subcategory typically includes products that use the SAS or SATA interface and can handle up to tens of thousand IOPS, whereas the latter includes products with extremely high performance and endurance characteristics that employ a PCIe interface and offer hundreds of thousand of IOPS. According to our survey, we choose two different SSD configurations for our simulation experiments. Table III shows the details of their structures and the underlying NAND flash characteristics. The SSD-MLC configuration is based on MLC NAND technology and is chosen to comply with the properties of client and midrange enterprise products containing lower numbers of flash chips. Further, the SSD-SLC is based on SLC NAND flash technology and conforms to the properties of a typical high-end enterprise SSD with higher numbers of flash chips and excessive performance and endurance requirements.

Workloads. Table IV shows the set of selected real application traces and their statistical characteristics. The `iozn` and `postm` traces were collected from IOZone [Norcott 2014] and Postmark [Katcher 1997] file system benchmark suites. The `tpce` and `tpcc` traces are examples of online transaction processing (OLTP) applications collected by Microsoft Corporation [2008a]; `fin1` and `fin2` traces are other examples of OLTP applications collected at two large financial institutions [UMass Trace Repository 2014]. The `wsrch` trace was collected at a popular search engine [UMass Trace Repository 2014]. The `msnfs-b` and `msn-c` traces were collected from MSN storage file servers by Microsoft Corporation [2008b]. The `exch` trace is a Microsoft Exchange 2007 mail server trace [Microsoft Corporation 2008a]. The `dev` trace was collected from a file server accessed by more than 3,000 users to download daily releases of Microsoft Visual Studio [Microsoft Corporation 2008b]. All other traces were gathered from Microsoft Research Cambridge data-center servers [Microsoft Corporation 2008c] (detailed characterization is available in Narayanan et al. [2008]). This large set of workloads enables

Table IV. Characteristics of the Evaluated I/O Workloads

Trace	Read Ratio	Size R/W[a]	Int Arr.[b]	Req. Count	LPAs[c]	Trace	Read Ratio	Size R/W[a]	Int Arr.[b]	Req. Count	LPAs[c]
Read-Dominant Small-Sized						*Write-Dominant Small-Sized*					
dev	88%	14.7/10.9	2.7	1326917	1101717	exch	28%	16.1/14.2	1.1	2618264	2173363
fin2	82%	2.3/2.9	11.1	3698864	58954	fin1	23%	2.3/3.8	8.2	5334949	69440
msn-b	67%	9.6/11.1	0.5	1175607	1003987	hm-0	36%	7.4/8.4	151.5	3993316	316442
msn-c	74%	8.7/12.6	4.9	4477510	3421708	mds-0	12%	23.7/7.3	499.4	1211034	410008
prn-1	75%	22.5/11.7	53.8	11233411	10894284	prn-0	11%	22.9/9.7	108.3	5585886	1948695
proj-3	95%	9.0/23.7	269.4	2244644	768778	pxy-0	03%	8.4/4.7	48.2	12518968	119932
proj-4	99%	23.7/11.1	93.5	6465639	16227056	src2-0	11%	8.1/7.1	448.6	1557814	100124
tpce	94%	8.0/9.5	0.5	1345035	2191996	stg-0	15%	24.9/9.2	297.8	2030915	837969
tpcc	67%	8.0/8.4	0.3	2459043	1196995	ts	18%	13.7/8.0	387.0	1801734	128883
wsrch	99%	15.1/8.6	3.4	4579809	846674	dev-0	20%	12.6/8.2	529.0	1143261	72110
Read-Dominant Large-Sized						*Write-Dominant Large-Sized*					
mds-1	93%	60.1/13.9	366.5	1637711	11076636	iozn	00%	–/335	5.3	1002847	16929531
postm	84%	48.0/336.2	3.0	1139771	3723778	proj-0	12%	17.9/40.9	143.2	4224524	422656
proj-1	89%	37.2/10.8	25.6	23639742	91553496	src1-2	25%	19.1/32.5	317.0	1907773	262496
src1-1	95%	36.0/14.7	13.2	45746222	15768544	src2-2	30%	68.1/51.1	547.6	1156885	2681076
stg-1	64%	59.6/7.2	275.3	2196861	10475174	usr-0	40%	40.9/11.3	270.3	2237889	327001
usr-2	81%	50.8/13.9	57.2	10570046	49735625	web-0	30%	30.0/8.6	297.9	2029945	958629

[a]Request size in KiB; [b]Interarrival time in milliseconds; [c]Total number of LPA accesses, assuming 8KiB page size.

us to do investigations under diverse working conditions. We classify the workloads into four main groups for better understanding and clear presentation of the results. Our investigations result in two main criteria for classification: *type of requests* (read or write) and *average request size*. A workload is *read dominant* if the ratio of read requests is larger than 50% of the total requests; otherwise, it is *write dominant*. In addition, a workload is *small-sized* if its average request size is smaller than 24KiB[2]; a workload with an average request size of larger than 24KiB is considered *large-sized*. Similar to prior studies [Van Houdt 2014], a preprocessing phase is performed ahead of trace execution to ensure existence of mapping table entries for all read operations. During this phase, the mentioned segmentation process of Figure 2 is virtually performed for each I/O request to find out which read LPA requires an entry in the mapping table. More clearly, if a read request accesses an LPA that is not written with an earlier write request of the trace, then a mapping table entry is created using *PLAlloc* and *BLAlloc* primitives and the corresponding physical page is marked as valid. This way, blocks of a plane are written sequentially, and valid pages are initially placed in an unfragmented area of consecutive physical blocks starting from the first block of the plane. The remaining physical blocks comprise free pages only. Initially, the first $1 - S_f$ fraction of plane blocks will be written, and the remaining S_f portion will be free due to overprovisioning. Microbenchmarking is also used to gain deep insight into SSD behavior. To this end, we use synthetic traces generated by the well-known DiskSim trace generator [Bucy et al. 2008].

4. DYNAMIC PAGE ALLOCATION

The *PLAlloc* primitive determines the target CID, WID, DID, and PID of a page-sized flash transaction following a specific priority order of the mentioned four parallelism levels. This prioritization strategy dictates the SSD behavior under various I/O access

[2]24KiB limit for page size equals three pages in both NAND flash chips (Table III). Our empirical investigations show that workloads with an average request size of three pages or less behave almost similarly with respect to various allocation strategies.

patterns and affects the chance of parallel accesses by specifying the method used to distribute transactions over SSD resources. Let us return to the I/O handling scenario of the SSD illustrated in Figure 2. As we stated previously, a host I/O request is handled through multiple page-sized flash transactions. If *PLAlloc* properly stripes the LPAs over channels, ways, dies, and planes, then the TSU can simultaneously conduct transactions of an I/O request to achieve lower RTs. This type of usage is known as *intrarequest* parallelism [Jung and Kandemir 2012]. In addition, if we consider the whole set of queued I/O requests, a proper striping scheme further helps to concurrently manage flash transactions of consecutive I/O requests and reduces their average waiting time. This is called *interrequest* parallelism [Jung and Kandemir 2012] and leads to further improvement of overall IOPS and RT.

Unfortunately, making an appropriate decision for *PLAlloc* prioritization is not trivial, as each choice acts in favor of some parallelism levels and diminishes the chance of utilizing others. Worse, some of the parallelism levels, such as plane level, have inherent restrictions and require extra design considerations to achieve reasonable performance [Jung et al. 2012]. Hence, prioritizing the plane level may even lead to lower performance if not managed properly [Jung et al. 2012].

ALGORITHM 1: CWDP Allocation Strategy

```
function PLAlloc(transaction)
    LPA ← transaction.LPA
    transaction.CID ← LPA%C
    transaction.WID ← (LPA/C)%W
    transaction.DID ← (LPA/(C × W))%D
    transaction.PID ← (LPA/(C × W × D))%P
```

ALGORITHM 2: DPWC Allocation Strategy

```
function PLAlloc(transaction)
    LPA ← transaction.LPA
    transaction.DID ← LPA%D
    transaction.PID ← (LPA/D)%P
    transaction.WID ← (LPA/(D × P))%W
    transaction.CID ← (LPA/(D × P × W))%C
```

4.1. Traditional Methods of Page Allocation

Most of the previous studies concentrated on *static PLAlloc* strategies [Shin et al. 2009; Hu et al. 2011; Jung and Kandemir 2012; Hu et al. 2013], where CID, WID, DID, and PID are calculated by successive divisions of the LPA. Algorithms 1 and 2 highlight this process for two sample strategies: CWDP and DPWC. CWDP, at the first step, uses the reminder of dividing the LPA to the number of channels (C) to determine the target CID; the quotient is further used for WID calculation, and the process is repeated for assigning DID and PID in the next steps. In a similar way, calculations in DPWC take place in the order of DID, PID, WID, and CID. Generally, a static allocation strategy can be represented by any of the $4! = 24$ different combinations as follows:

$$L_1L_2L_3L_4, \forall i, L_i \in \{C, W, D, P\} \quad for \quad L_i \neq L_j, \quad i \neq j. \tag{1}$$

The order of the letters in this notation (from left to right) determines the order of calculations and reflects the address striping strategy. Figure 3 shows an example of the behavior of different allocation strategies in a typical SSD with four channels, two ways (flash chips) per channel, two dies per flash chip, and two planes per die. The ID of each resource within its container is depicted in Figure 3(a), whereas Figure 3(b) shows the outcome of address translation using the CWDP strategy. As can be seen in the figure, this strategy first stripes logical addresses over channels ($[LPA = 0]$ is assigned to $[C\#0]$, $[LPA = 1]$ is assigned to $[C\#1]$, etc.) and can decrease the probability of intrarequest contention over this critical resource type [Hu et al. 2013]. However, due to its static nature, this strategy cannot resolve some of the resource conflicts. For example, both $[LPA = 0]$ and $[LPA = 32]$ are mapped to $[C\#0, W\#0, D\#0, P\#0]$, and hence one program operation should wait until the accomplishment of the other while there are still free resources available. The other three unresolvable contentions are highlighted with red color in Figure 3(b).

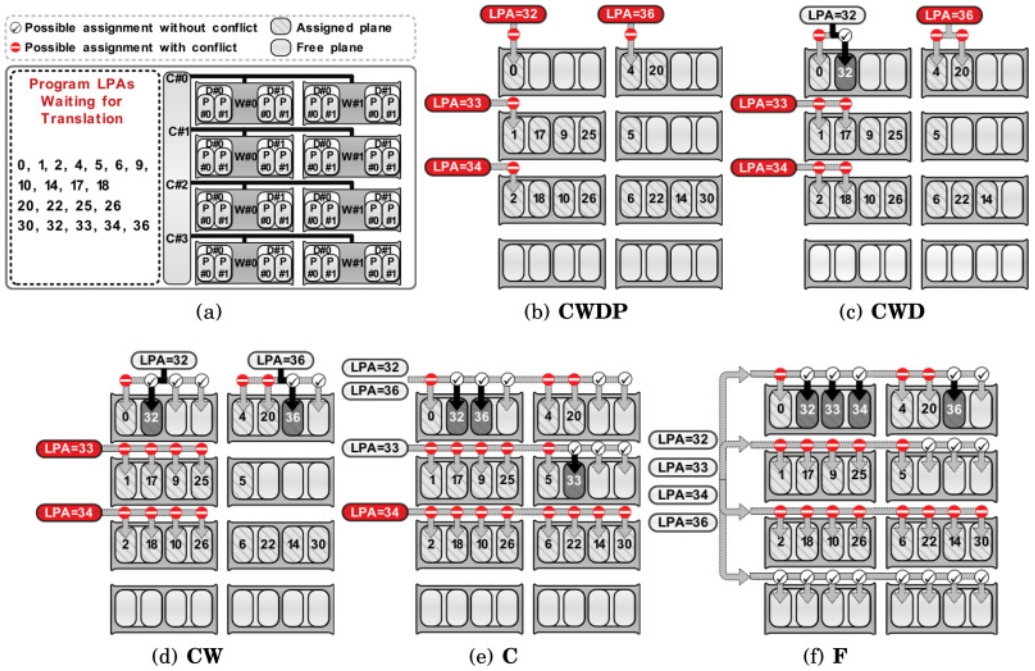


Fig. 3. Example scenario that shows the behavior of different allocation strategies. (a) IDs of SSD resources within their container and the list of current program LPAs, waiting to be translated to PPAs. (b) Result of the CWDP strategy with four unresolvable conflicts. (c) CWD provides freedom in assigning PID and successfully resolves resource conflict for [$LPA = 32$]. (d) CW provides freedom in assigning PID and DID and resolves conflicts for both [$LPA = 32$] and [$LPA = 36$]. (e) C provides freedom in assigning PID, DID, and WID, and all resources are successfully assigned except for [$LPA = 34$]. (f) F provides degree-4 freedom where all resources are dynamically allocated and all conflicts are resolved.

4.2. Use of Dynamism in Page Allocation

The method of calculating resource IDs in static approaches makes their write performance highly dependent on LPA access patterns. As an alternative, we suggest relaxing deterministic assignment of resource ID(s) in one or more levels of parallelism and using dynamic assignment instead. Dynamism may be provided by either round-robin or load-aware schemes. We use a mixture of round-robin and load-aware schemes in which round-robin allocation is normally used, but if a resource is busy during its turn, then the token is passed to the next resource at the same level. For instance, in dynamic CID assignment, if the FTL finds that the current channel (e.g., [C#2]) is busy, then it tries to assign the next channel (e.g., [C#3]) for LPA translation. This strategy results in better striping and resource utilization, especially in resource-limited client and midrange SSDs.

Before presenting our new dynamic *PLAlloc* strategies, we define the notion of *degree of freedom* as the number of resource IDs that are dynamically allocated. For example, a strategy has a freedom degree of one (shortly, degree-1) when it allocates one of resource IDs (CID, WID, DID, or PID) based on the mentioned load-aware round-robin model, and all others are statically determined. According to this definition, 41 dynamic allocation strategies are possible, which are classified into four categories by considering their degree of freedom.

Degree-1. Just one resource ID is determined dynamically, and other IDs are assigned in the same way done by the static approach. These strategies are represented as

follows:

$$L_1L_2L_3, \forall i, L_i \in \{C, W, D, P\} \quad for \quad L_i \neq L_j, \quad i \neq j. \tag{2}$$

The letters show the statically allocated resources, and their order (from left to right) determines the priority of calculations. Based on this definition, we have $4 \times 3!$ new allocation strategies. For instance, CWD is a degree-1 dynamic strategy that provides freedom in PID allocation. Figure 3(c) depicts the outcome of address translation using CWD for our example scenario of Figure 3(a). As can be seen, this strategy successfully allocates resources for [$LPA = 32$] with the help of dynamic PID allocation. Nonetheless, it cannot resolve the other three conflicts, as there are no free planes within target dies. For instance, [$LPA = 36$] is remained unassigned in the die address [C#0, W#1, D#0] as planes [P#0] and [P#1] are dedicated for execution of program operations at [$LPA = 4$] and [$LPA = 20$], respectively.

Degree-2. Two resource IDs are determined dynamically, and others are extracted statically. These allocation strategies are represented as follows:

$$L_1L_2, \forall i, L_i \in \{C, W, D, P\} \quad for \quad L_1 \neq L_2. \tag{3}$$

There are 12 *PLAlloc* strategies in this category. Since more than one resource ID is determined dynamically, dynamic allocation must be performed based on a priority order of parallelism levels. In Section 4.3, we discuss the prioritization policy and its implementation issues. Figure 3(d) shows how CW, as a degree-2 strategy, resolves contentions for program operations at [$LPA = 32$] and [$LPA = 36$] via dynamic die and plane allocation. For each of these LPAs, four different dynamic choices are provided. Among these choices, [$LPA = 32$] can be assigned to one of the three free planes at [C#0, W#0, D#0, P#1], [C#0, W#0, D#1, P#0], or [C#0, W#0, D#1, P#1], whereas for [$LPA = 36$], either of two free planes at [C#0, W#1, D#1, P#0] or [C#0, W#1, D#1, P#1] can be used. However, there exists no contention-free assignment for program operations at [$LPA = 33$] and [$LPA = 34$].

Degree-3. Only one resource ID is statically calculated. There are four strategies in this category, which are represented as follows:

$$L_1, L_1 \in \{C, W, D, P\}. \tag{4}$$

Figure 3(e) shows how C, as a degree-3 strategy, provides 8 different possible choices for each of unallocated LPAs in Figure 3(b). This way, translation is successfully performed for [$LPA = 32$], [$LPA = 33$] and [$LPA = 36$] but [$LPA = 34$] remains unallocated.

Degree-4. There is one *PLAlloc* strategy, namely F (fully dynamic), in which all resource IDs are dynamically assigned. This strategy was previously presented as the only choice of dynamic allocation [Park et al. 2010b; Hu et al. 2013]. As shown in Figure 3(f), this strategy can assign an LPA to any free plane within the SSD, and thus all resource contentions are successfully resolved. According to our discussions about the impact of dynamism on resolving resource conflicts among program operations, one might deduce that degree-4 strategy may provide the best performance. However, when considering the impact of allocation strategy on overall SSD performance, there are other critical issues that must be taken into account. Strictly speaking, F may hardly satisfy constraints of plane-level parallelism for both read and program operations, and as empirically shown in Section 5, utilization of intrachip parallelism levels in F is lower than that of many other dynamic allocation strategies. Moreover, F has no advantage over other dynamic strategies regarding read performance. In fact, dynamism generally helps to exploit parallelism just for program operations, but future read operations may follow different LPA access patterns, and hence dynamism may not lead to proper speedup for reads.

ALGORITHM 3: General Scheme of *PLAlloc* for Dynamic Allocation Strategies

```
function PLAlloc(transaction)
    LPA ← transaction.LPA
    if static CID allocation is used then
        transaction.CID ← MathematicalCIDCalculation(LPA)
    if static WID is used then
        transaction.WID ← MathematicalWIDCalculation(LPA)
    if static DID is used then
        transaction.DID ← MathematicalDIDCalculation(LPA)
    if static PID is used then
        transaction.PID ← MathematicalPIDCalculation(LPA)
```

ALGORITHM 4: General Scheme of Dynamic Resource Allocation Performed in the TSU

```
function TSU_DynamicAllocation(transaction)
    if static CID allocation is used then
        AssignWay(transaction)
    else
        while translation is not successful and all channels are not checked do
            if Channels[NextChannel] is idle then
                transaction.CID ← NextChannel
                AssignWay(transaction)
            NextChannel ← (NextChannel + 1) mod C                ▷ round-robin update of next channel

function AssignWay(transaction)
    if static WID allocation is used then
        AssignDie(transaction)
    else
        Channel ← Channels[transaction.CID]
        while translation is not successful and all chips of Channel are not checked do
            if Channel.Chips[Channel.NextWay] is idle then
                transaction.WID ← Channel.NextWay
                AssignDie(transaction)
            if translation is not successful or striping is fully performed inside Channel.Chips[Channel.NextWay]
then
                Channel.NextWay ← (Channel.NextWay + 1) mod W    ▷ round-robin update of next way

function AssignDie(transaction)
    if static DID allocation is used then
        AssignPlane(transaction)
    else
        Chip ← Channels[transaction.CID].Chips[transaction.WID]
        while translation is not successful and all dies of Chip are not checked do
            if Chip.Dies[Chip.NextDie] is free then
                transaction.DID ← Chip.NextDie
                AssignPlane(transaction)
            Chip.NextDie ← (Chip.NextDie + 1) mod D              ▷ round-robin update of next die

function AssignPlane(transaction)
    if static PID allocation is used then
        if multiplane constraint is satisfied for transaction.DID and transaction.PID then
            transaction.PPA ← BLAlloc(transaction)
    else
        Die ← Channels[transaction.CID].Chips[transaction.WID].Dies[transaction.DID]
        while translation is not successful and all planes of Die are not checked do
            if Die.Planes[Die.NextPlane] is free and multiplane constraint is satisfied then
                transaction.PID ← Die.NextPlane
                transaction.PPA ← BLAlloc(transaction)
            Die.NextPlane ← (Die.NextPlane + 1) mod P            ▷ round-robin update of next plane
```

ALGORITHM 5: Realization of CD Allocation Strategy Based on Algorithms 3 and 4

```
function PLAlloc(transaction)
    transaction.CID ← transaction.LPA%C
    transaction.DID ← (transaction.LPA/C)%D

function TSU_DynamicAllocation(transaction)
    Channel ← Channels[transaction.CID]
    while translation is not successful and all chips of Channel are not checked do
        if Channel.Chips[Channel.NextWay] is idle then
            transaction.WID ← Channel.NextWay
            Die ← Channel.Chips[transaction.WID].Dies[transaction.DID]
            while translation is not successful and all planes of Die are not checked do
                if Die.Planes[Die.NextPlane] is free and multiplane constraint is satisfied then
                    transaction.PID ← Die.NextPlane
                    transaction.PPA ← BLAlloc(transaction)
                Die.NextPlane ← (Die.NextPlane + 1) mod P
        if translation is not successful or striping is fully performed inside Channel.Chips[Channel.NextWay] then
            Channel.NextWay ← (Channel.NextWay + 1) mod W
```

4.3. Implementing Dynamic Allocation

As mentioned in Section 2.2, the TSU is responsible for resolving resource contentions among flash transactions to simultaneously conduct multiple operations in parallel and achieve high overall performance. As a prerequisite, page allocation strategy should fairly stripe flash transactions over SSD parallel resources to aid the TSU by reducing the contention probability and increasing the chance of parallel operation execution. Consequently, the address striping policy and priority order of parallelism levels are critical for effective exploitation of parallelism. For a dynamic allocation strategy, Hu et al. [2011, 2013] theoretically and experimentally concluded that the most beneficial priority order of parallelism levels is channel first, die second, plane third, and way as the last one. Therefore, we use the same order of parallelism levels for our proposed dynamic allocation strategies of degree-2, degree-3, and degree-4.

Since a dynamic allocation strategy requires being aware of the current status of SSD resources, dynamic allocation of CID, WID, DID, and PID can be performed inside the TSU. This is simply done in conjunction with normal TSU tasks at the cost of a few more instructions, which is negligible when compared to complicated scheduling mechanisms required for effective usage of parallel resources [Park et al. 2010b; Jung et al. 2012; Jung and Kandemir 2014]. Therefore, to realize a dynamic strategy, *PLAlloc* just calculates the statically allocated resource IDs whereas all dynamic allocations are performed inside the TSU. Algorithms 3 and 4 illustrate the general scheme of *PLAlloc* and the TSU for dynamic allocation strategies, respectively. During dynamic allocation, resource IDs are determined in the order of channel, way, die, and plane, since IDs of lower-level resources are defined within the address domain of higher-level container resources. For instance, there are different dies that have the same die address of [*D*#0] (see Figure 3(a)), each belonging to a different flash chip. It is worth noting that this order of resource ID allocation does not modify the mentioned priority order of parallelism levels. For instance, LPA striping is still performed in the order of channel, die, plane, and way despite the mentioned channel, way, die, and plane order for resource ID allocation. This is simply accomplished by adding a single condition at the end of the AssignWay function in such a way that *NextWay* of a channel is not updated until LPA striping is fully performed over dies and planes of the current way (flash chip). Strictly speaking, the priority order of parallelism levels is directly related to the update policy of the corresponding round-robin variables (i.e., *NextChannel*, *NextWay*, *NextDie*, and *NextPlane*). If these variables are updated without any restriction, then the priority order of parallelism levels is directly determined based on the resource allocation order. However, restricting the update of a higher-level round-robin variable, through a

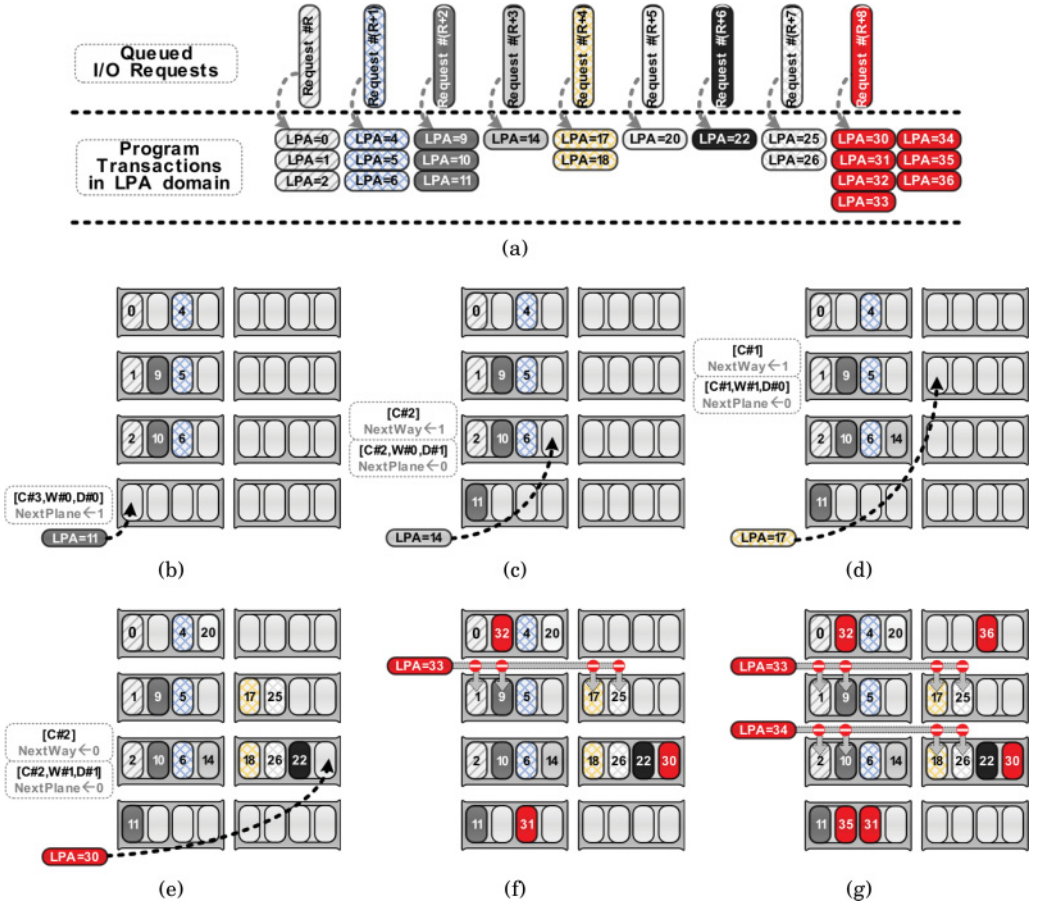


Fig. 4. Example scenario that shows snapshots of CD behavior in an SSD configuration similar to Figure 3(a). (a) Queued write I/O requests and their corresponding flash program transactions. (b) After translation of [$LPA = 11$], $NextPlane$ of die [$C\#3$, $W\#0$, $D\#0$] is updated; this type of update is seen for all program operations. (c) After translation of [$LPA = 14$], all parallelism levels inside flash chip [$C\#2$, $W\#0$] are completely utilized; consequently, $NextWay$ of channel [$C\#2$] is updated. (d) There is no free plane in die [$C\#1$, $W\#0$, $D\#0$] to be allocated to [$LPA = 17$]; hence, an early update of $NextWay$ variable is required. (e) After translation of [$LPA = 30$], $NextWay$ of channel [$C\#2$] is updated due to full utilization of all parallelism levels inside flash chip [$C\#2$, $W\#1$]. (f) Unsuccessful translation of [$LPA = 33$] due to resource conflicts. (g) Final result of the translation process that includes two unallocated LPAs.

conditional statement, reduces the priority order of its corresponding parallelism level. Accordingly, the least priority is given to way-level address striping, as updating of the $NextWay$ variable is controlled through a condition that checks for complete striping of LPAs over dies and planes within the current way (flash chip). Algorithms 3 and 4, however, should be customized for realization of a specific dynamic allocation strategy. For instance, the whole steps of *PLAlloc* and the TSU for LPA to PPA translation are reduced to the lines depicted in Algorithm 5 for implementation of CD strategy.

To better describe the behavior of a dynamic allocation strategy and clarify the critical role of round-robin variables, Figure 4 provides an example scenario of some waiting I/O requests and the outcomes of the address translation process using CD strategy. As illustrated in Figure 4(a), there are nine queued I/O requests, which are processed from left to right (from Request [$\#R$] to Request [$\#(R + 8)$]) and are segmented to page-sized

transactions, each having a specific LPA address. In the remainder, we just concentrate on important snapshots of the translation process and escape the other parts for brevity. Figure 4(b) shows the snapshot at which CD strategy determines the physical address for [$LPA = 11$]. First, CID and DID are mathematically calculated using the equations 35 provided in Algorithm 5. This way, the channel [$C\#3$] ($11\%4 = 3$) and die [$D\#0$] ($11/4\%2 = 0$) are assigned in the *PLAlloc* function. Then, flash chip [$C\#3$, $W\#0$] is selected based on the current value of $NextWay$ dedicated to channel [$C\#3$]. Finally, PID is determined using $NextPlane$ of die [$C\#3$, $W\#0$, $D\#0$], and hence [$LPA = 11$] is mapped to the plane [$C\#3$, $W\#0$, $D\#0$, $P\#0$]. Variable $NextPlane$ is updated right after PID allocation to record the next plane of die [$C\#3$, $W\#0$, $D\#0$]. Contrarily, $NextWay$ remains unchanged due to the lower priority of way-level parallelism. In fact, CD strategy waits for full LPA distribution over dies and planes inside flash chip [$C\#3$, $W\#0$] before using the neighbor flash chip at [$C\#3$, $W\#1$]. The next snapshot of address translation is depicted in Figure 4(c), where the $NextWay$ normal update condition is satisfied. As can be seen, [$LPA = 14$] is mapped to the plane at [$C\#2$, $W\#0$, $D\#1$, $P\#1$], which is the last free plane inside flash chip [$C\#2$, $W\#0$]. Therefore, the corresponding $NextWay$ value is updated to point to the next flash chip inside channel [$C\#2$] (i.e., flash chip [$C\#2$, $W\#1$]). Thus far, the translation process is successfully performed for all transactions of Request [$\#R$] to Request [$\#(R+3)$]. Figure 4(d) shows the translation process for [$LPA = 17$], which is dedicated to the next request in the queue. Here, the process leads to a situation where an early update of $NextWay$ is required despite the incomplete utilization of the flash chip internal parallelism. In fact, mathematical calculations lead to channel [$C\#1$] and die [$D\#0$] for physical mapping of [$LPA = 17$]. A complete search within the current flash chip of channel [$C\#1$] (i.e., chip [$C\#1$, $W\#0$]) reveals that all planes of die [$C\#1$, $W\#0$, $D\#0$] are occupied. This search is performed in the inner loop of the $TSU\_DynamicAllocation$ function proposed in Algorithm 5. To perform a successful LPA translation, an early update of $NextWay$ needs to occur in the outer loop of this function, and flash chip [$C\#1$, $W\#1$] is now dedicated for program LPA translation. The translation process consequently assigns [$LPA = 17$] to the free plane at [$C\#1$, $W\#1$, $D\#0$, $P\#0$]. Figure 4(e) depicts another snapshot of the translation process where $NextWay$ of channel [$C\#2$] is normally updated due to complete striping inside flash chip [$C\#2$, $W\#1$]. However, the new flash chip [$C\#2$, $W\#0$] is fully occupied and cannot service program operations directed toward channel [$C\#0$]. Figure 4(f) shows a snapshot of address translation where unresolvable resource contention occurs for [$LPA = 33$]. This LPA must be allocated to channel [$C\#1$] and die [$D\#0$]. But the dies at [$C\#1$, $W\#0$, $D\#0$] and [$C\#1$, $W\#1$, $D\#0$] are fully occupied, and the program transaction related to [$LPA = 33$] must wait until the completion of program operations at one of these dies. Figure 4(g) illustrates the final outcomes of address translation and the two unresolvable contentions related to Request [$\#(R + 8)$].

4.4. Mapping Table Size

Considering the I/O handling scenario described in Figure 2, a dynamic allocation strategy may increase the size of the mapping table. In fact, the FTL has to store any dynamically allocated resource ID together with the outcome of *BLAlloc* (i.e., BID and PaID) in DRAM to be able to retrieve the corresponding PPA in future read address translations. However, if static allocation is used, only BID and PaID must be stored, and all other resource IDs can be simply retrieved through mathematical calculations. Suppose that there are a total of c channels in the SSD, w flash chips in each channel, d dies in each flash chip, p planes within a die, b blocks within a plane, and k pages in each block. Then, each entry in the mapping table of a static allocation strategy requires $\log b + \log k$ bits, hereafter referred as *base*. In a dynamic strategy with degree-1 freedom, the size of the mapping table entry is correspondingly increased

Table V. Implementation Cost of Dynamic Allocation Strategies for SSD-MLC and SSD-SLC in Terms of Extra Mapping Table Size

		Static	Degree-1				Degree-2						Degree-3				F
Dynamic Allocated	CID		✓				✓	✓	✓				✓	✓	✓		✓
	WID			✓			✓			✓	✓		✓	✓		✓	✓
	DID				✓			✓		✓		✓	✓		✓	✓	✓
	PID					✓			✓		✓	✓		✓	✓	✓	✓
SSD-MLC																	
Bit increase (%)		—	11	11	11	5	21	21	16	21	16	16	32	26	26	26	37
Byte increase (%)		—	0	0	0	0	0	0	0	0	0	0	33	0	0	0	33
Map size (MB)		176	176	176	176	176	176	176	176	176	176	176	234	176	176	176	234
SSD-SLC																	
Bit increase (%)		—	17	17	11	6	33	28	22	28	22	17	44	39	33	33	50
Byte increase (%)		—	0	0	0	0	0	0	0	0	0	0	33	33	0	0	33
Map size (MB)		256	258	258	258	258	258	258	258	258	258	258	344	344	258	258	344

Note: Mos dynamic strategies incur zero byte-level overhead.

to $base + \log c$, $base + \log w$, $base + \log d$, or $base + \log p$ to store the information of the sole dynamically allocated resource ID. This way, F strategy requires a mapping entry size of up to $base + \log c + \log w + \log d + \log p$ bits for storing CID, WID, DID, and PID, as well as BID and PaID. To have a numerical estimation of the dynamism effects on the size of the mapping table, we consider the two target SSD configurations introduced in Table III; based on the specifications, the *base* (static allocation entry) size for SSD-MLC and SSD-SLC is equal to 19 bits and 18 bits, respectively. Nevertheless, mapping table entries are stored in a word (multiple bytes), and therefore it is more realistic to concentrate on a byte-level estimation. Accordingly, in both configurations, each table entry requires 3B of storage, and thus a total space of 176MB and 256MB must be dedicated for storing the mapping table of static allocations in SSD-MLC and SSD-SLC, respectively. Table V presents the mapping table size for different degrees of freedom and different types of dynamically allocated resources. As results show, the overhead in terms of the number of bits is moderate and varies between 5% and 37% for SSD-MLC and 6% and 50% for SSD-SLC. However, more realistic byte-level estimations show lower or even no overhead of mapping table size. As illustrated in Table V, a 33% memory space overhead is incurred only for P and F strategies in SSD-MLC and for P, D, and F strategies in SSD-SLC. Interestingly, all other dynamic allocation strategies do not require extra DRAM space for mapping table storage. It is also noteworthy that different mapping table caching mechanisms can be used to substantially decrease the required DRAM space of dynamic strategies [Wu et al. 2006; Gupta et al. 2009; Park et al. 2010a; Hu et al. 2010; Budilovsky et al. 2011]. As proposed Gupta et al. [2009], the temporal locality of real workloads can be exploited to implement a demand-based caching of mapping entries. This way, entries are mainly stored on flash storage, and only the hot (recently accessed) ones are kept inside a small portion of DRAM. Even if access locality is not seen in the workloads, other complementary mechanisms, such as HAT [Hu et al. 2010], can be used to prevent performance degradation of demand-based caching. HAT accelerates accesses to the missing mapping entries through a dedicated access path to a solid-state memory device exclusively used for mapping table storage. On the other hand, the large and inexpensive host DRAM memory can be used to cache mapping entries in low-cost SSDs similar to SSD-MLC [Budilovsky et al. 2011].

5. THE ROLE OF DYNAMISM FOR BETTER EXPLOITING PARALLELISM

In this section, we analyze the performance of 41 dynamic allocation strategies versus 24 static ones for SSD-MLC and SSD-SLC. First, we provide a detailed explanation of

the metrics used in our evaluations, and in the next two sections, simulation results are thoroughly discussed for page allocation strategies in SSD-MLC and SSD-SLC. In our discussions, we first investigate for the impact of allocation strategies on the fairness of flash operation distribution over the whole set of planes within the SSD (fairness, for short). The fairness metric is the *standard deviation of the number of executed flash read/program operations* over SSD planes. The lower values for standard deviation show that the allocation strategy better stripes flash operations over SSD resources and hence the resource utilization, and the chance of parallel operation execution are improved. Figure 5 shows the achieved fairness results for the workload categories mentioned in Table IV. For the sake of comparison, the results of *PLAlloc* strategies for a specific workload (e.g., `fin2`) are normalized with respect to those of CWDP strategy (the left-most item). Then, for the set of workloads within a category, the average of the normalized values is illustrated. The same method is also used for data presentation in Figures 6, 7, and 8.

We then provide details of the executed die-interleaved and multiplane operations in Figure 6 to help studying the impact of allocation strategies on the utilization of die- and plane-level parallelisms, respectively. Many previous studies have shown the critical role of these two parallelism levels to improve SSD performance [Park et al. 2010b; Jung and Kandemir 2012; Hu et al. 2013]. In fact, higher utilization of die- and plane-level parallelisms increases the opportunity for parallel I/O transactions on SSD back-end (flash) resources, and hence higher aggregate performance will be perceived at the front-end (host interface). For program operation, the plots show the percentage of exclusive interleaved, exclusive multiplane, and simultaneous interleaved and multiplane operations. The remaining portion of programs, which are absent in the plots, are handled through simple operations with no die- and plane-level parallelism. For read operations, however, there are just exclusive interleaved and exclusive multiplane operations.[3]

Next, we investigate the effects of dynamism on the waiting time of read/program operations, based on the results presented in Figure 7. This figure reveals how well the allocation strategy reduces the probability of resource contention. More precisely, when the TSU receives a flash operation, it waits until all resource conflicts at channel-, way-, die-, and plane level are resolved and then asks the FCC to initiate a communication with target die for sending operation command and data. Moreover, when a read operation finishes, the TSU waits for the corresponding communication channel to become free and then asks the FCC to transmit read data back to the controller. A good *PLAlloc* strategy decreases the average waiting time of flash operations through better striping of LPAs.

Finally, based on our discussions regarding Figures 5, 6, and 7, we analyze the performance results of SSD-MLC and SSD-SLC presented in Figure 8 in terms of maximum host IOPS and average RT. RT is defined as the time elapsed from I/O request arrival until the response is sent back via the host interface. For maximum IOPS calculation, the number of completed I/O operations is counted in the unit of time when the SSD is under a full stress condition. To reach this condition, the interarrival time of I/O events is ignored, and requests are successively serviced when there is at least one free slot in the I/O request queue. Variations of this methodology were used in previous studies for IOPS calculation [Narayanan et al. 2009; Wu and He 2012; Grupp et al. 2013], whereas exactly the same method was already used in Tavakkol et al. [2014]. Although the behavior of dynamic strategies in SSD-MLC and SSD-SLC is generally similar in terms of fairness and executed operation type, we see different behaviors for

[3]Since the read execution time is comparable to the flash data transfer time, simultaneous usage of interleaved and multiplane reads may not have noticeable performance gain.

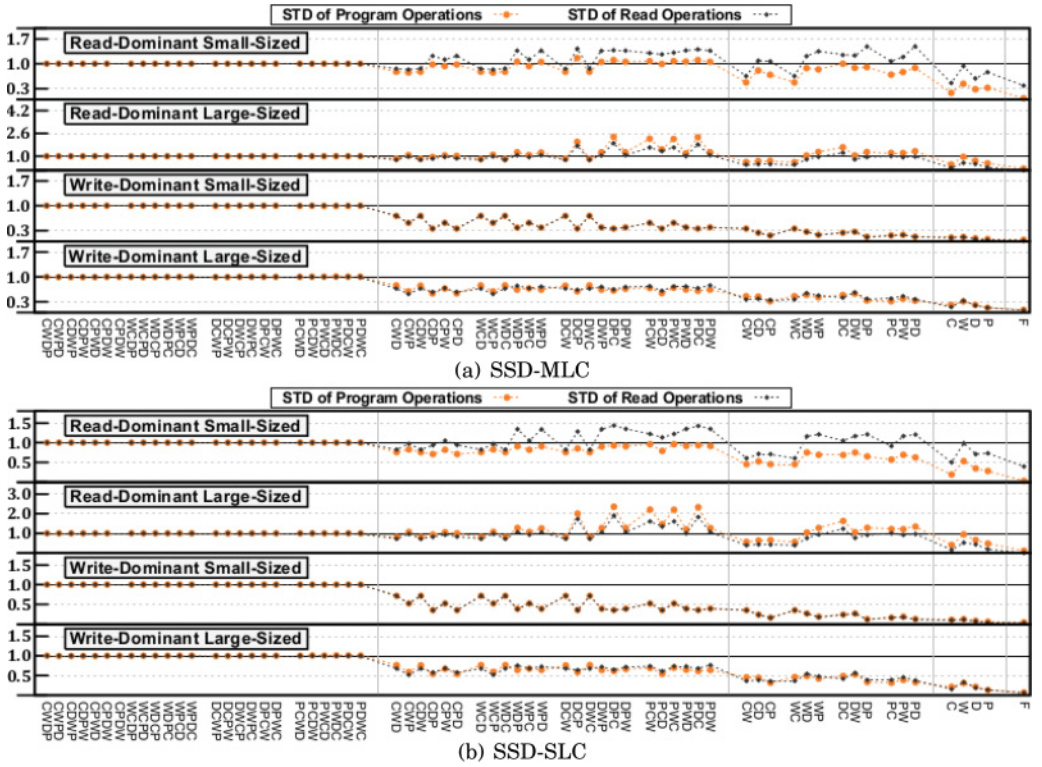


Fig. 5. Fairness of operation distribution among the whole set of planes inside the SSD in SSD-MLC (a) and SSD-SLC (b). The fairness metric is the standard deviation of the number of executed read/program operations per plane. For better comparison, the results are normalized with respect to the value of CWDP (left-most) strategy. Smaller normalized values are desirable, as they translate into better fairness, and thus degree-3 and degree-4 strategies are ranked first.

their waiting time and performance. Hence, simulation results are separately discussed for SSD-MLC and SSD-SLC in Sections 5.1 and 5.2, respectively.

5.1. Client and Midrange SSDs Based on MLC NAND Flash

Let us start with fairness results of SSD-MLC shown in Figure 5(a). As can be seen, there are minor variations in the outcomes of static allocation strategies because their behavior depends only on LPA access patterns, and it is the mathematical method of resource allocation that keeps fairness always the same. However, the results of dynamic strategies substantially change based on the degree of freedom and priority order of resource allocation. Despite undesirable results for some strategies with degree-1 or degree-2 freedom in read-dominant workloads, the fairness is improved for higher freedom degrees. In other words, increasing the degree of freedom greatly reduces the dependency of flash operation distribution to LPAs, and the round-robin busy-aware allocation approach has the opportunity to evenly distribute operations over SSD resources. In sum, F, C, and P strategies are always ranked first, whereas D, CW, and WC are ranked second regarding fairness of both programs and reads.

Figure 6(a) illustrates the types of executed operations in SSD-MLC. For static allocation strategies, there is a direct relation between the percentage of each operation type and priority order of resource assignment. Strictly speaking, the percentage of

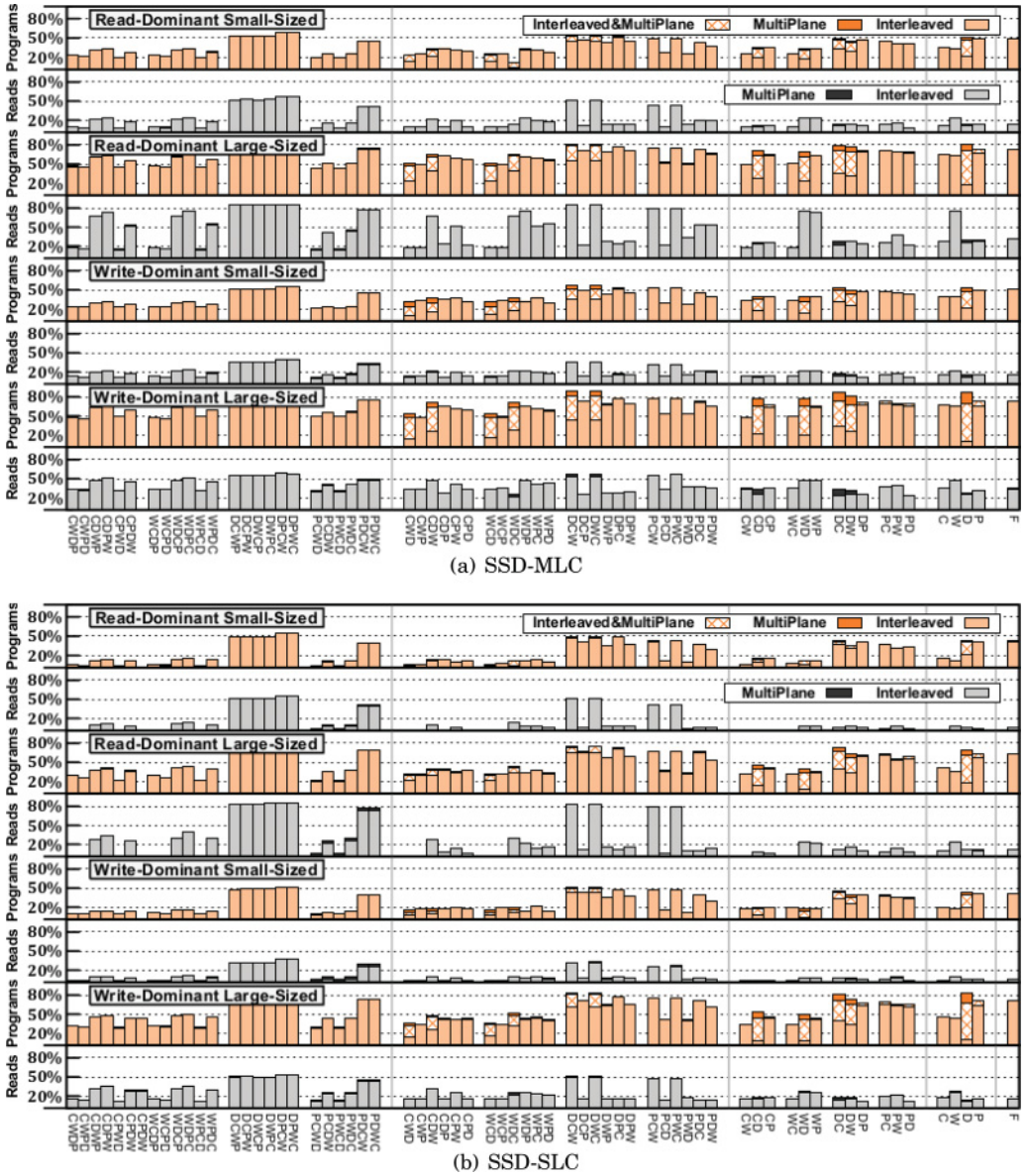


Fig. 6. Details of the executed operation types for different page allocation strategies in SSD-MLC (a) and SSD-SLC (b). The results show that two factors in dynamic strategies help the FTL to better exploit intrachip parallelism: static allocation of DID and increasing the degree of freedom. Thus, in most workload categories, D, DC, and DW strategies achieve the highest percentage of multiplane and interleaved operation execution.

interleaved/multiplane operations is increased when higher priority is given to DID/PID allocation. For example, a higher number of interleaved operations are executed in WPDC with respect to WPCD due to higher priority of DID allocation. Thus, allocation strategies such as DPWC and PDWC achieve the best results for interleaved and multiplane operations, respectively. In dynamic strategies, the ratio of each operation type is related to both priority of resource allocation and the degree of freedom:

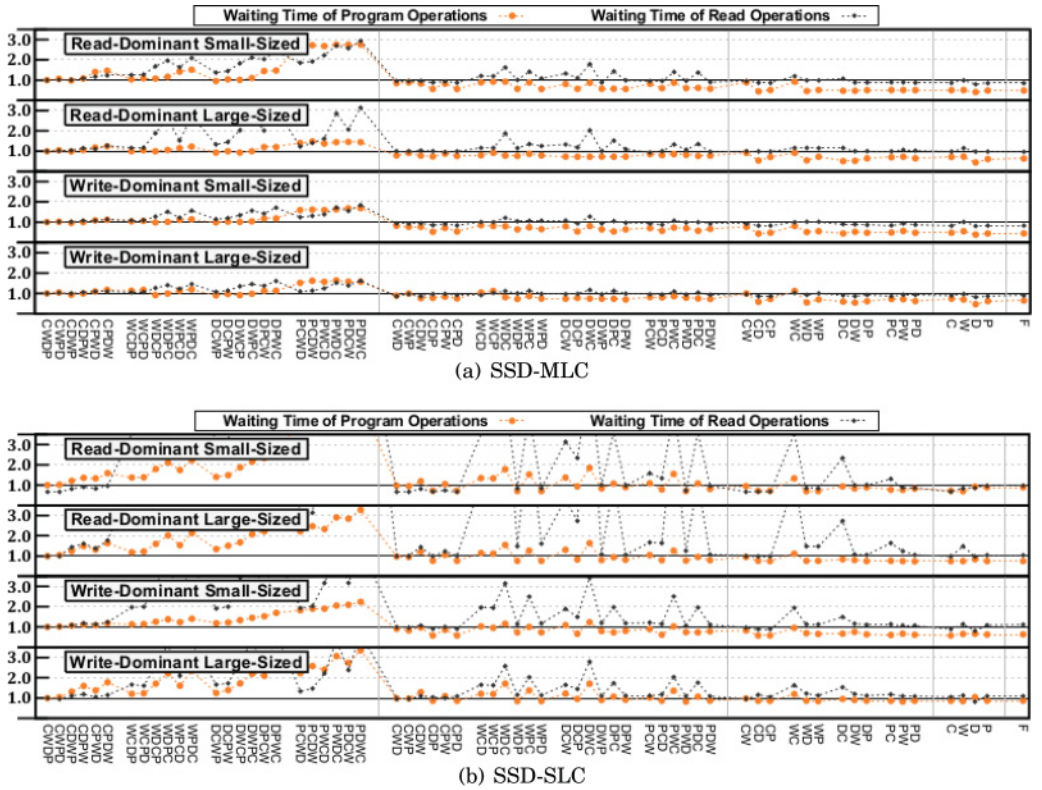


Fig. 7. Waiting time of flash operations in SSD-MLC (a) and SSD-SLC (b). The results are normalized in a similar manner to Figure 5. A small waiting time value is desirable, as it works in favor of better performance. For both SSDs, dynamism helps to reduce the waiting time of program operation while affecting read operation negligibly. Best results are achieved for degree-3 and degree-4 strategies.

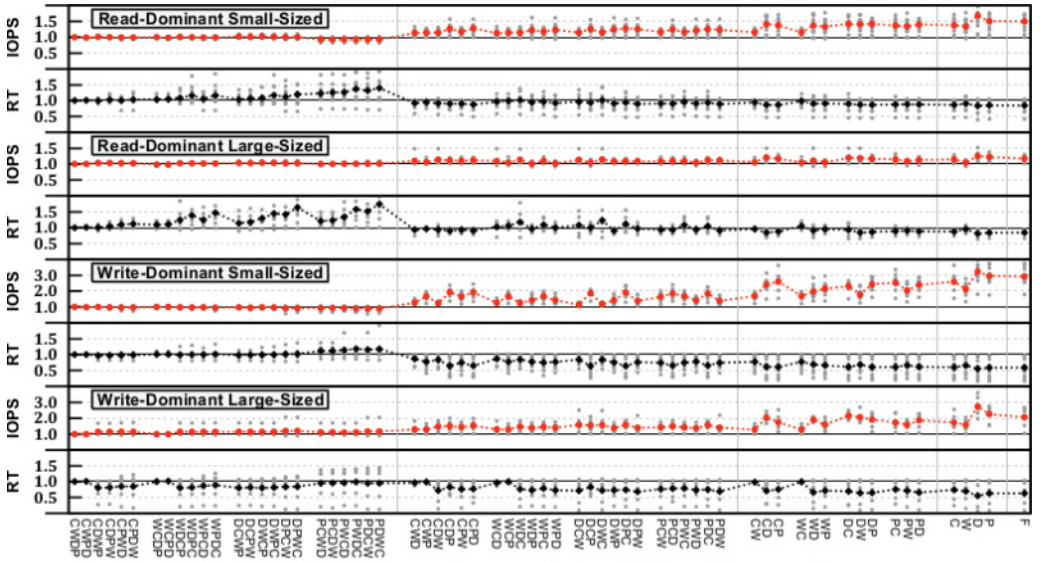
—*Multiplane operations*: As results show, the share of multiplane operations is enhanced if static DID calculation is used together with dynamic assignment of PID. Accordingly, strategies such as CWD, CDW, WCD, WDC, DCW, and DWC provide the best results for exploiting plane-level parallelism in the degree-1 category. Higher degrees of freedom can further improve the ratio of multiplane operations, and D strategy is ranked first among all dynamic strategies. The ratio of multiplane operations in F strategy is too small, as it gives low priority to dynamic PID allocation to achieve maximum fairness. Consequently, the chance of exploiting plane-level parallelism is significantly reduced in F.
—*Interleaved operations*: The results of interleaved operations reveal that if one of the DIDs and PIDs is calculated statically and at the same time the other one is assigned dynamically, then the ratio of interleaved operations is increased. If DID is statically calculated, then more improvement can be achieved through prioritizing DID calculation.

The waiting time results of SSD-MLC in Figure 7(a) show that an allocation strategy may undesirably obtain a large waiting time despite the fact that it provides good results of fairness and executed operation types. For instance, the fairness values for static strategies such as DPWC and PDWC are similar to CWDP, and they even achieve a higher percentage of interleaved and multiplane operations. Nonetheless, their

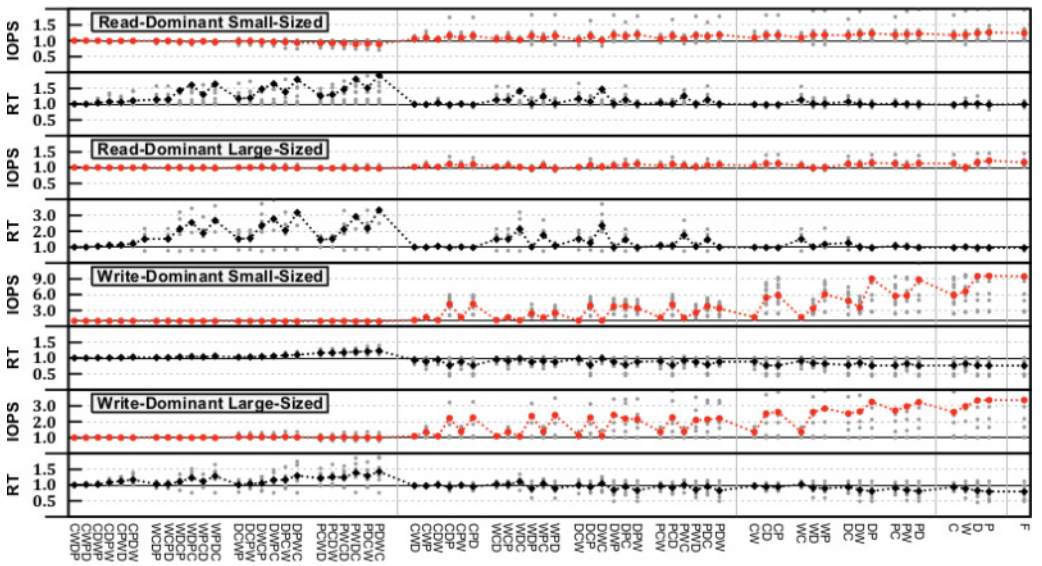
waiting time is extremely worse than that of CWDP due to their negative impacts on contentions at the channel domain. Strictly speaking, if an allocation strategy increases the probability of channel-level contention, then the achieved gains due to better fairness and higher ratio of interleaved and multiplane operations will completely vanish. Suppose that an I/O request including [$LPA = 0$] to [$LPA = 7$] is handled in our example SSD configuration of Figure 3(a). If the FTL uses DPWC and PDWC strategies, then all LPAs are mapped to [C#0], whereas using CWDP, they are equally striped over [C#0] to [C#3] and can better exploit SSD channels for increasing intrarequest parallelism and decreasing waiting time. In general, resolving contentions at the channel domain plays a key role in waiting time reduction, and simulation results show that giving higher priority to static allocation of CID, as well as dynamic allocation of CID, decreases the average waiting time. As a consequence, degree-2 strategies, such as CD and DW, and most degree-3 and degree-4 strategies provide the best waiting time results for both read and program operations. In sum, a good allocation strategy should minimize the probability of contention over SSD channels, and at the same time, it must increase the utilization of intrachip parallelisms (die- and plane level). We also see an interesting behavior in the plots. For instance, degree-1 and degree-2 strategies may increase the waiting time of read operation in read-dominant workloads but are usually helpful for program operation. This supports our discussion at the end of Section 4.2 on the role of dynamism for read and program operations due to dissimilarities in their LPA accesses. More precisely, there is no guarantee that future read requests follow the same LPA access pattern as in previous writes; as a result, new unresolvable resource contentions may be introduced for reads, particularly in degree-1 and degree-2 strategies with low-priority CID allocation. However, other dynamic strategies, such as D, P, F, C, CP, and CD, are always ranked first in terms of read waiting time, as they decrease the probability of channel-level contentions. As the ratio of write requests in the workload increases, these strategies can also indirectly reduce read waiting time through accelerating program operations and releasing resources required by read operations.

The performance results of SSD-MLC, shown in Figure 8(a), lead to the following key points. First, IOPS is greatly improved as a result of relaxing static resource ID assignment in one or more levels of parallelism. As we mentioned previously, dynamism helps to enhance resource utilization through improved fairness and increased percentage of interleaved and multiplane operations. Thus, the FTL can simultaneously conduct more flash operations on SSD resources to better exploit intra- and interrequest parallelism and handle more I/O requests in units of time. The results show that the maximum IOPS improvement is as high as 3.2x for write-dominant workloads, whereas it is limited to 1.7x for read-dominant ones. The allocation strategies with degree-3 freedom are always ranked first in terms of IOPS, whereas other dynamic strategies, such as F, CD, and DC, can also achieve very high IOPS. Second, dynamic allocation leads to great improvement for RT, particularly in write-dominant workloads. As stated before, dynamism can substantially reduce the average waiting time of program operations, whereas the improvement rate for read waiting time is less considerable, especially in read-dominant workloads. Consequently, a maximum of 44% RT improvement can be achieved in write-dominant workloads, whereas the improvement rate is limited to 18% for read-dominant ones.

The overall ranking process of allocation strategies can be considered a multiobjective decision problem where both low RT and high IOPS values are desirable. Such a problem is difficult to solve because the scalar concept of optimality cannot be applied for strategy ranking based on two different performance metrics. An allocation strategy may provide the best result for one metric, but it may worsen the other one. Hence, there is no trivial ranking methodology, and it is required to define a decision scheme for determining acceptable trade-off optimal strategies. The nature of this problem



(a) SSD-MLC, the best values are achieved for D and P strategies.



(b) SSD-SLC, the best values are achieved for F and P strategies

Fig. 8. Comparison of IOPS and average RT results under real workloads for SSD-MLC (a) and SSD-SLC (b). For each allocation strategy, gray points represent individual workload results being normalized with respect to the corresponding value of CWDP, whereas bold points provide the average of normalized values. Note that higher IOPS and lower RT values are desirable; plot areas are zoomed to clarify the differences among bold points, and hence a few gray points are not visible.

Table VI. Pareto Fronts of Allocation Strategies in Terms of IOPS and RT Criteria for SSD-MLC

	Rank-0	Rank-1	Optimal Set (IOPS/RT)
Read-Dominant Small-Sized	D, F, P	CD	CD(1.6x/17%), D(2.0x/17%), P(1.8x/17%), F(18x/17%)
Read-Dominant Large-Sized	D	CD, DW, P, F	CD(1.2x/16%), DW(1.2x/16%), D(1.3x/18%), P(1.2x/16%), F(1.2x/16%)
Write-Dominant Small-Sized	D	P, F	D(3.2x/44%), P(3.0x/41%), F(2.9x/41%)
Write-Dominant Large-Sized	D	DC, P, F	DC(2.2x/30%), D(2.8x/44%), P(2.2x/37%), F(2.1x/37%)

Note: For each workload category, the optimal set is defined to include members of both rank-0 and rank-1 Pareto fronts.

leads us to the widely used concept of *Pareto-optimality* [Marler and Arora 2004]. An allocation strategy is Pareto-optimal if all other strategies either have a worse value for at least one of the RT and IOPS criteria or have the same value for both. In this way, we will have a *Pareto front* set of optimal strategies that provide an appropriate trade-off between both metrics. To include a larger set of allocation strategies in our analysis, we can use the notion of Pareto ranking [Marler and Arora 2004]. More precisely, all Pareto-optimal frontiers receive a rank of zero (simply rank-0). Then, these rank-0 allocation strategies are temporarily removed from consideration and the Pareto front set is determined for the remaining ones. The strategies of the new Pareto front are given a rank of one (simply rank-1) and are considered the second-class strategies for optimal *PLAlloc* implementation.

For different workload categories, Table VI shows the list of allocation strategies that are on the Pareto fronts of rank-0 and rank-1. As can be seen, D is a rank-0 frontier for all workload categories and provides the best performance results among all allocation strategies. P and F strategies are on either of rank-0 or rank-1 frontiers for all workload categories and hence may be suitable choices for *PLAlloc*. For each workload category, we further define the *optimal set* of allocation strategies consisting of the members of both rank-0 and rank-1 Pareto fronts. Members of the optimal set can be considered as appropriate *PLAlloc* design choices for the corresponding workload category. For the read-dominant small-sized category, CD, D, P, and F strategies constitute the optimal set, and IOPS and RT can be improved by up to 2.0x and 17%, respectively. For read-dominant large-sized workloads, optimal set includes CD, DW, D, P, and F dynamic allocation strategies. A maximum of 1.3x IOPS improvement and 18% RT improvement is achieved for this workload category. The write-dominant small-sized workloads greatly benefit from dynamic resource allocation, and their optimal set includes D, P, and F strategies with maximum IOPS and RT improvements of 3.2x and 44%, respectively. Finally, for write-dominant large-sized workloads, DC, D, P, and F strategies form the optimal set with up to 2.8x IOPS and 44% RT improvements.

5.2. High-End SSDs Based on SLC NAND Flash

Figure 5(b) depicts the fairness results of *PLAlloc* strategies for the SSD-SLC configuration. The trend in the results is quite similar to that for SSD-MLC (Figure 5(a)) (i.e., fairness is proportionally improved when the freedom degree is increased). The best results are always achieved for F, C, P, and D strategies, whereas strategies with degree-2 freedom, such as CW, CD, CP, and WC are ranked second. In addition, the results of Figure 6 show a similar behavior in SSD-SLC and SSD-MLC for the type of executed operations. It is noteworthy that the share of multiplane operations is reduced in SSD-SLC because of its higher number of dies (256 vs. 64), which decreases the probability of finding flash operations within a die satisfying multiplane execution conditions.

Comparison of the waiting time results for SSD-SLC and SSD-MLC in Figure 7 shows that dynamism has less impact on program operation in SSD-SLC, whereas it may

Table VII. Pareto Fronts of Allocation Strategies in Terms of IOPS and RT Criteria for SSD-SLC

	Rank-0	Rank-1	Optimal Set (IOPS / RT)
Read-Dominant Small-Sized	P	CP, PD, C, F	CP(1.2x/2%), PD(1.2x/1%), C(1.2x/2%), P(1.3x/3%), F(1.2x/1%)
Read-Dominant Large-Sized	P, F	D	D(1.1x/4%), P(1.2x/4%), F(1.2x/5%)
Write-Dominant Small-Sized	P	PD, D, F	PD(8.9x/24%), D(9.5x/23%), P(9.6x/25%), F(9.5x/24%)
Write-Dominant Large-Sized	P, F	PD, D	PD(3.2x/19%), D(3.3x/17%), P(3.4x/20%), F(3.4x/20%)

Note: For each workload category, the optimal set is defined to include members of both rank-0 and rank-1 Pareto fronts.

negatively increase read waiting time of many degree-1 and degree-2 strategies, especially in read-dominant workloads. However, read/program waiting time may desirably be improved when CID allocation is prioritized over other resources, as in CWD, CWP, CW, CD, CP, C, and D. In short, availability of channel-level and way-level resources in this high-end configuration reduces the probability of intrarequest resource contention, and hence dynamism has less chance of enhancing LPA striping with respect to static strategies. Therefore, the maximum RT improvement, shown in Figure 8(b), is almost inconsiderable in read-dominant workloads, but it is increased to 25% for write-dominant ones. On the other hand, the better fairness and die- and plane-level parallelisms in dynamic allocation strategies can be used in favor of efficient utilization of highly parallel architecture of SSD-SLC to improve IOPS in write-dominant workloads. The maximum IOPS is thus surprisingly improved by up to 9.5x for write-dominant workloads, whereas it is limited to 1.2x for read-dominant ones.

Table VII provides the summary of SSD-SLC results in terms of Pareto fronts of rank-0 and rank-1 and the optimal set of allocation strategies. Here, we separately describe the results for each workload category. The optimal set of read-dominant small-sized workloads includes CP, PD, C, P, and F strategies, and the maximum IOPS and RT improvement rates are limited to 1.3x and 3%, respectively. For the read-dominant large-sized category, D, P, and F strategies constitute the optimal set, and the maximum IOPS and RT improvements are as small as 1.2x and 5%, respectively. On the contrary, the write-dominant small-sized workloads greatly benefit from dynamic allocation. In this category, the maximum gain of using dynamism is as high as 9.6x for IOPS and 25% for RT, and the optimal set includes PD, D, P, and F strategies. Finally, the optimal set for write-dominant large-sized workloads includes PD, D, P, and F strategies with up to 3.4x IOPS and 20% RT improvement.

6. THE STEADY-STATE BEHAVIOR OF ALLOCATION STRATEGIES

Thus far, we have analyzed the effects of dynamic allocation strategies with no concern on GC. Such an evaluation, however, is not thorough, as SSDs spend a major fraction of their lifetime in the presence of GC. Many previous studies have shown the negative impacts of GC execution on both I/O performance and flash endurance [Bux et al. 2012; Lee et al. 2013; Van Houdt 2013b; Iliadis 2014; Van Houdt 2014]. Our experiments also show that in write-dominant workloads, GC can block numerous read/program operations, and once a GC starts, the average operation waiting time and RT are increased. In addition, GC overheads reduce the available bandwidth, especially in resource-limited SSD-MLC, and IOPS may be decreased. In this section, we take into account the interaction between allocation strategies and GC to make our findings finer, especially for write-dominant workloads. To this end, we investigate the impact of GC on the performance of SSD-MLC and SSD-SLC in Section 6.1. In particular, we show how much of the GC overheads can be eliminated using dynamic resource allocation. Next, in Section 6.2, we study the impact of dynamism on P/E cycles of flash blocks. Our results show that better fairness of dynamic allocation greatly helps to achieve finer

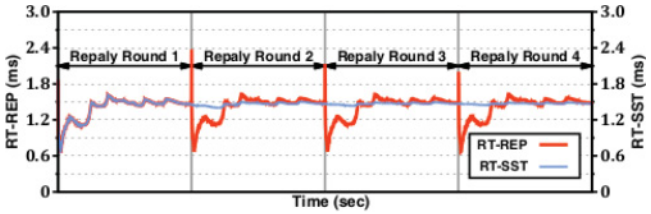


Fig. 9. Behavior of the SSD in the first four rounds of replaying `prn-0`. The average RT value from the beginning of each replay round (RT-REP) and the average RT value from simulation start point (RT-SST) are shown.

wear leveling and reduces the erase variance of flash blocks. Finally, in Section 6.3, we provide deeper insight into the behavior of dynamic allocation strategies through microbenchmarking. Note that for the purpose of simple illustration and to concentrate on best design alternatives, all evaluations of this section are just performed on the strategies included in the optimal sets presented in Tables VI and VII. Therefore, CD, DC, DW, D, P, and F strategies are used for SSD-MLC evaluation, whereas CP, PD, C, D, P, and F strategies are used for SSD-SLC. In addition, we used CWDP as a representative of static allocation strategies and a reference for comparison. For most workload types, CWDP is ranked among top five best static strategies.

6.1. The Steady-State Performance

We assume an SSD to be in its steady state when GC is regularly performed and its rate is nearly stable over a long time. We employ workload *replay* to force an SSD working in the steady state [Murugan and Du 2011; Li et al. 2013]. Here, a workload trace is repeatedly, and in different rounds, fed into the simulator, and the replay process continues up to a point that GC is triggered. Then, simulation is continued until variations in *average GC rate* and *average RT* become lower than 1% for the last five rounds. This method introduces an acceptable criterion for asserting a simulated system to be in its steady state [Pawlikowski 1990]. For better understanding of the replay process, Figure 9 shows SSD behavior within a window of the first four replays for a typical workload (i.e., `prn-0`). For performance analysis in the steady state, we define two metrics using the variations of RT in long-term simulation: RT-SST (*Start to Steady-sTate*) and RT-REP (*REPlay*). At a specific simulation time, RT-SST refers to the average RT seen by I/O requests from the beginning of simulation, and RT-REP represents the average RT from the beginning of the current reply round. We remark that (1) RT-SST and RT-REP curves are similar during the first replay round, and (2) after the initial simulation cycles, RT-SST converges to a fixed value, but RT-REP shows fluctuations following the temporal variations of the workload.

We applied our steady-state simulation methodology to SSD-SLC and SSD-MLC considering all 32 workloads listed in Table IV. We observed that for read-dominant workloads, the performance variation between initial and steady-state results was negligible (as shown in Figure 11). Hence, we mainly concentrate on the behavior of allocation strategies under write-dominant workloads. Figures 10(a) and 10(b) respectively show the steady-state behavior of SSD-MLC and SSD-SLC for different allocation strategies. Due to lack of space, we have presented the results for `exch` as an example of write-dominant workloads (also, DW is not shown for SSD-MLC and CP for SSD-SLC). Each chart corresponds to one allocation strategy and reports the performance in terms of RT-SST (left y-axis of the upper plot) and RT-REP (right y-axis of the upper plot), as well as the average GC rate from the start time until the steady state (GC-SST in the lower plot expressed in the unit of GC execution counts per second). Clearly, GC overheads affect the performance of allocation strategies, but

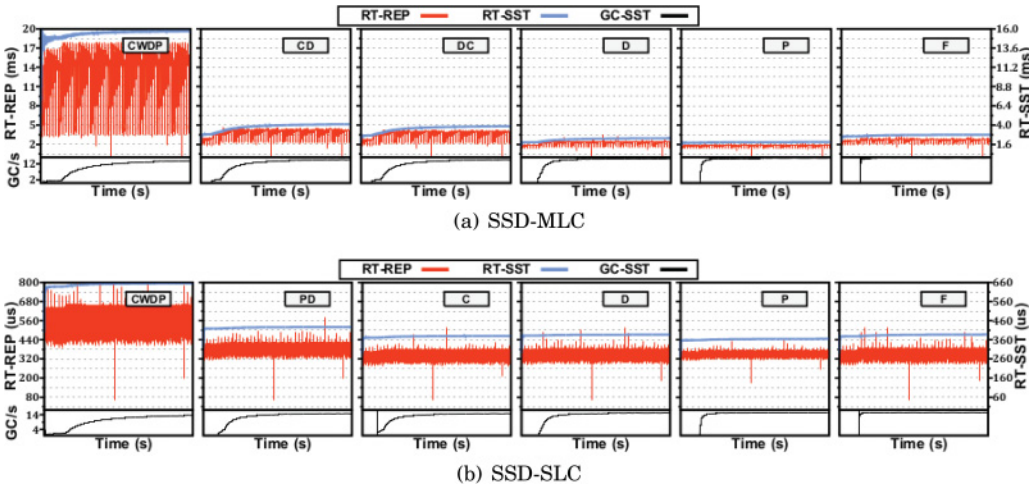


Fig. 10. The effect of *PLAlloc* strategies on the steady-state behavior of the SSD in SSD-MLC (a) and SSD-SLC (b). The results are achieved for replaying `exch` workload, and the scales of RT-REP and RT-SST are set to be different for better illustration. The average GC rate from the beginning of the simulation is depicted in the lower plots (black curves) in the unit of GC execution counts per second (GC/s).

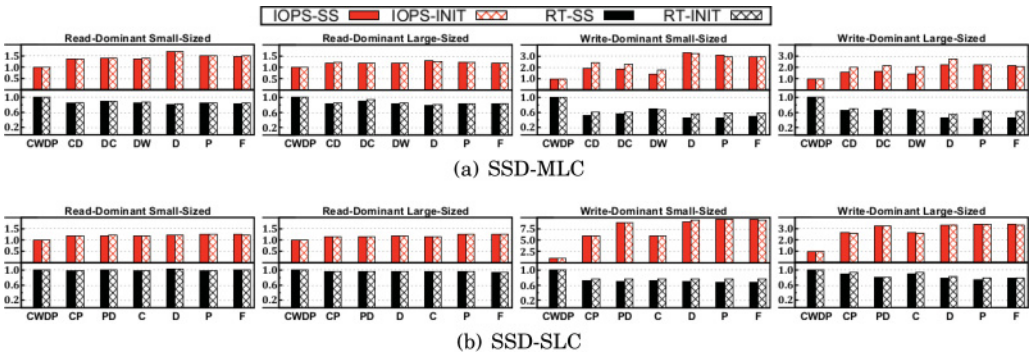


Fig. 11. Steady-state performance of allocation strategies in SSD-MLC (a) and SSD-SLC (b). SS and INIT abbreviations stand for steady-state and initial state results, respectively. The results are normalized with respect to CWDP in a similar manner to Figure 5. Higher IOPS and lower RT values are desirable. Dynamic allocations always achieve greater RT improvement in the steady state, but IOPS improvement may be negatively reduced due to GC interference.

the severity of variations/degradations in performance highly depends on the degree of freedom, priority of resources allocation, and SSD configuration. For SSD-MLC, the RT-SST curve of CWDP, CD, DC, and D strategies rises as the GC execution rate increases in GC-SST, whereas the impact of GC execution on RT-SST is almost negligible for P and F strategies. The values of the RT-REP curve for D, P, and F strategies vary in a much smaller range when compared to the results of CWDP, CD, and DC. In particular, CWDP results show a sudden increase, and the range of variations is six times larger than that of D, P, and F. This means that SSD performance is more predictable under dynamic allocation strategies, particularly those with degree-3 or degree-4 freedom. In a nutshell, the average and instantaneous performance of static allocation strategies are more sensitive to GC execution, whereas dynamic allocations with high freedom degrees can greatly tolerate the negative effects of GC. The best RT results are achieved for P strategy. For SSD-SLC, the variations of RT-SST and

RT-REP due to GC execution is insignificant, as higher number of channel-level and way-level resources together with better performance of SLC NAND flash help to hide the negative impact of read, program, and erase operations of GC. Yet RT-REP values vary in a wider range for CWDP, whereas the changes are limited into a narrow range in dynamic allocations; note that similar to SSD-MLC, P strategy provides the best results.

Figure 11 depicts the steady-state performance of SSD-MLC and SSD-SLC under different workload categories (labeled IOPS-SS and RT-SS). The normal simulation results, achieved in Section 5, are also included in the plots (labeled IOPS-INIT and RT-INIT) to highlight performance variations from the initial to steady state. As mentioned before, performance variations in read-dominant workloads are almost negligible, whereas under write-dominant workloads, IOPS and RT are greatly affected by GC. Interestingly, RT-SS values of dynamic strategies are always better (smaller) than RT-INIT results in both SSD-MLC and SSD-SLC under write-dominant workloads. In fact, if a program operation is blocked due to GC tasks, then a dynamic strategy has the chance to map it to an idle resource to relieve the long blocking time of normal I/O operations due to GC. However, comparing IOPS-INIT and IOPS-SS of SSD-MLC reveals that the IOPS improvement rate does not change noticeably for P and F, and it may even decrease for CD, DC, DW, and D strategies. As a matter of fact, in the steady state, components of the resource-limited SSD-MLC may be engaged in a GC process, and hence dynamic strategies have less chance to exploit interrequest parallelism in favor of IOPS. Contrarily, in SSD-SLC, the higher amount of resources are still enough for simultaneous handling of consecutive I/O requests even in the presence of GC. Hence, IOPS of SSD-SLC is less affected in the steady state for most dynamic allocation strategies. In sum, steady-state simulations show that RT improvement of dynamic strategies is as high as 56% for SSD-MLC and 32% for SSD-SLC. Furthermore, dynamism can provide up to 3.3x and 9.6x IOPS improvement for SSD-MLC and SSD-SLC, respectively.

6.2. The Impact of Dynamism on P/E Cycles

A good allocation strategy can support a wear-leveling primitive of the FTL through better distribution of P/E operations among the whole set of planes within the SSD. This reduces the variations of erase counts among flash blocks of different planes. Static strategies cannot help wear leveling due to their deterministic nature of plane allocation, which may lead to unfair distribution of program operations based on LPA accesses. Dynamic allocation strategies, on the other hand, can fairly distribute flash operations, as our previous discussion on the results of Figure 5 revealed the excellent fairness of degree-3 and degree-4 strategies.

Figure 12 illustrates the impact of dynamism on SSD endurance in terms of the *standard deviation of block erase count* (STD-ERASE) and *difference of maximum-minimum block erase count* (DIFF-ERASE). The lower values for the mentioned criteria mean finer distribution of block erases and better wear leveling. Based on the results, we can deduce that dynamism greatly improves wear leveling of SSD-MLC and SSD-SLC and hence helps to enhance their life span. The results show that dynamism can lead to up to 88% reduction in STD-ERASE and 87% reduction in DIFF-ERASE for SSD-MLC. In SSD-SLC, STD-ERASE and DIFF-ERASE are sharply reduced by up to 96% and 96%, respectively.

6.3. Microbenchmarking

We also conducted microbenchmarking simulations by varying I/O request sizes and read/write ratios for steady-state investigation of SSD-MLC and SSD-SLC. In our experiments, LPAs are uniformly distributed over user storage space, and the temporal

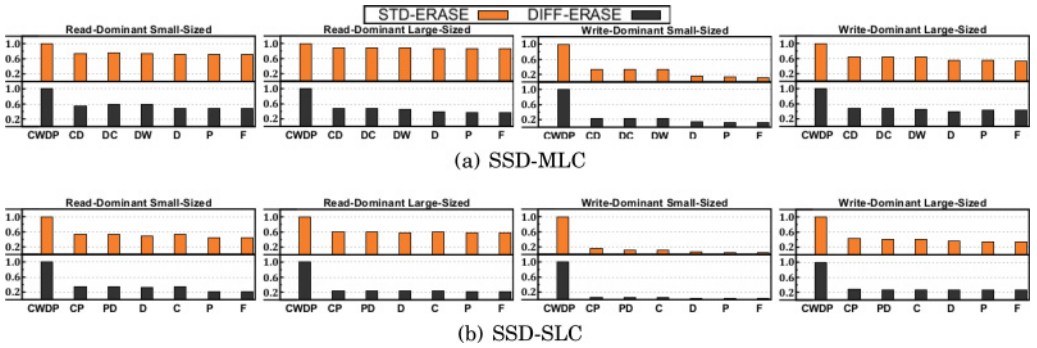


Fig. 12. The effect of dynamic allocation strategies on P/E cycles of flash blocks in SSD-MLC (a) and SSD-SLC (b). The metrics are a standard deviation of the block erase count (STD-ERASE) and the difference between maximum and minimum block erase counts (DIFF-ERASE). The lower values of both criteria are desirable, as they work in favor of SSD endurance. The results are normalized in a similar manner to Figure 5.

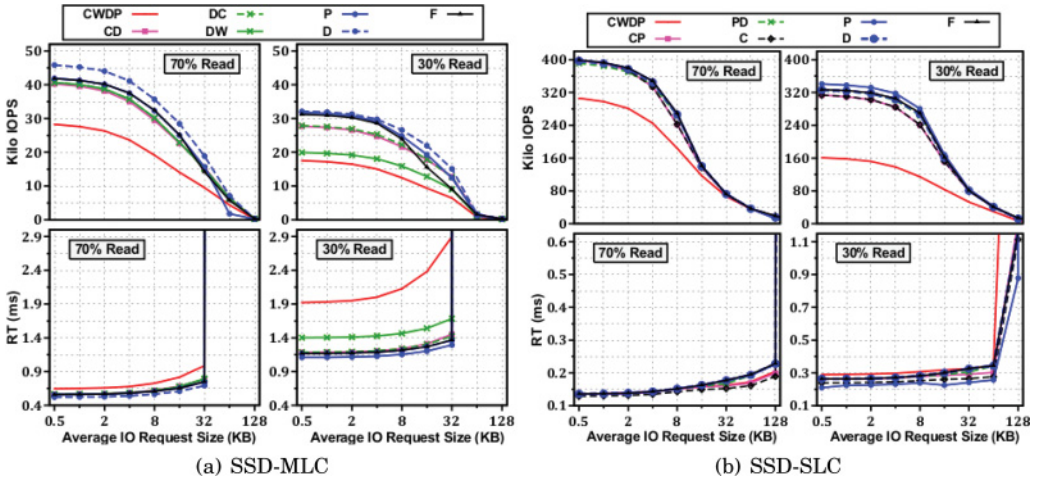


Fig. 13. Microbenchmarking results for steady-state evaluation of allocation strategies in SSD-MLC (a) and SSD-SLC (b). In most cases, degree-3 strategies are ranked first.

model of request generation is Poisson with average interarrival time of 1 ms; the percentage of read requests is set to be 30% and 70% for different scenarios, and the average request size varies between 0.5KB and 128KB for each scenario. As can be seen in Figure 13(a), the results for SSD-MLC show impressive performance impact of dynamic strategies, especially when using D. In fact, IOPS results of D are considerably higher than those of CWDP (and even better than other dynamic strategies). Regarding RT results, D ranked first in the 70%-read (read-dominant) scenario, whereas best results are achieved for P in the 30%-read (write-dominant scenario).

Figure 13(b) depicts the microbenchmarking results of SSD-SLC. As expected, dynamism enhances the IOPS of SSD-SLC in the read-dominant scenario, and the results of F, P, D, and PD are a little better than other dynamic strategies. Increasing the write ratio greatly enhances the IOPS improvement rate of dynamic strategies, and P is ranked first in the 30%-read scenario. Regarding RT results, the values of C and CP are slightly better than other strategies in the 70%-read scenario, whereas P is ranked first in the 30%-read scenario. As a final point, RT results of the 70%-read scenario

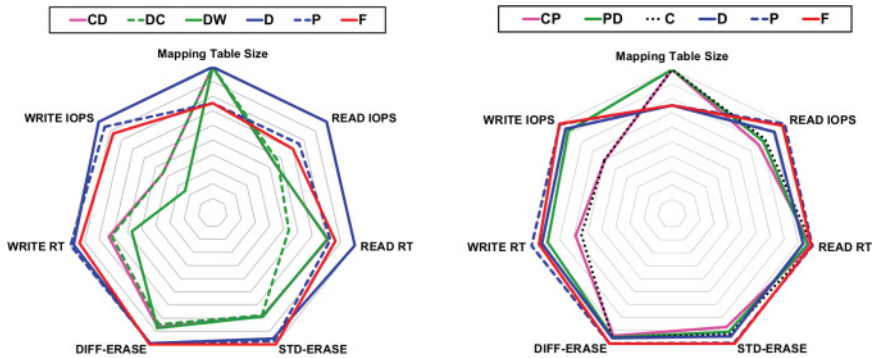


Fig. 14. Comparison of dynamic allocation strategies regarding the mapping table size and performance and endurance metrics in SSD-MLC (a) and SSD-SLC (b). For each SSD configuration, strategies in the optimal sets are considered.

support our previous conclusion that the resources of SSD-SLC are enough to equally decrease the chance of intrarequest contention for both static and dynamic strategies.

7. SUMMARY

Selecting an appropriate allocation strategy generally depends on design goals and restrictions, and even a low performance strategy may be preferred due to memory size or endurance considerations of the design. Therefore, our final suggestion for SSD-MLC and SSD-SLC includes all allocation strategies in their optimal sets. Figure 14 illustrates a comparison of these strategies based on their mapping table size and IOPS, RT, and endurance results. For better illustration, a single average value is reported as a representative of small-sized and large-sized workloads. For example, READ IOPS is the average value of IOPS improvement for read-dominant small-sized and read-dominant large-sized categories. In SSD-MLC, D provides excellent read/write performance with no overhead on the mapping table size, whereas F is ranked first regarding endurance metrics but requires a larger mapping table size and achieves lower I/O performance, especially for reads. In SSD-SLC, CP, PD, and C may be preferred due to their better READ RT and lower table size overheads, whereas P is a suitable choice due to its outstanding write performance. Furthermore, F provides the best endurance and ranked second regarding performance metrics. However, F incurs the highest cost on the size of the mapping table. Please note that our conclusions generally characterize dynamic allocation strategies for SSD-MLC and SSD-SLC, but the final selection should be made based on the configuration and working conditions of the target SSD.

8. RELATED WORK

Parallelism plays a vital role in achieving performance desires in modern SSDs, and efficient utilization of highly parallel resources has been the main focus of various studies [Chen et al. 2011; Jung et al. 2012, 2014]. Among them, some studies have concentrated on design space exploration of page allocation strategies to better exploit parallelism through enhanced address striping. Shin et al. [2009] investigated a small set of static allocation strategies with highest priority for CID allocation. Variations of fully dynamic allocation were also proposed and investigated in different works as the only solution to benefit from dynamism [Shin et al. 2009; Park et al. 2010b]. Hu et al. [2011] investigated the interaction of different levels of parallelism and analyzed the impact of SSD configuration on the performance of static and fully dynamic

strategies. Jung and Kandemir [2012] proposed a thorough analysis of all possible 24 static allocation strategies and provided new insight about the role of intrachip parallelism for performance enhancement. To the best of our knowledge, there has been no work that explicitly investigates the benefits of utilizing different freedom degrees in allocation strategies. Moreover, past studies have reported their results with no discussion on steady-state behavior, whereas in this work we have thoroughly explored our idea under such realistic conditions.

Other studies have concentrated on the scheduling policies that help to resolve resource contention among flash transactions and to increase the chance of parallel executions at different levels of the SSD internal parallelism. As we discussed in Section 4.3, a good allocation strategy should be augmented with a fine-tuned scheduler to efficiently exploit SSD parallel resources. Park et al. proposed a dynamic transaction rescheduling algorithm to increase the chance of multiplane command execution [Park et al. 2010b]. Ozone provides a hardware-assisted scheduling mechanism for I/O requests [Nam et al. 2011]. This mechanism explores data dependency among I/O requests and reorders safe requests to better exploit channel- and chip-level parallelism. Jung et al. proposed PAQ, a software-assisted dynamic scheduler that concentrates on out-of-order execution of read transactions to increase the utilization of channel-, chip-, and plane-level parallelism [Jung et al. 2012]. Sprinkler is a state-of-the-art I/O scheduler that maximizes resource utilization at all parallelism levels and reduces the overall number of flash transactions through enhanced incorporation of multiplane and interleaved commands [Jung and Kandemir 2014]. Recently, HIOS, a new host interface I/O scheduler, was proposed that tries to reduce the negative performance impacts of GC execution and meet the QoS requirements of I/O requests [Jung et al. 2014]. HIOS tries to distribute the resource conflict overheads and GC costs over noncritical I/O requests to meet the deadline requirements of host I/O requests and reduce RT variations. Such high-performance schedulers in conjunction with a dynamic allocation strategy of degree-3 can substantially reduce the steady-state SSD performance sensitivity to GC execution overheads.

9. CONCLUSION

Efficient utilization of parallel resources greatly impacts the performance of the SSD. In this article, we proposed a new set of allocation strategies based on dynamic resource assignment to achieve better striping of flash transactions and decrease the probability of resource conflicts. Our extensive simulation experiments uncovered the superior performance of dynamic allocation strategies with three degrees of freedom for both midrange and high-end SSDs. Our results especially showed that dynamic strategies are most favored (with respect to static ones) in resource-limited midrange and client SSDs. Furthermore, our steady-state simulation revealed that dynamism greatly helps to mitigate performance and endurance side effects of GC.

REFERENCES

J. S. Bucy, J. Schindler, S. W. Schlosser, G. R. Ganger, and Contributors. 2008. *The DiskSim Simulation Environment Version 4.0 Reference Manual*. Technical Report CMU-PDL-08-101. Parallel Data Laboratory, Carnegie Mellon University, Pittsburgh, PA.

Evgeny Budilovsky, Sivan Toledo, and Aviad Zuck. 2011. Prototyping a high-performance low-cost solid-state disk. In *Proceedings of the 4th Annual International Conference on Systems and Storage (SYSTOR'11)*. 13:1–13:10. DOI:http://dx.doi.org/10.1145/1987816.1987834

Werner Bux, Xiao-Yu Hu, Ilias Iliadis, and Robert Haas. 2012. Scheduling in flash-based solid-state drives—performance modeling and optimization. In *Proceedings of the IEEE 20th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'13)*. 459–468. DOI:http://dx.doi.org/10.1109/MASCOTS.2012.58

Werner Bux and Ilias Iliadis. 2010. Performance of greedy garbage collection in flash-based solid-state drives. *Performance Evaluation* 67, 11, 1172–1186. DOI:http://dx.doi.org/10.1016/j.peva.2010.07.003

Li-Pin Chang, Tei-Wei Kuo, and Shi-Wu Lo. 2004. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. *ACM Transactions on Embedded Computing Systems* 3, 4, 837–863. DOI:http://dx.doi.org/10.1145/1027794.1027801

Feng Chen, Rubao Lee, and Xiaodong Zhang. 2011. 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 (HPCA'11)*. 266–277. DOI:http://dx.doi.org/10.1109/HPCA.2011.5749735

Marcelo Cintra and Niklas Linkewitsch. 2013. Characterizing the impact of process variation on write endurance enhancing techniques for non-volatile memory systems. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'13)*. 217–228. DOI:http://dx.doi.org/10.1145/2465529.2465755

Laura M. Grupp, John D. Davis, and Steven Swanson. 2012. The bleak future of NAND flash memory. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*. 17–24.

Laura M. Grupp, John D. Davis, and Steven Swanson. 2013. The harey tortoise: Managing heterogeneous write performance in SSDs. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference (USENIX ATC'13)*. 79–90.

Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIV)*. 229–240. DOI:http://dx.doi.org/10.1145/1508244.1508271

Jen-Wei Hsieh, Han-Yi Lin, and Dong-Lin Yang. 2014. Multi-channel architecture-based FTL for reliable and high-performance SSD. *IEEE Transactions on Computers* 63, 12, 3079–3091. DOI:http://dx.doi.org/10.1109/TC.2013.169

Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Chao Ren. 2013. Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance. *IEEE Transactions on Computers* 62, 6, 1141–1155. DOI:http://dx.doi.org/10.1109/TC.2012.60

Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. 2011. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. In *Proceedings of the International Conference on Supercomputing (ICS'11)*. 96–107. DOI:http://dx.doi.org/10.1145/1995896.1995912

Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Shuping Zhang, Jingning Liu, Wei Tong, Yi Qin, and Liuzheng Wang. 2010. Achieving page-mapping FTL performance at block-mapping FTL cost by hiding address translation. In *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST'10)*. 1–12. DOI:http://dx.doi.org/10.1109/MSST.2010.5496970

Ilias Iliadis. 2014. Rectifying pitfalls in the performance evaluation of flash solid-state drives. *Performance Evaluation* 79, 235–257. DOI:http://dx.doi.org/10.1016/j.peva.2014.07.015

Myoungsoo Jung, Wonil Choi, Shekhar Srikantaiah, Joonhyuk Yoo, and Mahmut T. Kandemir. 2014. HIOS: A host interface I/O scheduler for solid state disks. In *Proceedings of 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA'14)*. 289–300. DOI:http://dx.doi.org/10.1145/2665671.2665715

Myoungsoo Jung and Mahmut T. Kandemir. 2012. An evaluation of different page allocation strategies on high-speed SSDs. In *Proceedings of the 4th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage'12)*.

Myoungsoo Jung and Mahmut T. Kandemir. 2013. 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 (SIGMETRICS'13)*. 203–216. DOI:http://dx.doi.org/10.1145/2494232.2465548

Myoungsoo Jung and Mahmut T. Kandemir. 2014. Sprinkler: Maximizing resource utilization in many-chip solid state disks. In *Proceedings of the IEEE 20th International Symposium on High Performance Computer Architecture (HPCA'14)*. 524–535. DOI:http://dx.doi.org/10.1109/HPCA.2014.6835961

Myoungsoo Jung, Ellis H. Wilson, III, and Mahmut Kandemir. 2012. Physically addressed queueing (PAQ): Improving parallelism in solid state disks. In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA'12)*. 404–415. DOI:http://dx.doi.org/10.1145/2337159.2337206

Jeffrey Katcher. 1997. *PostMark: A New File System Benchmark*. Technical Report. Network Appliance.

Sungjin Lee, Dongkun Shin, and Jihong Kim. 2013. BAGC: Buffer-aware garbage collection for flash-based storage systems. *IEEE Transactions on Computers* 62, 11, 2141–2154. DOI:http://dx.doi.org/10.1109/TC.2012.227

Yongkun Li, Patrick P. C. Lee, and John C. S. Lui. 2013. Stochastic modeling of large-scale solid-state storage systems: Analysis, design tradeoffs and optimization. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'13)*. 179–190. DOI:http://dx.doi.org/ 10.1145/2465529.2465546

R. Timothy Marler and Jasbir S. Arora. 2004. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization* 26, 6, 369–395. DOI:http://dx.doi.org/10.1007/s00158-003-0368-6

Micron Technology, Inc. 2008. *Technical Note: Wear-Leveling Techniques in NAND Flash Devices*. TN-29-42. Micron.

Micron Technology, Inc. 2010a. *Application Note: Wear Leveling in NAND Flash Memory*. AN1822. Micron.

Micron Technology, Inc. 2010b. *MT29E256G08CMCAB NAND Flash Memory Datasheet*. Micron.

Micron Technology, Inc. 2010c. *MT29F128G08AMCAB NAND Flash Memory Datasheet*. Micron.

Microsoft Corporation. 2008a. Microsoft Enterprise Traces. Retrieved March 27, 2016, from http://iotta.snia.org/traces/130.

Microsoft Corporation. 2008b. Microsoft Production Server Traces. Retrieved March 27, 2016, from http://iotta.snia.org/traces/158.

Microsoft Corporation. 2008c. MSR Cambridge Traces. Retrieved March 27, 2016, from http://iotta.snia.org/traces/388.

Muthukumar Murugan and David H. C. Du. 2011. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In *Proceedings of the IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST'11)*. 1–12. DOI:http://dx.doi.org/ 10.1109/MSST.2011.5937225

Eyee Hyun Nam, Bryan Suk Joon Kim, Hyeonsang Eom, and Sang Lyul Min. 2011. Ozone (o3): An out-of-order flash memory controller architecture. *IEEE Transactions on Computers* 60, 5, 653–666. DOI:http://dx.doi.org/10.1109/ TC.2010.209

Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. 2008. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage* 4, 3, 10:1–10:23. DOI:http://dx.doi.org/10.1145/1416944.1416949

Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. 2009. Migrating server storage to SSDs: Analysis of tradeoffs. In *Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys'09)*. 145–158. DOI:http://dx.doi.org/10.1145/1519065.1519081

William D. Norcott. 2014. IOzone File System Benchmark. Retrieved March 9, 2014, from http://www.iozone.org.

C. Park, Euiseong Seo, Ji-Yong Shin, Seungryoul Maeng, and Joonwon Lee. 2010b. Exploiting internal parallelism of flash-based SSDs. *Computer Architecture Letters* 9, 1, 9–12. DOI:http://dx.doi.org/10.1109/L-CA.2010.3

Dongchul Park, Biplob Debnath, and David Du. 2010a. CFTL: A convertible flash translation layer adaptive to data access patterns. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'10)*. 365–366. DOI:http://dx.doi.org/10.1145/1811039.1811089

Krzysztof Pawlikowski. 1990. Steady-state simulation of queueing processes: Survey of problems and solutions. *ACM Computing Surveys* 22, 2, 123–170. DOI:http://dx.doi.org/10.1145/78919.78921

Ji-Yong Shin, Zeng-Lin Xia, Ning-Yi Xu, Rui Gao, Xiong-Fei Cai, Seungryoul Maeng, and Feng-Hsiung Hsu. 2009. FTL design exploration in reconfigurable high-performance SSD for server applications. In *Proceedings of the 23rd International Conference on Supercomputing (ICS'09)*. 338–349. DOI:http://dx.doi.org/ 10.1145/ 1542275.1542324

John E. Shore. 1975. On the external storage fragmentation produced by first-fit and best-fit allocation strategies. *Communications of the ACM* 18, 8, 433–440. DOI:http://dx.doi.org/10.1145/360933.360949

Arash Tavakkol, Mohammad Arjomand, and Hamid Sarbazi-Azad. 2014. Design for scalability in enterprise SSDs. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation (PACT'14)*. 417–430. DOI:http://dx.doi.org/10.1145/2628071.2628098

UMass Trace Repository. 2014. UMass Trace Repository Home Page. Retrieved March 27, 2016, from http://traces.cs.umass.edu.

Benny Van Houdt. 2013a. A mean field model for a class of garbage collection algorithms in flash-based solid state drives. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'13)*. 191–202. DOI:http://dx.doi.org/10.1145/2465529.2465543

Benny Van Houdt. 2013b. Performance of garbage collection algorithms for flash-based solid state drives with hot/cold data. *Performance Evaluation* 70, 10, 692–703. DOI:http://dx.doi.org/10.1016/j.peva.2013.08.010

Benny Van Houdt. 2014. On the necessity of hot and cold data identification to reduce the write amplification in flash-based SSDs. *Performance Evaluation* 82, 1–14. DOI:http://dx.doi.org/10.1016/j.peva.2014.08.003

Chin-Hsien Wu, Tei-Wei Kuo, and Chia-Lin Yang. 2006. A space-efficient caching mechanism for flash-memory address translation. In *Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)*.

Guanying Wu and Xubin He. 2012. Reducing SSD read latency via NAND flash program and erase suspension. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*. 117–124.

Received December 2014; revised September 2015; accepted September 2015