

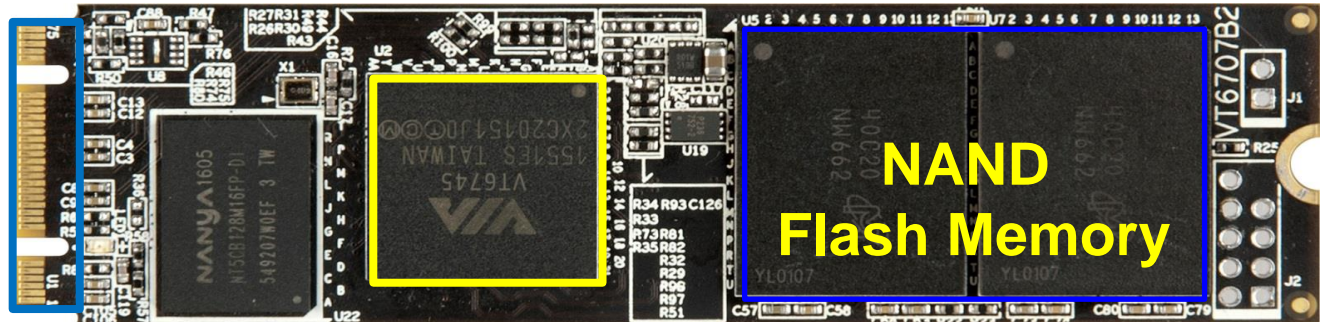


# High Performance FTL for PCIe/NVMe SSDs

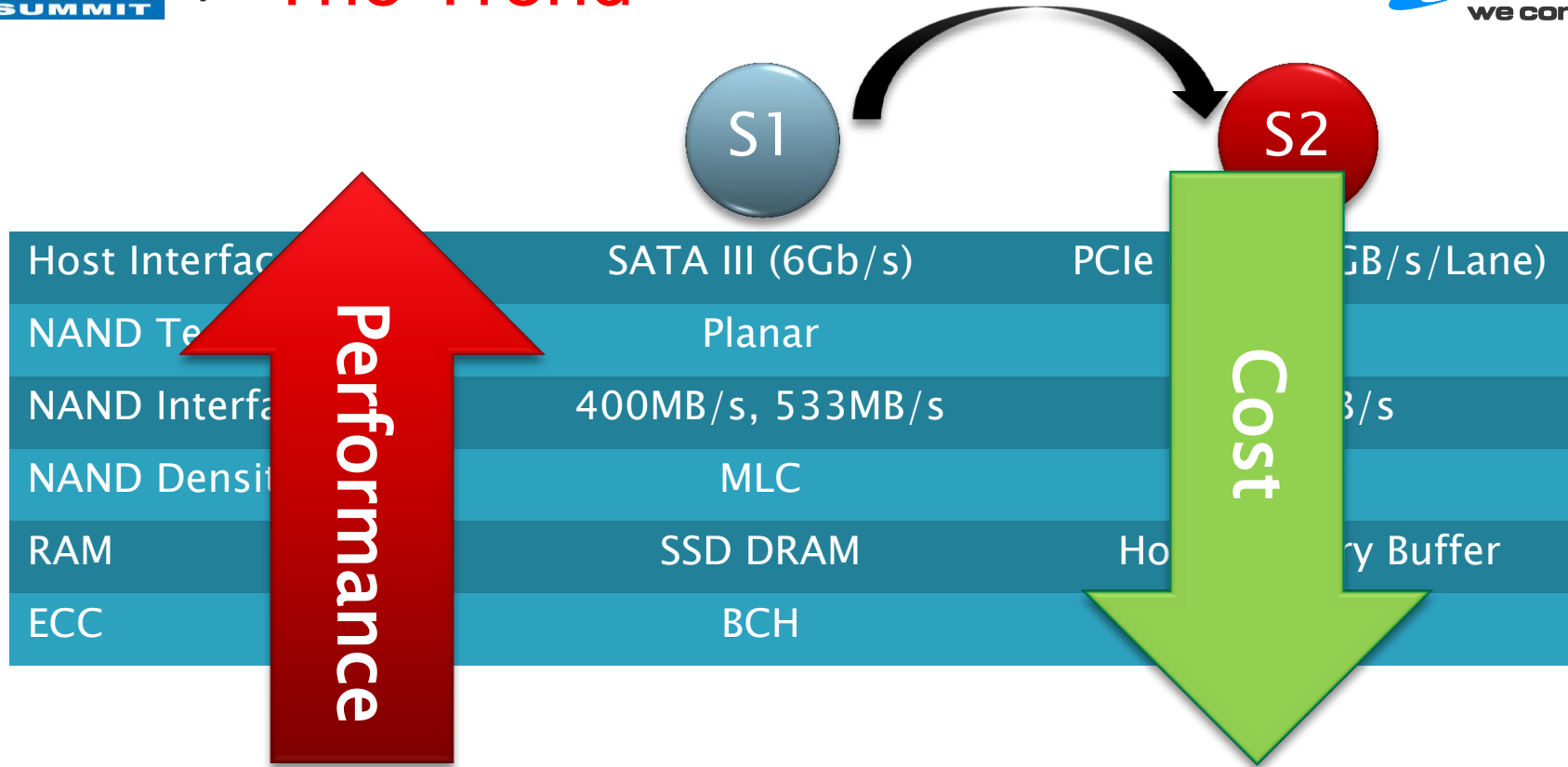
Ying Y. Tai, Ph.D.  
Senior Director  
VIA Technologies, Inc.  
[yingytai@viatech.com](mailto:yingytai@viatech.com)

- Growing popularity of NAND flash memory and SSD:
  - High read/write performance
  - Low power consumption
- SSD has changed the storage landscape on a wide range of applications: from embedded devices to data center.

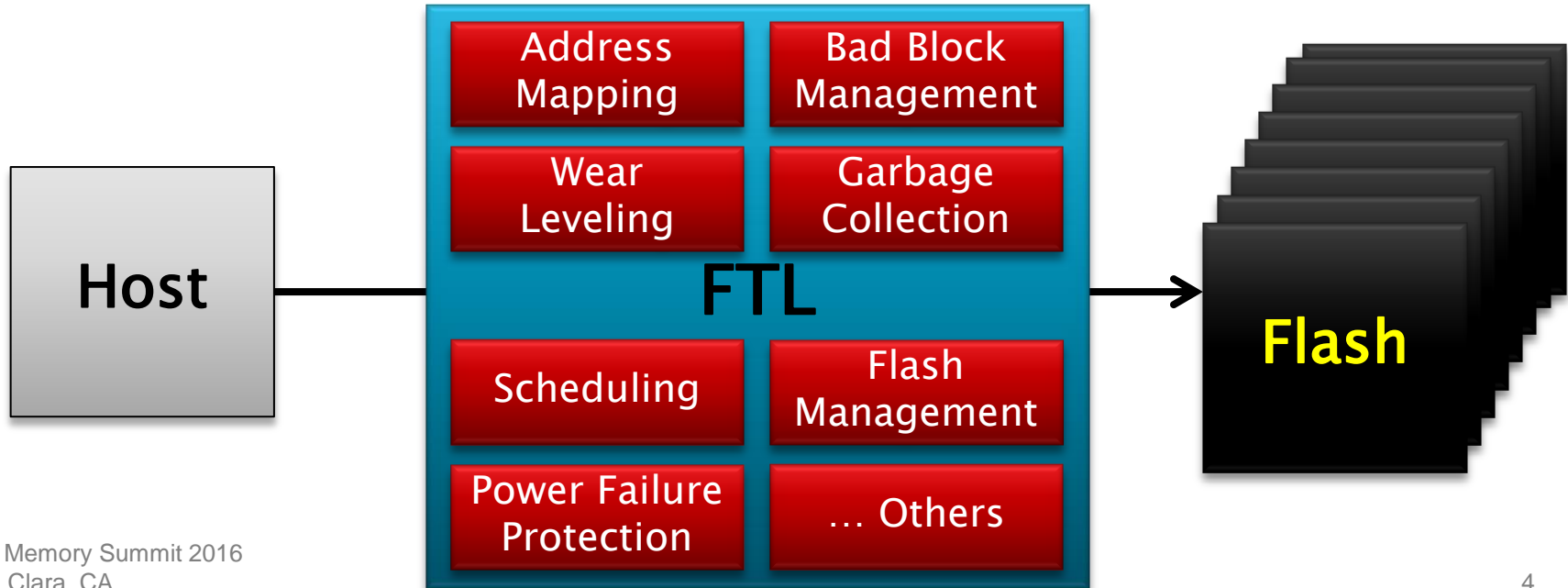
Interface



# The Trend



- The bridge between the host and the storage media.



Minimize	Maximize
Recovery Time	Parallelism
CPU Overhead	Garbage Collection Efficiency
Write Amplification	Wear Leveling Efficiency
Data Loss	
Latency	

- Parallelism
  - Multi-plane operations (internal)
  - Multi-channel architecture (external)
- Write amplification
  - Garbage collection is the main contributors
  - Write workloads

- Multi-channel interleaving
- Grouping blocks into a superblock

- D. Jung et al., "Superblock FTL: a superblock-based flash translation layer with a hybrid address translation scheme," ACM Transactions on Embedded Computing Systems, vol. 9, no. 4, 2012
- B. Peleato et al., "Analysis of trade-offs in v2p-table design for NAND flash," Asilomar 2012

## Superblock FTL: A Superblock-Based Flash Translation Layer with a Hybrid Address Translation Scheme

DAWOON JUNG  
Korea Advanced Institute of Science and Technology  
JEONG-UK KANG  
Samsung Electronics Co.  
HEESEUNG JO  
Korea Advanced Institute of Science and Technology  
and  
JIN-SOO KIM and JOONWON LEE  
Sungkyunkwan University

In NAND flash-based storage systems, an intermediate software layer called a Flash Layer (FTL) is usually employed to hide the erase-before-write characteristics of memory. We propose a novel superblock-based FTL scheme, which combines a set of logical blocks into a superblock. In the proposed Superblock FTL, superblocks are mapped to physical blocks. In the proposed Superblock FTL, superblocks are mapped to physical blocks, while pages inside the superblock are mapped freely at fine granularity in several physical blocks. To reduce extra storage and flash memory operations, mapping information is stored in the spare area of NAND flash memory. This hybrid translation scheme has the flexibility provided by fine-grain address translation, while the memory overhead to the level of coarse-grain address translation. Our experiments show that the proposed FTL scheme significantly outperforms previous block-mapped with roughly the same memory overhead.

Categories and Subject Descriptors: D.4.2 [Operating Systems]: Storage Management; B.7.1 [Integrated Circuits]: Types and Design Styles—Memory

This work was supported by the Ministry of Knowledge Economy, Korea, under the Technology Research Center Support program supervised by the (Institute of Information Technology Advancement) (ITA-2009-C1090-0902-0020), and by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. R01-2007-0 Authors' addresses: D. Jung, H. Jo, Computer Science Department, KAIST, Daejeon, Korea; email: {dewjng, hoesun}@lab.kaist.ac.kr; J.-U. Kang, Memory Division, Samsung Electronics Co., Gyeonggi, Republic of Korea; email: ju.kang@samsung.com; J.-S. Kim, J. L. Information and Communication Engineering, Sungkyunkwan University, Suwon, 440-746, Republic of Korea; email: {jinsookim, joonwon}@skku.edu; Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a presentation along with the full citation. Copyrights for components of this work owned by others than the author(s) must be acknowledged. Abstracting with credit is permitted. To copy otherwise, to republish, to post on a list, or to use any component of this work in other works requires prior permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permission@acm.org. © 2010 ACM 1539-9087/2010/03-ART40 \$10.00

DOI 10.1145/1721696.1721706 <http://doi.acm.org/10.1145/1721696.1721706>

ACM Transactions on Embedded Computing Systems, Vol. 9, No. 4, Article 40, Publication date: 2010.

## ANALYSIS OF TRADE-OFFS IN V2P-TABLE DESIGN FOR NAND FLASH

Borja Peleato, Rajiv Agarwal and John Cioffi

Electrical Engineering Department  
Stanford University  
Stanford CA 94305

### ABSTRACT

Flash memory uses relocate-on-write, also called out-of-place write for performance reasons. Data files from the host are spread across several non-sequential NAND physical pages. In order to retrieve host data at a later point a virtual-to-physical address table mapping the files to their physical addresses must be maintained. This process entails two basic steps. The first is to divide the NAND physical space in a hierarchical manner for efficiency of address lookup. The second is to store the resulting address lookup table, also called a virtual-to-physical (V2P) table in an efficient manner on the flash. This paper explores different architectures for constructing such table and storing it, thereby characterizing the trade-off that they offer in terms of complexity, write speed, and endurance of the flash memory.

### 1. INTRODUCTION

NAND flash has unique characteristics that pose challenges to the SSD system design. The basic unit of NAND physical space is a block, consisting of a fixed number of pages, typically 64 pages of 4 KB each. A block is the elementary unit for erase operations, whereas reads and writes are processed in terms of pages. Before data can be written to a page (i.e., the page is programmed) with that data, the block must have been erased. Moreover, NAND flash memories have a limited program-erase (PE) cycle count, or equivalently there is a limit to the number of times information can be re-written.

Flash memory uses relocate-on-write, also called out-of-place write [5], mainly for performance reasons. If write-in-place is used instead, flash will exhibit high latency due to the necessary reading, erasing, and re-programming of the entire block in which data is being updated. However, relocate-on-write requires maintaining a virtual-to-physical (V2P) table mapping the files to their physical addresses. This process entails two basic steps. The first is to divide the NAND physical space in a hierarchical manner for efficiency of address lookup. The second is to store the V2P table in an efficient manner on the flash. The architecture used for constructing such table and storing it plays a significant role in the overall performance of the memory.

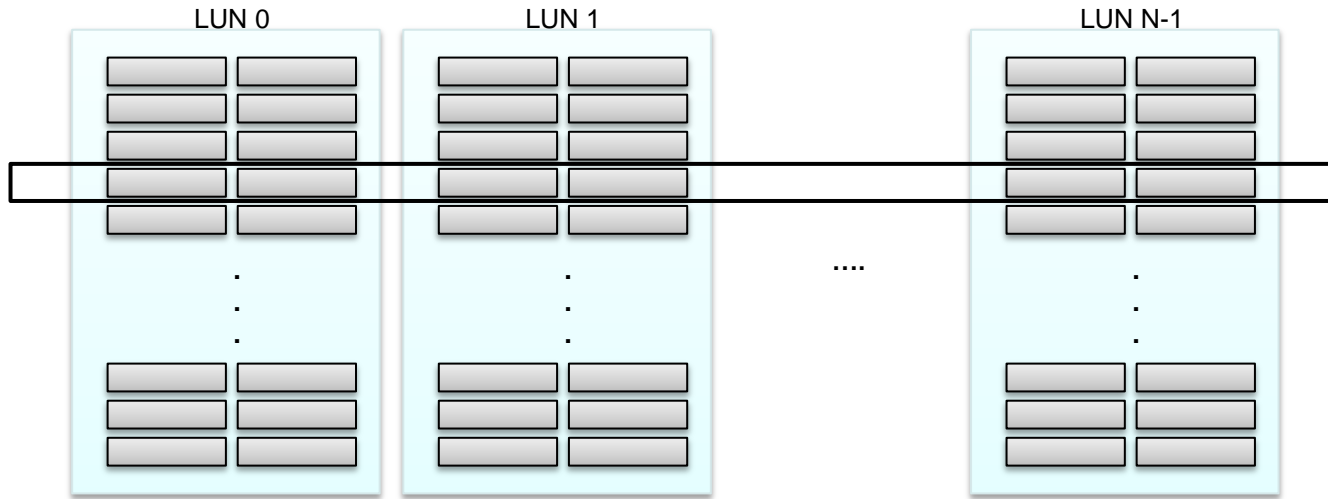
Additionally, relocate-on-write requires a garbage-collection process involving additional read and write operations [4]. This effect is known as write amplification and it reduces both endurance and write throughput [6]. Different garbage collection policies result in different write amplification performance. Here, endurance refers to the lifetime of the memory and write throughput refers to the rate at which incoming host data is transferred to Flash memory.

Write amplification, which depends on overprovisioning<sup>1</sup>, can be reduced by writing sequentially, since any file is then condensed in a small number of blocks. Additionally, when a file, or a portion thereof, occupies physically sequential pages, it can be located using only its starting location and length. As a result, writing sequentially would also reduce the size of the virtual-to-physical address table (V2P-table) mapping the files to their physical addresses. However, spreading the file across several blocks allows for faster reading and writing, since different blocks can be read in parallel via different channels. Additionally, the latter provides better error protection: if a block gets corrupted or loses information due to passage of time, the errors are spread across several files, and hence any single block can be corrected using a small amount of redundancy for each file.

Another important factor to consider is wear leveling. The program and erase operations damage the dielectric barrier in the flash cells, until they reach a point in which they can no longer information reliably. When a certain number of the blocks reach this point, the memory is considered dead. If a subset of blocks are programmed and erased more often than the others, they will suffer more wear leveling, and die earlier, effectively decreasing the lifetime of the memory. In practice, it is common for a memory to store both cold data, which is very rarely updated, and hot data, which is updated often. Blocks storing hot data are programmed and erased more often than those storing cold data. Wear leveling algorithms are therefore used to increase the memory's lifetime by occasionally moving cold data between blocks. The amount of data that needs to be moved during wear leveling, as well as how often this needs to be done depends on the architecture used. Wear

<sup>1</sup>Overprovisioning is defined as the difference between the total space in the drive and the user-accessible space, normalized by the user-accessible drive size.

- Block management is based on the granularity of a superblock.



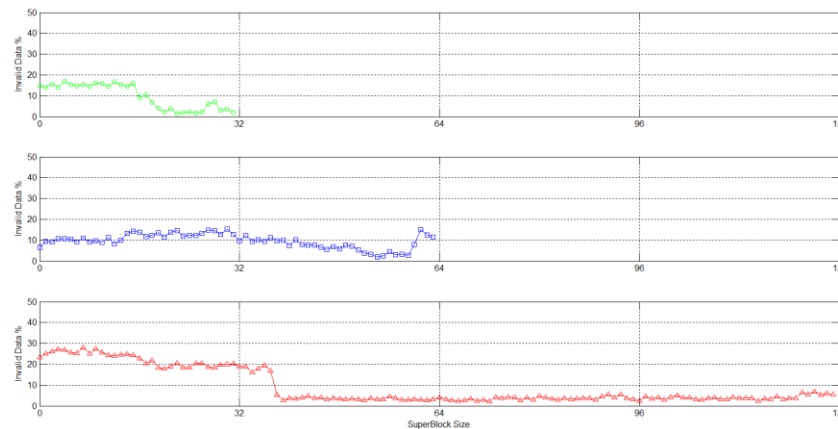
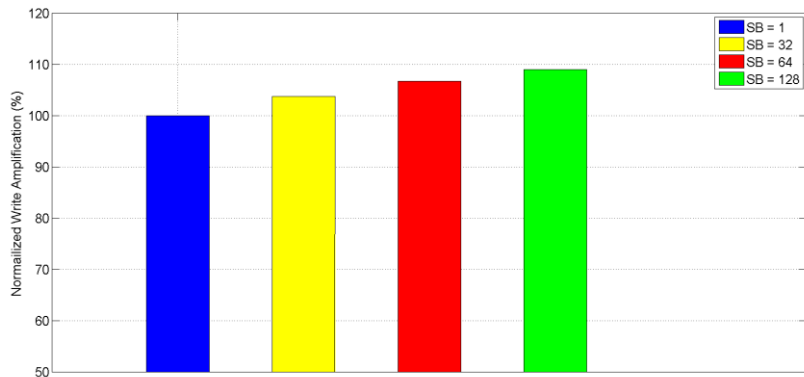
Across Channels



- Pros:
  - Superblock spans multiple channels for concurrent write.
  - Block management overhead is reduced.
  - Conducive to RAID 5 implementation.
- Cons:
  - GC efficiency depends on data locality (hot/cold and sequential/random data).

- Over-Provisioning
- I/O applications
- GC efficiency
- Data integrity writes
- Hot and cold data separation:
  - Algorithm (internal)
  - Host hints (external)

- Workload: 4K random write with 18% hot data.



Server	Function
usr	User home directories
proj	Project directories
prn	Print server
hm	Hardware monitoring
rsrch	Research projects
prxy	Firewall/web proxy
src1	Source control
src2	Source control
stg	Web staging
ts	Terminal server
web	Web/SQL server
mds	Media server
wdev	Text web server

- D. Narayana et al. "Write off-loading: practical power management for enterprise storage," ACM Transactions on Storage, Vol. 4 Issue 3. 2008.

## Write Off-Loading: Practical Power Management for Enterprise Storage

Dushyanth Narayanan

Austin Donnelly  
Microsoft Research Ltd.

Antony Rowstron

{dnarayan,austind,antr}@microsoft.com

### Abstract

In enterprise data centers power usage is a problem impacting server density and the total cost of ownership. Storage uses a significant fraction of the power budget and there are no widely deployed power-saving solutions for enterprise storage systems. The traditional view is that enterprise workloads make spinning disks down ineffective because idle periods are too short. We analyzed block-level traces from 36 volumes in an enterprise data center for one week and concluded that significant idle periods exist, and that they can be further increased by modifying the read/write patterns using *write off-loading*. Write off-loading allows write requests on spun-down disks to be temporarily redirected to persistent storage elsewhere in the data center.

The key challenge is doing this transparently and efficiently at the block level, without sacrificing consistency or failure resilience. We describe our write off-loading design and implementation that achieves these goals. We evaluate it by replaying portions of our traces on a rack-based testbed. Results show that just spinning disks down when idle saves 28-36% of energy, and write off-loading further increases the savings to 45-60%.

### 1 Introduction

Power consumption is a major problem for enterprise data centers, impacting the density of servers and the total cost of ownership. This is causing changes in data center configuration and management. Some components already support power management features: for example, server CPUs can use low-power states and dynamic clock and voltage scaling to reduce power consumption significantly during idle periods. Enterprise storage subsystems do not have such advanced power management and consume a significant amount of power in the data center [32]. An enterprise grade disk such as the Seagate Cheetah 15K.4 consumes 12W even when

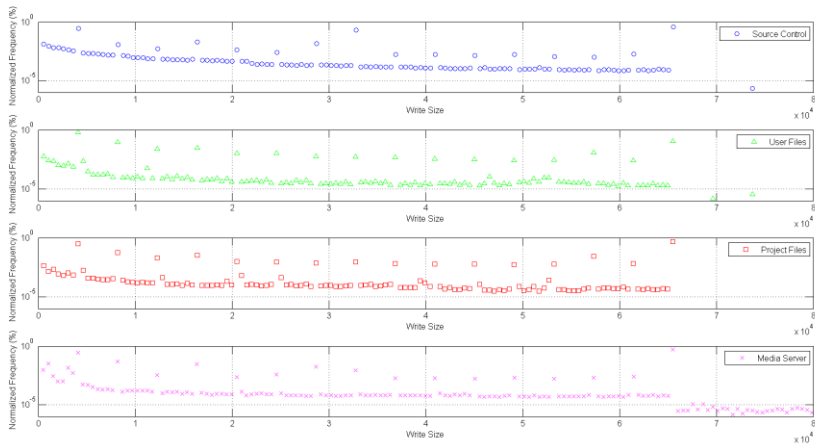
idle [26], whereas a dual-core Intel Xeon processor consumes 24W when idle [14]. Thus, an idle machine with one dual-core processor and two disks already spends as much power on disks as processors. For comparison, the 13 core servers in our building's data center have a total of 179 disks, more than 13 disks per machine on average.

Saving power in storage systems is difficult. Simply buying fewer disks is usually not an option, since this would reduce peak performance and/or capacity. The alternative is to spin down disks when they are not in use. The traditional view is that idle periods in server workloads are too short for this to be effective [5, 13, 32]. In this paper we present an analysis of block-level traces of storage volumes in an enterprise data center, which only partially supports this view. The traces are gathered from servers providing typical enterprise services, such as file servers, web servers, web caches, etc.

Previous work has suggested that main-memory caches are effective at absorbing reads but not writes [4]. Thus we would expect at the storage level to see periods where all the traffic is write traffic. Our analysis shows that this is indeed true, and that the request stream is write-dominated for a substantial fraction of time.

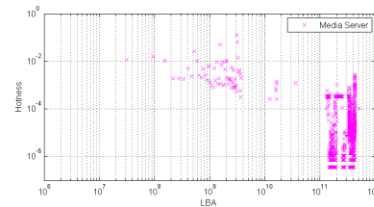
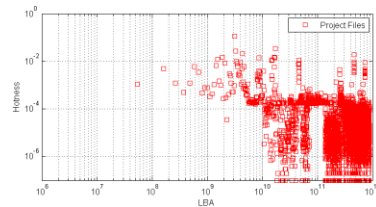
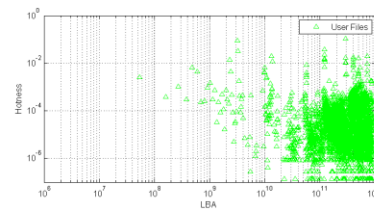
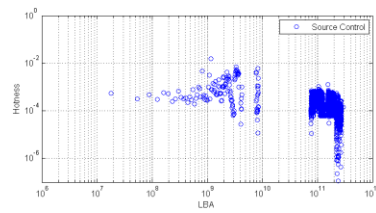
This analysis motivated a technique that we call *write off-loading*, which allows blocks written to one volume to be redirected to other storage elsewhere in the data center. During periods which are write-dominated, the disks are spun down and the writes are redirected, causing some of the volume's blocks to be off-loaded. Blocks are off-loaded temporarily, for a few minutes up to a few hours, and are reclaimed lazily in the background after the home volume's disks are spun up.

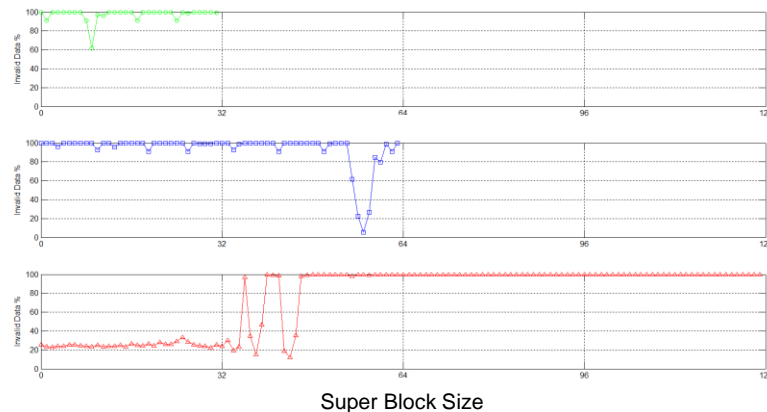
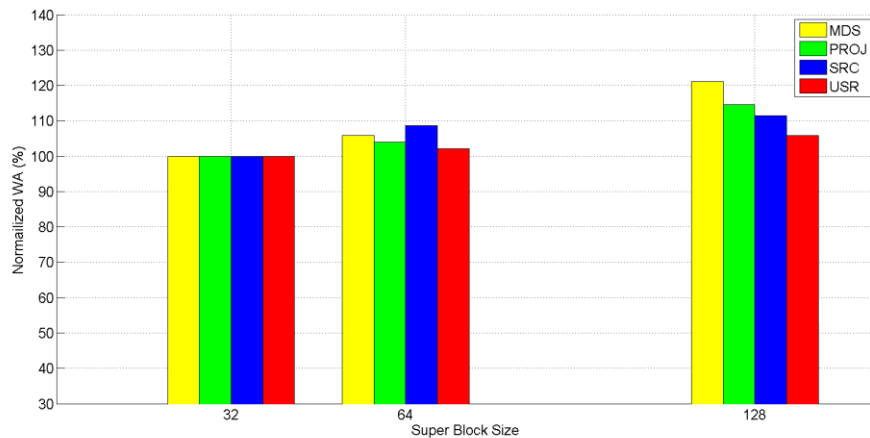
Write off-loading modifies the per-volume access patterns, creating idle periods during which all the volume's disks can be spun down. For our traces this causes volumes to be idle for 79% of the time on average. The cost of doing this is that when a read occurs for a non-off-loaded block, it incurs a significant latency while the disks spin up. However, our results show that this is rare.



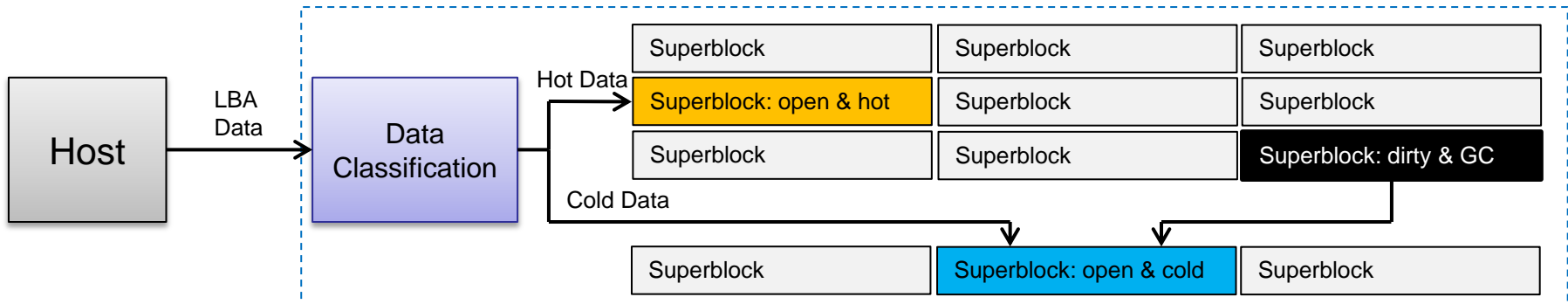
Data Size

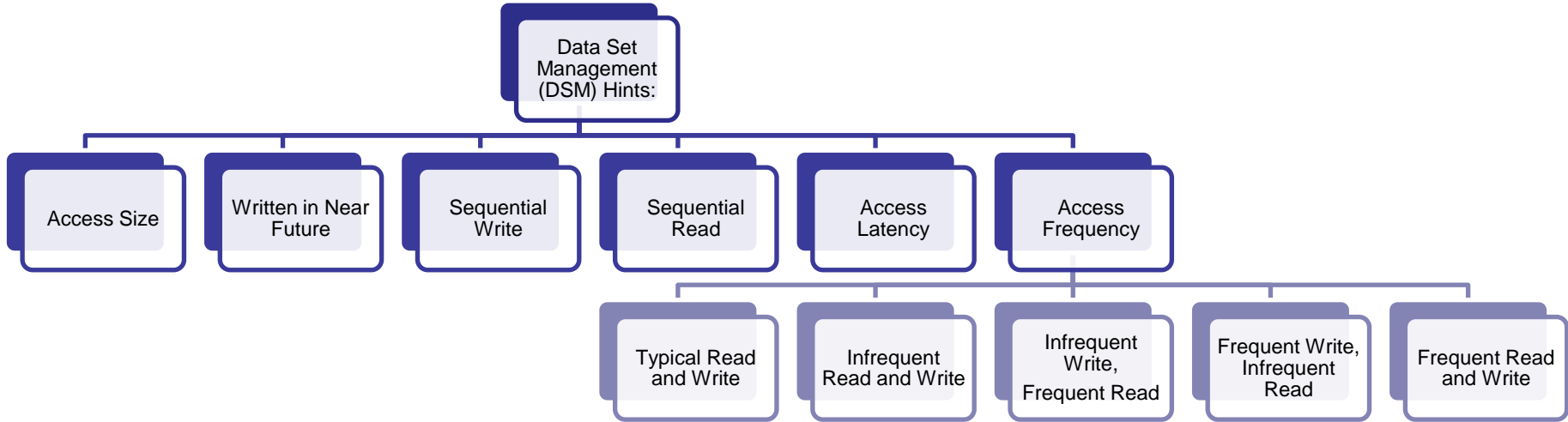
Data Hotness





- Logical block addressing (LBA):
  - Frequency: hot/cold data
  - Recency: workload changes
- Hot and cold data separation







- Proposed by Samsung.
  - A storage interface to inform (hint) SSDs about the data.
  - The host system opens “streams” for different write requests.
  - Data in a stream is written together to a related physical NAND flash space and separated from the data in other streams.
- 
- J. Kang et al. “The multi-streamed solid-state drive,” Proceedings of 6<sup>th</sup> USENIX conference on Hot Topics in Storage and File Systems, 2014.

## The Multi-streamed Solid-State Drive

Jeong-Uk Kang    Jeeseok Hyun    Hyunjoo Maeng    Sangyeun Cho

Memory Solutions Lab.  
Memory Division, Samsung Electronics Co.  
Hwasung, Korea  
E-mail: ju.kang@samsung.com

### Abstract

This paper makes a case for the *multi-streamed solid-state drive (SSD)*. It offers an intuitive storage interface for the host system to inform the SSD about the expected lifetime of data being written. We show through experimentation with a real multi-streamed SSD prototype that the worst-case update throughput of a Cassandra NoSQL DB system can be improved by nearly 56%. We discuss powerful use cases of the proposed SSD interface.

### 1 Introduction

NAND flash based solid-state drives (SSDs) are widely used for main storage, from mobile devices to servers to supercomputers, due to its low power consumption and high performance. Most SSD users do not (have to) realize that the underlying NAND flash medium disallows in-place update; the illusion of random data access is offered by the SSD-internal software, commonly referred to as flash translation layer or FTL. The block device abstraction paved the way for wide adoption of SSDs, because one can conveniently replace a HDD with an SSD without compatibility issues.

Unfortunately, maintaining the illusion of random data access through the block device interface comes at costs. For example, as the SSD is continuously written, the underlying NAND flash medium can become fragmented. When the FTL tries to reclaim free space to absorb further write traffic, internal data movement operations are incurred between NAND flash locations (i.e., garbage collection or GC) [6], leaving the device busy and sometimes unable to properly process user requests. The resultant changing performance behavior of a given SSD is hard to predict or reason about, and remains an impediment to full-system optimization [1].

In order to address the problem from the root, we propose and explore *multi-streaming*, an interface mechanism that helps close the semantic gap between the host system and the SSD. With the multi-streamed SSD, the

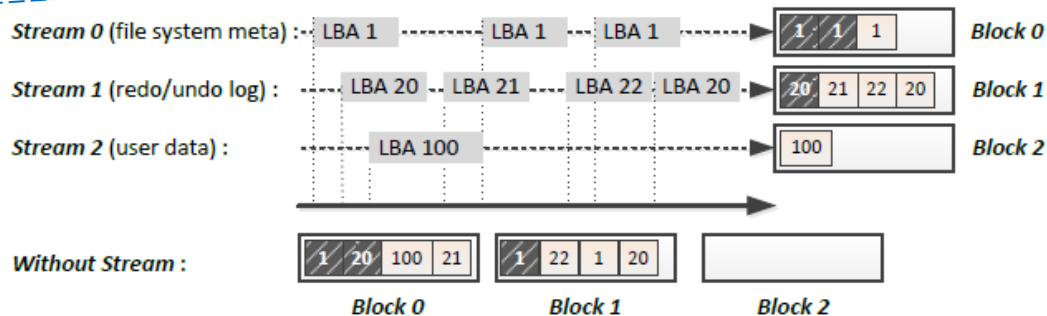
host system can explicitly open “streams” in the SSD and send write requests to different streams according to their expected lifetime. The multi-streamed SSD then ensures that the data in a stream are not only written together to a physically related NAND flash space (e.g., a NAND flash block or “erase unit”), but also separated from data in other streams. Ideally, we hope the GC process would find the NAND capacity unfragmented and proceed with no costly data movements.

In the remainder of this paper, we will delve first into the problem of SSD aging and data fragmentation in Section 2, along with previously proposed remedies in the literature. Section 3 will explain our approach in detail. Experimental evaluation with a prototype SSD will be presented in Section 4. Our evaluation looks at Cassandra [7], a popular open-source key-value store, and how an intuitive data mapping to streams can significantly improve the worst-case throughput of the system. We will conclude in Section 5.

### 2 Background

#### 2.1 Aging effects of SSD

SSD aging [16] explains why the SSD performance may gradually degrade over time; GC is executed more frequently as the SSD is filled with more data and fragmented. Aging effects start to manifest when the “clean” NAND flash capacity is consumed, and in this case, the FTL must proactively recover a sufficient amount of new capacity by “erasing” NAND flash blocks before it can digest new write data. The required erase operations are often preceded by costly GC, to make matters worse, a NAND block, a unit of erase operation, is fairly large in modern NAND flash memory with 128 or more pages in it [15]. When the SSD is filled up with more and more data, statistically, the FTL would need to copy more valid pages for GC before each NAND flash erase operation. This phenomenon is analogous to “segment cleaning” of a log-structured file system [13] and is well studied.



**Figure 2:** The multi-streamed SSD writes data into a related NAND flash block according to stream ID regardless of LBA. In this example, three streams are introduced to store different types of host system data.

- J. Kang et al. "The multi-streamed solid-state drive," Proceedings of 6<sup>th</sup> USENIX conference on Hot Topics in Storage and File Systems, 2014.

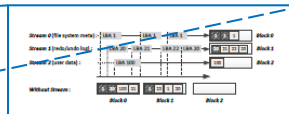


Figure 2: The multi-streamed SSD writes data into a related NAND flash block according to stream ID regardless of LBA. In this example, three streams are introduced to store different types of host system data.

Both the host system and the SSD share a unique stream ID for each open stream, and the host system augments each write with a proper stream ID. A multi-streamed SSD allocates physical capacity carefully, to place data in a stream together and not to mix data from different streams. Figure 2 illustrates how this can be achieved.

We believe that the multi-stream interface is abstract enough for the host system to be able to cope with convincing use cases and results (as discussed in Section 4). Furthermore, the level of information delivered through the interface is concrete enough for the SSD to optimize its behavior with. There are other proposals to specify write data attributes, like access frequency [11]. However, it is not straightforward for the SSD to derive data lifetime from the expected frequency of data updates.

### 3.2 Implementation

We implemented the proposed multi-stream interface on the currently market Samsung 840 Pro SSD [14]. Because 840 Pro is based on the SATA III interface, we piggyback stream ID on a reserved field of both regular and queued write commands as specified in the AT attached (ATA) command set [5]. Our multi-streamed SSD prototype currently supports four streams (Stream 1 to 4) on top of the default stream (Stream 0).

We modified the Linux kernel (3.13.3) to have a conduit between an application and the SSD, through the file system and the layers below. More specifically, an application passes a stream ID to the file system through the `fsync` system call, which, in turn, stores the stream ID in the `inode` of the virtual file system. When dirty pages are flushed into the SSD, or the application directly requests a write operation with the direct I/O facility, we send along the write request the stream ID (that can be retrieved from the associated `inode`).

### 4 Evaluation

#### 4.1 Experimental setup

To evaluate the multi-streamed SSD, we conduct experiments that run Cassandra [7] (version 1.2.10), a widely deployed open-source key value store. All experiments were performed on a commodity machine with a quad-

Table 1: Stream ID Assignment

	system		
	Commit-Log	flushed data	compaction data
Normal Single	0	0	0
Multi-Log	0	1	1
Multi-Data	0	1	2
Ratio of written data (%)	1.0	48.6	31.3
			4.4
			14.7

core Intel i7-3770 3.4GHz processor. We turned off power management for reliable measurements.

Cassandra optimizes I/O traffic by organizing its data set in or append-only "sorted strings tables" (SSTables) in disk. New data are first written to a commit log (CommitLog) and are put in a table in the main memory (MemTable) as they are inserted. Contents in the MemTable are flushed to a SSTable once they accumulate to a certain size. Since SSTables are immutable, several of them are "compacted" periodically to form a new (large) SSTable to reduce the space and time overheads of maintaining many (fragmented) SSTables. As the compaction process repeats, valid data gradually move from a (small) SSTable to another in a different size tier. We take into account how data are created and destroyed in Cassandra when we map writes to streams.

Table 1 lists four different mappings that we examine. Normal implies that all data are mapped to the default stream (Stream 0), equivalent to a conventional SSD with no multi-streaming support and is the baseline configuration. In Single, we separate all data from Cassandra into a stream (Stream 1). System data, not created by the workload itself, include the ext4 file system meta and journal data and still go to Stream 0. Multi-Log carves out the CommitLog traffic to a separate stream, making the total stream count three (including the default stream). Finally, Multi-Data further separates SSTables in different tiers to three independent streams. Intuitively, SSTables in the same tier would have similar lifetime while SSTables from different tiers would have disparate lifetime.

For workloads, we employ the Yahoo! Cloud Serving Benchmark (YCSB) [3] (0.1.4). We run both YCSB and Cassandra on the same machine, not to be limited by the 1Gb Ethernet. In addition, we limit the RAM size to 2GB to accelerate SSD aging by increasing Cassandra's flush frequency. The compaction throughput parameter of Cassandra was modified from 16 MB/s to 32 MB/s, as recommended by the community for SSD users.

#### 4.2 Results

Figure 3(a) plots the normalized update throughput of all mapping configurations studied. We introduce a normal configuration with the TRIM facility turned off, to gain insight about the impact of TRIM. We make the follow-

- PCIe with NVMe enables higher performance SSDs
- FTL: parallelism and write amplification management is critical
- Hot/cold data separation:
  - Data classification: adjusting with write workloads
  - Host hints: NVMe and other storage interfaces
- VIA VT6745 PCIe/NVMe SSD controller with FTL turnkey solution is the best option for high performance SSDs.

## VIA Technologies @ Flash Memory Summit

Time	Presentation
Forum E-21 Wed, Aug. 10 8:30 ~ 10:50am	High-Throughput LDPC Solution for Reliable and High Performance SSD
Forum M-22 Wed, Aug. 10 3:50 ~ 6:15pm	SSD Flash Management for 3D NAND Flash Memory
Session 302-D Thurs, Aug. 11 9:45 ~ 10:50am	High Performance FTL Architecture for PCIe/NVMe SSDs

