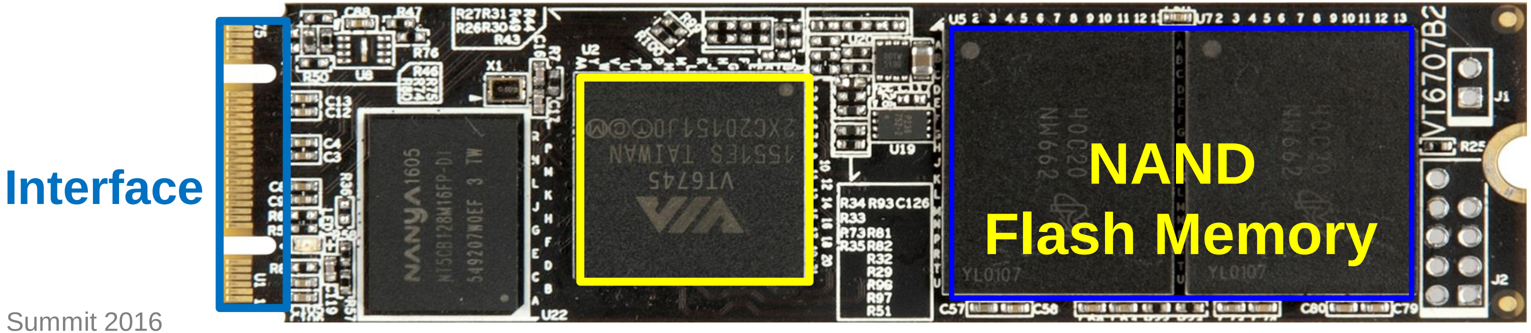
High Performance FTL for PCIe/NVMe SSDs

Ying Y. Tai, Ph.D.
Senior Director
VIA Technologies, Inc.
yingytai@viatech.com

NAND Flash Memory Storage

- 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.



The Trend

S1 S2

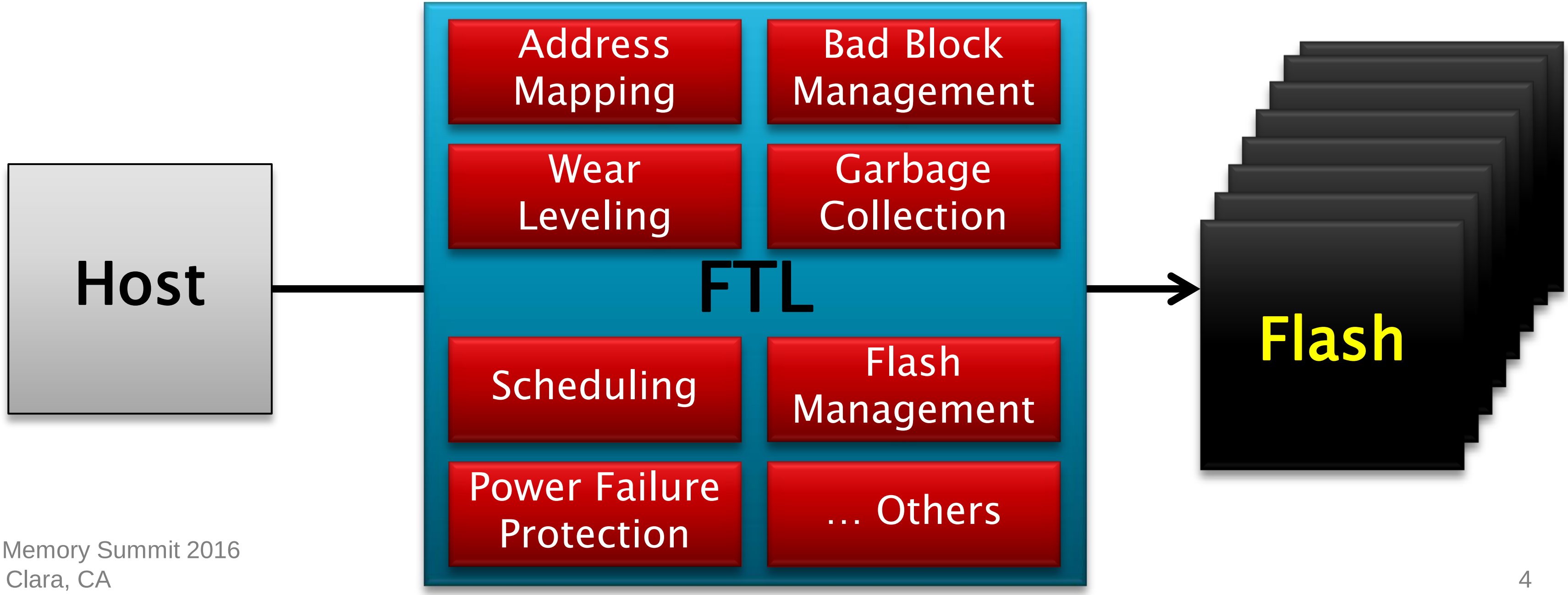
Performance

Cost

Host Interfac	SATA III (6Gb/s)	PCIe GB/s/Lane)
NAND Te	Planar	
NAND Interfa	400MB/s, 533MB/s	3/s
NAND Densit	MLC	
RAM	SSD DRAM	Ho y Buffer
ECC	BCH	

Flash Translation Layer

- The bridge between the host and the storage media.



FTL Recipe

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

2 Aspects

- Parallelism
  - Multi-plane operations (internal)
  - Multi-channel architecture (external)

- Write amplification
  - Garbage collection is the main contributors
  - Write workloads

Parallelism

- 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

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

Borja Peleato, Rajiv Agarwal and John Cioffi

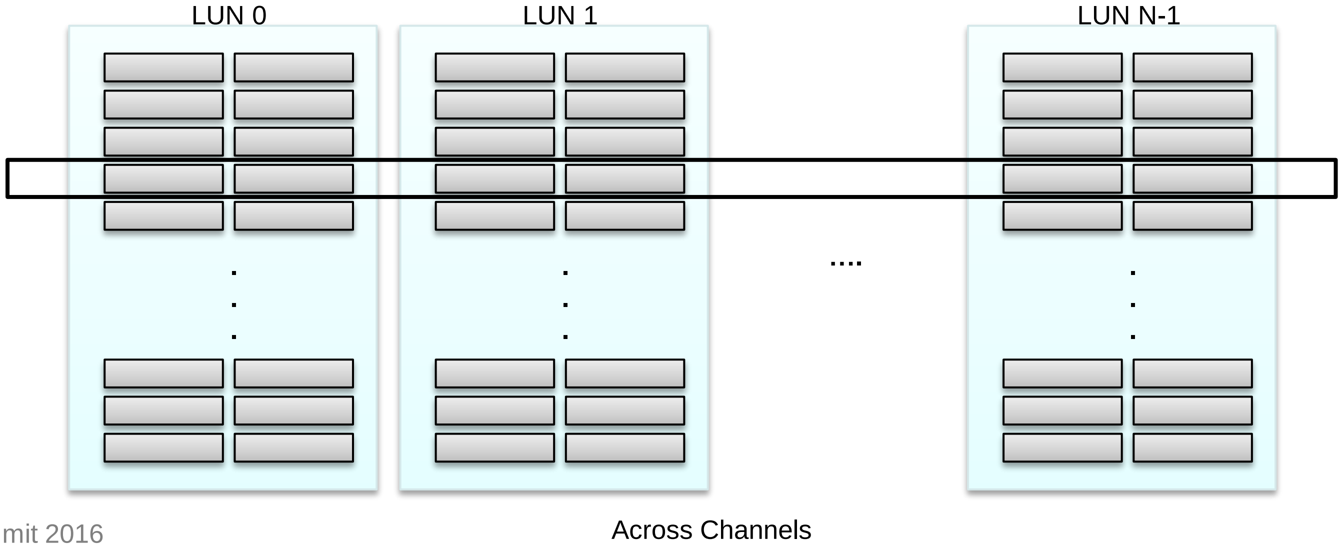
Electrical Engineering Department
Stanford University
Stanford CA 94305

ABSTRACT

1. INTRODUCTION

Superblock

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



Superblock Management

- Pros:
  - Superblock spans multiple channels for concurrent write.
  - Block management overhead is reduced.
  - Conductive to RAID 5 implementation.

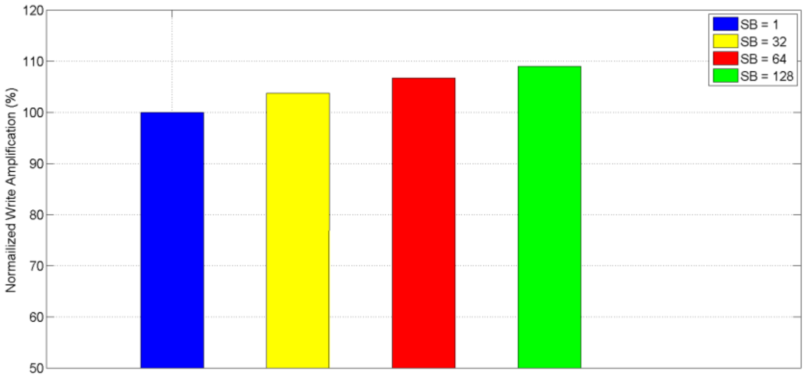
- Cons:
  - GC efficiency depends on data locality (hot/cold and sequential/random data).

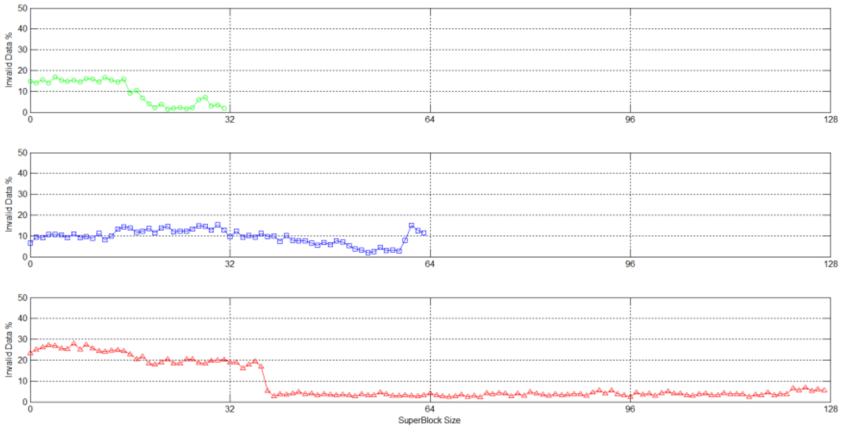
Write Amplification

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

Superblock GC

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





Real Workload Traces

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

Write Off-Loading: Practical Power Management for Enterprise Storage

Dushyanth Narayanan　Austin Donnelly　Antony Rowstron
Microsoft Research Ltd.
{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 12 W even when idle [26], whereas a dual-core Intel Xeon processor consumes 24 W 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.

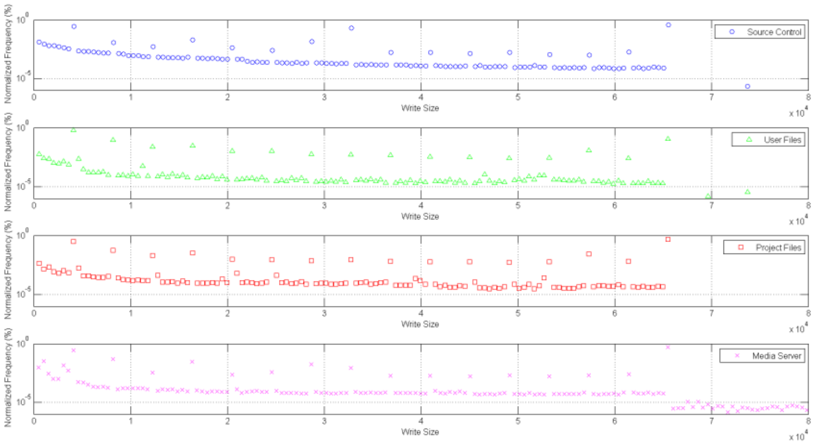
Previous work has suggested that main-memory caches are effective at absorbing reads but not writes [4]. Thus we would expect at the storage level to see periods where all the traffic is write traffic. Our analysis shows that this is indeed true, and that the request stream is write-dominated for a substantial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

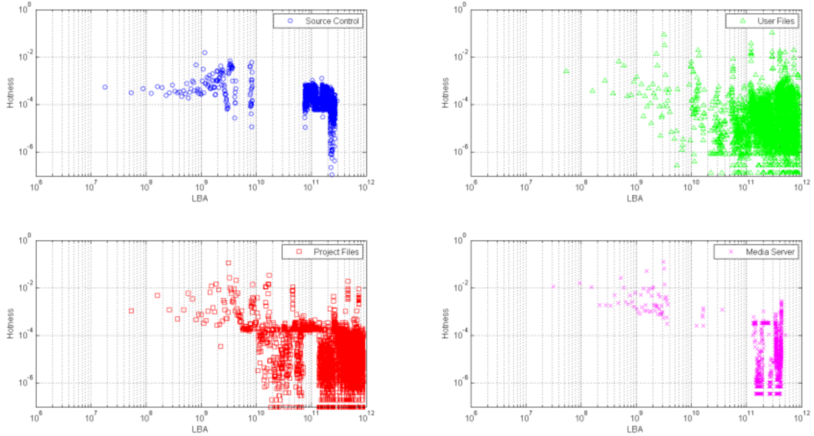
- D. Narayana et al. "Write off-loading: practical power management for enterprise storage," ACM Transactions on Storage, Vol. 4 Issue 3. 2008.

Write Workload Characteristics

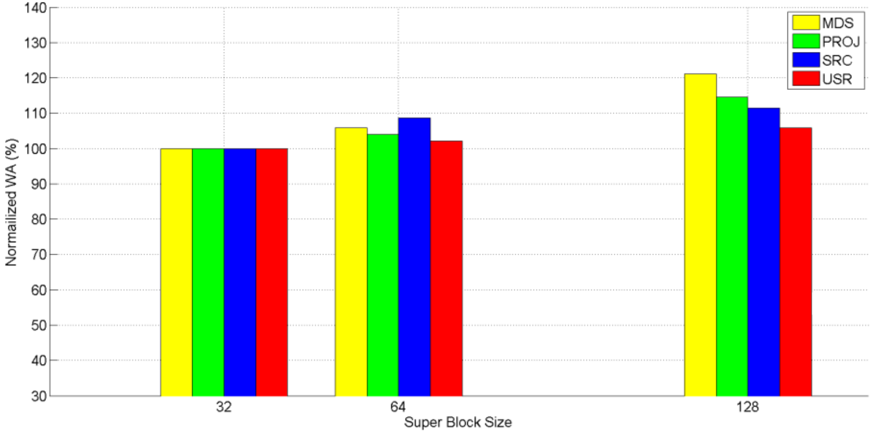


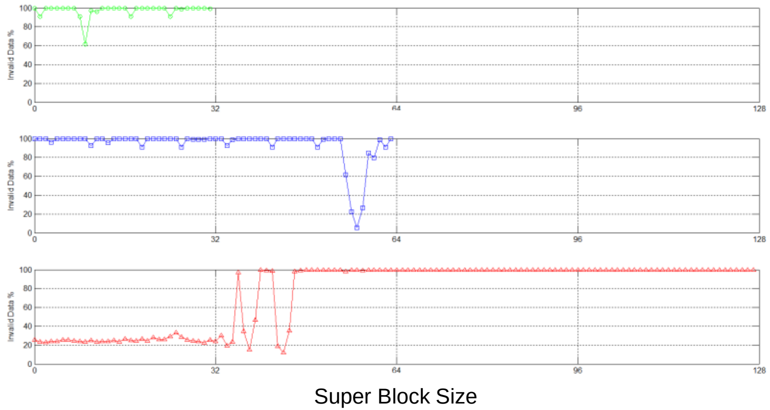
Data Size

Data Hotness



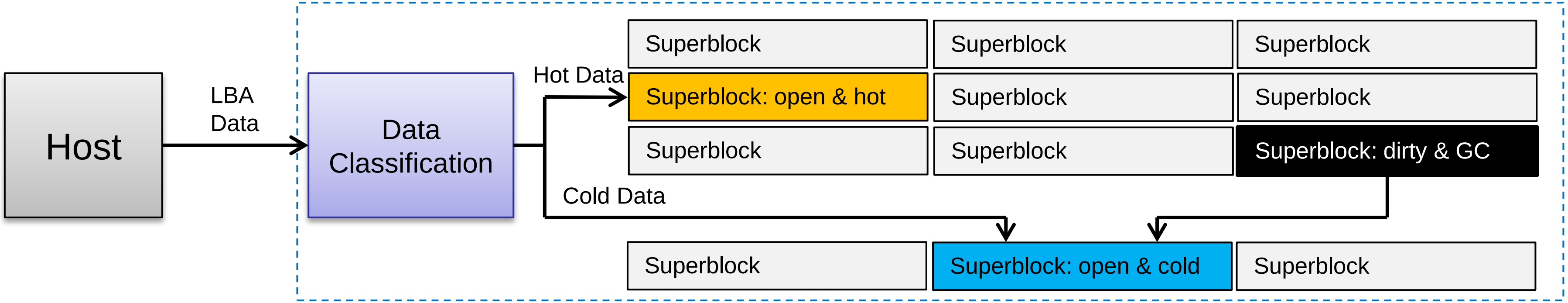
Real Workload Trace GC





Data Classification

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



Host Hints: NVMe

- Data Set Management (DSM) Hints:
  - Access Size
  - Written in Near Future
  - Sequential Write
  - Sequential Read
  - Access Latency
  - Access Frequency
    - Typical Read and Write
    - Infrequent Read and Write
    - Infrequent Write, Frequent Read
    - Frequent Write, Infrequent Read
    - Frequent Read and Write

Host Hints: Other Interface

- 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.

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.

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

Host Hints: Other Interface

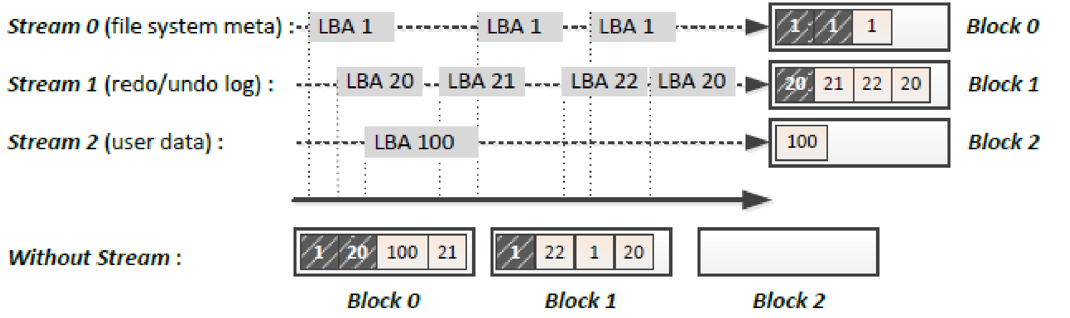


Figure 2: *The multi-streamed SSD writes data into a related NAND flash block according to stream ID regardless of LBA.* In this example, three streams are introduced to store different types of host system data.

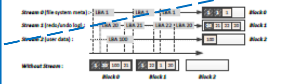


Figure 2: *The multi-streamed SSD writes data into a related NAND flash block according to stream ID regardless of LBA.* In this example, three streams are introduced to store different types of host system data.

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 tap, 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 marketed 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 `fadvise` 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-core Intel i7-3770 3.4GHz processor. We turned off power management for reliable measurements.

Table 1: Stream ID Assignment

	system	Commit-Log	flushed data	compaction data
Normal	0	0	0	0
Single	0	1	1	1
Multi-Log	0	1	2	2
Multi-Data	0	1	2	3~4
Ratio of written data (%)	1.0	48.6	31.3	4.4, 14.7

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-

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

Summary

- 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.

We Are @ Booth #818

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