Solid-State Drives (SSDs)

Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)
Systems Software & Architecture Lab.
Seoul National University

Spring 2019
Anatomy of an SSD

- Samsung 850 Evo

(Messy) Storage Interfaces

J. Pappas, Annual Update on Interfaces, FMS, 2015.
Moving Closer to the Processor

- SATA HDD: 10 ms
- SATA/SAS SSD: 150 μs
- PCIe SSD: 100 μs
- Intel Optane SSD: 10-20 μs
- Samsung Z-SSD: 10-20 μs
- DRAM NVDIMM-N: 35 ns

Operations Per Second

Milliseconds
Microseconds
Nanoseconds
Serial ATA (SATA)

- Primary internal storage interconnect for desktop and mobile PCs
  - Evolved from (Parallel) ATA
  - More than 1.1 billion SATA drives shipped during 2001-2008
  - Market share (as of 2008): Desktop (99%), Mobile PC (97.7%), Enterprise (27.6%)

- Serial, point-to-point, half duplex

- Why SATA?
  - Lower pin count (cost, space), Lower voltage support (5V → 0.7V)
  - Higher performance: SATA 3 – 600MB/s @ 6Gbps
  - Simple drive configuration (no slave)
  - Greater reliability (CRC/packet)
  - Migration to servers (hot plug, NCQ, …)
SATA NCQ

- Enqueue up to 32 commands in the drive
- Process them in an out-of-order fashion
NVMe (NVM Express)

- The industry standard interface for high-performance NVM storage
  - NVMe 1.0 in 2011 by NVM Express Workgroup
  - NVMe 1.2 in 2014

- PCIe-based

- Lower latency
  - Direct connection to CPU
  - No HBA (Host Bus Adapter) required: reduced power and cost

- Scalable bandwidth
  - 1GB/s per lane (PCIe Gen3)
  - Up to 32 lanes
NVMe Overview

- Deep queue: 64K commands per queue, up to 64K queues
- Streamlined command set: only 13 required commands
- One register write to issue a command ("doorbell")
- Support for MSI-X and interrupt aggregation
NVMe SSD Form Factors

Add-in-card (AIC)

2242 2260 2280
M.2 (PCIe: Up to x4)

U.2
(SFF-8639: Up to x4)

SATA

SAS

U.2
SSD Internals
Block Management in Solid-State Devices

(A. Rajimwale et al., USENIX ATC, 2009)

Some of slides are borrowed from the authors’ presentation.
SSD – A Different Beast

- SSDs differ from disks
  - No mechanical or moving parts
  - Contain multiple flash elements
  - Different internal architecture
  - Complex internal operations

- SSDs differ among themselves
  - Low, medium, and high end devices
  - Firmware, interconnections, mapping, striping, ganging

- Will the existing file system assumptions hold?
Problem

- Several assumptions are no longer valid

<table>
<thead>
<tr>
<th>Assumptions</th>
<th>Disks</th>
<th>SSDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential accesses much faster than random</td>
<td>✅</td>
<td>✗</td>
</tr>
<tr>
<td>No write amplification</td>
<td>✅</td>
<td>✗</td>
</tr>
<tr>
<td>Little background activity</td>
<td>✅</td>
<td>✗</td>
</tr>
<tr>
<td>Media does not wear down</td>
<td>✅</td>
<td>✗</td>
</tr>
<tr>
<td>Distant LBNs lead to longer access time</td>
<td>✅</td>
<td>✗</td>
</tr>
</tbody>
</table>
Sequential vs. Random

- Random I/O is only a few times slower than sequential I/O
- File systems must reconsider the policies for block-level sequentiality
- Block management should be moved to SSD

<table>
<thead>
<tr>
<th>Device</th>
<th>Read</th>
<th></th>
<th>Write</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seq</td>
<td>Rand</td>
<td>Ratio</td>
<td>Seq</td>
</tr>
<tr>
<td>HDD</td>
<td>86.2</td>
<td>0.6</td>
<td>143.7</td>
<td>86.8</td>
</tr>
<tr>
<td>$S_{1_{slc}}$</td>
<td>205.6</td>
<td>18.7</td>
<td>11.0</td>
<td>169.4</td>
</tr>
<tr>
<td>$S_{2_{slc}}$</td>
<td>40.3</td>
<td>4.4</td>
<td>9.2</td>
<td>32.8</td>
</tr>
<tr>
<td>$S_{3_{slc}}$</td>
<td>72.5</td>
<td>29.9</td>
<td>2.4</td>
<td>75.8</td>
</tr>
<tr>
<td>$S_{4_{slc_{sim}}}$</td>
<td>30.5</td>
<td>29.1</td>
<td>1.1</td>
<td>24.4</td>
</tr>
<tr>
<td>$S_{5_{mlc}}$</td>
<td>68.3</td>
<td>21.3</td>
<td>3.2</td>
<td>22.5</td>
</tr>
</tbody>
</table>
Write Amplification

- When write size is smaller than stripe size (read-modify-write)
- Also due to garbage collection
Background Activity

- SSDs perform a considerable amount of background activity (e.g., cleaning, wear-leveling, etc.)
- Improving QoS: Priority-aware cleaning
Block Wear

- Flash blocks have a limited P/E (Program/Erase) cycles
- The effectiveness of cleaning and wear-leveling can be improved by using file-system-level semantics
- Informed cleaning: e.g. TRIM or DISCARD command
Logical vs. Physical Distance

- Nearby LBNs do not mean physical proximity due to indirection in SSD (logical-to-physical mapping)
- File system accesses must be in terms of objects
- SSD must handle the low-level sector-specific scheduling
Object-based Storage Device?

Traditional

OSD

Application
System Call Interface
File System User Component
File System Storage Management
Sector/LBA Interface
Block I/O Manager
Physical Media

Application
System Call Interface
File System User Component
File System Storage Management
OSD Interface
OSD Storage Management
Block I/O Manager
Physical Media

Host

Storage Device
Using Data Clustering to Improve Cleaning Performance for Flash Memory

(M.-L. Chaing et al., SP&E, 1999)
Separating Hot/Cold data

- Separating hot data from cold data can reduce garbage collection overhead.

To make 1 free page, 3 pages should be copied.

To make 3 free pages, only one page should be copied.
**Dynamic dAta Clustering**

- Logically partitioning flash memory into regions
- A separate update block for each region
- Separate hot pages from cold pages
- Cluster pages of similar write access frequencies in the same region
Data Clustering

- **Key idea**
  - Dynamically clusters data not only during garbage collection, but also during data update
  - Similar to generational garbage collection
State Transitions

- Move data toward the Top region
  - Update frequencies increase
  - When the data is updated within a time threshold

- Move data toward the Bottom region
  - Update frequencies decrease
  - When cleaning the data after a time threshold (garbage collection)

- Victim selection in GC
  - Greedy, Cost-benefit, or CAT
Implementation in Page Mapping

- **Page-level mapping table**
  - Current region number for each entry
  - Timestamp (for time threshold)

- **Update blocks**
  - Maintained separately for each region

- **Per-block information**
  - Region number (optional)
  - Timestamp (for Cost-benefit and CAT)
DAC Architecture

Page Map Table

<table>
<thead>
<tr>
<th>PPN</th>
<th>Region</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>68</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>103</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>104</td>
<td>4</td>
</tr>
</tbody>
</table>

Region 1

Region 2

Region 3

Region 4

Update blocks

1

2

3
DAC Performance

- hplajw trace
  - without time threshold

- The effects of time threshold in 4 regions

No time threshold used
DAC

- **Pros**
  - Data are clustered with a low overhead during data update and GC
  - Data classification is more fine-grained than the traditional hot vs. cold classification
  - Easy to integrate in the page mapping

- **Cons**
  - The optimal number of regions depends on the workload
  - The effectiveness of the time threshold depends on the workloads
The Multi-streamed Solid-State Drive

(J.-U. Kang et al., HotStorage, 2014)

Some of slides are borrowed from the authors' presentation.
Effects of Write Patterns

- Previous write patterns (= current state) matter

Sequential LBA updates into Block 2

Need valid page copying from Block 0 & Block 1

Random LBA updates into Block 2

Just erase Block 0
Stream

SSD

Write to stream 1
Lifetime 1

Write to stream 2
Lifetime 2

Write to stream 3
Lifetime 3
The Multi-streamed SSD

- Mapping data with different lifetime to different streams

![Diagram illustrating multi-streamed SSD concept]

- Provide information about data lifetime
- Place data with similar lifetime into the same erase unit
Working Example

- High GC efficiency $\rightarrow$ Performance improvement

For effective multi-streaming, proper mapping of data to streams is essential!
Architecture

Host
- Application
- OS
- File System

Generic Block Layer

Multi-streaming Interface

SSD
- FTL
- NAND Flash memory

Application
- VFS
- EXT4
- Device

Store Stream ID
In buffer head

fadvise (fd, Stream ID)
inode field = Stream ID

SSD
Case Study: Cassandra
Cassandra’s Write Patterns

- Write operations when Cassandra runs
Mapping #1: Conventional

- Just one stream ID (= conventional SSD)
Mapping #2: Multi-App

- Separate application writes (ID 1) from system traffic (ID 0)
Mapping #3: Multi-Log

- Use three streams; further separate Commit Log
Mapping #4: Multi-Data

- Give distinct streams to different tiers of SSTables
Results: Conventional

- Cassandra’s normalized update throughput
  - Conventional “TRIM off”
Results: Conventional with TRIM

- Cassandra’s normalized update throughput
  - Conventional “TRIM on”
Results: Multi-App

- Cassandra’s normalized update throughput
  - “Multi-App” (System data vs. Cassandra data)
Results: Multi-Log

- Cassandra’s normalized update throughput
  - “Multi-Log” (System data vs. Commit-Log vs. Flushed data)
Results: Multi-Data

- Cassandra’s normalized update throughput
  - “Multi-Data” (System data vs. Commit-Log vs. Flushed data vs. Compaction Data)
Results: GC Overheads

- Cassandra’s GC overheads

The throughput is very well correlated with GC overheads
Results: Latency

- Cassandra’s cumulated latency distribution
  - Multi-streaming improves write latency
  - At 99.9%, Multi-Data lowers the latency by 53% compared to Normal
Summary

- Mapping application and system data with different lifetimes to SSD streams
  - Higher GC efficiency, lower latency
- Multi-streaming can be supported on a state-of-the-art SSD and co-exist with the traditional block interface
- Standardized in T10 SCSI (SAS SSDs) in 2015
- Standardized in NVMe 1.3 in 2017
Key-Value SSD (KVSSD)
KV Stores Common in Systems at Scale
Why KVSSD?

Key Value Store is everywhere!

- **Host S/W**
  - WiredTiger
  - RocksDB
  - levelDB

- **Thin KV Library**
  - TX/s
  - WAF, RAF, Latency

- **Traditional KV Store**
  - Block Device Driver
  - Block Device

- **KV Stacks**
  - KV Device Driver
  - KV Device
KVSSD Prototype

- Samsung KV-PM983

NGSFF KV SSD

Form factor: NGSFF/U.2
Capacity: 1-16TB
Interface: NVMe PCIe Gen.3
KVSSD Design

▪ Key size: up to 255B
▪ Value size: up to 2MB
▪ https://github.com/OpenMPDK/KVSSD
Open-Channel SSD (OCSSD)
Why OCSSD?

I/O Isolation
Enable I/O isolation between tenants by allocating your SSD into separate parallel units.

Predictable Latency
No more guessing when an IO completes. You know which parallel unit is accessed on disk.

Data Placement & I/O Scheduling
Manage your non-volatile memory as a block device, through a file-system or inside your application.

OCSSD Architecture