Prototyping a High-Performance Low-Cost Solid-State Disk

Evgeny Budilovsky, Aviad Zuck, Sivan Toledo
Tel-Aviv university
Introduction
Magnetic Disk is a Block Device

Magnetic Disk

read block
write block
SSD is Yet Another Block Device

SSD

Read(LBA)
Write(LBA, data)
We Want the Current Block Device API to Be Richer

SSD

Read(LBA)
Write(LBA, data)

?
Our Design Beats the Competition

Average Running Times

![Bar chart showing performance comparison between Existing Design and Our Design for different read and write operations. The chart includes categories like chunk read, chunk write, page read, and page write.]
Flash Background
How NAND Flash Works

- Solid state (no moving parts)
Page is Write Unit
Block is the Erase Unit

- No overwrite in-place
Block Level Mapping

SSD

Map

LBAs

0-11

24-36

12-23
SSD

LBAs
0-11
24-36
12-23

Map

Read(16)

- obsolete
- valid
- erased
Mapping data structure significantly larger
- Page-level mapping of 256GB of flash requires 256MB of RAM
- Most SSDs have small RAM (tens of MB)
Every Action Still Has Overhead

- Every request requires accessing and changing mapping data structure
- Committing and reading chunks to/from flash incurs overhead
- Random access more sensitive to this kind of overhead
Freeing Space Adds Overhead

1. 

2.
Similarly, need to change relevant mapping chunks

Only then, can we erase old block
The Design
Two-level Mapping

The root array in RAM

<table>
<thead>
<tr>
<th>LBAs 0 to c-1</th>
<th>version</th>
<th>phy. address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2,11</td>
</tr>
</tbody>
</table>

Data pages and mapping chunk pages on flash
Our Mapping Chunks are Small

- Mapping chunk size $<<$ Page size
- Writing the mapping to flash causes very little overhead
  - Chunks buffered and committed lazily to flash
  - Chunk read latency $<$ full-page read latency
- Baseline design (DFTL) used page-sized chunks
Small Mapping Chunks Improve Performance

Average Running Times

Chunks buffered and committed lazily

DFTL

Our Design
We Want to Do Even Better

Average Running Times

Existing Design

Our Design

Reduce overhead
Exploiting the Host’s **HUGE** Memory

- RAM in SSD is small
- RAM on host is large
- Perhaps we should store the mapping on the host
  - (No SSD does this)
- Keeping the host & the SSD consistent is hard
  - The SSD needs to modify the mapping (reclamations)
- **Lets cache mapping chunks on the host but treat them as hints, not as authoritative mappings**
- Send back as hints before any read/write request
- Dedicated kernel module on host-side
- Pseudo-LUN on SSD-side
Where are the Savings?

Redundant chunk read

Buffered chunk write
Implementation & Results
Prototype Implementation

- Concurrent SSD simulator, each flash chip simulated by a separate thread
- Controller code executes SCSI requests and drives simulated buses and simulated flash chips
- Garbage collection (kept it simple)
- Code runs under *tgt* (a user-space SCSI framework)
- Host-side code: single kernel module (hints cache)
Experimental Setup

- VirtualBox machine ran a Linux kernel with our hinting device driver
- SSD prototype runs on the same machine under $tgt$, and exported an iSCSI disk
- SSD configuration:
  - 8 NAND flash chips
  - 4 buses
  - 4GB Capacity
  - RAM usage in the SSD is 1MB
- Block-device synthetic workloads for all access patterns (Rand./Seq. Write/Read)
- Performance metric – actual flash accesses per SCSI request
- (Simulator is not cycle accurate)

- Comparison with DFTL (our implementation)
  - Page-size mapping chunks
  - No hinting
Small Chunks & Hinting Improve Performance

![Graph showing performance improvements with small chunks and hinting. The graph compares different operations (read and write) and shows a reduction in DFTL (Data Fetch Time Latency) and improved performance with our design.]
Performance Close to Hardware Limit

All Bars
Our Design

Hinting off

Hinting on

Full page program latency

Full page read latency

Reading
Mapping
Chunks
The Benefits of Hinting Scale with the Size of the Hints Cache

- % of SCSI requests that require a chunk read
- All random writes
- Random write workload
- 50% random
What if? Hinting More Important when Flash Latency is High

- faster bus
  - DFTL
  - Our Design

- slower flash chip
  - hinting on
  - hinting off
Lessons Learned (About Research)

- We really nailed the way to design SSDs, but
- In terms of the research, we probably should have
  - Built a cycle-accurate simulator
  - Separated the performance simulations from validation on the iSCSI framework
SSDs can be Better

- Two-level page mapping with small chunks delivers great performance, **even for random writes**
- Even with low-end SSDs (small RAM)
- Caching the entire mapping in RAM → close to optimal performance
- Either with an expensive SSD (lots of RAM)
- Or with a richer host-SSD interface (hints)
Open Source (prototype+kernel module), code at http://www.cs.tau.ac.il/~stoledo

Thank you!