Andromeda: Building the Next-Generation High-Density Storage Interface for Successful Adoption

Laura Caulfield, Michael Xing, Zeke Tan, Robin Alexander

Cloud Server Infrastructure Engineering

Microsoft Azure
Design Principles For Cloud Hardware

• Support a broad set of applications on shared hardware
  Azure (>600 services), Bing, Exchange, O365, others

• Scale requires vendor neutrality & supply chain diversity
  Azure operates in 38 regions globally, more than any other cloud provider

• Rapid enablement of new NAND generations
  New NAND every 18 months, hours to precondition, hundreds of workloads

• Flexible enough for software to evolve faster than hardware
  SSDs rated for 3-5 years, heavy process for FW update, software updated daily
### SSD Architecture

**Address Map**
- Flash Page: 16kB
- Flash Block: 4MB - 9MB
- Map Granularity: 4kB

**Data Cache**
- Enough data to fill a page

**NAND Flash**
- 4kB Writes

**Garbage Collection**
1. Copy valid data (Write Amplification)
2. Erase Block

**Attribute**
- **Size**
  - Flash Page: 16kB
  - Flash Block: 4MB - 9MB
  - Map Granularity: 4kB

**Write Amplification Factor (WAF)**
- **IO Size (MB)**
- **Host Buffer Size**

**Expectation**
- X

---

*Microsoft*
SSD Architecture

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Page</td>
<td>16kB 1MB</td>
</tr>
<tr>
<td>Flash Block</td>
<td>4MB - 9MB 1GB</td>
</tr>
<tr>
<td>Map Granularity</td>
<td>4kB</td>
</tr>
</tbody>
</table>

Address Map

Data Cache

NAND Flash

足够数据来填充条带页面

1MB写入

Block Size (MB)

Die Capacity (Gbit) -- Log Scale

Microsoft
Cloud-Scale Workloads

What is the most efficient placement of their data in an SSD’s NAND Flash Array?

<table>
<thead>
<tr>
<th>Azure Storage Backend (SOSP ‘11)</th>
<th>Vertical Stripe</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Lowest tier in hierarchy (“streaming”)</td>
<td>• High throughput through aggregation</td>
</tr>
<tr>
<td>• Write Perf. ↑, Stream Count ↑</td>
<td>• Smallest possible effective block size</td>
</tr>
<tr>
<td>• Read QoS via small reclaim unit</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application in Virtual Machine (VM)</th>
<th>Horizontal Stripe</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Small updates</td>
<td>• Each write receives peak performance</td>
</tr>
<tr>
<td>• Unaligned Peak Traffic (Bursty)</td>
<td>• Erase blocks when VM closes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>New Application in VM</th>
<th>Hybrid Stripe</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Same resources as any VM guest</td>
<td>• VM Host allocates horizontal stripe</td>
</tr>
<tr>
<td>• Adaptable to flash sizes</td>
<td>• VM Guest partitions it further</td>
</tr>
</tbody>
</table>

Allow these and other stripe dimensions simultaneously in the same SSD.
Key Observations

• Block abstraction is showing its age
  – Forces expensive reserve of flash and DRAM
  – Mixes distinct streams of data, increasing WAF
  – Even HDDs are host-managed (SMR)

• Reliability should be near media
  – Warranty & Production-scale quality are essential
  – Tuning the host to different NANDs is not practical
  – Expertise is in-house at SSD companies

• Data placement policy should be near applications
  – Trade-off between reclaim size & throughput
  – Each application has a different place on spectrum
  – Expertise is in-house at software companies

Baidu first to propose in ASPLOS ‘14
Increase Raw BW from 40% to 95%
Increase capacity from 50% to 99%

Fundamental requirements of the general purpose cloud
What do you mean by “Open Channel”?

FTL (Flash Translation Layer):
- Algorithms which allow flash to replace conventional HDDs
- Conventional systems contain the entire FTL in the Drive’s controller
- Target design divides the log and media mgmt. between host and drive

Log Managers:
- Receive random writes
- Transmit one or more streams of sequential writes
- Maintains address map, performs garbage collection

Media Managers:
- Tuned to a specific generation of media (such as Toshiba A19nm or Micron L95)
- Implement error correction such as ECC, RAID and read-retry
- Prevent media-specific errors through scrubbing, mapping out bad blocks, etc.
## Existing Proposals

<table>
<thead>
<tr>
<th>Proposal</th>
<th>Log Abstraction</th>
<th>In-Host Placement Policy</th>
<th>In-Drive Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightNVM / OCSSD 2.0 (FAST ‘17)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Software Defined Flash (ASPLOS ‘14)</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>IO Determinism (Fall ‘16)</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Multi-Streamed SSD (HotStor ‘14)</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Nameless Writes (FAST ‘12)</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Programmable Flash (ADMS ‘11)</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

Next: Iterate on OCSSD 2.0 Spec. to create production-ready system (manufacturable, warrantable, etc.)
Physical Page Addressing (PPA) Interface

Host and drive communicate with physical addresses

• Terminology
  – Channel: SSD Channel
  – Parallel Unit (PU): NAND Die
  – Chunk: multi-plane block
  – Sector: 512B or 4k region of NAND page

• All addresses map to a fixed physical location

• Host IO Requirements:
  – Erase a chunk before writing any sectors
  – Write sectors within the chunk sequentially
  – Read any sector that is not within “cache minimum write size”

Address Format:

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>Parallel Unit</td>
</tr>
</tbody>
</table>

**Chunk Address: Logical or Physical?**

*Host requires no resource guarantees rooted in physical location for chunk – option to make chunk address logical*

**If Chunk Address Remains Physical**

- Host manages all wear leveling
  - Monitors erase count on every block
  - Uses vector copy command
- Drive monitors wear, provides backup
  - Low water mark: notifies host of uneven wear
  - High water mark: moves data and notifies host of new address

**If Chunk Address Were Logical**

- Drive guarantees even wear within die
- Wear leveling integrated with scrubbing
- Simple wear leveling
  - Full, block-granular map
  - Swap most/least worn block at regular period
- Start-gap style wear-leveling
  - No map (equation-based)
  - Move 1 block of data at regular period
- Host issues maintenance “go” and “stop”

Remapping the block number within the drive has much lower performance and space overheads vs. current SSD designs
Reliability and QoS

**Tight QoS Guarantees**

- OCSSD 2.0
  - Read and write times include ECC
  - “Heroic ECC” is binary

- Future Enhancement
  - Use reported times as “base” performance
  - Table to select reliability/latency
  - Selectable per chunk? Per stripe?

**RAID**

- RAID and isolation are at odds
  - Independent work scheduled to each die
  - Must build parity and decode on all dies

- Higher level replication
  - RAID within drive is *sometimes* redundant
  - Require information & mechanisms for host to selectively use in-drive RAID
  - Leverage vector reads/writes
Cache Minimum Write Size

Mitigating Open Memory Cell Read Disturb, defining the right abstraction for the “open block timer”

- Open NAND cells susceptible to read disturb
- Cache the last 3-5 pages written to any write point
- Single-shot programming is not susceptible
- Host-Device Contract:
  - Cache min write size (CMWS) = max kB in open cells
  - Host queries for CMWS (0kB if drive caches enough)
  - Host does not read from last CMWS of data
  - Drive fails reads to CMWS region

“Vulnerabilities in MLC NAND Flash Memory Programming.” Yu Cai et al. HPCA 2017
Telemetry for Physical Addressing

• OCSSD 2.0: “Chunk Report”
  – Enhancement: use an NVMe log page
  – Supported only in open channel drives
  – Include additional telemetry (at right)

• Vendor-specific log page
  – Standard address
  – Vendor-specific content
  – May be temporary, used only while the open channel interface matures

Additional Standard Telemetry

• Erase count per PU
  Host may use this to determine which PUs to swap during cross-die wear leveling.

• Erase count per chunk
  If chunk addressing is logical, host may use this to reduce drive activity by freeing little-worn chunks.

• Read disturb count per sector
  Host may use this to reduce drive activity by scrubbing or distributing reads.
Conclusions

• Storage interface is changing, let’s architect it for the long term
  – Correct division of responsibilities between Host and SSD
  – Control to define heterogeneous block stripes
  – HyperScale: Hundreds or thousands of workers per TB

• Final solution must include expertise from community
  – Currently working through the division between host and SSD
  – Contact me to discuss (Laura.Caulfield@microsoft.com)
  – Read more in our FAST 2017 paper: FlashBlox
References

• Azure Storage Backend (SOSP ‘11)
  *Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency*

• FlashBlox (FAST ‘17)
  *FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs*

• LightNVM (FAST ‘17)
  *LightNVM: The Linux Open-Channel SSD Subsystem*

• Read Determinism (SDC ‘16)
  *Standards for improving SSD performance and endurance*

• Software-Defined Flash (ASPLOS ‘14)
  *SDF: Software-Defined Flash for Web-Scale Internet Storage Systems*

• Multi-Streamed SSD (HotStor ‘14)
  *The Multi-streamed Solid-State Drive*

• De-Indirection (FAST ‘12)
  *De-Indirection for Flash-based SSDs with Nameless Writes*

• Programmable Flash (ADMS ‘11)
  *Fast, Energy Efficient Scan inside Flash Memory SSDs*