DENALI

THE NEXT-GENERATION HIGH-DENSITY STORAGE INTERFACE

Laura Caulfield
Senior Software Engineer

Arie van der Hoeven
Principal Program Manager

Microsoft

OPEN. FOR BUSINESS.
Outline

• Technology Trends & Application Requirements

• Proof-of-Concept

• Host-Drive Specification
Outline

• Technology Trends & Application Requirements

• Proof-of-Concept

• Host-Drive Specification
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 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**

![Diagram of SSD Architecture]

- **Address Map**
- **Data Cache**
- **Flash Page**
- **Flash Block**

---

**Write Amplification Factor (WAF)**

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

---

**Garbage Collection:**

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

---

**Expectation**

- Enough data to fill a page

---

**Graph:**

- **Write Amplification Factor (WAF)** vs. **IO Size (MB)**

---

**Legend:**

- A1-960GB
- B1-480GB
- C1-480GB
- D1-480GB
- D2-480GB
- E1-480GB
- E2-960GB
- E2-480GB

---

**NAND Flash Architecture**

- **Address Map**
- **Data Cache**
- **Flash Page**
- **Flash Block**

---

**Attribute**

- **Size**

---

**4kB Writes**
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>

1MB Writes

Enough data to fill a striped page

Address Map

Data Cache

NAND Flash

Die Capacity (Gbit) -- Log Scale

Block Size (MB)
## Cloud-Scale Workloads

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

### Azure Storage Backend (SOSP ’11)
- Lowest tier in hierarchy (“streaming”)
- Write Perf. ↑, Stream Count ↑
- Read QoS via small reclaim unit

### Application in Virtual Machine (VM)
- Small updates
- Unaligned Peak Traffic (Bursty)

### New Application in VM
- Same resources as any VM guest
- Adaptable to flash sizes

<table>
<thead>
<tr>
<th>Vertical Stripe</th>
</tr>
</thead>
<tbody>
<tr>
<td>• High throughput through aggregation</td>
</tr>
<tr>
<td>• Smallest possible effective block size</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Horizontal Stripe</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Each write receives peak performance</td>
</tr>
<tr>
<td>• Erase blocks when VM closes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hybrid Stripe</th>
</tr>
</thead>
<tbody>
<tr>
<td>• VM Host allocates horizontal stripe</td>
</tr>
<tr>
<td>• VM Guest partitions it further</td>
</tr>
</tbody>
</table>

Allow these and other stripe dimensions simultaneously in the same SSD
Outline

• Technology Trends & Application Requirements

• Proof-of-Concept

• Host-Drive Specification
**Terminology**

**Open Channel SSD:**
Drive exposes physical addresses such as channels

**Denali SSD:**
Drive exposes logical hierarchy of addresses that map to physical attributes

**FTL (Flash Translation Layer):**
Algorithms which allow SSD to replace conventional HDDs

**Log Manager:**
Receives random writes
Transmits one or more streams of sequential writes
Maintains address map, performs garbage collection

**Media Manager:**
Written for a specific generation of media
Implements error correction such as ECC, RAID and read-retry
Prevents errors through scrubbing, mapping out bad blocks, etc.

**Evolution of the Architecture**

**Standard SSD**
- Host
- Drive
- FTL
- Log Mgmt.
- Media Mgmt.

**Open Channel SSD**
- Host
- Drive
- FTL
- Log Mgmt.
- Media Mgmt.

**Denali SSD**
- Host
- Drive
- FTL
- Log Mgmt.
- Media Mgmt.

**Evolution**
- Prototype OCSSD 1.2
- Production Ready OCSSD 2.0
POC Test Configuration

✓ POC Goal
Migrate FTL to Azure’s kernel
# Results: Optimizing System’s Overheads

*FW-based algorithms overheads are static, the host has information and flexibility to reduce them dynamically*

Write Amplification (4k Random Writes)
- Better end-to-end WAF – logic in FTL library is efficient
- Optimize host-side WAF using workload information

Memory
- 1GB of DRAM / TB of flash for address map
- Optimize map: sparse, granularity, dynamic allocation

CPU
- Implementation Specific Overheads in prototype
- Further optimization through end-to-end WAF reductions

<table>
<thead>
<tr>
<th>Write Amplification Factor</th>
<th>Host</th>
<th>Drive</th>
<th>End-to-End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional SSD</td>
<td>0.0</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Open Channel SSD</td>
<td>0.6</td>
<td>0.8</td>
<td>1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory (GB per TB of Flash)</th>
<th>Conventional SSD</th>
<th>Open Channel SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.0</td>
<td>1.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU (Cores/Drive)</th>
<th>Conventional SSD</th>
<th>Open Channel SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.0</td>
<td>3.5</td>
</tr>
</tbody>
</table>

![Optimize](Optimize)

**POC Goal**
Quantify opportunity for optimization of resources
Results: Performance Parity

• Workloads:
  – Seq: 4 threads, QD 32, 128kB
  – Mix: 4 threads, QD 4, 4kB
  – Rand: 4 threads, QD 32, 4kB, 70/30
• Read Perf.: Top in Class
• Write Perf.: Pending Typical Optimizations

• Workload:
  – Measured: 4kB random reads, QD 1
  – Background: 256kB random writes
• Top-in-class

✓ POC Goal
Remain competitive with conventional SSDs’ performance
Outline

• Technology Trends & Application Requirements

• Proof-of-Concept

• Host-Drive Specification

Logical Hierarchical Addressing

• Each field maps to logical part of architecture
  – Flexibility in HW to manage NAND (such as mapping out bad blocks)
  – System can implement 2-part wear leveling*
  – Overheads significantly lower than conventional SSDs

• Host IO Requirements
  – Allocate a fresh a chunk before writing any sectors
  – Write sectors within the chunk sequentially
  – Some new elements to abstract NAND management, for example, the cache minimum write size

Address Format:

<table>
<thead>
<tr>
<th>Group</th>
<th>Parallel Unit</th>
<th>Chunk</th>
<th>Sector</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>Group: SSD Channel</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parallel Unit (PU): NAND Die</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chunk: multi-plane block</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSB</td>
<td>Sector: 512B or 4k region of NAND page</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* “FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs” Huang et al, USENIX-FAST 2017
**Cache Minimum Write Size (CMWS)**

*Defining a logical abstraction for an idiosyncrasy of NAND flash physics*

- Open NAND cells susceptible to read disturb
- Example: Cache the last 3-5 pages written to any write point
- Host-Device Contract:
  - CMWS = max kB in open cells
  - CMWS = 0kB if drive caches to mitigate the effect
  - Host queries for CMWS
  - Drive fails reads to CMWS region

---

“Vulnerabilities in MLC NAND Flash Memory Programming.” Yu Cai et al. HPCA 2017
Reliability and QoS

This is the same challenge that the IO Determinism community is working to solve.

• RAID and isolation are at odds
  (Small tenant == high RAID overheads)

• Mechanism must enable spectrum of users
  – Many tenants use cross server replication, don’t require RAID
  – Some require standard reliability

• Solution: IO Determinism’s Read Recovery Levels
Conclusions

• Let’s architect the new storage interface 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

• Successful proof-of-concept
  – System overheads: as expected & ready for optimization
  – Performance parity on standard microbenchmarks
  – Next step: complete interface for warrantable Open-Channel SSD

• Final solution must include expertise from community
  – Currently working through the division between host and SSD
  – Contact us to discuss
  – 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