From server-side to host-side: Flash memory for enterprise storage

Jiri Schindler et al. (see credits)
Advanced Technology Group
NetApp

May 9, 2012
How do we make effective use of flash SSDs while preserving the benefits of shared storage?
Step I. Replace HDDs with SSDs
SAS Disk Shelves

- Each disk in a carrier
  - Hot-swappable
  - 3.5” or 2.5” form-factor
- Serial-attached SCSI expanders
  - 36-port cross-bar switch
  - Single link: 3 or 6 Gb/s, ~60-80K IOPS
Daisy-chaining Disk Shelves

- Single Flash SSD
  - 10-12K IOPS
  - ~125 MB/s

- Port-to-port links
  - Opened individually

- As chain grows:
  - IOPS diminished
  - BW limitations

SAS back-end (w/ many shelves) can be an IOPS bottleneck
Step II. Optimize for ONTAP Data Path
Flash Cache (PAM-II Card) Overview

- NetApp-designed card
  - No COTS design existed
  - FPGA controller
  - 512GB SLC Flash

Up to 4 cards in a single FAS controller (up to 8 in FAS6xx0 series)

- Specific to Data ONTAP® I/O data path
  - Read-only victim cache behind RAM buffer cache
- Minimal SW changes
  - Leverage existing RAM-based PAM card design
  - Buffer tags in RAM, simple FTL
Managing Flash Cache

- Flash acts as buffer for read-only (clean) data
  - No “in-place” overwrites of cached data
  - Invalidation of existing mapping on new write

- Simple FTL
  - Circular buffer w/ generation garbage collector
    - implicit wear leveling

- Tag store for buffer headers in RAM
  - Takes away RAM buffer cache space for data
    - Non-trivial with 8 PAM-II cards/4TB of Flash
OLTP-like Workload Performance

Baseline: FAS 3160 with 16 shelves of 15k RPM 300GB HDDs

<table>
<thead>
<tr>
<th></th>
<th>Baseline (84 HDDs)</th>
<th>Add Disks (140 HDDs)</th>
<th>Add PAM-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOPS</td>
<td>31504</td>
<td>55297</td>
<td>55230</td>
</tr>
<tr>
<td>Resp. Time (ms)</td>
<td>27.5</td>
<td>27.5</td>
<td>16.9</td>
</tr>
<tr>
<td>Power (kW)</td>
<td>3.1</td>
<td>6.8</td>
<td>3.1</td>
</tr>
<tr>
<td>Rack Space (U's)</td>
<td>24</td>
<td>54</td>
<td>24</td>
</tr>
</tbody>
</table>

1.6x Same operational costs, 30% COGS price reduction w/ PAM-II

### SPECsfs2008 (nfs.v3) Performance

Baseline: FAS 3160 with 16 shelves of 15k RPM 300GB HDDs

<table>
<thead>
<tr>
<th>Throughput (IOPS)</th>
<th>Baseline (FC-AL HDDs)</th>
<th>FC-AL Disks w/ PAM II</th>
<th>SATA Disks w/ PAM-II</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IOPS</strong></td>
<td>60409</td>
<td>60507</td>
<td>60389</td>
</tr>
<tr>
<td><strong>Resp. Time (ms)</strong></td>
<td>2.18</td>
<td>1.58</td>
<td>2.18</td>
</tr>
<tr>
<td><strong>Disks</strong></td>
<td>224</td>
<td>56</td>
<td>96</td>
</tr>
<tr>
<td><strong>Rack Space (U's)</strong></td>
<td>35</td>
<td>11</td>
<td>17</td>
</tr>
</tbody>
</table>

**Cost savings: replace FC-AL with fewer SATA HDDs & PAM-II**

Step III. Combine with Host-side

Project Mercury

Presented at MSST ’12 Conference
Data Centers with Flash SSDs

Shared (Virtualized) Compute

Shared Storage

Reliability and availability is different at host-side
Available and Durable

Goal
- Never lose data in any situation

Consequence
- Write-through policy

Chose common denominator policy
Other policies possible that leverage app’s specifics
Correct and Consistent

Goals
- Consistency with storage array
- Consistent with peers

Consequences
- Cache non-shared objects
- Invalidate on migration, restore, etc.

Horizontal Consistency

Vertical Consistency

VM
Hypervisor Host
Host Cache
Storage System

VM
Hypervisor Host
Host Cache

VM
Administrative Host
Datacenter Management Integration

Goal
- Simple and transparent management

Consequence
- Hypervisor integration

Most important for the end-user deployment
Design & Implementation
Prototype Implementation Overview

- **Write-through**
  - Simple cache consistency

- **Persistent**
  - Warm cache on restart
  - Cache durability after a crash is ongoing work

Diagram:
- Guest VM
- QEMU
- Virtual device emulation
- Mercury Cache
- I/O Stack
- Linux with KVM
- Shared Storage
- Local Flash
- FCoE/ISCSI/NFS/CIFS
- SSD
- User
- Kernel
- Write-through
- Persistent
Detecting Cache Hits

- All cache metadata in RAM for speed.
  - Mercury is a second-level cache → modest hit rate → minimize cache overhead
  - Memory-to-cache ratio is 0.5%
    (e.g., 500 GB cache requires 2.5 GB of RAM)
- Cache headers
  - One header for each block in the cache
- Address Map
  - (primary storage, LBA) keys, header index values
  - Implemented with hash table, O(1) lookup time
Cache Insertion

- Specialize I/O access patterns for flash
  - LFS-style writes
    - Large chunks match erase (meta) block size
    - Minimizes cleaning/slowdown at the SSD FTL
Admittance Policies

- Unrestricted (default)
  - All writes and read misses inserted in the cache

- Write-Around
  - Writes skip the cache

- Sequential I/O Bypass (ongoing work)
  - Sequential reads, writes, or both skip the cache
Results
Evaluation Setup

- Two workloads:
  - Microsoft® Exchange Jetstress
  - NetApp® Enterprise Workload

- Flash cache
  - PCIe device with SLC (single-level cell) flash
    - Paper contains SLC and MLC SSD results

- Other hardware
  - x86 Server with Linux, KVM/QEMU
  - NetApp FAS3270 with iSCSI LUN(s)

---

1 S. Daniel et al., A portable, open-source implementation of the SPC-1 workload.
Significant Response Time Improvement

Enterprise workload. Unrestricted admittance policy. CLOCK eviction policy.

Response time over 30ms fails the test

Cache capacity 11.25% of dataset
Cache capacity 3.5% of dataset
No cache
Reducing Access to Networked Storage

Jetstress workload. Unrestricted admittance policy. FIFO eviction policy.

3x fewer reads
Warming Cache: Takes a Long Time

Enterprise workload. Unrestricted admittance policy. Flash capacity set to 11.25% of dataset.

Many to reach steady state
Unrestricted Beats Write-Around

Enterprise workload. CLOCK eviction policy. Flash device capacity set to 11.25% of dataset.
Host-side Flash Summary

- **Host-side flash**
  - minimizes flash access latency

- **Hypervisor-based I/O cache**
  - simplifies deployment

- **Persistent**
  - cache is warm on a restart

- **Write-through**
  - consistent with primary storage
Concluding Remarks

- Working with real-world constraints
  - Timing is everything

- Design for the long haul
  - Deliver *something* useful fast

- Learn from the users
  - Collect field data

- Improve design in iterations over-time
Credits

- Efforts of many product engineers

- Project Mercury
  
  Steve Byan          James Lentini  
  Anshul Madan       Luis Pabón    
  Michael Condict    Jeff Kimmel   
  Steve Kleiman      Christopher Small  
  Mark Storer

Advanced Technology Group
NetApp
Thank you