Mercury: Host-side Flash Caching for the Data Center

Steve Byan
Anshul Madan
Michael Condict
Steve Kleiman
Mark Storer

James Lentini
Luis Pabón
Jeff Kimmel
Christopher Small

Advanced Technology Group
NetApp

© 2012 NetApp, Inc. All rights reserved.
Data Center with Flash SSDs

How do we make effective use of flash SSDs while preserving the benefits of shared storage?
Outline

- Part I: Architecture
- Part II: Design and Implementation
- Part III: Evaluation
Part I. Architecture
Four Architectural Goals

- Consistently High Performance
- Highly Available
- Correct and Consistent
- Simple Management Integration
Consistently High Performance

Goals
- Realize the low latency access
- Meet Service Level Objective (SLO) after restart

Consequences
- Direct-attached to host
- Persistent, preferably durable
Highly Available

Goal
- Never lose data in any situation

Consequence
- Write-through
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
Simple Management Integration

Goal
- Simple and transparent management

Consequence
- Hypervisor integration
Part II. Design and Implementation
Implementation Overview

- **Write-through**
  - Simplifies cache consistency

- **Persistent**
  - Warm cache on restart
  - Cache durability after a crash is future work
Simplified I/O Flow

Start → Write? → Yes → Perform I/O → Admit? → No → End

Write? → Yes → Hit? → No → End

Hit? → Yes → Invalidate → Yes → Insert Into Cache → No → Read from Cache

Hit? → No → No (read) → Hit? → Yes
Read Processing Example (1 of 6)

Start → Write? Yes → Hit? No → Perform I/O

Yes → Invalidate

No (read) → Hit? No → End

Yes → Read from Cache

Perform I/O → Admit? No → End

Yes → Insert Into Cache

Hit? Yes → Read from Cache
Read Processing Example (2 of 6)

Start → Write? → Yes → Hit? → No → Perform I/O → Admit? → No → End
→ No (read) → Hit? → No → Invalidate

Yes → Perform I/O

Yes → Admit?

Yes → Insert Into Cache

No → Read from Cache
Read Processing Example (3 of 6)

Start

Write? Yes

Hit? No

Perform I/O

Admit? No

End

No (read)

Hit? No

Invalidate

Yes

Insert Into Cache

Yes

Read from Cache

No

Yes

Yes
Read Processing Example (4 of 6)

How is a hit detected?

Flowchart:
- Start
  - Write?
    - Yes
      - Hit?
        - Yes
          - Perform I/O
        - No
          - Admit?
            - Yes
              - Insert Into Cache
            - No
              - End
    - No (read)
      - Hit?
        - Yes
          - Read from Cache
        - No
          - End
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
  - Implemented as a simple array

- Address Map
  - Dictionary maps (primary storage, LBA) keys to header index values
  - Implemented with hash table, O(1) lookup time
Data Structures

Address Map

Header Array

Current Chunk

In memory
On flash

Superblock | Map Region | Chunk 0 | Chunk 1 | Chunk 2 | Chunk 3 | Chunk 4 | … | Chunk n

Block Addresses

© 2012 NetApp, Inc. All rights reserved.
Read Processing Example (5 of 6)

- Start
  - Write? Yes → Hit? No → Perform I/O → Admit? Yes → Insert Into Cache → End
  - Write? No (read) → Hit? No → Read from Cache
  - Write? No (read) → Hit? Yes → Invalidate

- Hit? Yes → Perform I/O
- Hit? No → End
Read Processing Example (6 of 6)

Start → Write? → Yes → Hit? → No → Perform I/O → Admit? → No → End

Yes → Invalidate

No (read) → Yes

Hit? → No

Yes → Read from Cache

No

Insert Into Cache → Yes
Write Processing Example (2 of 11)

Start → Write? → Yes → Hit? → No → Perform I/O → Admit? → No → End

No (read) → Hit? → No → Invalidate

Yes → Perform I/O

Admit? → Yes → Insert Into Cache

Yes → Read from Cache
Write Processing Example (3 of 11)

Start → Write? → Yes → Hit? → No → Perform I/O → Admit? → No → End

No (read) → Hit? → No → Invalidate

Yes → Insert Into Cache → Yes

Read from Cache
Write Processing Example (4 of 11)

Start
→ Write? → Yes → Hit? → No → Perform I/O → Admit? → No → End

No (read)
↓
→ Hit? → No

Yes
↓
→ Invalidation

Yes
↓
→ Insert Into Cache

No
↓
→ Read from Cache

Yes
Write Processing Example (5 of 11)

Start → Write?  Yes → Hit?  No → Perform I/O → Admit?  No → End

Yes → Invalidate

No (read) → Hit?  No

Hit?  Yes → Read from Cache

Yes → Insert Into Cache
Write Processing Example (6 of 11)

Start

Write?

Yes

No (read)

Hit?

No

Hit?

Yes

Invalidate

Yes

Perform I/O

Admit?

No

End

No

Yes

Insert Into Cache

Read from Cache
Write Processing Example (7 of 11)

- Start
- Write?
  - Yes
    - Hit?
      - Yes
        - Perform I/O
          - Admit?
            - Yes
              - Insert Into Cache
            - No
              - Read from Cache
      - No
        - No (read)
          - Hit?
            - Yes
              - Perform I/O
            - No
              - Read from Cache
  - No

What is the admittance policy?
Admittance Policies

- Unrestricted (default)
  - All writes and read misses are inserted into the cache
- Write-Around
  - Writes skip the cache
- Sequential I/O Bypass (future work)
  - Sequential reads, writes, or both skip the cache
Write Processing Example (8 of 11)

Start

- Write? Yes
  - Hit? No
    - Perform I/O
    - Admit? No
      - End
    - No (read)
      - Hit? No
        - Read from Cache
      - Yes
    - Invalidation
  - Hit? No
    - Insert Into Cache
    - Yes
      - End
  - No (read)

© 2012 NetApp, Inc. All rights reserved.
Write Processing Example (10 of 11)

What is the eviction policy?

- Insert Into Cache
- Read from Cache
- Yes
- No
- Hit?
- No (read)
- Yes
- Hit?
- No
- Yes
- Perform
- Admit?
- No
- End
- Invalidate
- Yes
- Hit?
- No
- Write?
- Yes
- Start
Eviction Policies

- **First In First Out (FIFO)**
  - Less I/O to clean log, but lower hit rate
    - Eliminates reads during log cleaning.

- **CLOCK**
  - Higher hit rate, but more expensive log cleaning compared to FIFO.
Write Processing Example (9 of 11)

How do we efficiently insert into the cache?
Cache Insertion

- Specialize I/O access patterns for flash
  - Log-structured writes with erase block size chunks to minimize SSD FTL’s (flash translation layer) cleaning
Write Processing Example (11 of 11)

Start

Write? Yes → Hit? No → Perform I/O

No (read)

Hit? No → Read from Cache

Yes

Invalidate

Yes

Admit? No → End

Yes

Insert Into Cache
Part III. Evaluation
Evaluation

- Two workloads:
  - Microsoft® Exchange Jetstress
  - NetApp® Enterprise Workload\(^1\)
- PCIe device with SLC (single-level cell) flash
  - Paper contains SLC and MLC SSD results
- 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.*
Cache reduces access to network storage

Jetstress workload. Unrestricted admittance policy. FIFO eviction policy. PCIe flash device.
Warming the cache takes a long time

Enterprise workload. Unrestricted admittance policy. PCIe flash device capacity 11.25% of dataset.
Enterprise workload. CLOCK eviction policy. PCIe flash device capacity 11.25% of dataset.
Significant Response Time Improvement

Enterprise workload. Unrestricted admittance policy. CLOCK eviction policy. PCIe flash device.

A response time of over 30ms fails the test.
Summary

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

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

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

- **Write-through**
  - consistent with primary storage
Thank you