Request-Oriented Durable Write Caching for Application Performance

Sangwook Kim¹, Hwanju Kim², Sang-Hoon Kim³, Joonwon Lee¹, and Jinkyu Jeong¹

Sungkyunkwan University¹
University of Cambridge²
Korea Advanced Institute of Science and Technology (KAIST)³
Non-volatile Write Cache

- **Volatile** DRAM cache is ineffective for write
  - Writes are dominant I/Os [FAST’09, FAST’10, FAST’14]

- Non-volatile write cache (NVWC) provides
  - **Fast response** for write w/o loss of durability

- NVWC candidates

[**Bhadkamkar et al., FAST’09**] BORG: Block-reORGanization for self-optimizing storage systems
[**Koller et al., FAST’10**] I/O deduplication: Utilizing content similarity to improve I/O performance
[**Harter et al., FAST’14**] Analysis of HDFS under HBase: a Facebook messages case study
Non-volatile Write Cache

- Simple caching policy

![Diagram showing the flow of data from Application to Backing Storage through NVWC and Operating System.]

Application

P1 \(\rightarrow\) Write

P2 \(\rightarrow\) Write

P3 \(\rightarrow\) Write

Operating System

NVWC

Backing Storage

Blindly caching all writes

Lazily writing back to storage
Non-volatile Write Cache

- Simple caching policy

No consideration for application performance

Operating System

NVWC

Backing Storage

Blindly caching all writes

Lazily writing back to storage
Impact on Application Performance

- Illustrative experiment
  - TPC-C workload
    - PostgreSQL database
  - 2 NVWC devices
    - 32MB NV-DRAM (emulated via ramdisk)
    - 4GB flash SSD
Impact on Application Performance

- Experimental result

![Bar chart showing impact on IOPS and transactions per minute. The chart indicates a 2.1X improvement in IOPS and a 1.7X improvement in transactions per minute.]
Impact on Application Performance

- Experimental result

* System perf.
  - ~ 2.1X improved

* Application perf.
  - ~ 50% degraded
What’s the Problem?

- Criticality-agnostic contention
Criticality-Agnostic Contention

- Different write criticality
Criticality-Agnostic Contention

- Different write criticality

Client

Request → P1 → P2 → P3 → Application

Response

Operating System

Background process/thread

NVWC

Backing Storage
Criticality-Agnostic Contention

- Different write criticality

Diagram:

- Client
  - Request
  - Response
  - Application

- P1
  - Critical

- P2
  - Non-critical

- P3

- Operating System

- NVWC

- Backing Storage
Criticality-Agnostic Contention

- Different write criticality

* Contentions
  - Capacity contention
  - Bandwidth contention
Criticality-Agnostic Contention

- Capacity contention

![Diagram showing client, request, response, application, operating system, and backing storage with critical and non-critical levels and frequent write stalls and bounded writeback throughput.]
Criticality-Agnostic Contention

- Bandwidth contention

Excessive queueing delay

Sufficient free blocks

Backward Storage

Client

Request

Response

Application

P1

P2

P3

Critical

Non-critical

Critical

Non-critical

Critical
Our Approach

- Request-oriented caching policy

* Definitions
  - **Critical process (CP):** a process handling request
  - **Critical write:** a write awaited by a critical proc.
Challenge

- How to accurately detect critical writes

Types of critical write
- Sync. writes from critical processes
- Dependency-induced critical writes
  - Process dependency-induced
  - I/O dependency-induced
Dependency Problem

- Process dependency
Dependency Problem

- I/O dependency

* Example scenarios:
  - CP `fsync()` to a block under writeback issued by NCP
  - CP tries to **overwrite** fs journal buffer under writeback
Critical Write Detection

- Critical process identification
  - Application-guided identification
Critical Process Identification

- Application-guided identification

![Diagram showing Critical Process Identification]
Critical Write Detection

- Critical process identification
  - Application-guided identification

- Dependency resolution
  - Criticality inheritance protocols
    - Process criticality inheritance
    - I/O criticality inheritance
    - Blocking object tracking
Critical Write Detection

- Critical process identification
  - Application-guided identification

- Dependency resolution
  - Criticality inheritance protocols
    - Process criticality inheritance
    - I/O criticality inheritance
    - Blocking object tracking
Criticality Inheritance Protocols

- Process criticality inheritance

![Diagram of criticality inheritance protocols]

- NCP
- Lock
- Inherit
- Wake
- CP
- B1
- B2
- B3
- B4
Criticality Inheritance Protocols

- I/O criticality inheritance

Key issue:
caching the dependent write outstanding to disk w/o side effects
Evaluation

• Implementation on Linux 3.13 w/ FlashCache 3.1

• Application studies
  • PostgreSQL database
  • Redis key-value store

Client 1, 2, 3, ...

Back end1, Back end2, Check pointer, Log writer, Writer

Master, Snap shotter, Log rewriter
Evaluation

- Experimental setup

Server Machine

- PostgreSQL / Redis
- FlashCache
- 4GB ramdisk / 256GB SSD
- 10K RPM HDD x2

Client Machine

TPC-C / YCSB

1Gbps
Evaluation

- Experimental setup

* Caching policies
  - ALL (default)

Server Machine

POSTGRESQL / REDIS

FLASHCACHÉ

4GB RAMDISK / 256GB SSD

10K RPM HDD x2

No discretion

1Gbps

Client Machine

TPC-C / YCSB
Evaluation

- Experimental setup

* Caching policies
  - ALL (default)
  - SYNC

Server Machine

- PostgreSQL / Redis
- FlashCache
- 4GB ramdisk / 256GB SSD
- 10K RPM HDD x2

Sync. writes

Client Machine

1Gbps

TPC-C / YCSB
Evaluation

• Experimental setup

- PostgreSQL / Redis
- FlashCache
- 4GB ramdisk / 256GB SSD
- 10K RPM HDD x2

* Caching policies
  - ALL (default)
  - SYNC
  - CP

Server Machine

Client Machine

TPC-C / YCSB

1Gbps

CP sync. writes
Evaluation

- Experimental setup

* Caching policies
  - ALL (default)
  - SYNC
  - CP
  - CP+PI

Server Machine

10K RPM HDD x2

4GB ramdisk / 256GB SSD

FlashCache

PostgreSQL / Redis

+ Process criticality inheritance

Client Machine

TPC-C / YCSB

1Gbps
Evaluation

- Experimental setup

* Caching policies
  - ALL (default)
  - SYNC
  - CP
  - CP+PI
  - CP+PI+IOI
Evaluation

• Experimental setup

* Caching policies
  - ALL (default)
  - SYNC
  - CP
  - CP+PI
  - CP+PI+IOI
  - WAL (PostgreSQL)

TPC-C / YCSB
PostgreSQL Performance

- TPC-C workload w/ ramdisk

Same performance w/ 72% less cached writes

Our scheme resolves capacity contention & runtime dependencies

Scarce

Sufficient
PostgreSQL Performance

- TPC-C workload w/ SSD

Our scheme resolves bandwidth contention & runtime dependencies.
Redis Performance

- Update-heavy workload w/ 16GB SSD

- **47% better throughput**
- **Improved tail latency**
  - 13X better @ 99.9th %ile
  - (50ms vs. 649ms)
- *Our scheme improves request throughput & request latency*
Conclusions

- **Key observation**
  - Each write has different performance-criticality

- **Request-oriented caching policy**
  - Solely utilizes NVWC for application performance
  - **Improves** performance while **reducing** cached writes

- **Future work**
  - System-level critical process identification
  - Application to user-interactive environments
Thank You!

• Questions and comments

• Contact
  • sw.kim@skku.edu