RFLUSH: Rethink the Flush

USENIX FAST’18

Jeseong Yeon\textsuperscript{1}, Minseong Jeong\textsuperscript{1}, Sungjin Lee\textsuperscript{2}, Eunji Lee\textsuperscript{1,3}

\textsuperscript{1}Chungbuk National University, \textsuperscript{2}DGIST, \textsuperscript{3}University of Wisconsin–Madison
Storage Interfaces are Evolving!

- **Standard block interface**
  - Commands over logical block address space
  - READ, WRITE and FLUSH
  - Abstract of a storage device allows a host to use a device in an efficient manner
Storage Interfaces are Evolving!

- **Extend interfaces for new media**
  - **Command** – TRIM, Nameless Write, Remap, In-storage Copy, Atomic Update, Multi-stream SSD.
  - **Protocol** – NVMe / PCI-express
  - **Architecture** – Host-manageable SSD, User-programmable SSD
Rethink the Flush

- **Volatile buffer in storage**
  - Absorb write and minimize seeks in HDD
  - Improve random write performance and mask limited endurance in SSD
  - **Data loss** and **improper** ordering in power outage

- **FLUSH command**
  - Force all pending writes immediately
  - Ensure persistence and proper serialization
Rethink the Flush

- **Lump-sum approach**

- (-) **Slow down** non-volatile materialization of the writes that need to be made durable

- (-) **Deprive** non-urgent data of an opportunity to **absorb** future write and **coalesce in a buffer**
SSD Trends

- **FLUSH cost rises up** with next-gen SSD
- **Larger buffer** to compensate for steadily increasing latencies and endurance limits
  - 512MB to 2GB
  - Samsung 3D V-NAND 16 TB has 32GB DRAM

- **DRAM-less SSD**
  - Use host memory (Few GBs)
  - Limited form-factor
  - Competitive price
Alternatives

- **Supercapacitor**
  - Provide energy to force all dirty data in volatile buffer at power outage

- **Non-volatile memory**
  - Volatile read cache and non-volatile write buffer

- **Challenges**
  - Increase manufacturing cost
  - Write endurance
  - Limited form factor
Range Flush Primitive

- **RFLUSH**
  - **Fine-grained** control over forcing buffered data
  - Transfer a range of LBAs to persist with it

(+) Speed up non-volatile materialization of target data

(+) Enhance buffering and coalescing of other dirty data
Getting Real

- Realizing RFLUSH is not without **challenges**
  - How to **implement** RFLUSH?
  - Where to use RFLUSH?
  - How much performance gain can be achieved?

- **Prototype** Implementation
  - RFLUSH is implemented in **open-source flash** development platform (BlueDBM)
  - **F2FS** is modified to use RFLUSH during handling of fsync/fdatasync
Contents

- Motivation
- Problem and Solution
- Implementation
- Performance Evaluation
- Conclusion
Challenges

- Where to Use RFLUSH
- How to Address the Associated Data
- How to Handle Metadata
- How to Integrate into a Storage Protocol
Where to use RFLUSH

- More general than FLUSH
- Allow fine-grained control over what to flush
- FLUSH usage cases
  - E.g., Transaction commit in file system/ DB / KV-store

- Our Study – fsync / fdatasync
  - No application modifications – fsync semantics are preserved
  - Easy to determine associate data for RFLUSH
  - Noticeable performance gains when isolating regions to flush
How to Specify Associated Data

- **fsync**
  - Transfer LBAs of dirty pages of target file

![Diagram](image-url)
How to Specify Associated Data

- **fsync**
  - Transfer LBAs of dirty pages - *Incorrect*

Pages could have been sent to storage!

If the data are missing, semantics of fsync are violated
How to Specify Associated Data

- **fsync**
  - Transfer LBAs of dirty pages – *Incorrect*
  - Transfer **ALL** LBAs of the file

---

![Diagram showing how to specify associated data](image-url)

- **RFLUSH(LBA: 5-7)**
- **LBA of a file are likely to be adjacent**
How to Specify Associated Data

- **fsync**
  - Transfer LBAs of dirty pages – *Incorrect*
  - Transfer **ALL** LBAs of the file – **Correct**, but **costly**

![Diagram](image)
How to Specify Associated Data

- **fsync**
  - Transfer LBAs of dirty pages – *Incorrect*
  - Transfer **ALL** LBAs of the file – *Correct*, but **costly**
  - Transfer **Inode number** of the target file

![Diagram showing fsync operation]

Page Cache

![Page Cache Diagram]

Storage Media

![Storage Media Diagram]
How to Specify Associated Data

- Ino-based RFLUSH protocol

In the standard SSD:
- WRITE( ino : 40 )
- RFLUSH( ino : 40 )

In the open-channel SSD:
- RFLUSH( ino : 40 )

inode : 40
LBAs : 50-80, 100-200

Inode tag + Hash

HOST

Storage

Refer to host kernel

C` E`
Buffer

Standard SSD

C` E`
Buffer

Open-channel SSD
How to Handle Metadata

- fsync
  - Proper synchronization of dependent metadata
  - Small and random metadata updates
  - Encode a full range of metadata area

F2FS On-storage Layout (USENIX FAST ’15)
Integration into a Storage Protocol

- Open-channel SSD: BlueDBM
  - Developed by MIT CSAIL
  - SSD emulator with allocated memory

**Diagram:**
- **HOST**
  - File System
  - Block I/O Layer
    - Device Driver
    - FTL
  - NAND Flash
- **Storage**
  - Nand Flash
  - SRAM
  - Processor
  - Cache Buffer Controller
  - NAND Flash Controller
  - Host Interface
  - Cache Buffer
Contents

- Motivation
- Problem and Solution
- Implementation
- Performance Evaluation
- Conclusion
Performance Evaluation

- Linux 4.7.2
- Benchmark – FIO, Linkbench, TPC-C, Fileserver
- SSD Platform Setup

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page / Block size</td>
<td>4KB / 64 Pages</td>
</tr>
<tr>
<td>Read / Write Latency</td>
<td>100us / 1300us</td>
</tr>
<tr>
<td>Block Erase Latency</td>
<td>1.5ms</td>
</tr>
<tr>
<td>Data Transfer Latency</td>
<td>100us (for 4KB)</td>
</tr>
<tr>
<td>Overprovisioning Ratio</td>
<td>3%</td>
</tr>
<tr>
<td>SSD capacity</td>
<td>37 GB</td>
</tr>
<tr>
<td>In-storage Buffer</td>
<td>256 MB / 1 GB</td>
</tr>
</tbody>
</table>
FIO Benchmark - IOPS

- Single synching thread: write + periodic sync
- Multiple non-syncing threads: write
Tail Latency

- Response time (FIO Benchmark)

**Syncing Thread**

- FLUSH
- RFLUSH
- NOFLUSH

**Non-syncing Thread**

- FLUSH
- RFLUSH
- NOFLUSH

RFLUSH is critical to providing a **consistent performance** from a storage device.
Write Traffic

- Reduce write traffic by 24% to 43%
Real-world Benchmark

- Fileserver, Linkbench, and TPC-C
- Measure performance when a pair is run

<table>
<thead>
<tr>
<th>Component</th>
<th>IOPS</th>
<th>Req/s</th>
<th>Tpm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fileserver</td>
<td>1.2X</td>
<td>5.3X</td>
<td>4.2X</td>
</tr>
<tr>
<td>Linkbench</td>
<td>1.2X</td>
<td>1.2X</td>
<td>1.2X</td>
</tr>
<tr>
<td>TPC-C</td>
<td>1.2X</td>
<td>1.2X</td>
<td>1.2X</td>
</tr>
</tbody>
</table>
Conclusion

- Raise the issue of the **negative performance** impact of a **lump-sum** approach to persisting buffered data within storage
- Present **RFLUSH** that allows fine-grained persistence control
- RFLUSH prototype is implemented in F2FS and open-channel SSD
- Increase IOPS up to 5.3x, reduce write traffic by 43%
Thank you

Q&A