Adopting Barrier to Reduce Consistency Overheads in Linux I/O Stack

Jaemin Jung Hankeun Son Dam Quang Tuan Youjip Won
Hanyng University, Korea

Abstract

The synchronous write operation in modern filesystem is expensive. The typical example is fsync(). The dominant fraction of the overall latency is for flush-before-commit paradigm of modern storage device. We propose "barrier" in modern IO stack to mitigate the overhead of fsync() and to effectively exploit the potential performance of modern IO device. Modern storage IO interface standard, e.g. eMMC 5.1, supports cache enhancement barrier which ensures that the writes after the barrier is persisted only after the previous writes are persisted without immediate cache flushing. When a barrier is supported, a process can submit write requests after the barrier without cache flush.

Modern storage device supports command queuing. It allows the device to harbor a number of outstanding requests. This feature enables the storage device to better exploit the parallelism in the storage device and eventually to exhibit better throughput.

Flush-before-commit paradigm of modern IO stack and the command queueing feature of the storage device are not aligned well with each other. The fsync() call requires that the device writeback cache is flushed immediately and that the dirty page cache entries are persistently storage to storage surface before it returns. The flush-before-commit relieves the IO stack from the burden to understand the complicated ordering relationship among a number of IO requests. The flush-before-commit approach greatly simplfies the IO stack design. However, it deprives the IO device of the opportunity to exploiting the parallelism via maintaining a number of outstanding request and subsequently leaves the IO stack a significant room for performance improvement. Flush-before-commit paradigm is widely exploited in filesystem journaling, as well.

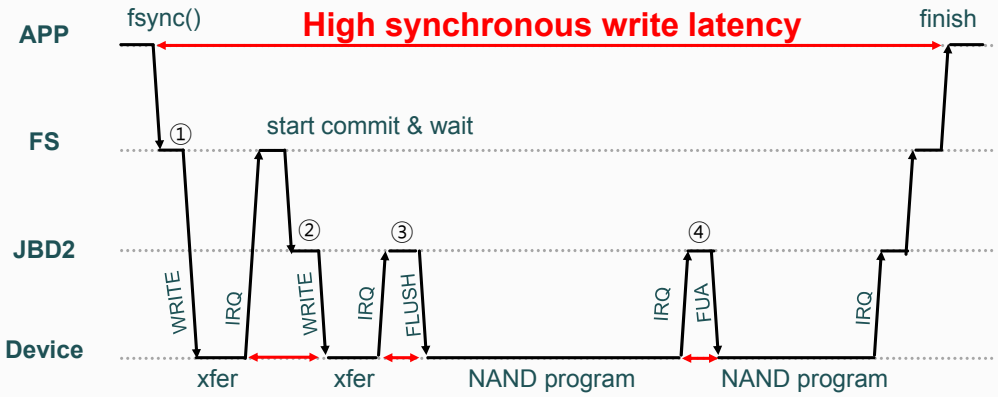
We are developing two barrier related system calls, fbarrier() and fdatabarrier() which correspond to fsync() and fdatasync(), respectively, in flush-before-commit based IO stack. They intend to fully exploits the command queuing. We are implementing the barrier feature on EXT4 including the journaling layer (JBD2). Imposing the ordering constraints among the IO requests may limit the parallelism, but it leaves greater degree of freedom for device IO scheduler than flush-before-commit paradigm does.

In this work, we like to develop barrier feature for current Linux I/O stack. Our work aims at providing two design principles. This work involves a barrier enhancement in all layers of the IO stack. First, we develop a barrier enabled storage firmware. Second, we develop the CFQ disk scheduler for barrier support. The ordering constraints imposed by barriers between write requests may restrict the behavior of I/O scheduling, request merging and reordering. Barrier feature should not interfere with the underlying IO scheduling policy. To preserve the existing I/O schedulingn policy, e.g, request merge, reordering and scheduling, we restrict a process to submit write requests until previous barrier operations is issued into the device. When a barrier is submitted, we block the process until the barrier goes out from the I/O subsystem. Note that we do not block the process until the barrier request is handled in the device. If there is no I/O contentions, the barrier will be issued as fast as possible. Therefore, the queue depth is not limited by this blocking. Our barrier handling mechanism allows I/O subsystems of existing kernel to safely perform request merge, reordering, and schedule without considering the ordering relations of requests. Although we only need to wait the barrier command to be issued to the device, this delay may have negative effect for the growth of the queue depth. To reduce the this delay and to reduce the number of command transfer, we propose to attach a barrier bit into the write command rather than issuing another barrier command.

Adopting Barrier to Reduce Consistency Overheads in Linux I/O Stack

Jaemin Jung Hankeun Son Dam Quang Tuan Youjip Won
Hanyang University, Korea

Problem: Synchronous Write in EXT4 with Ordered-mode Journaling



① Write data block
② Write journal
③ Issue FLUSH
④ Write commit block

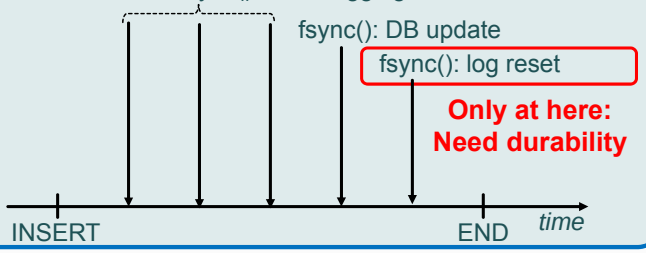
Significant Latency of Synchronous Write

- "flush-on-commit": process waits for data and journal are written into the NAND flash
- Serialized request processing: CPU cache flush → DMA setup → CMD issue → transfer → wait IRQ
  - Restrict parallelism of flash write
  - Idle device time with only a single I/O process

Barrier: Delayed Persistency

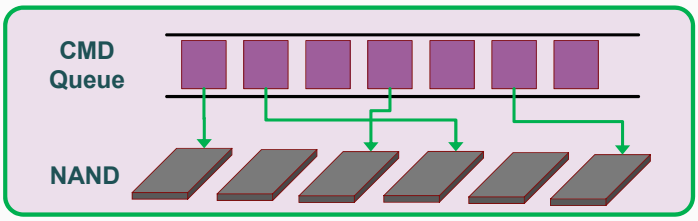
- eMMC 5.1 support *cache enhanced barrier* cmd
- Still **consistent**
- Durability for every fsync()? **No!**

SQLite Example (PERSIST Mode)



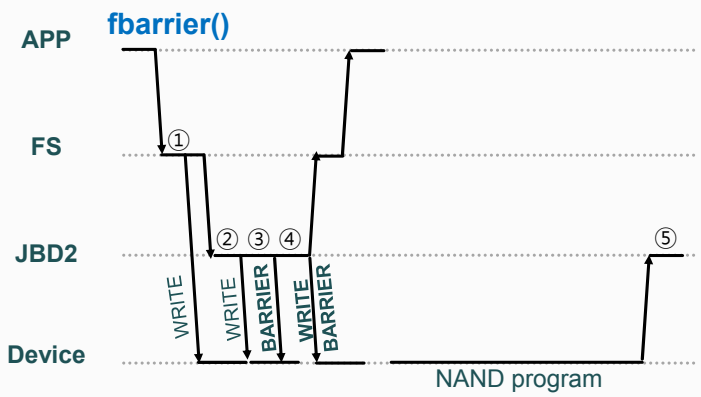
Command Queuing: Improve Performance

- Able to accept more requests during I/O
- Can improve parallelism in flash-based device



- Queue depth (QD) **cannot grow:**
  - Waitings for the completion of command
  - On 4KB write() + fsync(), QD is **mostly "1"**

Our Approach: barrier-on-commit Interface



- Significantly **low latency**
- Decrease the device idle time
- Exploits parallelism of request processing

① Write data block **without waiting for completion**
② Write journal **without waiting for completion**
③ Issue **BARRIER**
④ Write commit block with **BARRIER flag**
⑤ Make dirty journaled metadata to be checkpointed

14th USENIX Conference on File and Storage Technologies (FAST'16)
February 22-25, 2016, Santa Clara, CA, USA
Poster