An Investigation of Performance Problems with `msync()` System Calls on Filesystem DAX

Satoshi Iwata

Persistent Memory (PMem)

- Features

1. Fast compared to traditional storage devices like SSDs
2. Byte addressable (Load/Store instead of Read/Write)

A combination of filesystem direct access (DAX) and `mmap()` enables file access on PMem with its native ability.

- Avoid buffer cache
- Load/Store access

`msync()` vs. `pmem_persist()`

We need to call a CPU instruction such as `CLWB` or `CLFLUSHOPT` to synchronize CPU cache to persist stored data to a PMem.

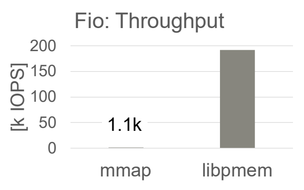
- `msync()`: A system call
  👍 Legacy applications which use `mmap()` and `msync()` for speeding up access to conventional storage devices are easy to port to PMem without any code modifications
- `pmem_persist()`: A function included in PMDK
  👍 Fast since it directly calls CPU instructions from user space

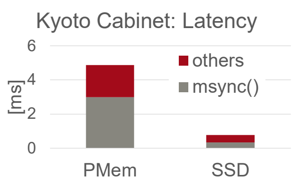
A Performance Problem

We faced a problem where `msync()` was tremendously slow compared to `pmem_persist()`

- 4KiB random store for a 100MiB file with `fio`

- Kyoto Cabinet's SET operation
  Kyoto Cabinet is a library of routines for managing a database. Though its original target is not a PMem, it uses `mmap()` and `msync()` for speeding up.





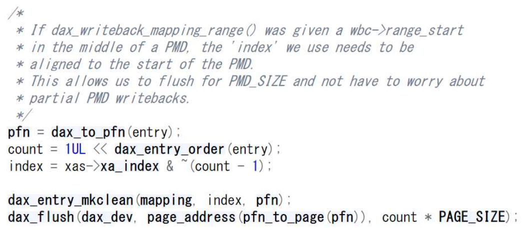
This measurement is done with Intel Optane DC Persistent Memory, ext4 filesystem on Linux Kernel 5.4.0, `fio` 3.23, and Kyoto Cabinet 1.2.79.

Root Cause of The Problem

It is a side effect of DAX hugepage.

When we try to map a region greater than 2MiB, the ext4 filesystem tries to map 2MiB pages instead of 4KiB pages. Hugepage support has several advantages, such as fewer page faults, smaller page tables, and less TLB contention.

Since one dirty flag is assigned to a whole 2MiB page, a 4KiB `msync()` incurs a 2MiB flush, which is 512 times larger.

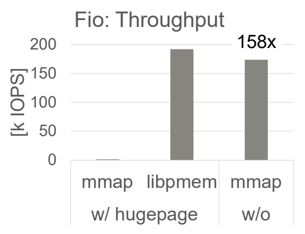


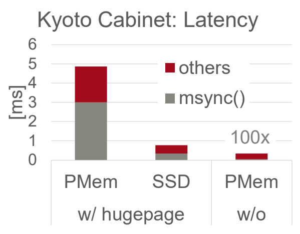
```
/*
 * If dax_writeback_mapping_range() was given a wbc->range_start
 * in the middle of a PMD, the 'index' we use needs to be
 * aligned to the start of the PMD.
 * This allows us to flush for PMD_SIZE and not have to worry about
 * partial PMD writebacks.
 */
pfn = dax_to_pfn(entry);
count = 1UL << dax_entry_order(entry);
index = xas->xa_index & ~(count - 1);

dax_entry_mkclean(mapping, index, pfn);
dax_flush(dax_dev, page_address(pfn_to_page(pfn)), count * PAGE_SIZE);
```

Measurement without Hugepage

These graphs show how hugepage's side effect is large. They are gained with a kernel in which `CONFIG_FS_DAX_PMD` is disabled.





Future Direction

Merely disabling hugepage cannot be the best solution.

Preparing 512 dirty flags for a hugepage every 4KiB would work, since the root cause is that a dirty flag manages a 2MiB page as a whole. This solution reduces the size of wasted cache line flush to nearly zero when `msync()` requests the flushing of a small region.

In addition, some optimizations may be required if a large flush is asked, like one near 2MiB, since flushing 512 4KiB pages would take longer compared to a 2MiB page flush.

FUJITSU LABORATORIES LTD.

shaping tomorrow with you