Optimising Major Page Faults

Barry Song

mmap_lock contention after introducing per-VMA locks

- Even after introducing per-VMA locks (and removing mmap_lock from madvise last year), mmap_lock contention remains significant, with page faults as the primary source.

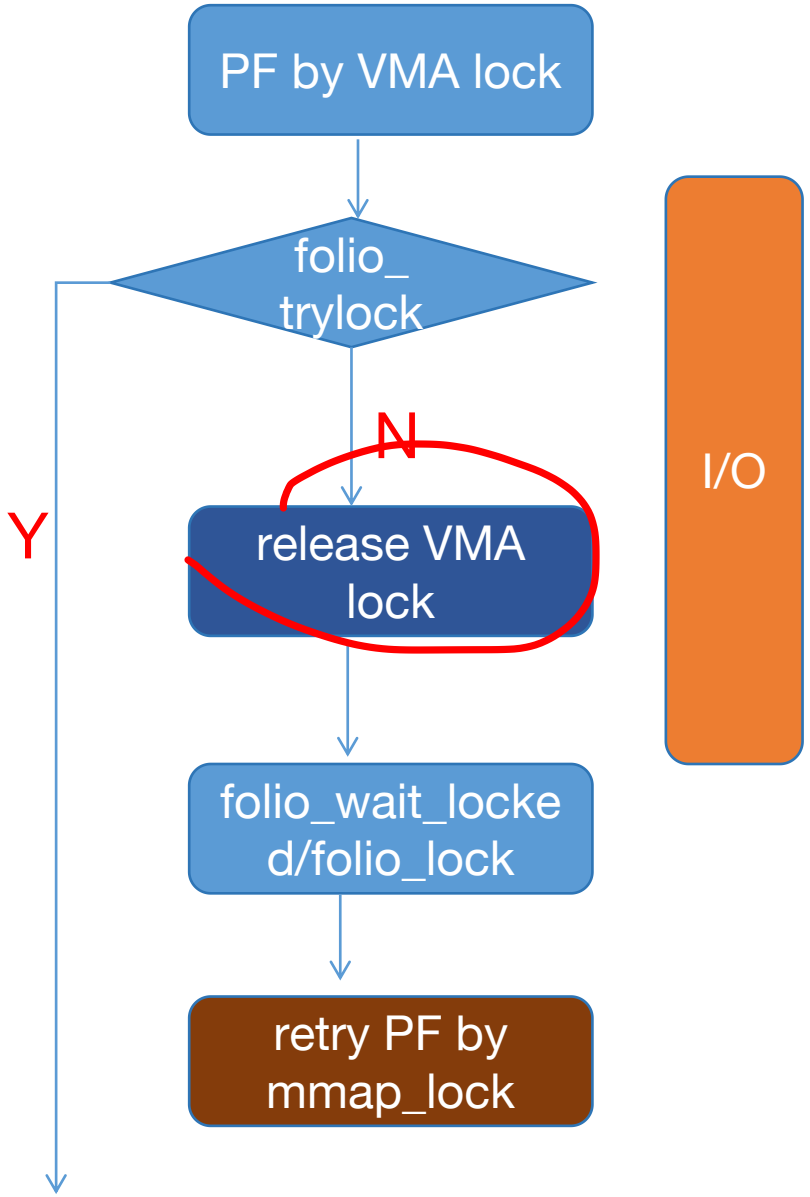
- **In filemap_fault():**

We use lock_folio_maybe_drop_mmap(), which may release the per-VMA lock via release_fault_lock(vmf), then acquire folio_lock()—mainly to wait for I/O completion, but also due to concurrent PTE installation—and then retry the page fault.

- **In do_swap_page():**

We use folio_lock_or_retry(), which may also release the per-VMA lock via release_fault_lock(vmf).

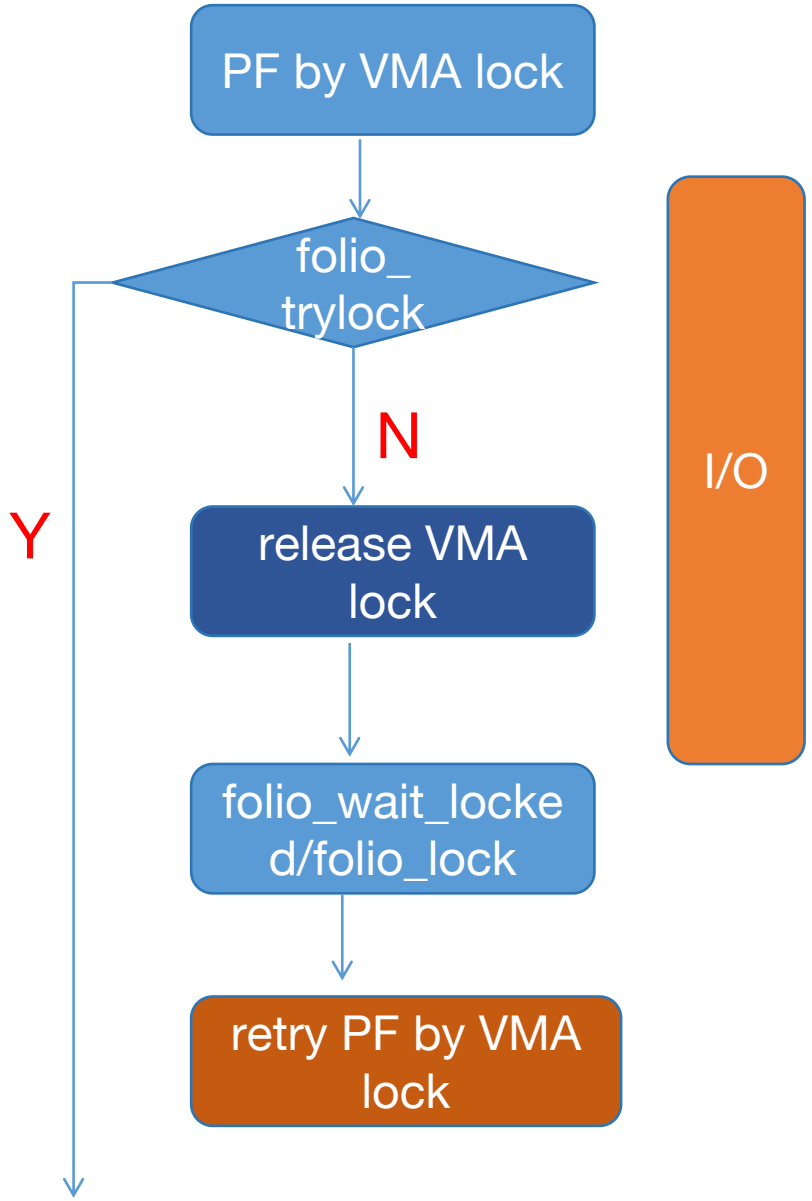
All retry paths acquire mmap_lock rather than the per-VMA lock.



Option 1:

- We retry PF by per-vma lock instead of mmap_lock

-Introduces additional complexity

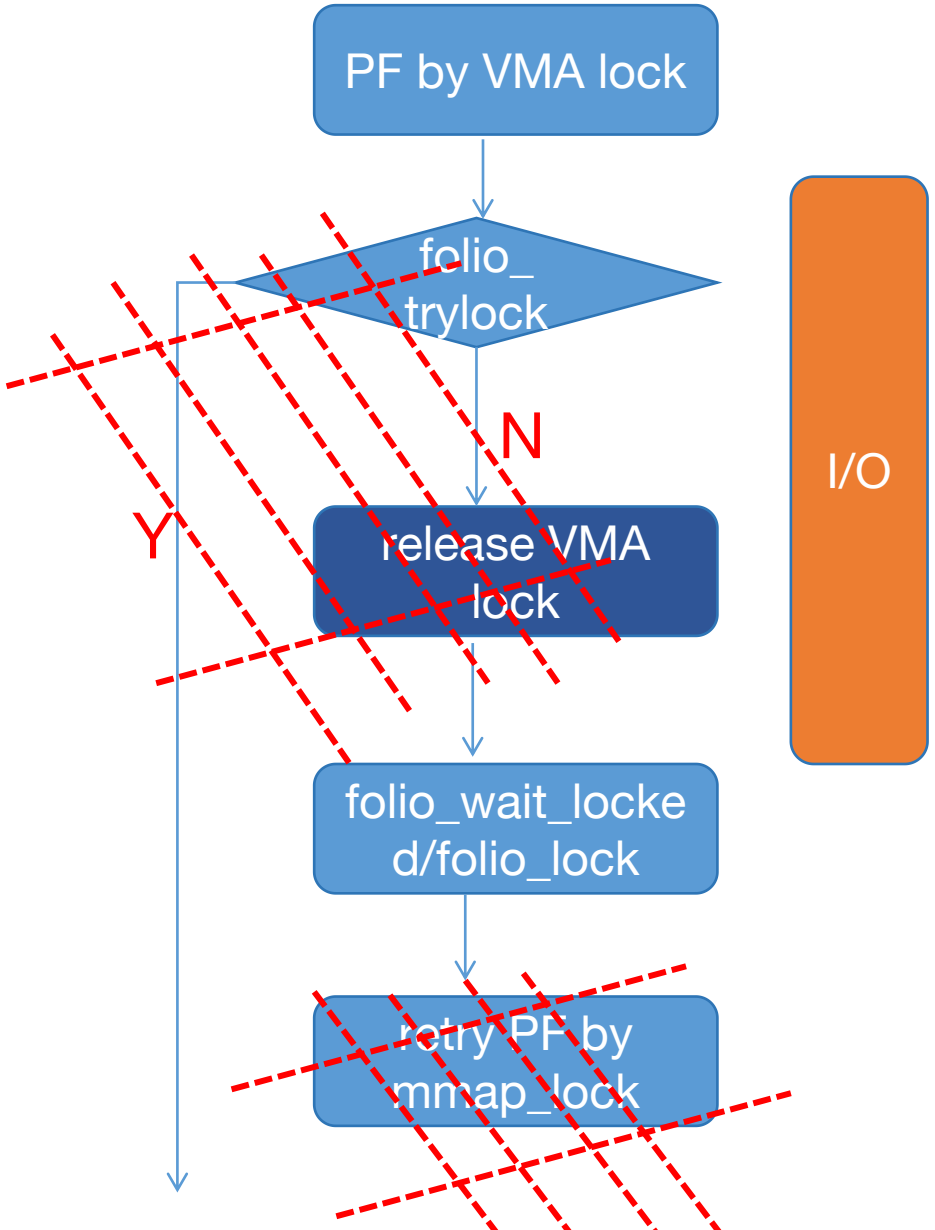


Option 2:

- We completely remove the page-fault retry code—

To quote Matthew, “rip it out.”

- Does this lead to priority inversion?



Threads in an Android app can have different priorities

- Better performance through threading

https://developer.android.com/topic/performance/threads

The main (UI) thread handles all on-screen work. If it is unable to complete its work within a single frame interval (e.g., 144 or 165 FPS on high-end devices), users may experience stutters, lag, or an unresponsive UI.

Holding the VMA lock during I/O may lead to potential priority inversion

thread1

- PF

PF by VMA lock

unpredictable I/O latency

thread2

- write VMA PF occurs

mmap_lock write

vma lock write

mmap_lock write unlock

thread3(high priority)

- write any VMA(mmap_lock write) or walk VMAs(mmap_lock read)

Wait for low priority I/O

mmap_lock write

Discussion

- Totally remove all locks and retries for both file-backed and anonymous memory?

- Separate approaches for file and anon?
  - Anon: VMA writes and page faults can occur simultaneously—for example, a high-level language GC may use mprotect or UFFD_REGISTER/UNREGISTER on a region.
  - File: A thorough test to determine whether priority inversion is a problem after “ripping it out” ?

- Checking folio_is_uptodate() and avoiding page-fault retries during concurrent PTE installation can improve performance - when folio_lock fails because another thread has already taken the folio lock to install PTEs in the same range.

- For simplicity, rather than using a whitelist mechanism for per-VMA retry, we could use a blacklist instead: default to always retry via VMA lock, and only for specific cases such as __vmf_anon_prepare() allow mmap_lock-based page-fault retry ?