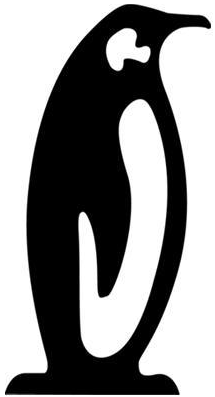


東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

Reducing Android Boot Times: Evaluating Proactive File Prefetching

Akilesh Kailash <akailash@google.com>
Takaya Saeki <takayas@google.com>

東京 2025 LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

History: Ureadahead on ChromeOS

- ureadahead is a disk prefetch daemon that makes boot faster
- Adopted in ChromeOS and showed improvement in boot time ~20%
- Ureadahead has two stages
    - Record
        - Records what file paths and their ranges are in the page caches after a boot
        - Track the I/O pattern by static tracepoint which tracks what pages are added to the page-cache
        - Results stored in a “pack file”
        - Runs when user boots first time
    - Replay
        - Pre-loads the pack file at any early-stage of the boot
        - Prefetch the I/O by populating file backed pages to page-cache
        - Runs on subsequent boots when pack file is present

Android Boot

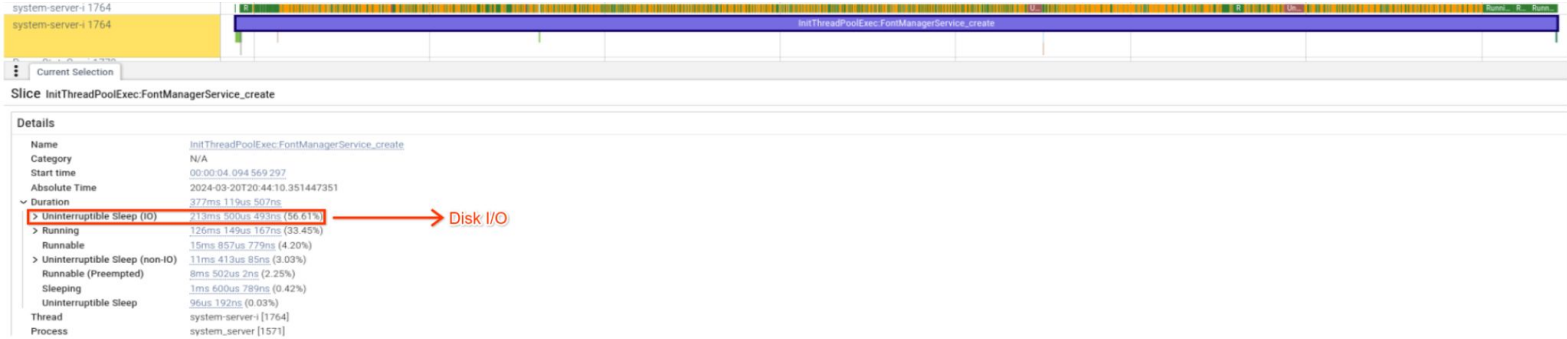
- Different stages of boot
    - Early-init, init, late-init, fs, post-fs, zygote-start, early-boot, boot
    - Deterministic I/O pattern but each stage I/O pattern varies
- ~1.2GB of data read during boot on Pixel (from read-only filesystems)
- Boot time varies across android verticals with varying I/O pattern (Phones, TV, Watch)

Prefetch - in Android

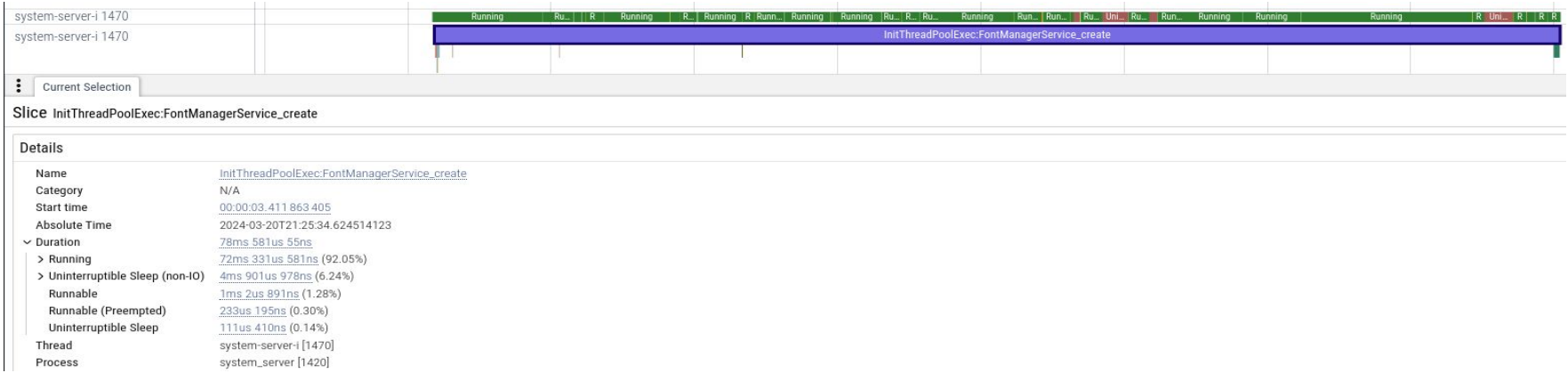
- Adoption of ureadahead in Android
- Prefetch is a daemon written in rust
- Similar concept as ChromeOS but quite a few tunables added based on experimentation
- ~6-8% improvement in boot time on Watch, TVs
- Evaluating on other verticals

Performance evaluation using Perfetto

Without Prefetch



With Prefetch



Challenges

- Inaccurate Page Cache Tracking: The current use of the static tracepoint *mm_filemap_add_to_page_cache* only tracks pages inserted into the page cache after disk I/O. It misses I/O activities for pages already present in the page cache during the "record" phase.

- Post-OTA Boot I/O Distortion:
  - Boot time is slower following an Over-The-Air (OTA) update due to updates to the read-only partition.
  - This post-OTA boot exhibits I/O activity that is not representative of a regular boot, making the "record" phase run during this period unreliable for a true boot I/O pattern.

- Memory pressure:
  - If there is memory pressure, prefetch is wasteful.
  - If the I/O is delayed for any reason, correct pages may not be available which then competes with the prefetch I/O

Challenges - Continued

- Risk of Memory Thrashing during early prefetch (Replay):
  - Triggering prefetch early in the boot process can lead to memory thrashing, especially on low-memory devices (~2GB).
  - A burst of I/O activity early on may cause the file-backed pages brought into the page cache to be immediately evicted by kswapd.
  - Careful tuning is required to determine the right interval and specific pages to bring into the page cache.

- Redundant “Record” phase execution
  - The “record” phase is unnecessarily re-run even when there are no significant changes in file updates or prefetch ranges
  - If a file hasn’t been updated during an OTA, re-running the record phase to capture the same I/O pattern between two versions is not necessary.

Potential Solutions - 1

- Record page-cache “access” not just the insertion
  - New static tracepoints in the kernel since 6.12
- Pages accessed by applications
  - Consists of get_pags, map_pages and page-fault activities
- Allows to do continuous record-replay instead of one off recording.
- Implemented in ChromeOS - ~3% improvement in boot time, ~12% improvement in OTA boot time
- Work in progress to adopt it in Android

Events

mm_filemap_get_pages

Process X read pages, inode X offset Y-Z

mm_filemap_map_pages

Process X mapped pages, inode X offset Y-Z

mm_filemap_fault

Process X page faulted. Inode X offset Y

Potential Solutions - 2

- Refining the "Record" Step:
  - Divide the "Record" process into multiple distinct phases.

- Android Boot Events for Prefetching:
  - Utilize specific Android boot events: Early-init, init, late-init, fs, post-fs, zygote-start, early-boot, and boot.
  - Create a dedicated "pack" file corresponding to each boot stage.
  - Execute the replay at the "n-1" or "n-2" stage to prefetch necessary pages just in time.

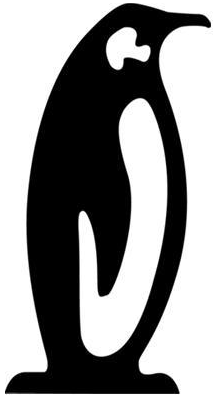
- Benefit:
  - This strategy is crucial for mitigating thrashing, particularly on devices with extended boot times (e.g., approximately 50+ seconds).

Next steps

- Integrate it to AVF (Android Virtualization Framework) - which runs debian OS inside a VM (Linux terminal)
  - Unlike Android, debian has read-write rootfs partition. Thus, I/O pattern may vary
  - Continuous record-replay may help

Thank you!

Akilesh Kailash <akailash@google.com>
Takaya Saeki <takayas@google.com>



東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025