Mapping Text with Large Folios

LSF/MM/BPF 2025

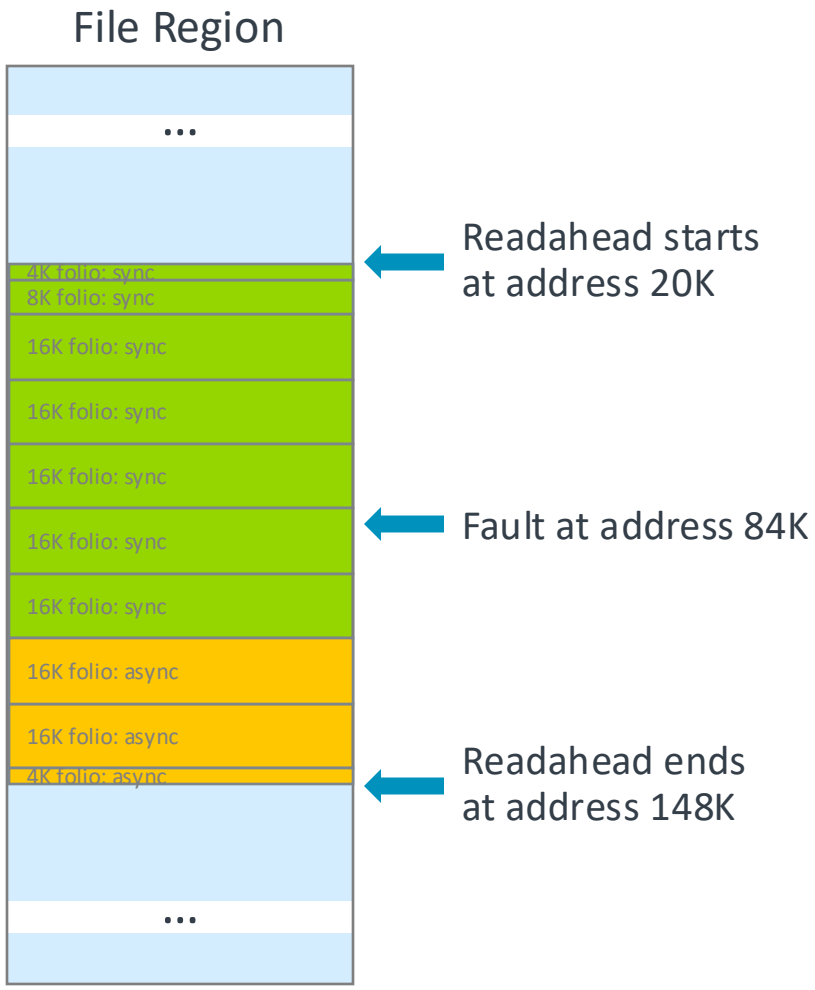
Ryan Roberts

March 2025

Goal: Improve performance by mapping text with larger folios by default

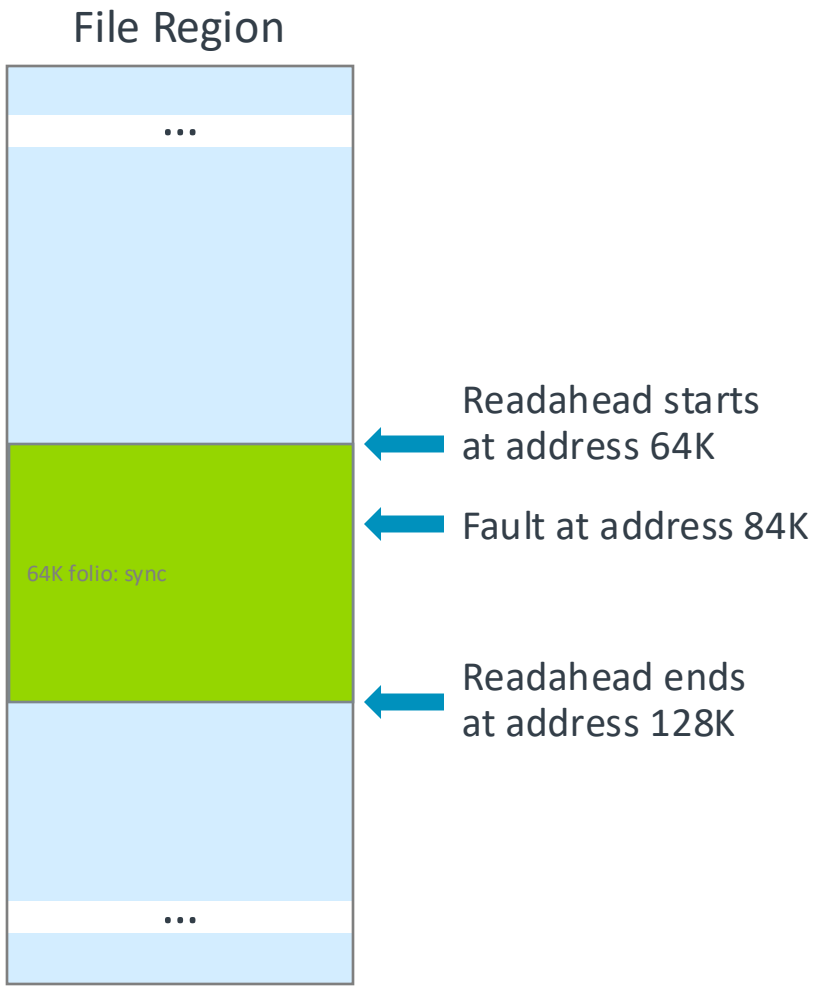
- Modern CPUs support multiple TLB entry sizes:
  - Arm64: 64K (order 4); with a “contiguous” PTE hint
  - AMD/x86: 32K (order 3); automatically coalesced by HW
- Larger mappings reduce TLB pressure and improve performance
- Good progress in supporting large folios:
  - Anon (private & shared): specify sizes to enable via sysfs
  - Page cache: readahead rapidly grows folio size as markers are hit
- But: readahead deliberately keeps folios small-ish (~16K) when file access is random
  - Text sections from ELF files rarely benefit from contpte mappings
  - Maintaining text in 64K folios will reduce iTLB pressure without increasing memory consumption

The Problem with Readahead



- Upon faulting on a page not present in page cache readahead is configured to read initial 128K (usually)
  - Centered around faulting address (“read-around”)
  - First 75% is synchronous, last 25% is asynchronous
  - Preferred folio size is 16K (order 2)
  - Lower orders used to align to natural boundaries
- If async folio is later accessed from page cache more readahead is performed
  - All async and size is increased
  - Preferred folio size is increased by 2 orders
- But text is accessed randomly so async folio is rarely accessed and readahead rarely moves beyond 16K folios

Proposed Behavior



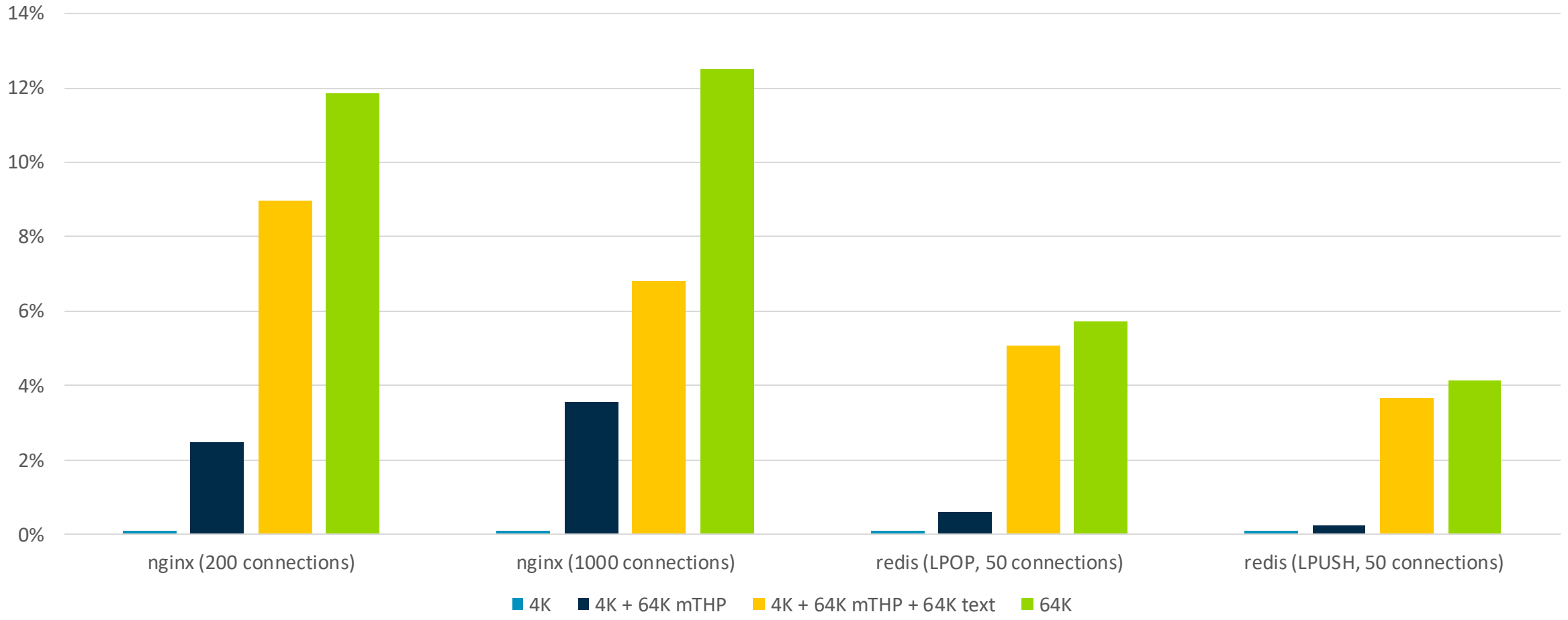
- Initiate readahead using the preferred folio size (64K on arm64) and ensure it is always naturally aligned
- Assuming “random” fault pattern for text, there is no value in having async portion
- Reduce overall read size from 128K to 64K (or could read 2x64K)
- No risk of write-back amplification since text is RO

- Result:
  - All (most) text in 64K folios
  - Fault-around causes whole folio to be mapped in 1 fault via set_ptes()
  - arm64 can efficiently set PTE contiguous bit

arm

Results

Relative performance improvement for different page size configs vs 4K



1: Arch-provided preferred folio size for PROT_EXEC mappings

- Architecture knows what folio size gives most benefit so can provide via opt-in hook
- Special-case readahead strategy for faults in VMAs with PROT_EXEC
  - Does not work when page cache already populated due to e.g. explicit readahead(), virus scanner, etc
  - Allows special casing for sub-region of file
- Contained in kernel and transparent to user space
- Exact pattern already used for special-casing MADV_HUGEPAGE

- First attempt implemented this
  - https://lore.kernel.org/all/20240215154059.2863126-1-ryan.roberts@arm.com/
  - Objection raised that arch should not dictate folio size

2: Sysfs-provided preferred folio size for PROT_EXEC mappings

- The same, except user space configures the preferred folio size globally via sysfs

- Second attempt implemented this
  - https://lore.kernel.org/all/20240717071257.4141363-1-ryan.roberts@arm.com/
  - Community not keen to add more sysfs knobs

3: Loader hints hugepage size

- Allow MADV_HUGEPAGE with process_madvise()
  - Extend to use flags to hint preferred size(s)
- Loader calls process_madvise(MADV_HUGEPAGE, 64K) for executable mappings directly after mmap()
- Moves policy to user space (yay) but creates ABI (boo)

4: Filesystem sets min folio size for ELF files

- Raised on list; including here for completeness
- Reuses same infrastructure as LBS
- Solves the readahead()/virus scanner problem
- Some issues to solve:
  - How does the filesystem know which files to raise the minimum folio size for?
  - How does it decide what to raise the minimum size to?
  - Applies to whole file; not restricted to text section
  - Would need to be implemented in every filesystem

arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు