CHINA LINUX KERNEL
中国Linux内核开发者大会

華中科技大學
网络空间安全学院
School of Cyber Science and Engineering, HUST

第19届中国Linux内核开发者大会

赞助单位

华中科技大学 | HUAWEI | 阿里云 | FUJITSU | 腾讯云
oppo | 火山引擎 | vivo | 龙芯中科 LOONGSON TECHNOLOGY | 蚂蚁集团 ANT GROUP

支持单位

清华大学 Tsinghua University | 迪捷软件 DIGIPROTO | OpenEuler | OpenAnolis 龙蜥社区 | OpenCloudOS

支持社区&媒体

CSDN | 云巅论剑 | 思否 | 阅码场 yomocode | InfoQ 极客邦旗下
51CTO | 开源江湖

2024年10月　湖北 • 武汉

華中科技大學

围绕 mTHP 展开的 SWAP 分配器优化

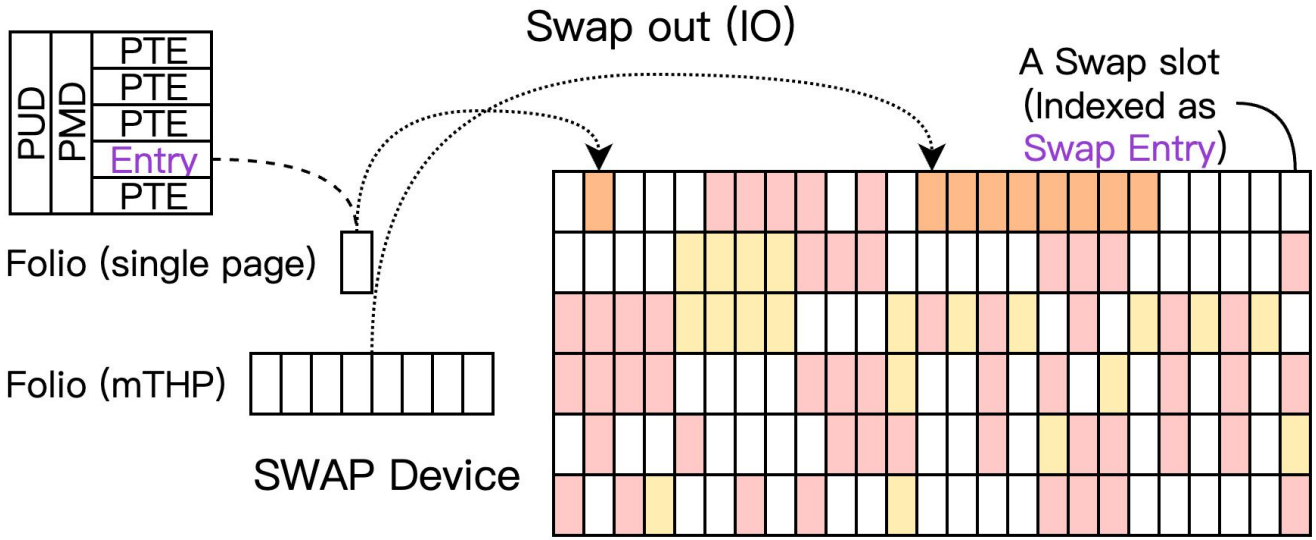
宋恺睿（腾讯），Chris Li（Google），刘海龙（OPPO）

围绕 mTHP 展开的 SWAP 分配器优化

宋恺睿（腾讯），Chris Li（Google），刘海龙（OPPO）

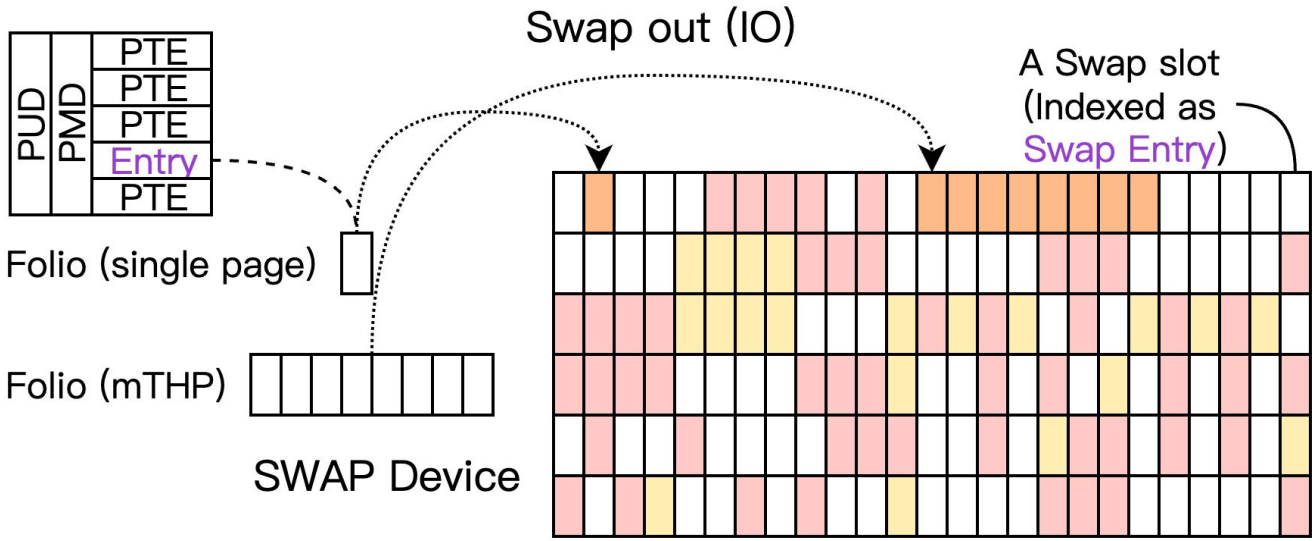
快速概括：SWAP 与 SWAP 分配器

- 匿名内存页从内存中换出到块设备。**Swap 设备**上存储单一页面（4K）的单位为 **Swap Slot**，内存中管理 Swap Slot 的基础数据为 **Swap Entry**。（Swap Entry / Slot 一一对应，相互指代）。



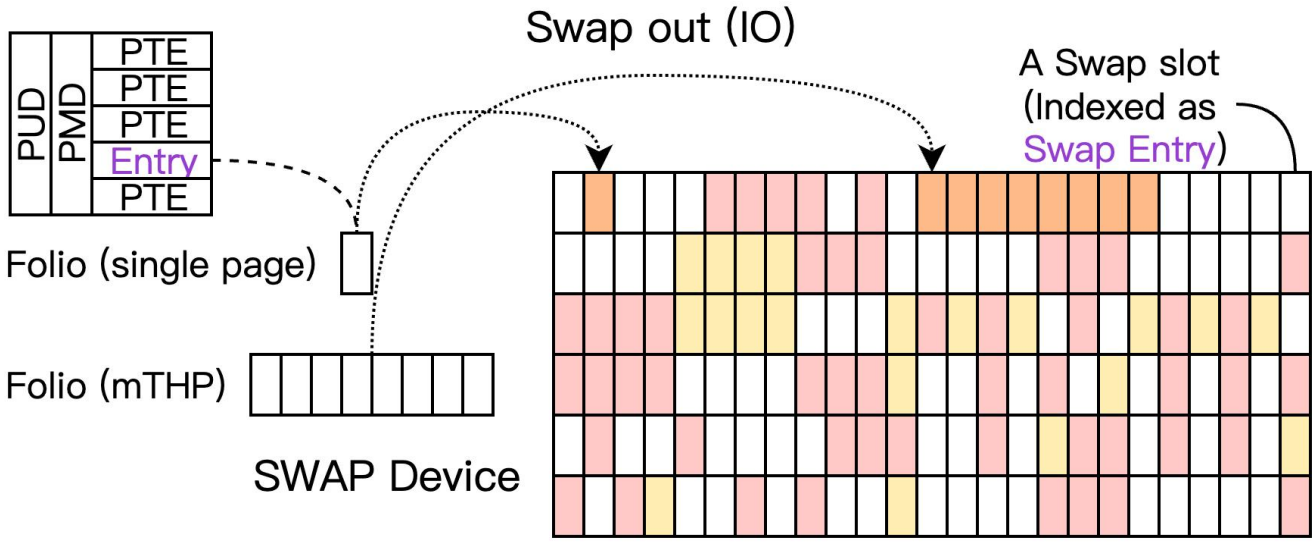
快速概括：SWAP 与 SWAP 分配器

- SWAP 分配器负责管理 SWAP Device 上的有限空间（Swap Slot）。
- 在 Folio （单页面 / mTHP）需要换出时，分配对应的可用的空间并返回 SWAP Entry。换入时则根据 Swap Entry 释放对应的空间。



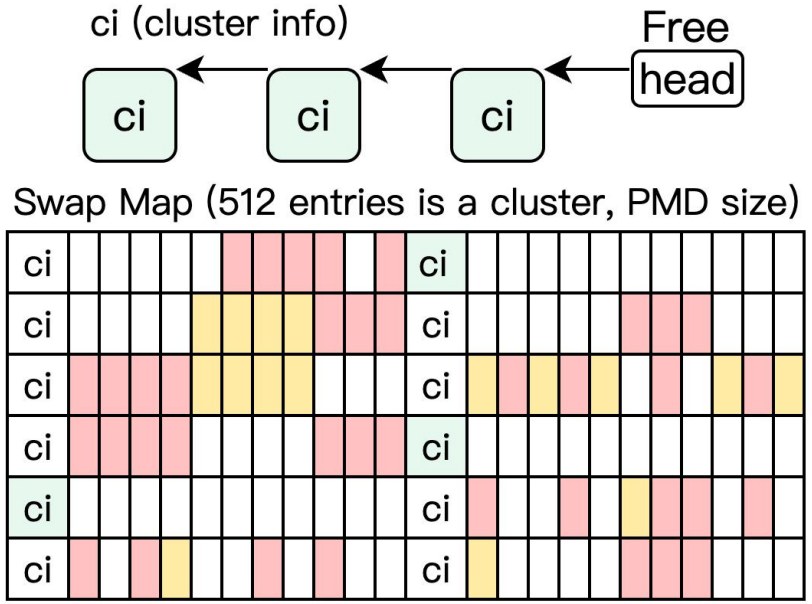
快速概括：SWAP 与 SWAP 分配器

- 挑战在于内存随机性极强，设备会快速碎片化，找到空闲空间会逐渐困难。
- 对性能和内存占用非常敏感，ZRAM 等基于内存的块设备更加剧这一问题。
- mTHP （Order > 0） 对连续性的需求造成了更大的挑战。



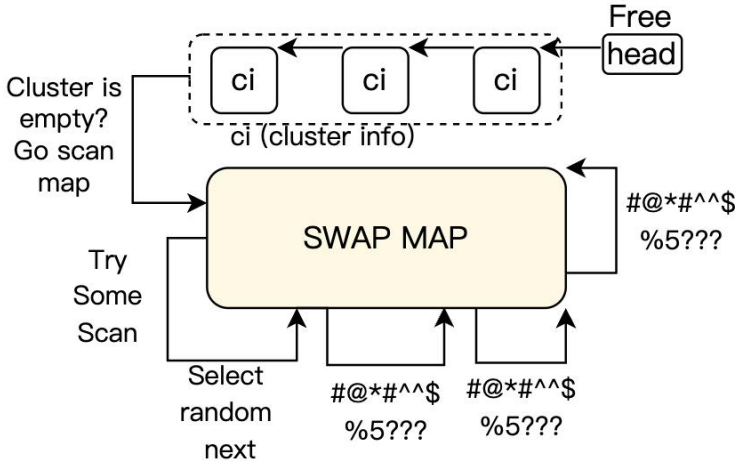
问题分析：SWAP 分配器之前的状态

- 预先分配 Swap Map 数组管理 Swap Slots / Entries（Array of bytes）
- 为了实现 THP 分配与提高 SSD 性能，引入了 Cluster 概念（PMD 大小）。
- Cluster 由单项链表链接组织，只使用头部的 Cluster。
- Cluster List 只包含完全空闲的 Cluster。



问题分析：SWAP 分配器之前的状态

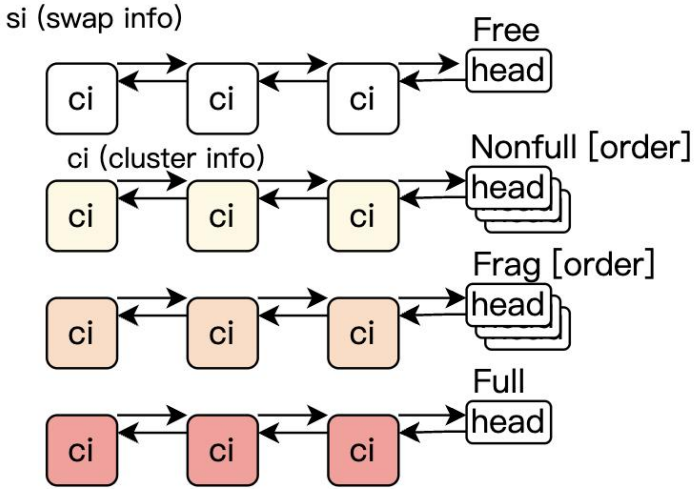
- scan_swap_map_try_ssd_cluster()
  - 尝试找到可用的 Free Cluster，不负责分配。
- scan_swap_map_slots()
  - 负责扫描 Swap Map，实际分配的发生处。
- Cluster 查询与实际分配关系错综复杂。
  - 常会发多次回落与多次重试。
  - 单链表导致的 Head 冲突。
  - 全空 Cluster 很容易耗尽。
  - 分配失败时会进行随机扫描。
  - Cluster 或被其他的 Swap Map Scan 抢用。



- Order > 0 分配失败率在 Free Cluster 耗尽后极高。Cluster 在设计中只是作为 Best Effort （尽力而为）的首选，Swap Map Scan 回落与重试率颇高。

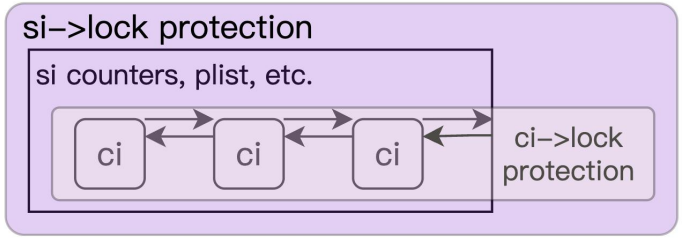
新的 Swap Allocator 设计

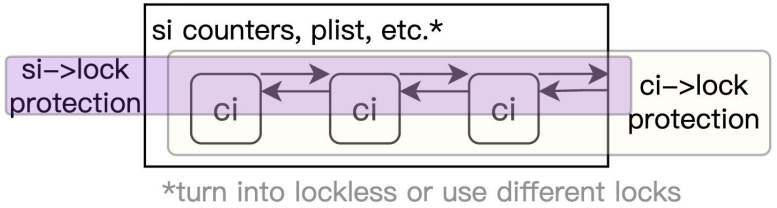
- Cluster 作为基础管理单位而非 Best effort
  - 完整的基于 Cluster 的管理过程。
  - 使用双链表。
  - 全空，非空，全满 Cluster 均被管理。
  - 完全基于 Cluster 的分配流程，不再有不同流程导致的大量分配冲突。
- 每个 Cluster 处于对应的链表中。
  - 每个 Order 所使用的 Cluster 均有独立链表，对 mTHP 友好。
  - 按照 Cluster 使用情况有多个分级链表：
    - Free cluster list
    - Nonfull/partial allocated list (per order).
    - fragmented list per order (per order).
    - Full list.
- 去除了扫描 Swap Map 的过程，永远使用 Cluster List 进行分配，提高了找到空闲空间的速度。
- Swap Cluster Allocator （6.12 合入）：https://lore.kernel.org/linux-mm/20240730-swap-allocator-v5-0-cb9c148b9297@kernel.org/



新 Swap Allocator 性能提升巨大

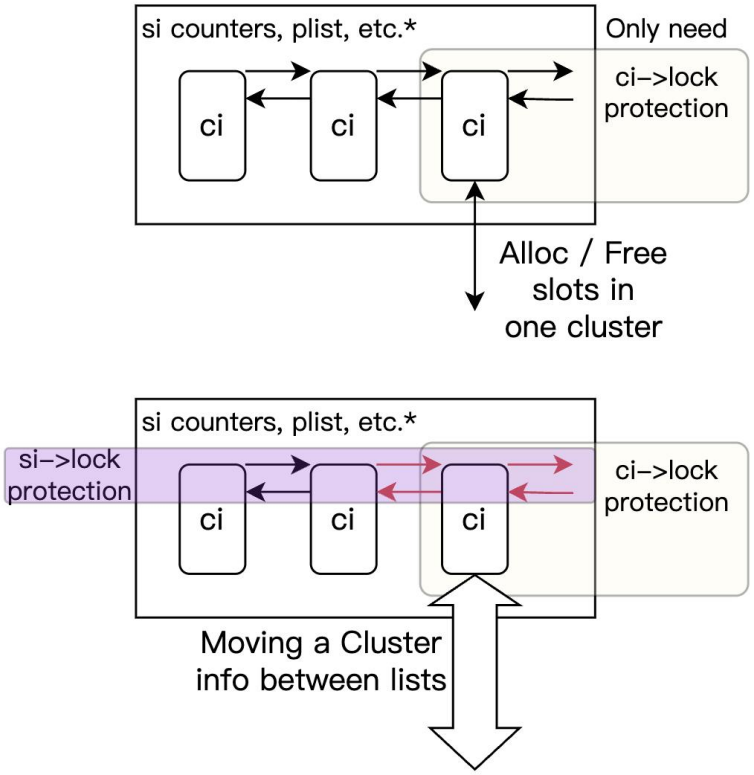
- 完全基于 Cluster 的分配避免了无效 Swap Map 扫描的同时，使得我们有机会重新思考与设计 Swap 锁。
- 当前 Swap 主要有两个锁：
  - si->lock (swap info - device lock, big lock)，**实际负载中争抢严重**。
  - ci->lock (cluster info - cluster lock, per cluster)，争抢相对轻微。
- ci -> lock 的引入之初让大量操作可以只持小锁，但大量常见操作仍旧需要同时持有两把锁：
  - 当前 si->lock 保护范围大于 ci->lock。
  - si->lock 临界区内包含 ci->lock。
  - 只能先获取 si->lock 后再获取 ci->lock，无法调转。
- 新的 Swap Allocator 设计中所有操作以 Cluster 为单位，故而有机会尽量只使用 ci->lock。
  - 降低 si->lock 的保护范围。
  - 与 Cluster 链表无关的数据全部移出 si->lock 范畴。
  - 调转 si->lock 与 ci->lock 的依赖关系。
  - 只在获取 ci->lock 后需要改动 list 时再获取 si->lock。





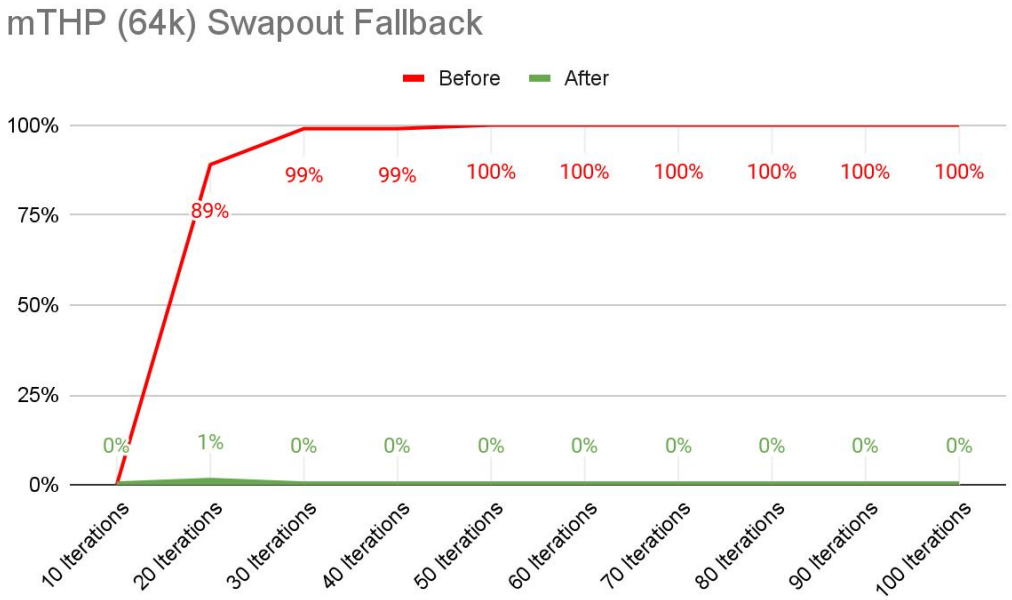
新 Swap Allocator 性能提升巨大

- 新分配器中大部分操作都在一个 Cluster 内部完成，不需要触碰 si->lock。
- 在释放路径（**Free**）：
  - full list -> nonfull list -> free list
  - 绝大多数情况 Cluster 停留在 nonfull list 上（直到 512 个 entry 全被释放）。
  - 不改变 list，不需要碰 si->lock。
  - Free 路径移除了 Slot Cache（swap_slot.c）。
- On **allocating** swap entry:
  - 分配器会尽量重复使用同一个 Cluster （已存在的机制）。
  - 在一个 Cluster 中 512 个 Entry 耗尽前不需要触碰 list，不需要碰 si->lock。
  - 可以进一步移除 Slot Cache？（swap_slot.c）
- Swap Allocator Lock Rework：https://lore.kernel.org/linux-mm/20241022192451.38138-1-ryncsn@gmail.com/



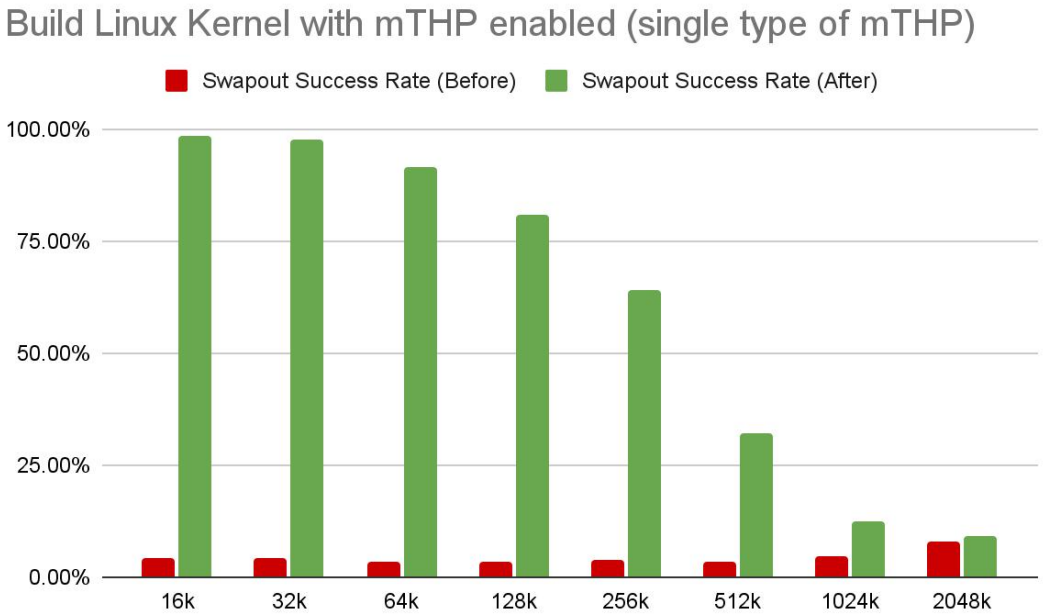
测试数据

- tools/mm/thp_swap_allocator_test.c 测试：
  - 模拟负载。
  - **碎片率降低 99%。**



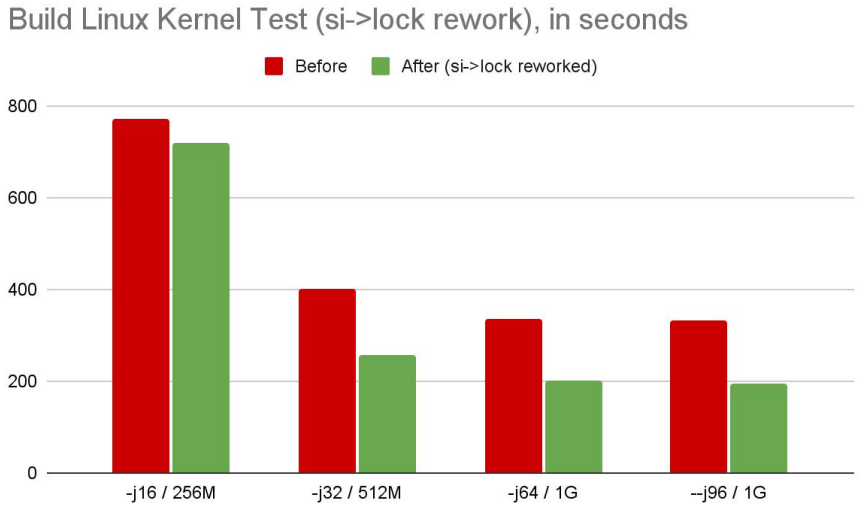
测试数据

- 6.12 中，使用 Linux Kernel Build 作为测试。
- 碎片化导致的换出失败率从 **~99%** 降低到 **~1% - ~90%** (<10% for 64K mTHP).
- SWAP 设备碎片化随时间会变得愈发严重。
- 在没有真实连续空间的情况下，分配器的工作更加困难。
- 当前的解决思路：
  - Reserving part of SWAP as mTHP only, avoid 0 order from polluting these clusters.
  - Non-continuous swapout.

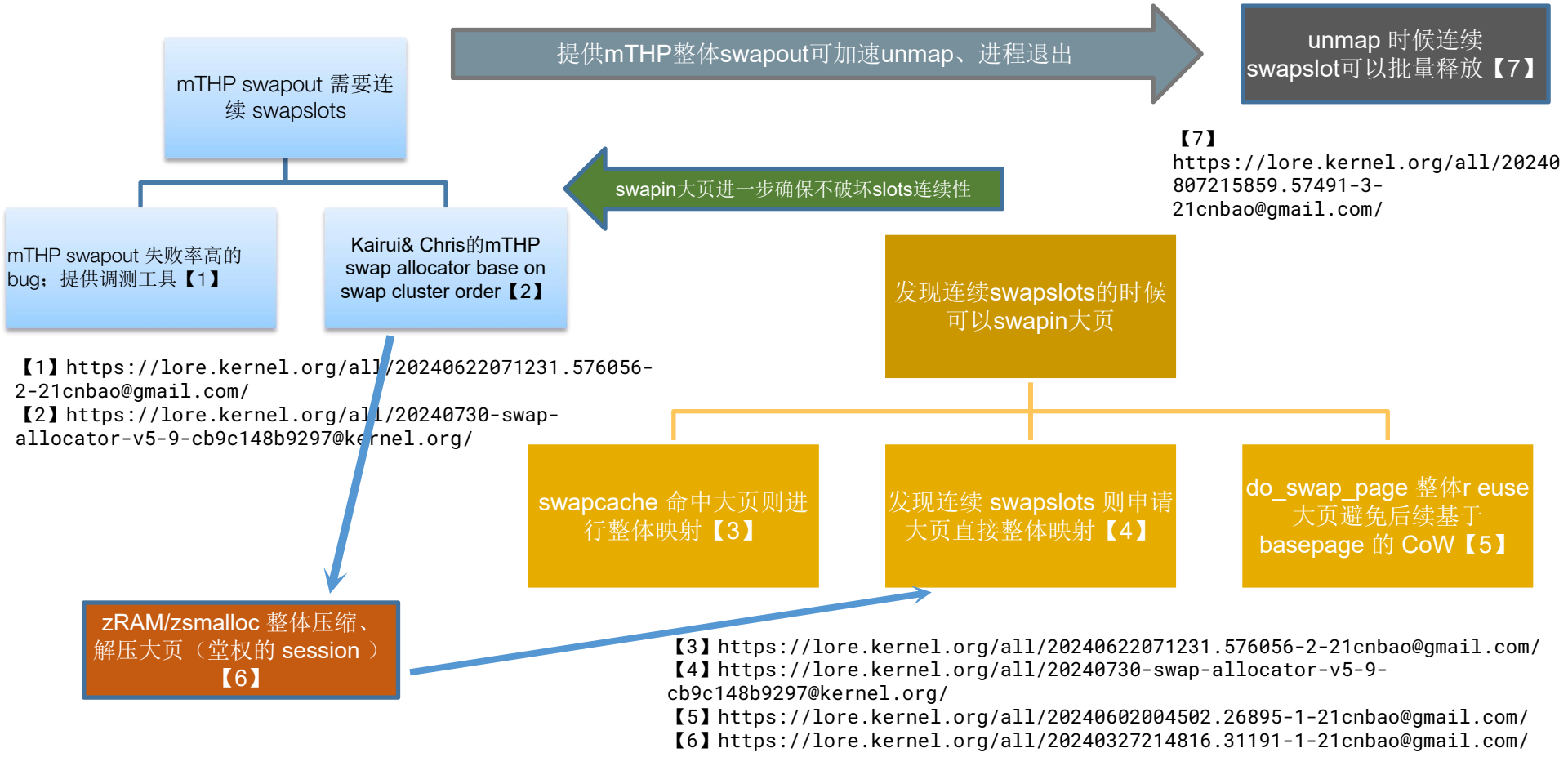


测试数据

- 编译内核测试：
- 6.12 中已合入的部分提升在 1 - 5% 左右。
- 进一步锁重构可以在多并发场景提高至 50% 的性能（mTHP / 4K）。
- make -j96 / 768M memcg, 4K pages, 10G ZRAM
  - Before：Sys time: **73578.30**, Real time: **864.05**
  - After: Sys time: **33314.76**, Real time: **437.67**
- sysbench - oltp_read_only.lua --tables=32 --table-size=1280000 --threads=96
  - Before: **6292.11** tps
  - After: **8972.73** tps



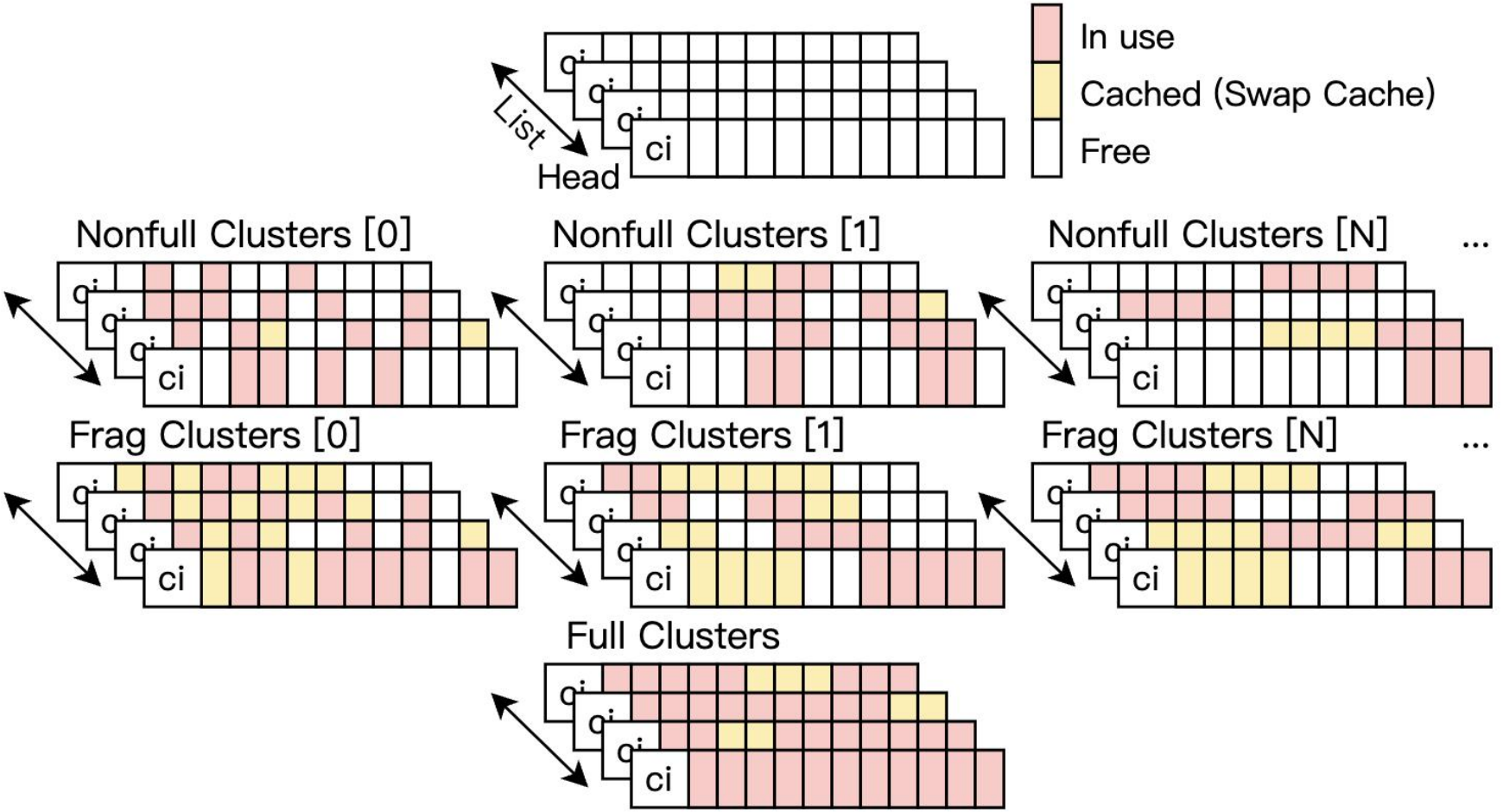
mTHP swap-out & swap-in

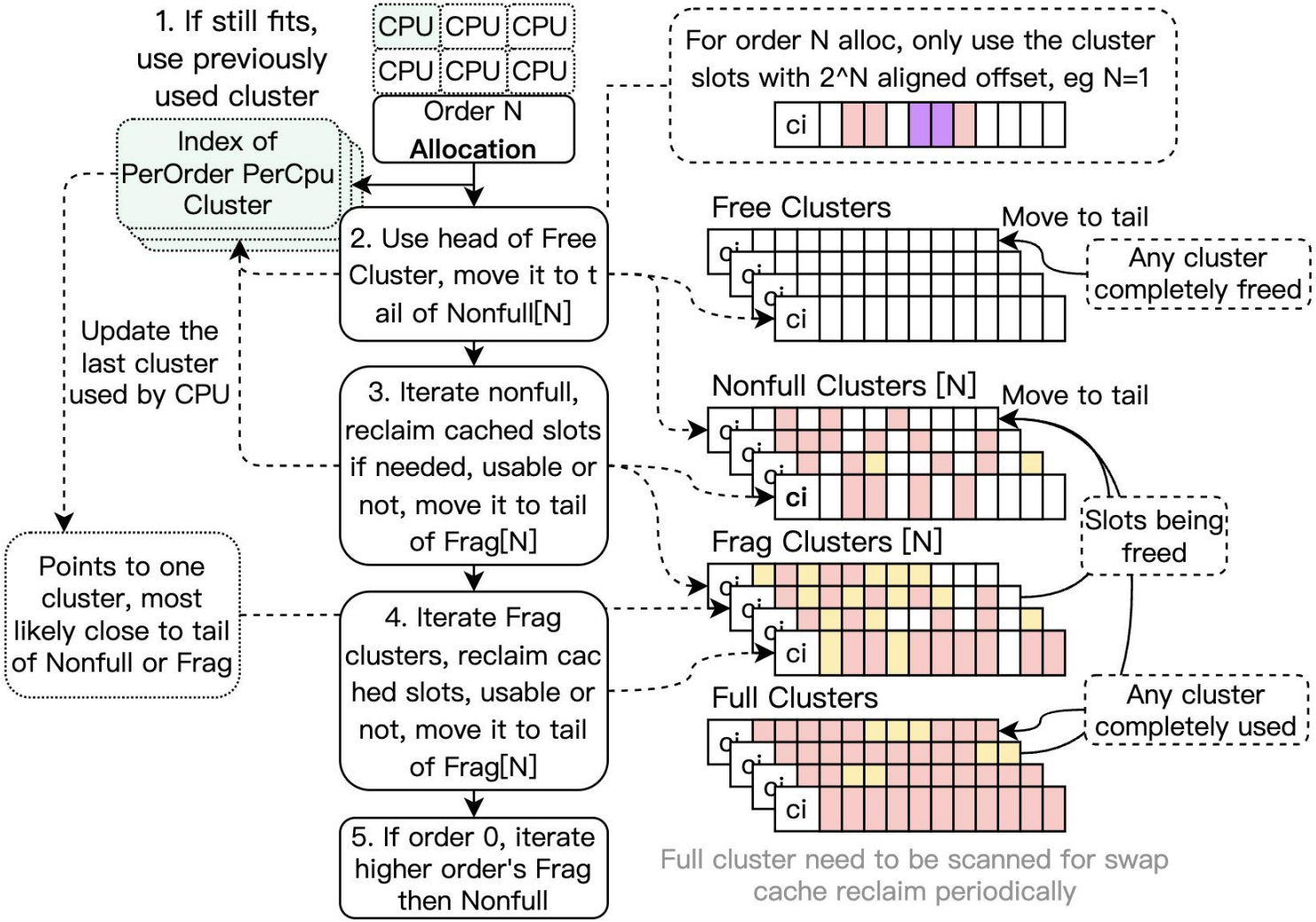


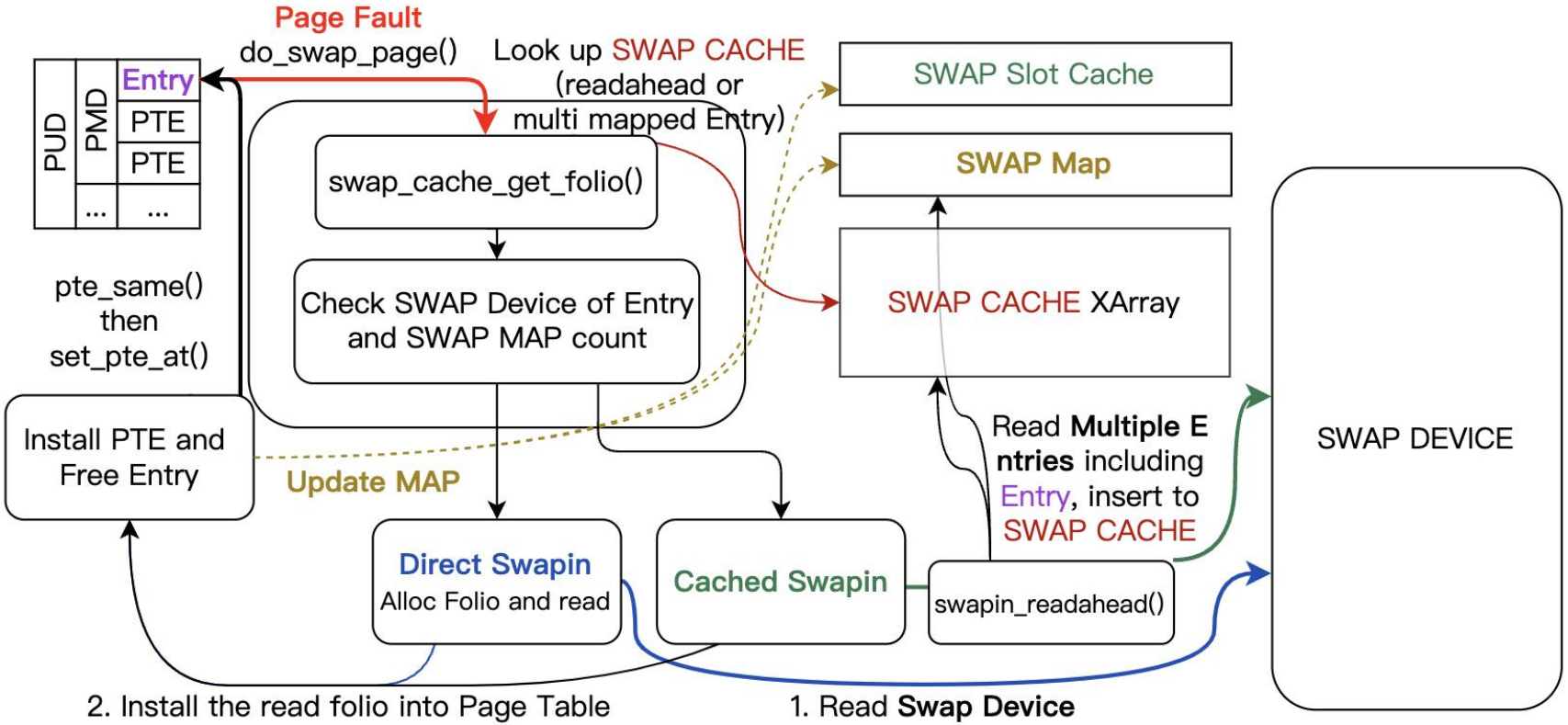
参考

- LPC Talk：https://lpc.events/event/18/contributions/1769/
- LPC Talk：https://lpc.events/event/18/contributions/1705/
- LPC Talk：https://lpc.events/event/18/contributions/1780/

附录


In use
Cached (Swap Cache)
Free
List
Head
ci
Nonfull Clusters [0]
Nonfull Clusters [1]
Nonfull Clusters [N]
...
Frag Clusters [0]
Frag Clusters [1]
Frag Clusters [N]
...
Full Clusters


1. If still fits, use previously used cluster
CPU CPU CPU
CPU CPU CPU
Order N Allocation
Index of PerOrder PerCpu Cluster
For order N alloc, only use the cluster slots with 2^N aligned offset, eg N=1
ci
2. Use head of Free Cluster, move it to t ail of Nonfull[N]
Free Clusters
Move to tail
Any cluster completely freed
Update the last cluster used by CPU
3. Iterate nonfull, reclaim cached slots if needed, usable or not, move it to tail of Frag[N]
Nonfull Clusters [N]
Move to tail
Slots being freed
Points to one cluster, most likely close to tail of Nonfull or Frag
4. Iterate Frag clusters, reclaim cac hed slots, usable or not, move it to tail of Frag[N]
Frag Clusters [N]
Full Clusters
Any cluster completely used
5. If order 0, iterate higher order's Frag then Nonfull
Full cluster need to be scanned for swap cache reclaim periodically


Page Fault
do_swap_page()
Look up SWAP CACHE
(readahead or
multi mapped Entry)
PUD
PMD
Entry
PTE
PTE
...
...
swap_cache_get_folio()
Check SWAP Device of Entry
and SWAP MAP count
SWAP Slot Cache
SWAP Map
SWAP CACHE XArray
pte_same()
then
set_pte_at()
Install PTE and
Free Entry
Update MAP
Direct Swapin
Alloc Folio and read
Cached Swapin
swapin_readahead()
Read Multiple E
ntries including
Entry, insert to
SWAP CACHE
SWAP DEVICE
2. Install the read folio into Page Table
1. Read Swap Device

感谢