

PR #39480 完整报告

sgl-project/sglang

[HiCache] Optimize buffer-mode storage existence bookkeeping

合并时间: 2026-09-16 01:49

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39480>

执行摘要

- 一句话: HiCache 存在缓存 LRU 扩至 512K 并优化查询热路径
- 推荐动作: 值得精读的 Python 热路径微优化范例: 展示了方法绑定 (hoisting)、EAFP 与 LBYL 的取舍、批量淘汰、以及容量 - 内存权衡的工程分析, 适合对缓存设计和性能优化感兴趣的工程师学习。但应意识到它缺少 benchmark 和专项测试, 落地到线上前建议补一次 buffer-mode 长上下文对比测试验证收益。

功能与动机

PR body 明确指出: `StorageExistenceCache` 位于 buffer-mode 热路径, 每次 prefetch 决策和 write-through 准入都要通过 `contains_all` / `covers_all` 走 page-hash 链; 在长上下文负载下, 128K 条目的 LRU 把 positive 挤出 belief set 的速度快于被复用的速度, 导致缓存对已存储页重复写入。因此需要让查询每页开销更小、并让缓存记住更多页, 从而减少冗余的 D2H 与存储写。

实现拆解

变更入口为 `python/sglang/srt/mem_cache/buffer_mode/storage_existence_cache.py`, 从容量与每页开销两个维度改造:

1. 容量放大: 将 `HICACHE_EXISTENCE_CACHE_MAX_ENTRIES` 从 $128 * 1024$ 提升为 $512 * 1024$, 注释同步更新为约 125 MiB 最坏内存、约 32M tokens (page size 64, 跨所有 pool) 覆盖量。容量是构造 `StorageExistenceCache` 的默认值, 外部未暴露配置项, 因此对调用方无接口影响。
2. 查询路径重写: `contains_all` 弃用 `all(self.contains(...) for ...)` 生成器, 改为单次绑定 `move_to_end` 的循环, 用 `move_to_end` 抛出的 `KeyError` 作为 miss 信号 (EAFP 风格), 命中时每个 key 只做一次 dict 查找; `covers_all` 在 `extra_cover` 为空时直接退化为 `contains_all`, 仅在非空时才逐页做额外成员判断, 避免默认空 `frozenset()` 带来的多余检查。
3. 写入与失效路径重写: `add` 将 `move_to_end` / `popitem` 绑定到局部变量, 插入完成后按 `len(entries) - self.max_entries` 计算淘汰数量并连续弹出最老条目, 替代旧版 `while` 循环中反复的 `len` 检查; `invalidate_beyond` 同样将 `pop` 绑定到循环外。两个方法最终保留的条目集合与旧实现等价。
4. 导入调整与测试配套: `Container` / `Iterable` / `Sequence` 从 `typing` 改为 `collections.abc` 导入 (运行时类型判定的正确来源)。未新增测试, 依赖现有 `test_buffer_mode_sidecar.p`

y 与 test_unified_radix_cache_unittest.py 覆盖，PR 说明本地均通过。

关键文件：

- python/sglang/srt/mem_cache/buffer_mode/storage_existence_cache.py (模块 存在缓存；类别 source；类型 core-logic；符号 StorageExistenceCache.add, StorageExistenceCache.contains, StorageExistenceCache.contains_all, StorageExistenceCache.covers_all)：唯一变更文件，既是容量配置的来源（HICACHE_EXISTENCE_CACHE_MAX_ENTRIES），也是 buffer-mode 热路径上所有存在性判断的载体；本次改动重写了 contains_all / covers_all / add / invalidate_beyond 四个方法的控制流。

关键符号：StorageExistenceCache.add, StorageExistenceCache.contains_all, StorageExistenceCache.covers_all, StorageExistenceCache.invalidate_beyond

关键源码片段

python/sglang/srt/mem_cache/buffer_mode/storage_existence_cache.py

唯一变更文件，既是容量配置的来源（HICACHE_EXISTENCE_CACHE_MAX_ENTRIES），也是 buffer-mode 热路径上所有存在性判断的载体；本次改动重写了 contains_all / covers_all / add / invalidate_beyond 四个方法的控制流。

```
from collections import OrderedDict
from collections.abc import Container, Iterable, Sequence
```

```
# 512K 条目 ≈ 125 MiB 最坏情况，覆盖约 32M tokens (page size 64, 跨所有 pool)。
# 相比原先 128K 条目，长上下文下 positive 更不容易在复用前被 LRU 挤出。
HICACHE_EXISTENCE_CACHE_MAX_ENTRIES = 512 * 1024
```

```
class StorageExistenceCache:
```

```
    """Buffer 模式下 host 侧“哪些页已写入 L3 存储”的 LRU 信念缓存。
```

```
    命中可跳过冗余 D2H 与存储写；误报最多导致一次冷重算，不影响正确性。
    所有变更都必须发生在 scheduler 线程的 lockstep 点，保证 TP 副本一致。
    """
```

```
    def __init__(self, max_entries: int = HICACHE_EXISTENCE_CACHE_MAX_ENTRIES):
        self.max_entries = max_entries
        self._entries: OrderedDict[tuple[str, str], None] = OrderedDict()
```

```
    def add(self, pool: str, hashes: Iterable[str]) -> None:
        entries = self._entries
        move_to_end = entries.move_to_end # 方法绑定到局部变量，省去循环内属性查找
        for h in hashes:
            key = (pool, h)
            entries[key] = None
            move_to_end(key) # 新 belief 标记为最新使用
        # 批量淘汰：先算超额数量再连续弹出，避免旧版 while 循环里反复 len 检查
```

```

pop_oldest = entries.popitem
for _ in range(len(entries) - self.max_entries):
    pop_oldest(last=False)

def contains(self, pool: str, page_hash: str) -> bool:
    entries = self._entries
    key = (pool, page_hash)
    if key not in entries:
        return False
    entries.move_to_end(key) # 命中即 LRU-touch
    return True

def contains_all(self, pool: str, hashes: Iterable[str]) -> bool:
    entries = self._entries
    move_to_end = entries.move_to_end
    for h in hashes:
        key = (pool, h)
        try:
            # EAFP 风格: move_to_end 内部只做一次 dict 查找,
            # 命中时比“先 in 检查再 move_to_end”少一次哈希查找
            move_to_end(key)
        except KeyError:
            return False # 链上任意一页缺失即整体不可用, 提前短路
    return True

def covers_all(
    self,
    pool: str,
    hashes: Iterable[str],
    extra_cover: Container[str] = frozenset(),
) -> bool:
    """每页相信已存储, 或位于 extra_cover (如 D2H 已发起、必达 storage-ack 的内容) 即返回 True。"""
    entries = self._entries
    move_to_end = entries.move_to_end
    if extra_cover:
        # 只在 extra_cover 非空时逐页判断, 避免默认空集合时的多余成员检查
        for h in hashes:
            key = (pool, h)
            if key in entries:
                move_to_end(key) # 相信已存储, 同时维护 LRU 顺序
            elif h not in extra_cover:
                return False
        return True
    # extra_cover 为空时退化为纯存在检查, 复用优化后的 contains_all
    return self.contains_all(pool, hashes)

def invalidate_beyond(
    self, pool: str, hashes: Sequence[str], keep_pages: int

```

) -> None:

```
"""来自 prefetch hit 查询的 ground-truth 修复：丢弃 keep_pages 之后的信念。"""
pop = self._entries.pop # 方法绑定到局部变量，循环外只查一次属性
for h in hashes[keep_pages:]:
    pop((pool, h), None) # 默认值 None，缺页时静默跳过
```

评论区精华

本 PR 没有任何 review 评论，是一次作者自合并的改动（merged_by: xiezhq-hermann，带 [bypass-fastfail](#) 标签）。唯一可观察到的外部信号是 CI：PR Test (Extra) 曾失败一次（Run #34939396445），作者连续 3 次发出 [/tag-and-rerun-ci](#) 触发重跑，最终主 CI (Base) 通过。失败原因未在评论中说明，结合上下文推测为 flaky 而非本次改动引入的确定性失败。

- CI Extra 失败与重跑 (other): 主 CI (Base) 最终通过；Extra 失败未见根因说明，推测为已存在的 flaky。

风险与影响

- 风险：
 1. 缺少量化验证：PR 明确写 'Benchmarking and Profiling: Not run for this PR'，性能收益是推断性的；EAFP 的 try/except KeyError 在命中率低的场景下反而可能因异常开销变慢，而容量放大后命中率是否显著提升没有数据支撑。
 2. 内存占用上升：512K 条目最坏约 125 MiB，且是调度器进程常驻内存；在小内存国产卡或 CPU 部署环境中可能引入额外压力。
 3. 无专项测试变更：extra_cover 非空分支、批量淘汰、KeyError 短路等新控制流没有对应单元测试，回归风险靠现有两个测试文件兜底。
 4. 批量淘汰的瞬时峰值：add 先全部插入再统一淘汰，当一次插入量大时，瞬时条目数可达 $\text{max_entries} + \text{len}(\text{hashes})$ ，略高于旧实现的逐个淘汰峰值。
 5. TP 确定性约束：文件注释强调所有变更必须发生在 scheduler 线程的 lockstep 点；本次改动未触碰该约束，但后续维护者若在类外新增调用点仍可能破坏确定性。- 影响：影响范围严格限定在 HiCache buffer-mode（Prefill-Disaggregation 中 KV 卸载到 L3 存储的场景）：每次 prefetch 决策和 write-through 准入都会走 contains_all / covers_all，长上下文负载下可减少重复存储写与 D2H 传输。对非 HiCache 用户无任何影响；无接口、配置或数据契约变化。团队侧，这是 HiCache 子系统近期优化序列中的一环，与 Router 存储层级感知、KV 事件归因等改动属于同一演进方向。- 风险标记：缺少 benchmark 数据，无专项测试变更，LRU 容量 4 倍提升推高内存占用

关联脉络

- PR #39516 [Fix] HiCache startup ImportError on the pinned kernel wheel: 同为 mem_cache / HiCache 子系统（pool_host/common.py、unified_radix_cache.py）的稳定性修复，与本次优化处于同一模块的可靠性主线。
- PR #39487 [Fix][DCP] Localize widened KV ids in MLA retraction CPU backup/restore: 同触及 mem_cache/memory_pool 与 pool_host 备份路径，与 buffer mode 的存储信念修复属于同一个子系统的相邻代码。

- PR #39368 [CI][PD] Skip the flaky decode HiCache file-backend disaggregation test: HiCache 相关测试稳定性处理，说明该子系统近期在补可靠性与清理冗余用例，与本次 hot path 优化的配套测试缺口形成对照。
- PR #37482 feat(agent sessions): attribute stored KV cache blocks to sessions: KV 事件与会话归因改造，与 buffer mode 存储簿记存在语义关联，同属 HiCache KV 存储体系演进。