PR #36227 完整报告

sgl-project/sglang

[HiCache] Retry L3 storage prefetch after a missed attempt

合并时间：2026-08-28 16:35

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36227

执行摘要

- 一句话：为排队请求重试 L3 存储预取，提升高并发下命中率
- 推荐动作：值得 HiCache、调度与缓存生命周期相关工程师精读：miss 标记一次性消费、按调度 pass 而非墙钟节奏重试、anchor 提前 pin 与 cap clamp 三个设计点有借鉴价值。由于缺少公开性能复现且功能默认关闭，生产启用前建议基于自身负载 benchmark 并监控调度延迟。

功能与动机

PR body 指出：高并发下排队请求的首次 L3 存储可用性检查往往运行在满足它的写回提交之前，预取被拒绝或撤销，请求随后落入全量重算，而内容在片刻后才落盘——在缓存最需要的时候造成有效存储命中率的大幅下降。同一时间窗口也伤害 buffer 模式 anchor：它在 hit-alloc 时才懒加载 pin，届时节点常已被淘汰，导致 load 在无锁状态下启动。

实现拆解

1. 缓存侧 miss 标记：python/sglang/srt/mem_cache/unified_radix_cache.py 新增 _storage_prefetch_missed_rids 集合，在 prefetch_from_storage() 的各类“已解析但不可用”分支记录 req_id（too-short/fully-matched 拒绝、rate-limit 拒绝、aux-alloc 放弃、controller 侧终止、低于阈值命中）；新增 pop_storage_prefetch_miss() 提供一次性消费语义；请求被调度（pop_prefetch_loaded_tokens）或中止时丢弃未服务标记，防止泄漏。
2. 调度器重试扫描：python/sglang/srt/managers/scheduler.py 新增 _retry_missed_storage_prefetches()，在 _get_new_batch_prefill_raw() 的 hicache 事件 tick 之后、enable_hicache_storage 开启时执行。它按调度 pass 计数而非墙钟时间控制节奏（保证 TP 各 rank 同一 pass 重发），并显式遍历整个 waiting_queue（准入循环在第一个不可调度请求处停止）。每个请求受 poll-interval 与 max-attempts 约束，默认 0 表示禁用。
3. buffer 模式 anchor 改进：python/sglang/srt/mem_cache/buffer_mode/pipeline.py 中 anchor_lock_cap_tokens 改为按 full_pool.size - max_context_len 裁剪并设下限 0，防止排队 pin 在池满时永久卡死准入；UnifiedRadixCache.prefetch_from_storage() 在预取发起时即调用 try_lock_anchor() 提前 pin 刚匹配的 anchor，hit-alloc 时的 pin 保留为幂等的第二次机会。
4. 配置与配套：python/sglang/srt/server_args.py 新增 --hicache-storage-prefetch-retry-poll-interval（默认 0= 禁用）与

--hicache-storage-prefetch-retry-max-attempts（默认 4）；python/sglang/srt/managers/schedule_batch.py 为请求补充重试状态字段；dense 路径 HiRadixCache.pop_storage_prefetch_miss() 返回 False，使重试机制在该路径下惰性生效。

5. 测试：test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py 新增两个用例（miss 标记一次性消费与重试后命中、abort 清理；anchor cap 按上下文余量裁剪），并固定到单一配置（page_size=1, sliding_window_size=4）以避免各 fixture 在 CI GPU 上累计约 2.5 GiB 显存占用。

关键文件：

- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 _retry_missed_storage_prefetches）：重试机制的调度侧入口：新增 _retry_missed_storage_prefetches() 并在每次调度 pass 的 hicache tick 后扫描整个等待队列，是整项功能的核心控制流。
- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 缓存层；类别 source；类型 core-logic；符号 pop_storage_prefetch_miss, prefetch_from_storage, pop_prefetch_loaded_tokens）：miss 标记的归属地：_storage_prefetch_missed_rids 记录各类解析后不可用的预取结果，pop_storage_prefetch_miss() 为调度器提供一次性消费接口，同时提前 pin buffer 模式 anchor。
- test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py（模块 缓存测试；类别 test；类型 test-coverage；符号 test_buffer_only_storage_prefetch_miss_marker_and_retry, test_buffer_only_anchor_lock_cap_clamped_by_context_headroom）：新增两个 buffer_only 用例覆盖 miss 标记生命周期、重试后命中与 anchor cap 裁剪，是功能验证的主体。
- python/sglang/srt/mem_cache/buffer_mode/pipeline.py（模块 缓冲管线；类别 source；类型 core-logic；符号 BufferModePipeline.init）：anchor 锁上限按准入余量裁剪，防止排队 pin 卡死准入，并打印解析后的 cap。
- python/sglang/srt/server_args.py（模块 启动配置；类别 source；类型 configuration）：新增两个重试参数，默认关闭保证行为中性。
- python/sglang/srt/mem_cache/hiradix_cache.py（模块 密集缓存；类别 source；类型 core-logic；符号 pop_storage_prefetch_miss）：dense 路径提供 inert 的 pop_storage_prefetch_miss()，明确重试机制适用范围。
- python/sglang/srt/managers/schedule_batch.py（模块 批次管理；类别 source；类型 configuration）：为请求补充重试计数与等待状态字段，支撑调度器按 pass 计数。

关键符号：_retry_missed_storage_prefetches, UnifiedRadixCache.pop_storage_prefetch_miss, UnifiedRadixCache.prefetch_from_storage, UnifiedRadixCache.pop_prefetch_loaded_tokens, HiRadixCache.pop_storage_prefetch_miss, BufferModePipeline.init

关键源码片段

python/sglang/srt/managers/scheduler.py

重试机制的调度侧入口：新增 _retry_missed_storage_prefetches() 并在每次调度 pass 的 hicache tick 后扫描整个等待队列，是整项功能的核心控制流。

```
def _retry_missed_storage_prefetches(self):
    """重新为等待队列中预取未命中的请求发起可用性检查。

    节奏按"调度 pass"计数而非墙钟时间：这样 TP 各 rank 会在同一
    pass 内重发，避免 rank 间错开。扫描整个等待队列而非在准入循环内
    处理，因为准入循环在第一个不可调度的请求处就会停止。
    interval <= 0 时整体禁用（默认关闭，行为中立）。
    """
    interval = get_memory().hicache_storage_prefetch_retry_poll_interval
    if interval <= 0 or not self.waiting_queue:
        return
    max_attempts = get_memory().hicache_storage_prefetch_retry_max_attempts
    for req in self.waiting_queue:
        # 每个 miss 标记只消费一次：取走即代表本次 miss 已进入重试节奏
        if self.tree_cache.pop_storage_prefetch_miss(req.rid):
            req.storage_prefetch_retry_pending = True
            req.storage_prefetch_retry_wait_polls = 0
        if (
            not req.storage_prefetch_retry_pending
            or req.storage_prefetch_retry_attempts >= max_attempts
        ):
            continue
        req.storage_prefetch_retry_wait_polls += 1
        if req.storage_prefetch_retry_wait_polls <= interval:
            continue
        # 等待足够 pass 后重发一次可用性检查；再次 miss 会重新触发
        req.storage_prefetch_retry_pending = False
        req.storage_prefetch_retry_attempts += 1
        logger.debug(
            "HiCache storage prefetch retry req=%s attempt=%d",
            req.rid,
            req.storage_prefetch_retry_attempts,
        )
        self._prefetch_kvcache(req)
```

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

anchor 锁上限按准入余量裁剪，防止排队 pin 卡死准入，并打印解析后的 cap。

```
# 仅展示构造函数中与 anchor 锁上限相关的核心分支，其余字段初始化省略
#  （写队列、锁表等与原实现一致）
def __init__(
    self,
    cache: UnifiedRadixCache,
    swa_window_pages: int,
    write_backlog_cap: int,
    max_context_len: int = 0,
```

```
):
    self._cache = cache
    # SWA 窗口大小在池组装后静态确定；backlog 上限决定新增 intent
    # 是否在准入时被丢弃（后续命中可重新触发）。
    self.write_backlog_cap = write_backlog_cap
    self.anchor_lock_enabled = envs.SGLANG_ENABLE_HICACHE_BUFFER_ANCHOR_LOCK.get()
    from sglang.srt.mem_cache.swa_memory_pool import SWAKVPool

    kvcache = cache.token_to_kv_pool_allocator.get_kvcache()
    full_pool = kvcache.full_kv_pool if isinstance(kvcache, SWAKVPool) else kvcache
    # 按准入余量裁剪 anchor 锁上限：pin 必须为最大允许请求留出空间，
    # 否则排队中的 pin 会在池满时永久卡死准入（无可回收对象）。
    # 无余量的池不允许任何 pin。
    self.anchor_lock_cap_tokens = max(
        0,
        min(
            int(envs.SGLANG_HICACHE_BUFFER_ANCHOR_LOCK_CAP.get() * full_pool.size),
            full_pool.size - max_context_len,
        ),
    )
    logger.info(
        "BufferModePipeline anchor_lock_enabled=%s cap_tokens=%d",
        self.anchor_lock_enabled,
        self.anchor_lock_cap_tokens,
    )
    self.reset()
```

评论区精华

本 PR 无实质 review 讨论：唯一 review 为 hanming-lu 的 APPROVED（无文字评论）；issue 评论仅有作者触发的 CI 重跑指令 /tag-and-rerun-ci。设计取舍主要记录在 PR body 与源码注释中。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 调度开销：启用 retry 后每个调度 pass 都会遍历整个 waiting_queue 做集合查询与计数，超大队列下需关注调度延迟。
  2. TP 节奏假设：按本地调度 pass 计数假设各 rank 进度一致，负载不均时重试时刻可能错开，scheduler.py 的 pacing 依赖该假设。
  3. 标记生命周期：miss 标记在 pop_prefetch_loaded_tokens 与 abort 路径清理，若存在其他未覆盖的退出路径会滞留（unified_radix_cache.py，测试已覆盖 abort 场景）。
  4. anchor pin 行为：提前 pin 延长占用窗口；cap 裁剪依赖 server_args.context_length，未配置时退化为不裁剪（pipeline.py）。

5. 验证缺口：无公开 serving 级 benchmark，内部部署测量未附可复现配方，收益量级需生产环境自行验证。 - 影响：默认关闭（poll-interval=0），对现有用户零行为变化；启用后预期在高并发与 L3 命中场景减少全量重算、降低尾延迟，但无公开量化数据。buffer 模式在 anchor lock 开启时行为有变：pin 时机提前且 cap 受上下文余量约束，影响 KV 池占用节奏。团队新增两个启动参数与请求级重试字段，文档（README/CLI help）未同步更新。 - 风险标记：默认关闭需显式开启，调度器每 pass 扫描等待队列，跨 TP rank 节奏同步假设，缺 serving 级 benchmark，文档未同步更新

关联脉络

- PR #35931 [HiCache] Reject load-back specs that claim nodes pinned by an in-flight load-back: 同属 HiCache 统一缓存：处理 pin/load-back 并发时序，与本 PR 的 anchor 提前 pin 形成互补。
- PR #36705 [HiCache] Stop populating host-pool mmaps twice (-13% allocation time): 同属 HiCache 存储 / 主机池性能线，本 PR 进一步消除 L3 预取时序导致的有效命中率损失。
- PR #36637 [mem_cache] Add free_full to release the full side of a tombstoned SWA node: 同属 mem_cache 状态机演进，关注缓存释放语义与生命周期管理，与本 PR 的 miss 标记生命周期清理相关。