

PR #39283 完整报告

sgl-project/sglang

[HiCache] Rework the buffer-mode storage prefetch pipeline and retry bookkeeping

合并时间: 2026-09-16 01:48

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39283>

执行摘要

- 一句话: 重构 HiCache buffer 预取管线, 修复命中记账错乱与重试失控
- 推荐动作: 值得精读。这是 HiCache buffer-mode 预取管线的核心重构, 建议重点关注三个设计决策: 一是 `add_one_req` 的选择 / 提交分离模式, 是“先记账后确认”型准入缺陷的通用解法; 二是 FULL-only pin 与 SWA segment lock 的分离理由; 三是 Rust/Python 双实现共享测试时的显式覆盖回退管理方式 (NotImplementedError stub + 测试 skip)。对缓存子系统的维护者, 本 PR 的 rebase 历史 (适配 #38483、#32888、#39318) 也是理解项目演化脉络的很好素材。

功能与动机

PR body 明确指出: buffer-mode HiCache 此前在 `prefill adder` 内部决定 staged L3 hit 的命运, 但 admission 已先按 host-hit token 记账, 一旦设备前缀在排队期间被收缩, 这些已记账的 token 无法再拼接, 负载下产生 phantom host-hit accounting、holds 卡住准入以及无界 storage-prefetch 重试; 此外 PD-prefill 与 PP 调度器在一次调度器重构后停止了 HiCache 事件排空。作者的目标是让 L3 hit 在准入选择前完成规划、只物化一次, 并把重试路径变为有界行为。

实现拆解

该 PR 的实现分五个层次推进:

1. 调度器与准入流程解耦: `PrefillAdder.add_one_req` 拆分为“选择”与“提交”两阶段——`_select_prefill_admission` 在节点锁下只计算 prefill shape (不分配槽位、不物化 host hit), `_commit_prefill_admission` 再通过 `tree_cache.init_load_back` 物化 host hit 并提交预算。`init_load_back` 可返回 `None` 触发重新选择; FULL 超额交付时收缩 extend range, 欠额交付则按零 host hit 重新选择, 部分交付会抛 `RuntimeError` 阻止半提交。SWA 预算判断收敛进 `_swa_admission_gate`, 保留上游 #38192 的 ring-slot 精确匹配语义。
2. 有界重试与事件排空: 新增 `mem_cache/storage_prefetch.py` 的 `StoragePrefetchRetries`, 统一管理 miss 的节流轮询 `poll_miss`、已知 hit 锚点移动后的立即重查 `refetch` 以及等待队列顺序化 `pop_ready`, 配合 `--hicache-storage-prefetch-retry-max-attempts` (默认 8) 封顶; `scheduler._process_storage_prefetch_retries` 消费到期项, `_prefetch_after_device_hit_loss` 处理排队期间设备前缀收缩后的重查。新增 `_process_hicache_events` 在 `get_next_batch_to_run` 开头统一执行事件排空, 恢复

PD-prefill 与 PP 循环的 HiCache 事件处理。

3. Buffer-mode 预取管线生命周期重构: staged prefetch 锚点改为通过 `match_full_device_prefix` 在活树上重新匹配, 并用 `inc_full_pin` / `dec_full_pin` 对 FULL 槽位加 FULL-only pin (有意不动 SWA segment lock, 因为其 receipt 在窗口重挂后失效); `prepare_prefetch` 只返回 `staging_tokens` 大小, 命中确认后才由 `alloc_prefetch_staging` 分配 staging 缓冲; 准入期设备容量 deferral 设上限 `max_staged_admission_defers = 32`, 超限即丢弃 hold 并重算。
4. SWA 墓碑窗口修复与 Rust 同步: Python tree core 新增 `_walk_span`、`swa_tombstone_ranges`、`attach_swa_window` 支撑 buffer-mode 消费时修补墓碑化的 SWA 窗口; Rust `unified_tree_core.rs` 补齐 `match_full_device_prefix`、`inc_full_pin`、`dec_full_pin` 以及 `staging_tokens` 参数, 但两个 SWA 修复助手暂以 `NotImplementedError` stub 挂载, 3 个共享测试在 Rust 后端跳过。
5. 测试与配置配套: 新增 `test_scheduler_hicache_events.py` (4 个场景验证 drain 时机与 feature gate) 和 `test_storage_prefetch_lifecycle.py` (498 行覆盖 trim 边界、双 rank 重试轨迹、锚点丢失); `test_unified_radix_cache_unittest.py` 增加 653 行并修复 atexit 显存钉死问题 (峰值 33 → 7 GiB); 新配置项 `hicache_storage_prefetch_retry_max_attempts` 与 `hicache_storage_prefetch_retry_poll_interval` 落入 `arg_groups/fields/memory.py`。

关键文件:

- `python/sglang/srt/managers/scheduler.py` (模块 调度器; 类别 source; 类型 core-logic; 符号 `_prefetch_kvcache`, `_process_storage_prefetch_retries`, `_retry_storage_prefetch`, `_prefetch_after_device_hit_loss`): 调度主路径: 新增 `_process_hicache_events` 统一事件排空, 重试逻辑改为 `_process_storage_prefetch_retries / _retry_storage_prefetch / _prefetch_after_device_hit_loss`, 恢复 PD-prefill 与 PP 循环的 HiCache 事件处理。
- `python/sglang/srt/managers/schedule_policy.py` (模块 预填准入; 类别 source; 类型 core-logic; 符号 `_PrefillAdmission`, `_swa_admission_gate`, `_select_prefill_admission`, `_commit_prefill_admission`): 准入策略核心: `add_one_req` 拆分为 `_select_prefill_admission / _commit_prefill_admission`, 引入 `_PrefillAdmission` 数据类与 `_swa_admission_gate`, 处理 load-back 部分 / 超额交付的重新选择与预算修正。
- `python/sglang/srt/mem_cache/unified_cache/unified_tree_core.py` (模块 缓存树; 类别 source; 类型 core-logic; 符号 `inc_full_pin`, `dec_full_pin`, `match_full_device_prefix`, `_walk_span`): Python 树核新增 FULL-only pin (`inc_full_pin` / `dec_full_pin`)、只读 FULL 设备前缀匹配 `match_full_device_prefix`, 以及 SWA 墓碑区间扫描与窗口重挂 (`swa_tombstone_ranges` / `attach_swa_window`)。
- `python/sglang/srt/mem_cache/buffer_mode/pipeline.py` (模块 预取管线; 类别 source; 类型 core-logic; 符号 `staged_splice_tokens`, `swa_transient_size`, `try_lock_anchor`, `prepare_staged_prefetch`): buffer-mode 预取管线核心: 重写 `_AnchorLock` 为 FULL-only pin、`try_lock_anchor` 改为重新匹配并区分 `cap_skip`/`cap_busy`/`anchor_lost`, 新增 `prepare_staged_prefetch`、`plan_staged_splice` 与 `swa_transient_size`。

- python/sglang/srt/mem_cache/storage_prefetch.py (模块 预取重试; 类别 source; 类型 core-logic; 符号 _StoragePrefetchRetry, StoragePrefetchRetries, init, poll_miss) : 新增模块: 用 StoragePrefetchRetries 统一管理 paced miss 轮询与锚点移动后的立即重查, 按等待队列顺序和 step 时钟保证 TP rank 同步, 是本次“有界重试”的核心簿记。
- python/sglang/srt/mem_cache/unified_radix_cache.py (模块 缓存集成; 类别 source; 类型 core-logic; 符号 _log_storage_prefetch_deferred, pop_storage_prefetch_miss, plan_staged_splice, staged_prefetch_swa_tokens) : 集成层: prefetch_from_storage 识别 storage_hit_end 并假设已存区间、改为按 staging_tokens 预分配、接入 StoragePrefetchRetries, 并补充 issued_assumed_stored 统计与 resolve_prefetch_threshold。
- rust/sglang-radix-tree/src/unified_tree_core.rs (模块 缓存树; 类别 source; 类型 core-logic; 符号 inc_full_pin, dec_full_pin, match_full_device_prefix) : Rust 树核同步实现 inc_full_pin / dec_full_pin / match_full_device_prefix, 并在 build_hicache_transfers 中贯通 staging_tokens 参数。
- python/sglang/srt/mem_cache/rust_tree_core/adaptor.py (模块 Rust 适配; 类别 source; 类型 core-logic; 符号 swa_tombstone_ranges, attach_swa_window, match_full_device_prefix, inc_full_pin) : Rust 适配层: match_full_device_prefix / inc_full_pin / dec_full_pin 桥接绑定, swa_tombstone_ranges 与 attach_swa_window 以显式 NotImplementedError stub 暴露已知覆盖回退。
- test/registered/unit/managers/test_scheduler_hicache_events.py (模块 调度测试; 类别 test; 类型 test-coverage; 符号 TestSchedulerHiCacheEvents, test_feature_gates, test_pd_prefill_drains_before_admission_even_with_empty_queue, test_unified_drains_when_prefill_is_deferred) : 新增调度器 HiCache 事件排空测试: 验证 feature gate 组合、PD-prefill 空队列也排空、prefill 延迟时仍排空、PP 循环排空顺序。
- test/registered/unit/mem_cache/test_storage_prefetch_lifecycle.py (模块 预取测试; 类别 test; 类型 test-coverage; 符号 TestStagedPrefetchLifecycle, test_trim_and_stage_preserve_raw_token_boundaries, test_two_rank_completion_capacity_and_anchor_loss, test_next_pass_uses_fresh_joint_match) : 新增 staged 预取生命周期测试: 覆盖 trim/stage 边界、双 rank 完成容量与 anchor 丢失、下一轮 pass 使用全新 joint match, 是 buffer-mode 管线行为的主要回归防线。

关键符号: _select_prefill_admission, _commit_prefill_admission, _swa_admission_gate, _process_hicache_events, _process_storage_prefetch_retries, _retry_storage_prefetch, _prefetch_after_device_hit_loss, match_full_device_prefix, inc_full_pin, dec_full_pin, swa_tombstone_ranges, attach_swa_window, prepare_staged_prefetch, try_lock_anchor, plan_staged_splice, pop_ready, poll_miss, refetch, cancel, alloc_prefetch_staging, swa_transient_size

关键源码片段

[python/sglang/srt/managers/scheduler.py](#)

调度主路径：新增 `_process_hicache_events` 统一事件排空，重试逻辑改为 `_process_storage_prefetch_retries / _retry_storage_prefetch / _prefetch_after_device_hit_loss`，恢复 PD-prefill 与 PP 循环的 HiCache 事件处理。

```
def _process_hicache_events(self) -> None:
```

```
    # HiCache 事件排空是 TP 全域共识：必须在 rank 本地决策（如
    # _should_defer_prefill）之前执行，否则各 rank 会进入不同集合通信
    if (self.enable_hierarchical_cache or get_memory().enable_flexkv
        or self.enable_unified_cache_external_linker):
        self.tree_cache.check_hicache_events()
    if self.enable_hicache_storage:
        self._process_storage_prefetch_retries()
```

```
def _retry_storage_prefetch(self, req, storage_hit_end=None) -> None:
```

```
    req.storage_prefetch_retry_attempts += 1
    max_attempts = get_memory().hicache_storage_prefetch_retry_max_attempts
    if req.storage_prefetch_retry_attempts >= max_attempts:
        # 达到重试上限：不再发起 L3 查询，直接按设备已有内容准入
        logger.warning(
            "HiCache storage prefetch reissue cap reached req=%s attempts=%d; "
            "the request is admitted without further L3 lookups",
            req.rid, req.storage_prefetch_retry_attempts)
    else:
        logger.debug("HiCache storage prefetch re-issue req=%s attempt=%d",
                     req.rid, req.storage_prefetch_retry_attempts)
    self._prefetch_kvcache(req, storage_hit_end)
```

```
def _prefetch_after_device_hit_loss(self, req: Req) -> bool:
```

```
    # 排队期间设备前缀被换出后，重新查询新暴露的 L3 区间；
    # 已有 staged hold 时不再重查，避免与已停靠的 fetch 冲突
    previous_match_len = req.storage_prefetch_last_match_len
    buffer_pipeline = self.tree_cache.buffer_pipeline
    if not previous_match_len or (
        buffer_pipeline is not None
        and buffer_pipeline.has_staged(req.cache_request_handle)
    ):
        return False
    current_match_len = len(req.prefix_indices) + req.host_hit_length
    if current_match_len >= previous_match_len:
        return False
    if req.storage_prefetch_retry_attempts >= get_memory().hicache_storage_prefetch_retry_max_attempts:
        # 超过重试上限：接受较短的实际匹配，不再重试
        req.storage_prefetch_last_match_len = current_match_len
        return False
    logger.warning(
        "HiCache device prefix shrank before admission req=%s "
```

```

        "lookup_match=%d current_match=%d; reissuing storage lookup",
        req.rid, previous_match_len, current_match_len)
    self._retry_storage_prefetch(req)
    return True

```

python/sglang/srt/mem_cache/unified_cache/unified_tree_core.py

Python 树核新增 FULL-only pin (inc_full_pin / dec_full_pin) 、只读 FULL 设备前缀匹配 match_full_device_prefix, 以及 SWA 墓碑区间扫描与窗口重挂 (swa_tombstone_ranges / attach_swa_window) 。

```

def swa_tombstone_ranges(self, key: RadixKey, start: int, end: int) -> list[tuple[int, int]]:
    """返回 ``[start, end)`` 内被 SWA 墓碑覆盖的最大连续区间。"""
    ranges: list[tuple[int, int]] = []
    for child, pos, prefix_len in self._walk_span(key, end):
        seg_end = pos + prefix_len
        if seg_end <= start:
            continue
        # SWA 设备 value 为 None 即墓碑段, 需从负载中补回
        if child.component_data[ComponentType.SWA].value is not None:
            continue
        lo, hi = max(start, pos), min(end, seg_end)
        if ranges and ranges[-1][1] == lo:
            ranges[-1] = (ranges[-1][0], hi) # 相邻区间合并
        else:
            ranges.append((lo, hi))
        if hi >= end:
            break
    return ranges

```

```

def attach_swa_window(self, key, window_start, window_end, swa_values):
    """把 L3 加载回来的 SWA 片段挂回墓碑化的树区间, 返回产生的切分动作。"""
    assert len(swa_values) == window_end - window_start, (
        f"attach_swa_window size mismatch: got {len(swa_values)} "
        f"for [{window_start}, {window_end})")
    actions: list = []
    covered = window_start
    for child, pos, prefix_len in list(self._walk_span(key, window_end)):
        seg_end = pos + prefix_len
        if seg_end <= window_start:
            continue
        seg_start = max(pos, window_start)
        assign_end = min(seg_end, window_end)
        # 只能挂到墓碑段上, 覆盖活 SWA 说明状态已不一致
        assert child.component_data[ComponentType.SWA].value is None, (
            f"attach_swa_window over live SWA at [{seg_start}, {assign_end}) "
            f"of [{window_start}, {window_end})")
        self._attach_swa_segment(
            child, pos, seg_start, assign_end,

```

```

        swa_values[seg_start - window_start: assign_end - window_start],
        actions)
    covered = assign_end
    if covered >= window_end:
        break
    assert covered == window_end, (
        f"attach_swa_window covered {covered} of [{window_start}, {window_end}]")
    return actions

```

python/sglang/srt/mem_cache/storage_prefetch.py

新增模块：用 StoragePrefetchRetries 统一管理 paced miss 轮询与锚点移动后的立即重查，按等待队列顺序和 step 时钟保证 TP rank 同步，是本次“有界重试”的核心簿记。

```

class StoragePrefetchRetries:
    """新 L3 尝试的统一簿记：step 时钟让 TP 各 rank 保持同步，
    重试按 waiting 队列顺序发出，超过预算的请求直接按现有设备内容准入。"""

    def __init__(self):
        self._pending: dict[str, _StoragePrefetchRetry] = {}
        self._step = 0

    def poll_miss(self, req_id: str, storage_hit_end=None) -> None:
        # 非 immediate 的 miss：按 poll 间隔等待，避免空转
        self._pending[req_id] = _StoragePrefetchRetry(False, storage_hit_end)

    def refetch(self, req_id: str, storage_hit_end=None) -> None:
        # 已知 hit 但锚点移动：下一 pass 立即重查
        self._pending[req_id] = _StoragePrefetchRetry(True, storage_hit_end)

    def cancel(self, req_id: str) -> None:
        self._pending.pop(req_id, None)

    def pop_ready(self, waiting_queue, interval, max_attempts):
        self._step += 1
        if not waiting_queue or not self._pending:
            return []
        # 推测性 miss 绝不能阻塞队头：队头的 spec miss 直接取消
        head_id = waiting_queue[0].rid
        head_retry = self._pending.get(head_id)
        if head_retry is not None and not head_retry.immediate:
            self.cancel(head_id)
        if not any(
            retry.due_step is None or retry.due_step <= self._step
            for retry in self._pending.values()
        ):
            return []

        ready = []
        for req in waiting_queue:

```

```

retry = self._pending.get(req.rid)
if retry is None:
    continue
if req.storage_prefetch_retry_attempts >= max_attempts:
    # 达到重试上限：放弃该 L3 查询，按设备现状准入
    self.cancel(req.rid)
    continue
if not retry.immediate:
    # paced miss: 按 interval 节流，interval <= 0 表示不重试
    if interval <= 0:
        self.cancel(req.rid)
        continue
    if retry.due_step is None:
        retry.due_step = self._step + interval
    if retry.due_step > self._step:
        continue
self.cancel(req.rid)
ready.append((req, retry.storage_hit_end))
return ready

```

评论区精华

该 PR 没有产生实质 code review 讨论（两位 reviewer 直接 approve，其中一位评论 “Awesome”），设计权衡主要由作者在 PR body 与 issue 评论中自述。值得记录的几个要点：

“The last one passed under Rust before this PR, so this is a deliberate coverage regression for buffer-only + SWA + L3 on the Rust core until the helpers are ported; the Python core is unaffected.”

作者明确承认 Rust 后端 3 个共享测试跳过是一次有意的覆盖回退，而非无意的遗漏。

“`StoragePrefetchRetries` stays keyed by `rid` (it paces requests in the waiting queue, so an attempt change must not orphan a pending retry).”

在 rebase 到 #39318 (attempt-scoped CacheRequestHandle) 时，作者决策：预取管控按 `rid` 键控，避免 attempt 变化导致等待队列中的重试孤儿化。

“Fixtures now unregister the hook via `addCleanup`; peak device memory for the suite drops from ~33 GiB to ~7 GiB.”

1-GPU CI shard OOM 的根因是 `init_hicache` 用 `atexit` 注册 `cache.shutdown`，钉住所有 HiCache fixture 直到进程退出；修复后显存峰值大幅下降。

- 准入期 load-back 交付与 admission 预算不一致的处理 (correctness): 全量提交或全不提交；部分交付属于契约违例，必须 fail-loud。
- Rust tree core 未移植 SWA 窗口修复助手 (design): 承认是有意的覆盖回退，Python core 不受影响，等待后续 Rust port。
- 测试套件 atexit 导致 1-GPU shard OOM (testing): fixture 改用 `addCleanup` 注销钩子，峰值设备内存从约 33 GiB 降到约 7 GiB。

- rebase 三次的上游适配决策 (design): 保留本 PR 逻辑并按 handle 键控管线状态; 重试簿记必须按 rid, 避免 attempt 变化孤儿化等待队列中的重试。

风险与影响

- 风险: 本 PR 存在四类主要风险:
- 1. 核心调度路径变更: `schedule_policy.py` 的 `add_one_req` 和 `scheduler.py` 的 `_process_hicache_events` 是请求准入主路径, 选择 / 提交拆分的任何边界错误都可能表现为吞吐回退或准入卡死; `init_load_back` 返回 `None` 触发重新选择的循环路径需要调度器确保不活锁。
- 2. Rust 后端覆盖回退: `rust_tree_core/adapter.py` 中 `swa_tombstone_ranges` 与 `attach_swa_window` 是显式 `NotImplementedError` stub, 启用 Rust 树核的 `buffer-only + SWA + L3` 部署将无法执行窗口修补路径, `test_buffer_only_load_back_trims_head_published_by_sibling` 等用例在 Rust 后端被跳过。
- 3. 新配置默认值的语义影响: `max_attempts = 8` 对超预算请求直接“按设备现状准入”, 极端 L3 拥塞下可能牺牲部分命中率; `poll_interval` 默认改为 8 会改变重试节奏, 需要监控命中率指标。
- 4. 双树核同步维护成本: `staging_tokens`、FULL pin 等契约同时在 Python 与 Rust 侧维护, 任何一侧漏改都会导致共享套件失败或行为分叉, 本 PR 已因该问题在 CI 修复过一次 `test_swa_prefetch_commit_end_to_end`。- 影响: 影响范围集中在启用 HiCache `buffer_only` 主机内存模式并接入 L3 存储的部署: 修复了高并发下 phantom host-hit 记账、准入 holds 卡死和重试无界三个问题, 内部验证显示 warm-cache 前缀命中率不再塌陷。对团队而言, 调度器与缓存子系统的内部接口 (admission 两阶段、FULL-only pin、staging 契约) 发生变化, 后续 HiCache 改动需同步 Python 与 Rust 两侧; Rust 树核的 SWA 窗口修补 port 是明确的待办技术债。未启用 HiCache 的普通路径不受影响, 因为事件排空与预取重试都受 feature gate 控制。- 风险标记: 核心调度路径变更, Rust 后端覆盖回退, 新配置参数影响行为, 显存占用回归风险, Python/Rust 双 core 耦合

关联脉络

- PR #39480 [HiCache] Optimize buffer-mode storage existence bookkeeping: 同模块 (buffer-mode + HiCache) 性能优化, 与本 PR 同属 buffer-mode 预取链路演进。
- PR #39318 [HiCache] Attempt-scoped CacheRequestHandle: 评论中明确提到 rebase 依赖: 本 PR 管线、缓存预取状态、锚点锁与 staged holds 均按 CacheRequestHandle 键控。
- PR #38483 [HiCache] Buffer pipeline anchor release: rebase 时采用了上游 #38483 的 buffer-pipeline anchor 释放语义, 与之等价。
- PR #38192 [Scheduler] SWA req ring exact-fit admission gate: PR body 专门提到 `_swa_admission_gate` 保留了 #38192 引入的 `_swa_req_ring` 精确匹配比较。
- PR #39108 [Router] Honor KV-event storage tiers in the cache-aware tree (1/4): 同一 HiCache 功能脉线: Router 前缀树按存储层级记录持有, 与本 PR 的 staged prefetch 物化策略互补。

- PR #39516 [Fix] HiCache startup ImportError on the pinned kernel wheel: 同模块稳定性修复，共同支撑 HiCache 在多种平台（AMD）上的可用性。