

PR #39567 完整报告

sgl-project/sglang

[HiCache] Forward prefix metadata to v2 storage calls

合并时间: 2026-09-16 14:05

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39567>

执行摘要

- 一句话: HiCache v2 调用补齐 prefix 元数据
- 推荐动作: 值得花 10 分钟精读, 尤其是三处细节: 一是 None 与 [] 的三态语义如何影响累积起点; 二是「非原地拼接」与 HiCacheStorageExtraInfo(prefix_keys=...) 共享 list 引用之间的隐式耦合, 读懂这一点才能理解为什么 _page_transfer_sidecar 还能拿到原始前缀; 三是 _page_transfer 先读 KV、_page_transfer_sidecar 后读 sidecar 的顺序依赖。合入后的建议: 补一个单元测试覆盖 prefix_keys 为 None / [] / 非空三种输入下 v2 后端实际收到的 extra_info, 并在累积处加一句注释固化「禁止改回原地 +=」的约束; 同时确认失败的那条 AMD ROCm 10 CI 与本改动无关。

功能与动机

PR body 明确指出问题: "With `hicache_storage_pass_prefix_keys` enabled, KV v1 calls receive prefix metadata, but the controller's v2 sidecar reads and writes omit it. Backends therefore lose prefix context for these transfers." 也就是说 v1 与 v2 两条存储调用链在 prefix 上下文上不一致, 后端拿不到前缀信息, 无法据此做命中判定或分层放置。修复目标是让 v2 调用与 v1 调用在同一个 batch 内看到同一份前缀上下文, 并且不破坏 hybrid 侧车读取依赖的原始前缀。

实现拆解

1. 统一 prefix 元数据的「三态」语义 (python/sglang/srt/managers/cache_controller.py): _page_transfer、_storage_hit_query、_page_backup 三处把 if prefix_keys and len(prefix_keys) > 0: prefix_keys += batch_hashes 改为 if prefix_keys is not None: prefix_keys = prefix_keys + batch_hashes。原因有两点: 一是 None (未启用特性) 与 [] (已启用、当前请求暂无前置命中) 必须区分, 否则空列表永远无法开始累积; 二是原地 += 会改写 extra_info.prefix_keys 引用的同一个 list 对象, 等于回溯修改已经交给后端的元数据。
2. 让 KV 派生池的 v2 读取拿到 batch 上下文 (CacheController._page_transfer_kv_batch): self.storage_backend.batch_get_v2(current_kv_derived_transfers) 改为 batch_get_v2(current_kv_derived_transfers, extra_info=extra_info), 使同一 batch 内 KV 池 (走 page_get_func) 与 indexer 等 KV 派生池的前端上下文保持一致; 命中页数仍按 min([kv_hits, *sidecar_hits.values()]) 收敛。

3. 补齐 hybrid 控制器的 v2 读写元数据 (python/sglang/srt/mem_cache/hybrid_cache/hybrid_cache_controller.py) : `_page_transfer_sidecar` 中构造 `HiCacheStorageExtraInfo(prefix_keys=operation.prefix_keys)` 后传给 `batch_get_v2(transfers_nonkv, extra_info=extra_info)`; `_page_backup` 中构造同样的 `extra_info` 传给 `batch_set_v2(backup_transfers, extra_info=extra_info)`。前缀按 operation 级下发, 因为尾部池 (SWA、Mamba 等) 可能只覆盖 operation 页面的一部分, PR body 也明确不引入 per-pool 前驱元数据。
4. 消除 `.copy()` 依赖: `_storage_hit_query` 原本写作 `prefix_keys = operation.prefix_keys.copy() if operation.prefix_keys else None`, 现在直接引用 `operation.prefix_keys`, 隔离性由第 1 步的非原地拼接承担。这也意味着累积逻辑的对象来源更清晰: `_page_transfer` 与 `_page_transfer_sidecar` 的先后顺序决定了 hybrid sidecar 读到的仍是原始前缀。
5. 测试与配置配套: 本次未新增或修改任何测试文件; `hicache_storage_pass_prefix_keys` 仍是 opt-in 开关, 未开启时 `prefix_keys is None` 会让所有新分支短路, 行为与改动前一致。PR body 说明本地无 PyTorch, 「Runtime tests have not been run」, Checklist 中 Runtime regression tests 未勾选, 最终依赖 run-ci / run-ci-extra 流水线兜底。

关键文件:

- python/sglang/srt/managers/cache_controller.py (模块 缓存控制器; 类别 source; 类型 core-logic; 符号 `_page_transfer`, `_page_transfer_kv_batch`, `_storage_hit_query`, `_page_backup`) : 核心修复文件: `_page_transfer` / `_page_backup` / `_storage_hit_query` 统一 prefix 三态语义并改为非原地累积, `_page_transfer_kv_batch` 把 `extra_info` 透传给 `batch_get_v2`, 是 v2 元数据缺口的正向修复点。
- python/sglang/srt/mem_cache/hybrid_cache/hybrid_cache_controller.py (模块 混合缓存; 类别 source; 类型 data-contract; 符号 `_page_transfer_sidecar`, `_page_backup`) : hybrid 控制器侧的元数据补齐: `_page_transfer_sidecar` 与 `_page_backup` 分别给 `batch_get_v2` / `batch_set_v2` 补上 `HiCacheStorageExtraInfo`, 也是依赖父类非原地累积才能读到原始前缀的地方。

关键符号: `CacheController._page_transfer`, `CacheController._page_transfer_kv_batch`, `CacheController._storage_hit_query`, `CacheController._page_backup`, `HybridCacheController._page_transfer_sidecar`, `HybridCacheController._page_backup`

关键源码片段

python/sglang/srt/managers/cache_controller.py

核心修复文件: `_page_transfer` / `_page_backup` / `_storage_hit_query` 统一 prefix 三态语义并改为非原地累积, `_page_transfer_kv_batch` 把 `extra_info` 透传给 `batch_get_v2`, 是 v2 元数据缺口的正向修复点。

```
# python/sglang/srt/managers/cache_controller.py
```

```
def _page_transfer(self, operation: PrefetchOperation) -> int:
    """按 STORAGE_BATCH_SIZE 分批把 KV 从 L3 读回 host。
```

这里维护的 prefix_keys 是发给存储后端的“前缀上下文”，它有三种语义：

None -> 未开启 hicache_storage_pass_prefix_keys，后端不应收到 prefix 元数据

[] -> 已开启但本请求没有前置命中，后续 batch 需要逐步累积

非空 -> 已有前置哈希，直接把整段前缀当作第 0 个 batch 的上下文

"""

不再做 .copy(): 隔离性交给下面的非原地拼接来保证

prefix_keys = operation.prefix_keys

all_success = True

completed_pages = 0

for i in range(0, len(operation.hash_value), STORAGE_BATCH_SIZE):

if all_success and operation.is_terminated():

all_success = False

if all_success:

batch_hashes = operation.hash_value[i : i + STORAGE_BATCH_SIZE]

batch_host_indices = operation.host_indices[

i * self.page_size : (i + len(batch_hashes)) * self.page_size

]

本 batch 的元数据快照：KV 池与 KV 派生池共用同一份

extra_info = HiCacheStorageExtraInfo(prefix_keys=prefix_keys)

hit_pages = self._page_transfer_kv_batch(

operation,

batch_hashes,

batch_host_indices,

extra_info,

kv_derived_transfers,

)

if hit_pages != len(batch_hashes):

all_success = False

本次修复的核心：重新绑定而非原地 +=，原因有三

1. extra_info.prefix_keys 持有同一个 list 引用，原地追加会“回溯改写”

已经交给后端的元数据；

2. operation.prefix_keys 必须保持原始值，供 hybrid sidecar 读复用；

3. is not None 让 enabled-but-empty ([]) 也能累积，与“未启用”区分开。

if prefix_keys is not None:

prefix_keys = prefix_keys + batch_hashes

completed_pages += hit_pages

每个 batch 无论成败都要投递一个 PrefetchAck，因为 prefetch_sync_thread

会在各 rank 上对结果做 all_reduce，数量必须严格对齐

self.prefetch_sync_queue.put(

PrefetchAck(

rid=operation.request_id,

completed_tokens=completed_pages * self.page_size,

operation=operation,

)

)

return completed_pages

```

def _page_transfer_kv_batch(
    self,
    operation: PrefetchOperation,
    batch_hashes: List[str],
    batch_host_indices: torch.Tensor,
    extra_info: HiCacheStorageExtraInfo,
    kv_derived_transfers: List[PoolTransfer],
) -> int:
    """读一个 batch 的 KV 与 KV 派生 sidecar (如 indexer pool) , 命中页数取最小值。"""
    # KV 池走 v1 风格的 page_get_func, 本身已经带 prefix 元数据
    kv_hits = self.page_get_func(
        operation, batch_hashes, batch_host_indices, extra_info
    )

    sidecar_hits: dict[str, int] = {}
    if len(kv_derived_transfers) > 0:
        current_kv_derived_transfers = [
            PoolTransfer(
                name=transfer.name,
                host_indices=batch_host_indices,
                keys=batch_hashes,
            )
            for transfer in kv_derived_transfers
        ]
        # 修复点: v2 读取此前丢掉了 prefix 元数据, 导致同一个 batch 内
        # KV 池与 KV 派生池拿到的前端上下文不一致; 现在透传同一个 extra_info。
        sidecar_results = self.storage_backend.batch_get_v2(
            current_kv_derived_transfers, extra_info=extra_info
        )
        sidecar_hits = count_pool_hits(sidecar_results)

    # 两侧命中数不一致时收敛到最小值, 避免读回不完整的页面
    return min([kv_hits, *sidecar_hits.values()])

```

python/sglang/srt/mem_cache/hybrid_cache/hybrid_cache_controller.py

hybrid 控制器侧的元数据补齐: `_page_transfer_sidecar` 与 `_page_backup` 分别给 `batch_get_v2` / `batch_set_v2` 补上 `HiCacheStorageExtraInfo`, 也是依赖父类非原地累积才能读到原始前缀的地方。

python/sglang/srt/mem_cache/hybrid_cache/hybrid_cache_controller.py

```

class HybridCacheController(CacheController):
    def _page_transfer(self, operation: PrefetchOperation) -> bool:
        # 顺序很关键: 先由父类完成 KV + KV 派生池的读取, 其命中页数决定本轮实际可用页数
        kv_completed_pages = super()._page_transfer(operation)
        # 再读非 KV 派生的 sidecar (SWA、Mamba 等), 它使用 operation 级的原始前缀
        self._page_transfer_sidecar(operation, kv_completed_pages)

    def _page_transfer_sidecar(

```

```

        self, operation: PrefetchOperation, kv_completed_pages: int
    ) -> None:
        if operation.pool_transfers is None:
            return

        # 只有 KV 完整命中时才继续读额外池; KV 提前终止 (IO 失败、超时、TP 不一致)
        # 时整段跳过, 避免数据错位
        pool_hits: dict[str, int] = {}
        if not operation.is_terminated() and kv_completed_pages == len(
            operation.hash_value
        ):
            # KV 派生池在父类 _page_transfer_kv_batch 中已处理, 这里只处理非 KV 派生池
            transfers_nonkv = [
                transfer
                for transfer in operation.pool_transfers
                if transfer.indices_from_pool != PoolName.KV
            ]
            self._sync_trailing_keys(
                transfers_nonkv, operation.hash_value, kv_completed_pages
            )
            self._resolve_sidecar_nonkv_derived_pool_transfers(operation)
            # 修复点一: sidecar 读取补上 prefix 元数据。这里之所以还能拿到“原始”前缀,
            # 正是因为父类 _page_transfer 改成了 prefix_keys = prefix_keys + batch_hashes
            # 的非原地累积, 不再污染 operation.prefix_keys。
            extra_info = HiCacheStorageExtraInfo(prefix_keys=operation.prefix_keys)
            results = self.storage_backend.batch_get_v2(
                transfers_nonkv, extra_info=extra_info
            )
            pool_hits = count_pool_hits(results)
        # 即使 operation 已被调度线程取消, 也必须投递 PrefetchAck,
        # 保证 prefetch_sync_thread 在各 rank 上的 all_reduce 次数一致
        self.prefetch_sync_queue.put(
            PrefetchAck(
                rid=operation.request_id,
                operation=operation,
                pool_hits=pool_hits,
            )
        )

def _page_backup(self, operation):
    # MLA KV 在 TP 之间是副本, 只由 TP0 写; rank 分片的 sidecar 则每个 TP rank 都要写
    backup_transfers = [
        transfer
        for transfer in operation.pool_transfers or []
        if self.should_backup(transfer)
    ]

    if backup_transfers:
        self._resolve_sidecar_kv_derived_pool_transfers(operation)

```

```

self._resolve_sidecar_nonkv_derived_pool_transfers(operation)
# 修复点二：写入侧同样带上 operation 级 prefix 元数据。
# 注意这是 operation 级（未逐 batch 累积）的上下文：尾部池可能只覆盖
# operation 页面的一部分，因此本 PR 明确不引入 per-pool 前驱元数据。
extra_info = HiCacheStorageExtraInfo(prefix_keys=operation.prefix_keys)
results = self.storage_backend.batch_set_v2(
    backup_transfers, extra_info=extra_info
)
pool_hits = count_pool_hits(results)
operation.pool_storage_result.update_extra_pool_hit_pages(pool_hits)

# KV 主池仍走父类的逐 batch 备份路径（其内部会做 prefix 累积）
if not self.backup_skip:
    super()._page_backup(operation)
# 其余分支负责在 backup_skip 下把 sidecar 结果折算成 completed_tokens

```

评论区精华

Review 讨论极其稀薄：只有 hzh0425 一条 **APPROVED**，没有正文和任何行内评论（`review_comments_count = 0`，`comments_count = 0`）。最新 commit 的 message 带有 **Co-authored-by: Zhangheng <hzh0425@apache.org>**，说明改动更像结对提交，prefix 三态语义与 v2 后端契约大概率在提交前就已对齐，因此没有再落文字讨论。值得记录的「未解决项」来自 PR body 与 Checklist 而非评论区：作者自己标明未跑运行时回归测试，并在草稿状态下等待运行时验证；PR body 还主动圈定了范围边界——不引入 per-pool 前驱元数据、不引入 session identity、不涉及后端分层策略，Mooncake direct-linker 路径不在本次控制器改动内。

- 唯一评审为 APPROVED，无行内技术讨论 (other)：没有争议点，但也没有留下对 None / [] 语义和 v2 后端契约的书面确认，后续维护者只能从代码与 PR body 反推约定。
- 运行时回归测试未执行即获批 (testing)：最终在缺少运行时用例的情况下合并，回归风险主要由 hicache_storage_pass_prefix_keys 默认关闭与现有 CI 兜底，hybrid sidecar 路径的元数据正确性没有专门的验证用例。

风险与影响

- 风险：1) 测试覆盖缺口：本次 0 个测试文件改动，而 hybrid sidecar (SWA / Mamba) 与 KV 派生池的 v2 路径恰恰是最需要多池对齐验证的地方，PR 自查也承认运行时测试未执行，仅 Base / Extra 两条 CI 通过（AMD ROCm 10 那条为失败，PR 中未给出解释）。2) 语义变更面：prefix_keys 为 [] 时现在会开始累积本 batch 哈希，这让 _storage_hit_query 里 v1 的 batch_exists 也会从「永远拿到 None」变成「拿到累积前缀」，若某个后端把 prefix 用于祖先链校验或分层放置，命中结果可能与改动前不同——这是本次改动中离 v1 路径最近的一处回归面。3) 性能：PR body 承认 "Prefix accumulation creates a new list per batch when prefix metadata is enabled"，每 batch 重建列表使拷贝成本随 batch 数近似线性增长（前缀长度本身也在增长），仅在 opt-in 且长序列时可见。4) 接口兼容：batch_get_v2 / batch_set_v2 新增 extra_info 关键字参数，若某个 v2 后端实现签名不接受该参数会直接抛 TypeError；未开启特性时传的是

HiCacheStorageExtraInfo(prefix_keys=None), 需要后端能容忍 None。5) 实现脆弱性: 非原地拼接一旦被后续改动改回 +=, 会同时污染 extra_info 已交给后端的元数据与 operation.prefix_keys, 而这一约束目前只存在于 PR 描述里, 缺少断言或注释型防护。

- 影响: 用户侧: 仅在开启 hicache_storage_pass_prefix_keys 且使用 v2 (hybrid / sidecar 池) 后端时, L3 读写的前缀上下文才恢复完整, 命中行为与缓存分层信息更接近 v1 语义; 未开启该开关时所有新分支短路, 行为与改动前完全一致, 风险基本为 0。系统侧: 影响 HiCache L3 元数据契约在 v1 与 v2 两条调用链上的一致性, 以及 KV 池与 KV 派生池在同一个 batch 内的命中对齐 (min 收敛逻辑不变); 同时修正了 operation.prefix_keys 被原地改写导致 hybrid 侧车读到被污染前缀的隐患。团队侧: 这是 HiCache v1/v2 元数据对齐的一条小切口, 后续若引入 per-pool 前驱元数据或 Mooncake direct-linker 路径补齐, 需要沿用同一套三态语义与不可变累积约定。
- 风险标记: 缺少测试覆盖, 运行时验证缺口, opt-in 路径语义变更, v2 后端接口签名依赖, CI 存在未解释失败

关联脉络

- PR #39280 [HiCache] Label radix-cache metrics per rank and split the "shrunk" prefetch reason: 同属 HiCache 前缀缓存与预取链路, 改动 base_prefix_cache.py、unified_radix_cache.py 等相邻模块, 是同一时期 HiCache 元数据 / 指标归因的并行工作。
- PR #38402 fix(npu): fix hybrid KV transfer with PP prefill in PD disaggregation: 同样触及 hybrid KV 传输与多池数据对齐 (memory_pool.py、disaggregation 连接层), 与本 PR 的 hybrid sidecar 完整命中判定属于同一条正确性脉络。
- PR #39111 [Router] Add a KV storage-tier coverage row to the Grafana dashboard (4/4): 主题同为 HiCache 分层存储 (storage tier), 可作为 v2 元数据补齐后在可观测性侧的配套, 帮助验证前缀元数据生效后的存储层命中表现。