

PR #38486 完整报告

sgl-project/sglang

[HiCache] Publish a host store event for storage-prefetch refills

合并时间: 2026-09-13 18:29

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38486>

执行摘要

- 一句话: HiCache 补发 host 存储事件, 修复路由器 L2 索引缺失
- 推荐动作: 值得精读, 尤其是负责路由、KV 事件或 unified radix cache 的工程师。理由: diff 极小 (源码 +2 行) 但语义跨度大, 是一个典型的“可观测性缺口会伪装成路由性能问题”的案例, PR body 的推理链 (为什么问题不立即暴露、demote 分支为何不补发事件、为什么不对称的 remove 让问题可见) 本身就有很高的阅读价值。读的时候重点关注三件事: 一是 insert_host 中新增 record_store 的插入位置与两条静默边界 (完整命中、write-through 丢弃) 是否覆盖了所有 host_value 物化路径; 二是 Python 与 Rust 两套实现的同步策略, 思考是否有更好的机制 (例如把发布收敛到 shared recorder / 在 C++/Rust 层统一) 而不是靠人工保持一行对一行; 三是考虑是否需要一条端到端或回放测试来锁定“L3 refill 后 router 能看到 host_pinned_blocks”的行为, 避免未来回归。如果只想过一遍结论: 可以把 Jialin 的 “Looks safe and lightweight enough to land.” 作为合入依据, 但仍建议核对你们下游 router 对重复 BlockStored(CPU_PINNED) 的幂等处理。

功能与动机

PR body 明确指出事件发布的不对称: SGLang 为 L1 发布 BlockStored(GPU), 为 L2 发布 BlockStored(CPU_PINNED), 但 L2 只有 write-through ack 这一个发布者; host 内存还有第二个生产者——storage (L3) prefetch refill, 它通过 tree_core.insert_host() 直接插入 host 树, 两套 unified tree core 在该路径上都不发事件, 导致这些 block 永远进不了 cache-aware router 的低层级索引。作者把问题上升为不变式: “every path that turns a node's host_value from None into a value should publish BlockStored(CPU_PINNED). insert_host was the only exception.”, 并指出 HiRadixCache._insert_helper_host 已经发布同样的事件, 两套 hicache 实现当前并不一致。缺失事件不会立刻暴露——refill 后 device 副本会被加载并被 BlockStored(GPU) 覆盖; 只有当 device 副本被 demote (evict_device_leaf 走 _demote 分支, 节点已 backedup 所以没有写回、也没有迟到事件) 而 host 副本仍存活时, 问题才显现: router 认为 worker 一无所有, 而实际上一次 H2D 拷贝就能服务请求——这正是 L2 hit weight 存在的意义。观测证据: 8 节点部署上的 Dynamo KV router, worker 从 Mooncake L3 拉回 169,472 tokens, 而该轮运行的全部 344 次路由决策中 host_pinned_blocks=0。此外移除侧是无条件发布的, _evict_host_leaf 与 _release_full_host_duplicate 会对同一批节点发 BlockRemoved(CPU_PINNED), 消费者可能观察到它从未见过的 stored 事件的 remove。

实现拆解

1. Python 侧发布点 (`python/sglang/srt/mem_cache/unified_cache/unified_tree_core.py`, `insert_host()`) : 在 `new_node` 建好、`host_value` 从 `None` 物化、`_update_evictable_leaf_sets()` 更新完可淘汰叶子集合、并写入 `result.inserted_host_node = new_node.id` 之后, 新增 `self.kv_events.record_store(new_node, medium=StorageMedium.CPU)` 再 `return result` (源码仅 +1 行)。选择这个位置是因为它是“host 值真正落地”的唯一分支, 并且此时节点的 `hash_value`、`key` 都已就位, 事件里能带上正确的 block hashes 与 `token_ids`。
2. Rust 侧同步 (`rust/sglang-radix-tree/src/unified_tree_core.rs`, `try_insert_host_in_namespace()`) : 在同样的收尾位置插入 `self.record_store_event(new_node_id, StorageMedium::Cpu)` 后 `Ok(result)` (+1 行)。两侧语义必须一致, 否则 Python tree core 与 Rust tree core 在 KV 事件上会出现行为分叉, 而 SGLang 的 unified radix cache 允许二者替换使用。
3. 保留的静默边界 (关键设计决策) :
 - 完整命中早返回 (`len(key) == 0`, 节点已存在、没有新内容物化) 不发布事件;
 - write-through (非 write-back) 策略下、父节点尚未 backup 时的 refill 被丢弃 (`result.host_insert_dropped = True`) 不发布事件——因为并没有新的 host 数据产生。PR body 明确把这两点列为 out of scope 且“already correct today”; 另说明 `--hcache-host-memory-mode buffer_only` 不走 `insert_host`, 同样不在覆盖范围。
4. 测试配套 (双侧) :
 - Rust: `rust/sglang-radix-tree/src/tests/unified_tree_core.rs` 新增 `insert_host_publishes_a_host_store_event` (+34 行), 用 `events_core(2)` 构造带事件记录的核心, 断言 `take_events()` 恰好得到一条 `KvCacheEvent::BlockStored { medium: StorageMedium::Cpu, block_size: 2, parent_block_hash: None, ... }`。
 - Python: `test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py` 新增 `test_hicache_storage_prefetch_publishes_host_only_suffix` (+30 行), 用 `enable_kv_cache_events=True` 的 fixture, 直接调用 `cache.tree_core.insert_host(...)` 模拟 L3→L2 refill, 断言 recorder 把父子相连的页合并成 1 条 `BlockStored(CPU_PINNED)`, `token_ids`、`block_size`、`hash` 列表与 `parent_block_hash is None` 全部对齐。
5. 演进与评审修订: 6 个 commit 中有 3 个是 main 合并, 实质修改为初版 + 两次修订——`9e0f2f0` 修复 Rust 测试编译错误 (`insert_host` 返回 `Result<InsertResult, TreeCoreRuntimeError>`, 需 `.expect("live test node")`, 否则读 `host_insert_dropped` 直接触发 E0609), `76bdc87` 采纳 review nit, 改用 `get_hash_str(..., page_size=cache.page_size)` 取代手写分页循环。
6. 性能说明: 事件关闭时两个 recorder 立即返回, 热路径零成本; 开启时每次 refill 增加一条 `BlockStored`, recorder 会合并 parent-linked 页, N 页 refill 只产生一条携带 N 个 block hash 的事件。

关键文件:

- `python/sglang/srt/mem_cache/unified_cache/unified_tree_core.py` (模块 统一树核心; 类别 source; 类型 core-logic; 符号 `insert_host`, `record_store`) : Python 侧唯一的功能改动点: `insert_host()` 在 `host_value` 物化并挂到父节点后新增

`self.kv_events.record_store(new_node, medium=StorageMedium.CPU)`, 是 cache-aware router 能看到 storage-prefetch refill 的 L2 数据的直接原因。

- `rust/sglang-radix-tree/src/unified_tree_core.rs` (模块 前缀树底层; 类别 source; 类型 core-logic; 符号 `try_insert_host_in_namespace`, `record_store_event`) : Rust 侧对等实现: `try_insert_host_in_namespace()` 在同样的物化收尾点调用 `self.record_store_event(new_node_id, StorageMedium::Cpu)`, 保证 unified radix cache 在两套 tree core 上发布一致的 KV 事件。
- `test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py` (模块 缓存单元测试; 类别 test; 类型 test-coverage; 符号 `test_hicache_storage_prefetch_publishes_host_only_suffix`) : Python 侧回归测试: 新增 `test_hicache_storage_prefetch_publishes_host_only_suffix`, 直接调用 `tree_core.insert_host()` 模拟 L3→L2 refill, 断言 recorder 合并成单条 `BlockStored(CPU_PINNED)`; 同时是 review nit (hash 计算重复分页) 的落点。
- `rust/sglang-radix-tree/src/tests/unified_tree_core.rs` (模块 树核心测试; 类别 test; 类型 test-coverage; 符号 `insert_host_publishes_a_host_store_event`) : Rust 侧回归测试: `insert_host_publishes_a_host_store_event` 用 `events_core(2)` 断言 `take_events()` 恰好产出单条 `KvCacheEvent::BlockStored { medium: StorageMedium::Cpu, block_size: 2, parent_block_hash: None }`; 该测试初版因 `insert_host` 返回 `Result` 未 `unwrap` 而产生 E0609 编译错误, 是 CI 首轮红灯的直接原因。

关键符号: `insert_host`, `try_insert_host_in_namespace`, `record_store`, `record_store_event`, `insert_host_publishes_a_host_store_event`, `test_hicache_storage_prefetch_publishes_host_only_suffix`, `_update_evictable_leaf_sets`, `_evict_host_leaf`

关键源码片段

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

Python 侧唯一的功能改动点: `insert_host()` 在 `host_value` 物化并挂到父节点后新增 `self.kv_events.record_store(new_node, medium=StorageMedium.CPU)`, 是 cache-aware router 能看到 storage-prefetch refill 的 L2 数据的直接原因。

```
# python/sglang/srt/mem_cache/unified_cache/unified_tree_core.py
# insert_host() 的核心分支 (前面的 prefix 匹配与 result 组装逻辑已省略):
# 只有走到“新建 host 节点”的这条分支, host_value 才会从 None 变成实值。
def insert_host(self, node_id, key, host_value, hash_value):
    # ... 省略前缀匹配, 得到 matched_length / total_len 并组装 result ...

    # 边界 1: 完整命中 (len(key) == 0)。节点已存在, 没有新内容物化, 因此保持静默。
    if len(key) == 0:
        if (
            node is not self.root_node
            and node.component_data[BASE_COMPONENT_TYPE].host_value is not None
        ):
            result.inserted_host_node = node.id
    return result
```

```

# 边界 2: write-through (非 write-back) 策略下, 父节点尚未 backup 的 refill 会被丢弃。
# 此时并没有产生新的 host 数据, 同样不应发布 BlockStored(CPU_PINNED)。
if node is not self.root_node and not node.backuped and not self.is_write_back:
    logger.info(
        "HiCache prefetch dropped %d-token refill under un-backed-up node %d",
        len(host_value),
        node.id,
    )
    result.host_insert_dropped = True
    return result

# 走到这里, host_value 一定会落地: 这是本 PR 唯一新增的事件发布点。
new_node = self._new_node(priority=node.priority)
new_node.parent = node
new_node.key = key
new_node.hash_value = hash_value
new_node.component_data[BASE_COMPONENT_TYPE].host_value = host_value.clone()
node.children[child_key] = new_node
self._update_evictable_leaf_sets(new_node)
self._update_evictable_leaf_sets(node)
result.inserted_host_node = new_node.id

# 新增: storage prefetch refill 直接从 L3 落进 host 池, 没有 write-through ack 替它发布,
# 只能由 host insert 自己上报, 否则 router 的 L2 索引里永远看不到这些 block。
self.kv_events.record_store(new_node, medium=StorageMedium.CPU)
return result

```

rust/sclang-radix-tree/src/unified_tree_core.rs

Rust 侧对等实现: `try_insert_host_in_namespace()` 在同样的物化收尾点调用 `self.record_store_event(new_node_id, StorageMedium::Cpu)`, 保证 unified radix cache 在两套 tree core 上发布一致的 KV 事件。

```

// rust/sclang-radix-tree/src/unified_tree_core.rs
// try_insert_host_in_namespace() 中 host 节点落地的收尾段 (前缀匹配部分已省略)。
{
    // 匹配到的头部被丢弃, 未匹配的后缀挪进一个大小合适的 list。
    let mut hash_value = hash_value;
    let suffix = hash_value.split_off(matched_length / self.page_size);
    self.arena.node_mut(new_node_id).hash_value = Some(suffix);
}

// host_value 只写入未匹配的后缀: 这里正是 None -> Some(...) 的物化点。
self.arena.set_host_value(
    new_node_id,
    FULL,
    host_value
        .narrow(0, matched_length as i64, (total_len - matched_length) as i64)
        .copy(),
);

```

```

let child_map_key = self.arena.node(new_node_id).key.child_key(self.page_size);
let displaced = self
    .arena
    .insert_child_edge(node_id, child_map_key, new_node_id);
assert!(
    displaced.is_none(),
    "insert_host: parent {node_id} already has a child on the new node's page"
);

self.update_evictable_leaf_sets_(new_node_id);
self.update_evictable_leaf_sets_(node_id);
result.inserted_host_node = Some(self.arena.node(new_node_id).id);

// 新增：与 Python 侧 insert_host() 语义一致——host 值一旦被物化就发布 BlockStored(CPU_
PINNED),
// 避免两套 tree core 在 KV 事件上出现行为分叉。
self.record_store_event_(new_node_id, StorageMedium::Cpu);
Ok(result)

```

test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

Python 侧回归测试：新增 `test_hicache_storage_prefetch_publishes_host_only_suffix`，直接调用 `tree_core.insert_host()` 模拟 L3→L2 refill，断言 recorder 合并成单条 BlockStored(CPU_PINNED)；同时是 review nit（hash 计算重复分页）的落点。

```

# test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py
def test_hicache_storage_prefetch_publishes_host_only_suffix(self):
    """A storage-prefetch refill has no write-through ack to publish it."""
    from sglang.srt.mem_cache.utils import get_hash_str, hash_str_to_int64

    cache, _, _ = build_fixture(self.cfg, enable_kv_cache_events=True)
    self._init_hicache(cache)
    cache.take_events() # 清掉 reset / init 阶段的事件噪声。

    tokens = [1, 2, 7, 8]
    # get_hash_str 传入 page_size 时已经按页切分，等价于 compute_node_hash_values;
    # 这里正是 review nit 的修订点（原实现手写了累积 hash 循环）。
    hash_values = get_hash_str(array("q", tokens), None, page_size=cache.page_size)
    # 绕过 device 侧，直接走 insert_host，模拟 L3 -> L2 的 storage prefetch refill。
    result = cache.tree_core.insert_host(
        cache.root_node_handle(),
        RadixKey(array("q", tokens)),
        torch.tensor([100, 101, 102, 103], dtype=torch.int64),
        hash_values,
    )
    self.assertFalse(result.host_insert_dropped)
    self.assertIsNotNone(result.inserted_host_node)

    # recorder 会把父子相连的多页合并成一条事件，因此这里只应看到 1 条 BlockStored(CPU_
    PINNED)。

```

```

stored_cpu = self._stored_events(cache, StorageMedium.CPU)
self.assertEqual(len(stored_cpu), 1)
self.assertEqual(list(stored_cpu[0].token_ids), tokens)
self.assertEqual(stored_cpu[0].block_size, cache.page_size)
self.assertEqual(
    self._event_hashes(stored_cpu),
    [hash_str_to_int64(value) for value in hash_values],
)
self.assertIsNone(stored_cpu[0].parent_block_hash)

```

评论区精华

整体是低争议、快速落地的小 diff，讨论集中在两处：

1. 测试里的 hash 计算冗余（唯一一条代码级 review 评论）：alphabetc1 在 `test_unified_radix_cache_unittest.py` 上给出 nit——“get_hash_str already pages when given page_size, same as compute_node_hash_values: `hash_values = get_hash_str(array("q", tokens), None, page_size=cache.page_size)`”，指出原实现手写了 `for start in range(0, len(tokens), cache.page_size)` 的累积 hash 循环。作者回复 “Thx, fixed in the latest push.”，并在 commit `76bdc874`（“…drop the manual loop. (review nit)”）中落地。
2. 是否可安全合入：Jialin 以 APPROVED 给出结论 “Looks safe and lightweight enough to land.”，主要依据是事件只在启用时才有开销、且不触碰模型输出（PR body 的 Accuracy Tests 一节写 “Not applicable; this only adds a KV cache event and does not touch model outputs”）。
3. CI 红灯归属（issue 评论线，7 条）：CI 出现过 Rust 新测试的编译错误，作者在 commit `9e0f2f0` 修复并本地用 pre-commit 的 clippy + rustfmt 验证，同时说明剩余 `call-gate` 红是因为缺少 `run-ci` 标签（无权自加），随后 Jialin `/tag-and-rerun-ci`、alphabetc1 `/rerun-test -c` 与 `/rerun-failed-ci`。最终作者逐条归因为存量问题（AMD stage-b rocm10 的 LoRA ROUGE-L 精度 flake 与 Triton deterministic test、MLX test_scheduler_mixin、Xeon CPU build），alphabetc1 用截图确认 “The failing CI is unrelated to the PR, and it also fails on main”，因此该 PR 最终带 `run-ci`、`bypass-fastfail` 标签合入。

未解决的疑虑：作者自述 workspace 级别的 clippy 未在本地执行（“The workspace-wide clippy run was not executed locally.”），属于已知的验证边界而非阻塞项。

- 测试中的 hash 计算是否重复分页 (style): 作者回复 “Thx, fixed in the latest push.”，并在 commit `76bdc874`（“[HiCache] Use get_hash_str's page_size in the host store event test”）中删除手写循环，改用 `get_hash_str(..., page_size=cache.page_size)`。
- 只加事件、不动主逻辑是否可安全合入 (design): 审阅者认可该变更风险与开销均可接受并批准合入；未提出额外的性能或设计修改要求。
- Rust 测试初版编译失败 (E0609) (correctness): commit `9e0f2f0` 用 `.expect("live test node")` 解包，与周围既有的 `insert_host` 测试写法保持一致；作者同时说明 workspace 级别的 clippy 未在本地跑过（“The workspace-wide clippy run was not executed locally.”）。

- CI 红灯是否由本 PR 引入 (testing): 存量失败被排除, PR 带 run-ci 与 bypass-fastfail 标签合入; /rerun-test -c 的 1-gpu-5090 用例 (registered/unit/mem_cache/test_unified_radix_cache_unittest.py) 通过。

风险与影响

- 风险:

1. 事件可能重复发布 (主要风险): storage prefetch refill 落进 host 树后, 若同一批 block 随后又因为 write-through ack 或 HiRadixCache._insert_helper_host 路径再发一次 BlockStored(CPU_PINNED), 消费端 (cache-aware router) 必须对同一 block_hash 的重复 store 幂等。本 PR 只保证“完整命中早返回”与“refill 被丢弃”两条路径静默, 未在事件层做去重; Python 与 Rust 两条实现的去重行为若有差异, 问题只会多节点路由侧暴露。
 2. 双实现同步属于长期维护负担: insert_host 的语义现在靠 Python 与 Rust 两侧各一行调用维持一致 (record_store / record_store_event_)。未来若新增 host 值物化路径 (新的 component 类型、新的 refill 入口), 很容易再次漏发——PR 作者本人就是用“不变式”来描述这个约束, 说明它缺乏机制性强制。
 3. 测试只覆盖单条路径、缺少端到端验证: Python 测试直接调 tree_core.insert_host(), 没有走完整的 L3→L2 prefetch refill 调度链 (storage backend 与 scheduler 的交互未覆盖); Rust 侧同样是单元级。PR body 中的 169,472 tokens / 344 次决策的真实症状没有对应的集成或回放测试来防止回归。
 4. 性能影响可控但有条件: kv_events 关闭时直返; 开启时每次 refill 多一次 record_store 调用与事件合并开销, 属于可接受量级, 但大规模 L3 命中场景下的事件量增长未做压测。
 5. 兼容性: 变更不涉及协议格式、配置项或模型输出, 但对 BlockStored(CPU_PINNED) 已做去重假设或从未处理过额外 store 事件的下游 router 版本, 需要在升级时重新验证。
 - 影响: 对系统: 修复后 cache-aware router 能正确看到 worker 持有的 L2-only 前缀, 避免本可一次 H2D 拷贝解决的请求退回 L3, 直接减少跨节点存储流量与首 token 延迟; 症状级证据是 8 节点部署上 169,472 tokens 的无效 L3 拉取与 344 次决策中 host_pinned_blocks=0 的索引缺失。对用户: 仅在启用 KV 事件与 L3 storage 后端 (如 Mooncake) 并配合 cache-aware 路由的多节点部署中可感知, 单机、纯 L1/L2 场景无行为差异, 模型输出不受影响。对团队: 确立了“host_value 由 None 物化即发布 BlockStored(CPU_PINNED)”不变式, 意味着后续任何新增 host 插入路径都要在两套 tree core (Python 与 Rust) 中同时补齐事件发布; unified radix cache 的 Python/Rust 对等性成为一个需要持续审计的维度。对下游消费者: _evict_host_leaf / _release_full_host_duplicate 的 BlockRemoved 不再出现“只删未见”的不对称, 路由侧的低层级索引语义变得更自治。
- 风险标记: 核心缓存路径变更, Python/Rust 双实现需同步维护, 同一 block 的存储事件可能重复发布, 测试仅覆盖单元级单条路径, 缺少多节点端到端验证, workspace 级 clippy 未本地验证

关联脉络

- PR #39120 Fix multimodal embedding cache retaining full batches through views: 同样修改 `python/sglang/srt/mem_cache/` 下的缓存生命周期语义（视图导致整批存储被保留），属于同一 `mem_cache` 层的正确性修复线，都强调缓存句柄 / 引用语义的自洽性。
- PR #39038 [Session] Work with PD and Fix empty continuations: 同为“缓存 / 会话状态在 PD 场景下语义不一致”的修复，且改动落在同一测试目录 `test/registered/unit/mem_cache/`（新增 `test_session_token_share_unit.py`），与本 PR 的 Python 单测位置相邻。
- PR #39182 Add a provider hook for prefill-buffer ceilings: 主题上高度呼应：该 PR 用 provider hook 统一两条路径的尺寸计算（tags 含 consistency），本 PR 则通过“不变式 + 双实现同步”统一 Python/Rust 两套 tree core 的事件发布，都体现当前仓库在收敛多实现分叉的努力。