PR #27072 完整报告

sgl-project/sglang

hicache: publish split write-through fragments

合并时间：2026-06-04 05:52

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27072

执行摘要

- 一句话：修复 HiCache 节点分裂时 write-through 事件丢失
- 推荐动作：建议相关人员精读实现，特别是 _replace_pending_write_through_node 中列表替换的逻辑和 _finish_write_through_ack 中对 storage 的持久化方式。设计上通过 publish_nodes 列表而非直接修改 ack 回调，是一种简洁的解耦方案。

功能与动机

在 HiCache 的 write-through 模式下，CPU 事件异步发布。如果节点在提交 host 索引后、DMA 确认到达前被分裂（例如因插入前缀匹配的新序列），则 pending 的 ack 仍指向原节点，只能发布后缀的事件，前缀的事件被静默丢失。该 PR 旨在消除这一竞态条件。

实现拆解

1. 新增节点字段：在 TreeNode 和 UnifiedTreeNode 类中添加 write_through_pending_id 属性，标记节点是否有待处理的 write-through 发布。
2. 扩展字典结构：将 ongoing_write_through 字典的值从 (node, lock_params/bi) 扩展为 (node, lock_params/bi, publish_nodes)，其中 publish_nodes 记录需要发布事件的节点列表，初始为 [node]。
3. 提取三个辅助方法：_track_write_through_node 负责录入待发布节点；_replace_pending_write_through_node 在节点分裂时被调用，用新节点替换待发布列表中的旧节点；_finish_write_through_ack 在 DMA 确认时遍历 publish_nodes 为每个节点生成 BlockStored 事件。
4. 挂接到分裂流程：在 _split_node 中，当分裂的子节点已 backuped（即有 pending write-through）时，调用 _replace_pending_write_through_node。
5. 替换原 write_backup 和 writing_check 中的逻辑：write_backup 中改用 _track_write_through_node；writing_check 中改用 _finish_write_through_ack，并在 enable_storage 情况下对每个 publish 节点执行 write_backup_storage，确保前缀也被持久化。
6. 添加单元测试：新增 test/registered/unit/mem_cache/test_hiradix_cache_unit.py，在 test_unified_radix_cache_unittest.py 中添加新方法，模拟分裂 pending write-through 场景，验证事件发布正确性。

关键文件：

- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 缓存层；类别 source；类型 core-logic；符号 _track_write_through_node, _replace_pending_write_through_node, _finish_write_through_ack）：实现了核心的三个辅助方法，并修改了 _split_node 和 write_backup 等关键路径。
- python/sglang/srt/mem_cache/hiradix_cache.py（模块 缓存层；类别 source；类型 core-logic；符号 _track_write_through_node, _replace_pending_write_through_node, _finish_write_through_ack）：实现了与 unified 版本对应的三个辅助方法，并修改了 write_backup 和 writing_check。
- test/registered/unit/mem_cache/test_hiradix_cache_unit.py（模块 缓存层测试；类别 test；类型 test-coverage；符号 TestHiRadixCacheKVEvents, setUpClass, _build_cache, _insert）：新增完整的测试类，覆盖分裂 pending write-through 的核心场景。
- test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py（模块 缓存层测试；类别 test；类型 test-coverage；符号 test_hicache_split_pending_write_through_publishes_fragments）：添加了类似的测试方法，并额外验证了 enable_storage 场景下 write_backup_storage 的调用次数和参数。
- python/sglang/srt/mem_cache/radix_cache.py（模块 缓存层；类别 source；类型 core-logic）：为 TreeNode 基类添加 write_through_pending_id 字段，被两个子类复用。

关键符号：_track_write_through_node, _replace_pending_write_through_node, _finish_write_through_ack, _split_node, write_backup, writing_check

关键源码片段

python/sglang/srt/mem_cache/unified_radix_cache.py

实现了核心的三个辅助方法，并修改了 _split_node 和 write_backup 等关键路径。

```
def _track_write_through_node(
    self,
    node: UnifiedTreeNode,
    lock_params: Optional[DecLockRefParams],
) -> None:
    # 标记节点有 pending 的 write-through，并将自身作为初始发布节点
    node.write_through_pending_id = node.id
    self.ongoing_write_through[node.id] = (node, lock_params, [node])

def _replace_pending_write_through_node(
    self, old_node: UnifiedTreeNode, new_nodes: list[UnifiedTreeNode]
) -> None:
    # 节点分裂时，将 old_node 从待发布列表中替换为分裂后的新节点
    ack_id = old_node.write_through_pending_id
    if ack_id is None:
        return
    pending = self.ongoing_write_through.get(ack_id)
    if pending is None:
        return
```

```
        lock_node, lock_params, publish_nodes = pending
        updated_nodes = []
        replaced = False
        for node in publish_nodes:
            if node is old_node:
                updated_nodes.extend(new_nodes)
                replaced = True
            else:
                updated_nodes.append(node)
        if not replaced:
            return
        for node in new_nodes:
            node.write_through_pending_id = ack_id
        self.ongoing_write_through[ack_id] = (lock_node, lock_params, updated_nodes)

    def _finish_write_through_ack(self, ack_id: int) -> None:
        # DMA 确认时，为所有待发布节点生成 CPU 存储事件
        lock_node, lock_params, publish_nodes = self.ongoing_write_through.pop(ack_id)
        for node in publish_nodes:
            if node.write_through_pending_id == ack_id:
                node.write_through_pending_id = None
            self._record_store_event(node, medium=StorageMedium.CPU)
        if lock_params is not None:
            self.dec_lock_ref(lock_node, lock_params)
        if self.enable_storage:
            # 对每个碎片都调用 write_backup_storage，确保前缀也被持久化
            for node in publish_nodes:
                self.write_backup_storage(node)
```

test/registered/unit/mem_cache/test_hiradix_cache_unit.py

新增完整的测试类，覆盖分裂 pending write-through 的核心场景。

```
    def test_split_pending_write_through_publishes_fragments(self):
        cache, allocator = self._build_cache()
        cache.take_events()

        self._insert(cache, allocator, [1, 2, 3, 4])
        node = self._leaf_for(cache, [1, 2, 3, 4])
        backed_up = cache.write_backup(node, write_back=True)
        self.assertGreater(backed_up, 0)

        # 分裂节点，此时 write-through DMA 尚未确认
        self._insert(cache, allocator, [1, 2, 5, 6])
        self.assertEqual(self._stored_cpu_events(cache), [])

        cache.writing_check(write_back=True)

        # 两个碎片都应发布事件，且 parentage 正确
        stored_cpu = self._stored_cpu_events(cache)
```

```
        self.assertEqual(
            [list(e.token_ids) for e in stored_cpu],
            [[1, 2], [3, 4]],
        )
        self.assertIsNone(stored_cpu[0].parent_block_hash)
        self.assertEqual(stored_cpu[1].parent_block_hash, stored_cpu[0].block_hashes[0])
```

评论区精华

来自 chatgpt-codex-connector 的 Review 指出（路径 unified_radix_cache.py）：当 enable_storage 开启且节点分裂后，_finish_write_through_ack 仅对 lock_node 调用 write_backup_storage，而 lock_node 此时只持有后缀，前缀的 CPU 事件虽已发布但从未持久化到存储后端，可能导致数据丢失。该问题在第三个提交中修复：改为对 publish_nodes 中所有节点调用 write_backup_storage。

- unified radix cache storage 持久化遗漏前缀 (correctness): 作者在第三个提交中修复：改为对 publish_nodes 中所有节点调用 write_backup_storage。

风险与影响

- 风险：主要风险包括：
  - 潜在死锁或竞态：新增的 write_through_pending_id 和节点列表替换逻辑可能引入新的并发问题，尤其在多线程或分布式环境下需仔细验证。
  - 性能开销：publish_nodes 列表增加了少量内存和遍历开销，但由于 write-through 路径本身涉及 DMA 和 event 读取，额外开销可忽略。
  - 兼容性：ongoing_write_through 字典结构变化，任何外部访问该字典的代码（目前无）需要适配。
  - 测试覆盖：测试仅覆盖了简单的分裂场景，未覆盖嵌套分裂、多节点 pending 等复杂情况。
  - 影响：直接影响所有使用 HiCache write-through 特性的用户和设备，修复了事件丢失的 bug，增强了缓存一致性。对不使用 HiCache 或使用 write-back 策略的用户无影响。系统无需配置变更，升级后自动生效。
  - 风险标记：竞态条件修复，核心数据结构变更，需要验证并发安全

关联脉络

- PR #25991 [HiCache] fix: truncate prefetch key on degraded allocation: 同样影响 HiCache 模块，修复类似的边情况 bug，属于同一子系统演进。