PR #25065 完整报告

sgl-project/sglang

[UnifiedTree] fix: backup SWA-split parent before child under write-through

合并时间：2026-05-25 00:21

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25065

执行摘要

- 一句话：修复 SWA 分裂叶子在 write-through 下丢失备份的 bug
- 推荐动作：值得精读：patch 虽小但修复了 write-through + SWA 路径下缓存一致性 bug，递归备份方案设计简洁，测试完整。关注 write_backup 中递归的边界条件，以及测试对 swa_evicted_seqlen 的构造方式。

功能与动机

修复 HiCache write_through 模式下 SWA 边界分裂叶子被静默丢弃的 bug：当插入跨越 swa_evicted_seqlen 时，产生的 split parent 没有 host_value，导致 child 的 write_backup 被跳过，后续设备驱逐时节点被删除而非降级为主机节点，违反已有不变性“每个备份节点的父节点也必须备份”。

实现拆解

1. 修改 write_backup 方法（python/sglang/srt/mem_cache/unified_radix_cache.py 第 1172 行）：当 write_back == False 且父节点非根且未备份时，不再直接返回 0，而是递归调用 self.write_backup(node.parent)，只有递归备份失败（返回值 ≤ 0）时返回 0。
2. 新增单元测试（test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py 第 1651-1684 行）：test_hicache_write_through_offloads_swa_split_leaf 测试构造一个 SWA 边界分裂场景，设置 write_through_threshold=1，插入后调用 writing_check(write_back=True) 和 evict，断言分裂叶子被驱逐、已备份、且位于可驱逐主机叶子集合中。
3. 递归保证：递归由正常的 write_backup 生命周期管理（锁引用、ongoing_write_through 簿记、主机池记账），无需独立的清理路径；递归深度受 radix 树深度限制；仅当父节点确实缺失主机备份时触发，不会导致已备份祖先的冗余写入。

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 write_backup）：修复核心：在 write_backup 中递归备份未备份的父节点，确保 write-through 不变性。
- test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_hicache_write_through_offloads_swa_split_leaf）：新增测试验证 SWA 边界分裂叶子在 write-through 下能被正确备份和驱逐，是回归

防护的关键。

关键符号：write_backup, test_hicache_write_through_offloads_swa_split_leaf

关键源码片段

python/sglang/srt/mem_cache/unified_radix_cache.py

修复核心：在 write_backup 中递归备份未备份的父节点，确保 write-through 不变性。

```
def write_backup(self, node: UnifiedTreeNode, write_back: bool = False) -> int:
    """Backup a node's data from device to host (D->H)."""
    if self.cache_controller is None:
        return 0

    # Backup invariant (write-through): parent must be backuped first
    if not write_back and (
        node.parent is not self.root_node and not node.parent.backuped
    ):
        # 递归备份父节点，若失败则返回 0
        if self.write_backup(node.parent) <= 0:
            return 0

    device_value = node.component_data[BASE_COMPONENT_TYPE].value
    kv_xfer = PoolTransfer(name=PoolName.KV, device_indices=device_value)

    # Build aux transfers, keyed per component.
    comp_xfers: dict[ComponentType, list] = {}
    for comp in self._components_tuple:
        if comp.component_type == BASE_COMPONENT_TYPE:
            continue
        t = comp.build_hicache_transfers(node, CacheTransferPhase.BACKUP_HOST)
        if t:
            comp_xfers[comp.component_type] = t
    sidecar_xfers = self._build_sidecar_transfers(
        CacheTransferPhase.BACKUP_HOST, kv_xfer, comp_xfers
    )

    # Pre-evict host if insufficient
    kv_tokens = len(device_value)
    host_avail = self.cache_controller.mem_pool_host.available_size()
    if host_avail < kv_tokens:
        needed = kv_tokens - host_avail
        evicted = self.evict_host(needed)
        if evicted < needed:
            return 0

    # ... 后续写入主机池逻辑不变
```

test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

新增测试验证 SWA 边界分裂叶子在 write-through 下能被正确备份和驱逐，是回归防护的关键。

```
  def test_hicache_write_through_offloads_swa_split_leaf(self):
      """A SWA boundary-split leaf should offload normally under write-through."""
      if not self.cfg.has_swa:
          self.skipTest("requires SWA")
      if self.cfg.has_mamba:
          self.skipTest("SWA-only path keeps the split setup simple")

      ps = self.cfg.page_size
      tree, allocator, _ = build_fixture(self.cfg)
      self._init_hicache(tree)
      tree.write_through_threshold = 1 # 每次命中都触发 write-through

      seq = self._make_seq(1, 2)
      value = self._alloc(allocator, len(seq))
      # 插入时设置 swa_evicted_seqlen 为 page_size，强制产生分裂
      result = tree.insert(
          InsertParams(
              key=RadixKey(seq),
              value=value,
              swa_evicted_seqlen=ps,
          )
      )
      self.assertEqual(result.prefix_len, 0)

      # 验证树结构：root -> split_parent -> split_leaf
      self.assertEqual(len(tree.root_node.children), 1)
      split_parent = next(iter(tree.root_node.children.values()))
      self.assertEqual(len(split_parent.children), 1)
      split_leaf = next(iter(split_parent.children.values()))

      # 触发 write_backup 和驱逐
      tree.writing_check(write_back=True)
      tree.evict(EvictParams(num_tokens=len(seq)))

      # 断言叶子已被驱逐、已备份、且出现在可驱逐主机叶子集合中
      self.assertTrue(split_leaf.evicted)
      self.assertTrue(split_leaf.backuped)
      self.assertIn(split_leaf, tree.evictable_host_leaves)
      tree.sanity_check() # 额外验证树结构完整性
```

评论区精华

Review 中 gemini-code-assist[bot] 建议移除 not write_back 守卫并传播 write_back 标志，以确保所有策略下的一致性；hzh0425 提出是否应该在 SWAComponent.commit_insert_component_data 或 _split_node 中触发 _inc_hit_count 来避免递归回溯所有祖先。alphabetc1 回应：在 commit_insert_component_data 中添加 _inc_hit_count 会使 SWA 感知树备份逻辑

，更合适的做法是放入 _split_node；且递归回溯仅发生在父节点缺失备份时，不会产生多余写入。最终 hzh0425 批准了当前方案。

- 递归备份 vs 在 SWA 分裂点触发备份 (design): 采用递归备份方案，因为递归深度受 radix 树深度限制且仅对未备份父节点触发，不会造成多余开销。
- 移除 not write_back 守卫 (correctness): 未采纳，当前修复仅针对 write-through 场景，非 write_back 路径下保持不变。

风险与影响

- 风险：递归调用 write_backup 可能增加递归深度，但 radix 树深度通常很小（受序列长度限制），风险可控。该递归仅在父节点未备份时触发，不会导致已备份祖先的冗余备份。若父节点备份失败（如主机空间不足），递归会返回 0，保持原有行为不变。对非 write-through 场景无影响。新测试覆盖了 SWA-only 路径，未覆盖同时有 MAMBA + SWA 的混合场景。
- 影响：影响范围仅限于 HiCache 的 write_through 策略 + SWA 组件。修复后，跨越 SWA 驱逐边界的叶子在 write-through 模式下能被正确备份和降级，避免重复计算。对非 SWA 或非 write-through 场景无影响。新增测试确保不变性不被破坏。
- 风险标记：递归深度风险低，仅 SWA+write-through 路径，缺少混合组件场景测试覆盖

关联脉络

- PR #25874 [CPU] add faster KV-cache writes: 涉及相同的 KV-cache 和内存池区域，HiCache 相关改动可能影响 CPU 路径的缓存行为。
- PR #26097 [VLM] try to reuse precomputed padded input ids in scheduler instead of padding: 调度器复用预计算输入，与缓存命中逻辑相关。
- PR #26225 fix(swa): downgrade translate_loc_from_full_to_swa key-change log from warning to debug: 同样涉及 SWA 内存池，属于 SWA 组件维护的一部分。
- PR #24610 [observability] add ServerArgs.stat_loggers for pluggable metrics backend: 涉及 HiRadixCache 和 HiMambaRadixCache，与被修改的 HiCache 组件共享基础结构。