PR #25068 完整报告

sgl-project/sglang

[UnifiedTree]: Fix the leaf determination logic in _cascade_evict.

合并时间：2026-05-13 10:57

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25068

执行摘要

- 一句话：修复 _cascade_evict 叶子判定逻辑
- 推荐动作：建议精读。该 PR 修正了 UnifiedTree 中一个关键的叶子节点判定逻辑，展示了在分层缓存系统中如何正确处理组件锁定状态与结构叶子之间的关系。_cascade_evict 的实现值得学习，尤其是组件优先级与叶子判定的结合。同时，新增的测试用例是隔离测试复杂驱逐场景的典范。

功能与动机

原有 _cascade_evict 中 is_leaf = len(node.children) == 0 的判定过于简单，在节点虽无子节点但被锁定（lock_ref > 0）或处于非可驱逐状态时，仍返回 True，导致驱逐优先级错误。Issue 评论中用户 icepoint666 报告了 agentic 工作负载下的异常行为，经复现确认该补丁修复了问题。

实现拆解

1. 修改叶子判定逻辑（python/sglang/srt/mem_cache/unified_radix_cache.py）：在 _cascade_evict 中，将 is_leaf 的计算从简单的 len(node.children) == 0 改为根据 target 层从 self.evictable_device_leaves 或 self.evictable_host_leaves 集合中查找。当 target 不是 DEVICE 或 HOST 时，is_leaf 默认 False。
2. 新增测试用例（test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py）：添加 test_aux_evict_full_locked_leaf_tombstones_aux_only，模拟一个叶子节点上 Full 组件被锁定、辅助组件（SWA 或 MAMBA）被标记为可驱逐的场景，验证仅驱逐辅助组件且不影响 Full 组件的正确性。

关键文件：

- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 层次缓存；类别 source；类型 core-logic；符号 _cascade_evict）：核心修复文件，修改了 _cascade_evict 中的叶子判定逻辑，是本次 PR 的主要变更。
- 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_aux_evict_full_locked_leaf_tombstones_aux_only）：新增测试用例 test_aux_evict_full_locked_leaf_tombstones_aux_only，覆盖了 Full 组件锁定且辅助组件可驱逐的场景，验证仅驱逐辅助组件。

关键符号：_cascade_evict

关键源码片段

python/sglang/srt/mem_cache/unified_radix_cache.py

核心修复文件，修改了 _cascade_evict 中的叶子判定逻辑，是本次 PR 的主要变更。

```
# python/sglang/srt/mem_cache/unified_radix_cache.py

def _cascade_evict(
    self,
    node: UnifiedTreeNode,
    trigger: TreeComponent,
    tracker: dict[ComponentType, int],
    target: EvictLayer = EvictLayer.DEVICE,
):
    """Cascade eviction from trigger to lower-or-equal priority components."""

    is_leaf = False
    if target == EvictLayer.DEVICE:
        # 仅在 DEVICE 层使用 evictable_device_leaves 判定叶子
        is_leaf = node in self.evictable_device_leaves
    elif target == EvictLayer.HOST:
        # 仅在 HOST 层使用 evictable_host_leaves 判定叶子
        is_leaf = node in self.evictable_host_leaves

    trigger_priority = trigger.eviction_priority(is_leaf)

    for comp in self._components_tuple:
        if comp.eviction_priority(is_leaf) <= trigger_priority:
            if comp is not trigger and comp.node_has_component_data(node, target):
                cd = node.component_data[comp.component_type]
                if EvictLayer.DEVICE in target:
                    assert cd.lock_ref == 0
                if EvictLayer.HOST in target:
                    assert cd.host_lock_ref == 0
                self._evict_component_and_detach_lru(
                    node, comp, target=target, tracker=tracker
                )

    # 略去后续 tomstone 处理 ...
```

test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

新增测试用例 test_aux_evict_full_locked_leaf_tombstones_aux_only，覆盖了 Full 组件锁定且辅助组件可驱逐的场景，验证仅驱逐辅助组件。

```
# test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

def test_aux_evict_full_locked_leaf_tombstones_aux_only(self):
    # 只支持 SWA 或 MAMBA 中恰好一种辅助组件的配置
    aux_types = [
```

```
    ct
    for ct in (ComponentType.SWA, ComponentType.MAMBA)
    if ct in self.cfg.components
]
if not aux_types:
    self.skipTest("requires an auxiliary component")
if len(aux_types) > 1:
    self.skipTest("single-aux case keeps cascade expectations precise")
aux = aux_types[0]

tree, allocator, req_to_token_pool = build_fixture(self.cfg)
seq = self._make_seq(1, 2)
self._insert(tree, allocator, req_to_token_pool, seq)

# 找到叶子节点，验证 Full 和 aux 组件都存在
match = tree.match_prefix(MatchPrefixParams(key=RadixKey(seq)))
node = match.last_device_node
full_cd = node.component_data[ComponentType.FULL]
aux_cd = node.component_data[aux]
self.assertEqual(len(node.children), 0)
self.assertIsNotNone(full_cd.value)
self.assertIsNotNone(aux_cd.value)

# 锁定节点，使节点不在 evictable_device_leaves 中
lock_result = tree.inc_lock_ref(node)
self.assertGreater(full_cd.lock_ref, 0)
self.assertGreater(aux_cd.lock_ref, 0)

# 手动将辅助组件的状态置为可驱逐，模拟部分锁定
aux_len = len(aux_cd.value)
tree.component_protected_size_[aux] -= aux_len
tree.component_evictable_size_[aux] += aux_len
aux_cd.lock_ref = 0
self.assertNotIn(node, tree.evictable_device_leaves)

# 发起仅驱逐辅助组件的请求
evict_params = EvictParams(num_tokens=0)
if aux == ComponentType.SWA:
    evict_params.swa_num_tokens = aux_len
else:
    evict_params.mamba_num = aux_len
result = tree.evict(evict_params)

# 验证：Full 组件保留，aux 组件被驱逐
self.assertEqual(result.num_tokens_evicted, 0)
if aux == ComponentType.SWA:
    self.assertEqual(result.swa_num_tokens_evicted, aux_len)
else:
    self.assertEqual(result.mamba_num_evicted, aux_len)
```

```
self.assertIsNotNone(full_cd.value)
self.assertIsNone(aux_cd.value)
self.assertFalse(tree.lru_lists[aux].in_list(node))

# 解锁并检查树的一致性
tree.dec_lock_ref(
    node,
    DecLockRefParams(swa_uuid_for_lock=lock_result.swa_uuid_for_lock),
)
tree.sanity_check()
```

评论区精华

gemini-code-assist[bot] 指出：当 target 不是 DEVICE 或 HOST（例如 EvictLayer.ALL）时，is_leaf 默认 False，可能将结构上的叶子节点错误视为内部节点。建议使用类似 is_leaf = len(node.children) == 0 作为 fallback。当前实现未采纳该建议，但通过仅对 DEVICE 和 HOST 层进行集合判断，在其他组合目标下回退为 False（非叶子），该设计有其合理性，因为组合目标情况下不应依赖单个叶子集判断。

- is_leaf 默认值为 False 对复合目标的影响 (correctness): 未采纳。当前实现有意避免在复合目标下依赖单个叶子集，因为复合目标场景下叶子判定本身就模糊，且实际调用中较少使用复合目标。

风险与影响

- 风险：
  1. EvictLayer.ALL 场景：当前实现中，当 target 为 EvictLayer.ALL 等复合目标时，is_leaf 固定为 False，可能导致在复合目标溢驱逐时优先级计算不准确。但复合目标在实际中较少使用，且影响面有限。
  2. 回归风险：修改了 _cascade_evict 的核心逻辑，可能影响其他依赖 is_leaf 判断的驱逐路径（如 host 层驱逐）。新增测试覆盖了该路径，降低了回归风险。
  3. 性能影响：从 O(1) 的 len(node.children) 变为集合查找操作，但集合查找开销仍为 O(1)，性能影响可忽略。 - 影响：影响范围：直接影响 UnifiedTree 中 _cascade_evict 的叶子节点判定，进而影响所有基于优先级驱逐的行为（device 和 host 层）。影响对象：启用 --enable-hierarchical-cache 和 --enable-unified-radix-tree 的用户，特别是使用多组件（Full+SWA/MAMBA）场景。影响程度：中等。修复了核心驱逐路径的语义错误，可能改善在复杂工作负载下的系统稳定性。
- 风险标记：复合目标路径未覆盖，核心驱逐逻辑变更

关联脉络

- 暂无明显关联 PR