PR #24585 完整报告

sgl-project/sglang

fix(unified radix cache w/ hicache): backup ancestor nodes before leaf in write_back eviction

合并时间：2026-05-17 15:58

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24585

执行摘要

- 一句话：修复 write_back 模式下 radix cache 驱逐时未等待写回及 sanity 检查误报
- 推荐动作：建议阅读此 PR 以理解 write_back 与 write_through 模式下缓存驱逐的不同约束。设计决策在于如何让 sanity_check 兼容不同写策略，避免误报。对于使用 HiCache write_back 的生产环境，此修复至关重要。

功能与动机

在 write_back 模式下，仅叶子节点被备份到 host，祖先节点不备份。原有的 sanity_check 强制要求所有备份节点的父节点也必须备份，这在 write_back 模式下不成立，导致断言失败和服务崩溃。另外，驱逐后没有等待写回完成，可能造成数据不一致。PR body 指出复现 bug 后 2 分钟内就会出现 sanity check 断言失败。

实现拆解

1. evict() 方法增加 writing_check：在 UnifiedRadixCache.evict() 中，遍历各组件完成驱逐后，检查 write_policy 是否为 'write_back'，若是则调用 self.writing_check(write_back=True) 确保异步写回操作完成。
2. sanity_check() 跳过 write_back 的父节点备份检查：在 sanity_check() 方法中，添加局部变量 write_back 标识当前策略；在检查 ' 节点备份但父节点未备份 ' 的逻辑前，如果 write_back 为 True，则跳过该检查。
3. 新增 write_back 单元测试：测试文件 test_unified_radix_cache_unittest.py 中新增 test_hicache_write_back_leaf_backup 测试用例，使用 write_back 策略初始化树，插入序列，锁定父节点，驱逐叶子节点 token，验证叶子节点被标记为 evicted 且 backuped，父节点不被备份，最后调用 sanity_check 验证无错误。

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 evict, sanity_check）：核心修复：evict() 添加 writing_check，sanity_check 增加 write_back 跳过条件
- 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 _init_hicache, test_hicache_write_back_leaf_backup）：新增 write_back 逐出叶子备份测试，验证修复正确性

关键符号：evict, sanity_check, _init_hicache, test_hicache_write_back_leaf_backup

关键源码片段

python/sglang/srt/mem_cache/unified_radix_cache.py

核心修复：evict() 添加 writing_check，sanity_check 增加 write_back 跳过条件

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

def evict(self, params: EvictParams) -> EvictResult:
    if self.disable:
        return EvictResult()
    start_time = time.perf_counter()
    tracker = {ct: 0 for ct in self.tree_components}

    for component in self._components_tuple:
        component.drive_eviction(params=params, tracker=tracker)

    # 新增：如果当前策略是 write_back，则在驱逐完成后等待所有异步写回操作完成
    if (
        self.cache_controller is not None
        and self.cache_controller.write_policy == 'write_back'
    ):
        self.writing_check(write_back=True)

    self.update_eviction_metrics(sum(tracker.values()), start_time)
    return EvictResult(...)

def sanity_check(self):
    # ... 省略树结构检查 ...
    # 新增：判断是否为 write_back 模式
    write_back = (
        self.cache_controller is not None
        and self.cache_controller.write_policy == 'write_back'
    )
    # ...
    # 原有检查：if full_hst and not p_hst:
    # 修改为：跳过祖先备份检查（write_back 模式下只备份叶子自身）
    if full_hst and not p_hst and not write_back:
        E(f'node {nid} backed up but parent {node.parent.id} not backed up')
    # ...
```

test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

新增 write_back 逐出叶子备份测试，验证修复正确性

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

def _init_hicache(self, tree, *, write_policy: str = 'write_through'):
    # 可接受 write_policy 参数，用于测试 write_back 模式
```

```
    # ... mock pool 构造 ...
    server_args = ServerArgs(
        model_path='dummy',
        page_size=self.cfg.page_size,
        hicache_io_backend='direct',
        hicache_write_policy=write_policy, # 使用传入的策略
    )
    # ...

def test_hicache_write_back_leaf_backup(self):
    '''write_back 驱逐情形：设备叶子被驱逐时应自动备份到 host'''
    if self._skip_unsupported_hicache_test():
        return
    tree, allocator, req_to_token_pool = build_fixture(self.cfg)
    self._init_hicache(tree, write_policy='write_back')

    base = self._make_seq(1, 2)
    leaf_seq = base + self._make_seq(500, 2)
    self._insert(tree, allocator, req_to_token_pool, base)
    self._insert(tree, allocator, req_to_token_pool, leaf_seq)

    m = tree.match_prefix(MatchPrefixParams(key=RadixKey(leaf_seq)))
    leaf = m.last_device_node
    parent = leaf.parent
    self.assertIsNot(parent, tree.root_node)

    # 初始状态：叶子及其父节点均未备份
    self.assertFalse(leaf.backuped)
    self.assertFalse(parent.backuped)

    # 锁定父节点，防止驱逐时拆解到它
    lr = tree.inc_lock_ref(parent)
    try:
        evict_tokens = len(leaf_seq) - len(base)
        tree.evict(EvictParams(num_tokens=evict_tokens))
    finally:
        tree.dec_lock_ref(
            parent,
            DecLockRefParams(
                swa_uuid_for_lock=getattr(lr, 'swa_uuid_for_lock', None)
            ),
        )

    # 验证叶子节点已被标记为 evicted 且 backuped
    self.assertTrue(leaf.evicted, 'leaf should be demoted to host')
    self.assertTrue(leaf.backuped, 'write_back must back up the leaf on eviction')
    # 父节点不应被备份（write_back 策略仅备份被驱逐的叶子）
    self.assertFalse(parent.backuped, 'parent must NOT be backed up under write_back')
```

```
# 最终一致性检查
tree.sanity_check()
```

评论区精华

核心讨论集中在 sanity_check 兼容性和 writing_check 必要性。hzh0425 指出“问题可能不是父节点缺少备份，而是 sanity_check 需要兼容 write_back 模式”。libertyeagle 同意并进一步指出“当前代码仍没有等待写回完成，需要添加 self.writing_check(write_back=True)”。最终方案同时采纳了这两点。此外，hzh0425 要求提供 benchmark 和单元测试，作者提供了 benchmark 数据并新增了测试用例。

- sanity_check 在 write_back 模式下的兼容性 (correctness): 在 sanity_check 中增加 write_back 条件判断跳过祖先备份检查，并在 evict() 末尾添加 writing_check。
- 添加 writing_check 的必要性 (correctness): 在 evict() 循环后、metrics 更新前，根据 write_back 模式调用 writing_check(write_back=True)。
- 请求 benchmark 和单元测试 (testing): libertyeagle 提供了 benchmark 数据，并添加了 test_hicache_write_back_leaf_backup 单元测试。

风险与影响

- 风险：本 PR 的变更仅限于 write_back 模式，默认 write_through 路径不变。修改 sanity_check 跳过一个检查可能会掩盖其他潜在的树不变量问题，但 write_back 模式下该检查本来就不适用。writing_check 可能引入额外的等待时间，但只发生在 write_back 模式的驱逐结束时，对性能影响有限。新增的单元测试覆盖了主要场景，但因 write_back 模式本身使用较少，可能在生产中存在未覆盖的边界情况。
- 影响：仅影响使用 --hicache-write-policy write_back 的用户。修复了之前导致服务崩溃的严重 bug，提升了 write_back 模式的稳定性。团队应合并此修复到包含 write_back 支持的发布分支。
- 风险标记：write_back 模式特有，sanity_check 跳过检查，异步写回等待

关联脉络

- PR #25477 Fix DeepSeek V4 HiCache layer count logic: 同属 HiCache 修复系列，修改了 radix cache 相关逻辑。