PR #29350 完整报告

sgl-project/sglang

[optimize] fix swa eviction boundary for unfinished inserts

合并时间：2026-06-30 00:19

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29350

执行摘要

- 一句话：修复未完成请求的 SWA 驱逐边界丢失问题
- 推荐动作：值得快速合入。该 PR 修复了一个隐蔽的边界条件 bug，修改量小、理由充分，且通过了 review 讨论。推荐阅读 swa_component.py 中的 prepare_for_caching_req 逻辑演变，以理解 radix cache 的组件化设计。

功能与动机

PR body 指出："Unfinished request insertion can happen after part of the request has already been evicted from the SWA window. The insert path needs that eviction boundary to distinguish live SWA KV from SWA tombstones." 即未完成请求可能部分已被 SWA 窗口驱逐，但旧代码在未完成分支未保留驱逐边界，导致插入时误将应标记为 tombstone 的 KV 当作有效数据。

实现拆解

1. 放宽边界复制条件：在 swa_component.py 的 prepare_for_caching_req 方法中，移除 if is_finished 条件判断，使 insert_params.swa_evicted_seqlen = req.swa_evicted_seqlen 对所有请求（无论完成与否）均执行。这样，未完成请求的已有驱逐边界得以保留。
2. 添加单元测试：在 test_unified_radix_cache_unittest.py 中新增 test_swa_unfinished_req_preserves_existing_eviction_boundary 测试用例。该测试构造一个具有 8 个 token、已驱逐 4 个的未完成请求，调用 cache_unfinished_req 后，验证根节点下第一个子节点（对应驱逐部分）的 SWA component 值为 None（即 tombstone），而后续子节点（对应剩余 token）的 SWA component 值非 None，并最终通过 match_prefix 确认整体前缀匹配无误。
3. 变更范围：仅涉及 swa_component.py 中 2 行核心逻辑修改（删除条件判断）和对应的测试文件新增 41 行测试代码。

关键文件：

- python/sglang/srt/mem_cache/unified_cache_components/swa_component.py（模块 缓存组件；类别 source；类型 core-logic）：核心修复文件，移除了 prepare_for_caching_req 中的 is_finished 条件判断，使所有请求均保留 SWA 驱逐边界。这是缓存一致性的关键逻辑。

- 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_swa_unfinished_req_preserves_existing_eviction_boundary）：新增测试用例 test_swa_unfinished_req_preserves_existing_eviction_boundary，验证修复的正确性，确保缓存树结构符合预期。

关键符号：prepare_for_caching_req, test_swa_unfinished_req_preserves_existing_eviction_boundary

关键源码片段

python/sglang/srt/mem_cache/unified_cache_components/swa_component.py

核心修复文件，移除了 prepare_for_caching_req 中的 is_finished 条件判断，使所有请求均保留 SWA 驱逐边界。这是缓存一致性的关键逻辑。

```
# python/sglang/srt/mem_cache/unified_cache_components/swa_component.py
def prepare_for_caching_req(
    self,
    req: Req,
    insert_params: InsertParams,
    token_ids_len: int,
    is_finished: bool,
) -> Optional[int]:
    # Unfinished requests can already have an SWA-evicted prefix; preserve
    # that boundary so insertion creates a tombstone instead of live SWA KV.
    insert_params.swa_evicted_seqlen = req.swa_evicted_seqlen
    return None

def free_out_of_window_slots(
    self, req: Req, pre_len: int, insert_params: InsertParams
) -> None:
    # ... ( 此函数未变更，但依赖 insert_params 中的边界 )
```

test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

新增测试用例 test_swa_unfinished_req_preserves_existing_eviction_boundary，验证修复的正确性，确保缓存树结构符合预期。

```
# test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py
def test_swa_unfinished_req_preserves_existing_eviction_boundary(self):
    # 仅在 SWA 启用且无 Mamba、page_size=1、sliding_window_size=4 时运行
    if not self.cfg.has_swa or self.cfg.has_mamba:
        self.skipTest("requires SWA without Mamba")
    if self.cfg.page_size != 1 or self.cfg.sliding_window_size != 4:
        self.skipTest("requires page_size=1, sliding_window_size=4")
    tree, allocator, req_to_token_pool = build_fixture(self.cfg)

    req = self._make_req(req_to_token_pool)
    tokens = self._make_seq(1, 8)
    evicted_len = 4 # 假设前 4 个 token 已被 SWA 驱逐
```

```
req.origin_input_ids = array("q", tokens)
req.output_ids = []
# ... 填充其他 req 字段 ...
req.swa_evicted_seqlen = evicted_len

tree.cache_unfinished_req(req)

# 检查驱逐部分被标记为 tombstone (value is None)
first = next(iter(tree.root_node.children.values()))
self.assertEqual(len(first.key), evicted_len)
self.assertIsNone(first.component_data[ComponentType.SWA].value)

# 检查剩余部分为 live SWA KV (value is not None)
live = next(iter(first.children.values()))
self.assertEqual(len(live.key), len(tokens) - evicted_len)
self.assertIsNotNone(live.component_data[ComponentType.SWA].value)

# 前缀匹配应能访问全部 token（包括驱逐部分）
m = tree.match_prefix(MatchPrefixParams(key=RadixKey(array("q", tokens))))
self.assertEqual(len(m.device_indices), len(tokens))
# ... 释放锁并检查结构一致性 ...
```

评论区精华

hzh0425 的质疑：在 prepare_for_caching_req 修改处（第 569 行）询问“这会导致 insert_helper 跳过叶子节点创建并释放叶子节点关联的值吗？@ispobock”。 ispobock 的答复：回应确认“是的，这是为了 SGLANG_OPT_UNIFIED_CACHE_FREE_OUT_OF_WINDOW_SLOTS”。表明该修改与 free-out-of-window-slots 优化选项协同，有意允许提前释放驱逐区域的 KV 值。 讨论未产生进一步分歧，ispobock 随后批准了 PR。

- 修改是否会导致 insert_helper 跳过叶子创建并释放值 (design): ispobock 确认是预期行为，旨在配合 SGLANG_OPT_UNIFIED_CACHE_FREE_OUT_OF_WINDOW_SLOTS 环境变量使用。

风险与影响

- 风险：该修改很小且逻辑清晰，主要风险是回归测试覆盖不足。由于废弃了条件判断，对于已完成的请求行为不变量（is_finished=True 时必然有非零的 swa_evicted_seqlen）继续保持。但是，如果未来某个代码路径意外在未完成请求上运行了 prepare_for_caching_req 两次，可能导致边界被错误覆盖。不过，鉴于 prepare_for_caching_req 在调用链中通常只执行一次，且已有单元测试验证了正确路径，此风险较低。
- 影响：直接影响 SWA（Sliding Window Attention）场景下的缓存一致性，修复后未完成请求的缓存插入能够正确区分 live KV 和 tombstone，避免后续解码阶段访问无效数据。影响范围局限于启用了 SWA 功能的模型部署。配套测试确保回归覆盖，对系统性能无负面影响。
- 风险标记：核心路径变更，缺少集成测试覆盖

关联脉络

- PR #29349 [optimize] fix swa eviction boundary for unfinished inserts (full set): 本 PR 是 PR #29349 的拆分第 1/5 部分，后续还有针对同一功能的其他修复或优化。