PR #26919 完整报告

sgl-project/sglang

Split SWA leaf to one window on insert

合并时间：2026-06-01 23:46

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26919

执行摘要

- 一句话：插入时分割 SWA 叶子，锁定仅占一个滑动窗口
- 推荐动作：该 PR 值得仔细阅读，特别是 _maybe_split_leaf_for_swa_lock 的设计：通过插入时立即裁剪来防止锁定长叶子过度占用 SWA 池，是一种简洁有效的资源治理策略。对于关注 SWA 或统一缓存的开发者，理解此模式有助于在其他类似场景中复用。

功能与动机

SWA 只关注最后 sliding_window_size 个 token，但锁定一个缓存叶子时会固定整个叶子的 SWA slots。使用分块预填时单个叶子可能包含数千个 token，一个锁定请求会占用远超实际需要的 SWA 池，导致 SWA 池过早填满并开始撤回请求。该 PR 通过插入时立即裁剪叶子，使锁定仅保护一个窗口大小的 SWA 资源。

实现拆解

1. 在 SWAComponent.commit_insert_component_data 末尾无条件调用 self._maybe_split_leaf_for_swa_lock(node)，确保每个新叶子（已经在 SWA 窗口内）都经过长度裁剪。
2. 新增 _maybe_split_leaf_for_swa_lock 方法：先计算覆盖滑动窗口所需的最小页面对齐大小 tail_size；若叶子长度大于 tail_size 且满足页面对齐条件，则通过 self.cache._split_node 将叶子分裂，前面部分成为可驱逐的普通前缀，后面部分保留 SWA 且长度为 tail_size。
3. 补充单元测试 test_swa_leaf_capped_to_window_on_insert：分别构造长叶子（超过一个窗口）和短叶子（不超过一个窗口），验证长叶子分裂后 SWA 值大小为窗口大小且存在父节点，短叶子保持原样；验证锁定后 SWA 保护大小等于窗口大小而 full attention 保护整个序列。
4. 修改 test_swa_lru_walk_down_does_not_refresh_ancestors_during_insert 测试中的节点数预期，因为 8 页叶子被分裂成两个节点，LRU 顺序节点数相应调整。

关键文件：

- python/sglang/srt/mem_cache/unified_cache_components/swa_component.py（模块 SWA 缓存；类别 source；类型 core-logic；符号 commit_insert_component_data, _maybe_split_leaf_for_swa_lock）：核心变更文件，新增 _maybe_split_leaf_for_swa_lock 方法并在 commit_insert_component_data 结尾调用，

实现 SWA 叶子插入时按窗口长度裁剪。

- 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_leaf_capped_to_window_on_insert, test_swa_lru_walk_down_does_not_refresh_ancestors_during_insert）：新增 test_swa_leaf_capped_to_window_on_insert 测试用例，并调整 test_swa_lru_walk_down_does_not_refresh_ancestors_during_insert 以适应新的分裂行为（节点数增加）。

关键符号：commit_insert_component_data, _maybe_split_leaf_for_swa_lock, test_swa_leaf_capped_to_window_on_insert

关键源码片段

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

核心变更文件，新增 _maybe_split_leaf_for_swa_lock 方法并在 commit_insert_component_data 结尾调用，实现 SWA 叶子插入时按窗口长度裁剪。

```
def commit_insert_component_data(
    self,
    node: UnifiedTreeNode,
    is_new_leaf: bool,
    params: InsertParams,
    result: InsertResult,
) -> None:
    # 仅在新建叶子时处理 SWA 元数据
    if not is_new_leaf:
        return

    node_start = result.prefix_len
    split_pos = params.swa_evicted_seqlen - node_start

    if split_pos <= 0:
        # 整个叶子在滑动窗口内，正常设置 SWA value
        swa_value = self._translate_full_to_swa(
            node.component_data[BASE_COMPONENT_TYPE].value
        )
        node.component_data[self.component_type].value = swa_value
        self.cache.lru_lists[self.component_type].insert_mru(node)
        self.cache.component_evictable_size_[self.component_type] += len(swa_value)
    elif split_pos < len(node.key):
        # 叶子跨越 SWA 驱逐边界：分裂出无 SWA 的父亲和有 SWA 的孩子
        self.cache._split_node(node.key, node, split_pos)
        swa_value = self._translate_full_to_swa(
            node.component_data[BASE_COMPONENT_TYPE].value
        )
        node.component_data[self.component_type].value = swa_value
        self.cache.lru_lists[self.component_type].insert_mru(node)
```

```
            self.cache.component_evictable_size_[self.component_type] += len(swa_value)
        else:
            # 整个叶子在滑动窗口之外，作为 tombstone 不设置 SWA
            return

        # 对新插入的叶子执行窗口大小裁剪，防止锁定过多 SWA 池
        self._maybe_split_leaf_for_swa_lock(node)

    def _maybe_split_leaf_for_swa_lock(self, leaf: UnifiedTreeNode) -> None:
        """将新叶子限制为一个页面对齐的滑动窗口长度，确保锁定时仅占用 SWA 池的一个窗口"""
        ct = self.component_type
        cd = leaf.component_data[ct]
        # 根节点、无 SWA value 或已被锁定的叶子不处理
        if leaf is self.cache.root_node or cd.value is None or cd.lock_ref > 0:
            return

        page_size = self.cache.page_size
        # 计算覆盖滑动窗口所需的最小页面对齐大小
        tail_size = (self.sliding_window_size + page_size - 1) // page_size * page_size
        leaf_len = len(leaf.key)
        if leaf_len <= tail_size:
            return # 叶子长度未超过一个窗口，无需分裂
        split_at = leaf_len - tail_size
        # 防御性检查：确保分裂点与叶子长度都是页面边界对齐的
        if page_size > 1 and (split_at % page_size != 0 or leaf_len % page_size != 0):
            return

        self.cache._split_node(leaf.key, leaf, split_at)
```

test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

新增 test_swa_leaf_capped_to_window_on_insert 测试用例，并调整 test_swa_lru_walk_down_does_not_refresh_ancestors_during_insert 以适应新的分裂行为（节点数增加）。

```
    def test_swa_leaf_capped_to_window_on_insert(self):
        """A long SWA leaf is split so locking it protects one window of SWA
        while full attention still protects the whole sequence."""
        if not self.cfg.has_swa:
            self.skipTest("requires SWA component")

        ps = self.cfg.page_size
        window = self.cfg.sliding_window_size
        tail_size = ((window + ps - 1) // ps) * ps
        tail_pages = tail_size // ps

        for case in ("long_splits", "short_keeps"):
            with self.subTest(case=case):
                tree, allocator, req_to_token_pool = build_fixture(self.cfg)
                num_pages = tail_pages + 2 if case == "long_splits" else tail_pages
```

```
seq = self._make_seq(1, num_pages)
self._insert(tree, allocator, req_to_token_pool, seq)
tree.sanity_check()

leaf = tree.match_prefix(
    MatchPrefixParams(key=RadixKey(array("q", seq)))
).last_device_node
swa_val = leaf.component_data[ComponentType.SWA].value
self.assertIsNotNone(swa_val)

if case == "long_splits":
    # 被裁剪为一个页面对齐的窗口，且存在真实的前缀父节点
    self.assertEqual(len(swa_val), tail_size)
    self.assertIsNot(leaf.parent, tree.root_node)
else:
    # 叶长不足一个窗口，保持原样
    self.assertEqual(len(swa_val), len(seq))
    self.assertIs(leaf.parent, tree.root_node)

lock_result = tree.inc_lock_ref(leaf)
# SWA 只保护一个窗口，而 full attention 保护整个序列
self.assertEqual(tree.swa_protected_size(), len(swa_val))
self.assertEqual(tree.full_protected_size(), len(seq))
tree.sanity_check()
tree.dec_lock_ref(
    leaf,
    DecLockRefParams(swa_uuid_for_lock=lock_result.swa_uuid_for_lock),
)
tree.sanity_check()
```

评论区精华

gemini-code-assist[bot] 在 _maybe_split_leaf_for_swa_lock 中指出：tail_size 和 leaf_len 均为 page_size 的倍数时，split_at 必定也是 page_size 的倍数，因此 split_at % page_size != 0 的检查是冗余的，建议简化为 if page_size > 1 and leaf_len % page_size != 0: return。但作者未采纳该建议，保留了原有的双重检查作为防御性验证。这是唯一讨论点。

- Check redundancy in page alignment validation (style): 作者未采纳该建议，保留了原有双重检查作为防御性验证。

风险与影响

- 风险：
  1. 回归风险：分裂逻辑影响所有新插入的 SWA 叶子，若 _split_node 或 LRU 链表操作有缺陷，可能导致内存损坏或死锁。但该路径已有现有测试覆盖。
  2. 性能影响：插入时多一次叶子分裂判断和可能的分裂操作，但仅发生在叶子首次创建时，且分裂很快（仅指针操作），开销可忽略。

3. 兼容性：改变了 LRU 节点数量（叶子分裂成两个节点），依赖 LRU 节点计数或顺序的代码（如部分测试）需要适配。已在 PR 中调整了相关测试。
4. 边界条件：split_at 和 leaf_len 非页面对齐时的防御性返回虽安全，但可能导致某些极端配置下不分裂，浪费 SWA 池。不过代码注释表明这是预期安全网。 - 影响：用户视角：对于启用 SWA（如 DeepSeek 模型）且使用分块预填的长序列请求，该 PR 显著降低单次锁定消耗的 SWA 池量，提高 SWA 池利用率，减少因池满而被迫撤回请求的概率。直接影响推理吞吐和延迟稳定性。系统视角：SWA 组件插入路径增加一次分裂逻辑，但开销低；LRU 链表节点数可能增加，但节点数量仍在合理范围。团队视角：该 PR 合入后，后续需要确保所有涉及 SWA 叶子锁定的测试和逻辑都与新的分裂行为一致。

- 风险标记：核心路径变更：SWA 插入分裂，缺少极端配置验证：非对齐场景直接跳过分裂

关联脉络

- PR #26615 [sgl] Window-aware LRU refresh for SWA prefix cache in unified cache: 同一模块（统一缓存 SWA 组件）的改进，新增了 LRU 刷新策略，与本 PR 的叶子分割逻辑共同优化 SWA 池管理。
- PR #26870 Make unified tree SWA hicache tests faithful to write-through backup: 同一测试文件 test_unified_radix_cache_unittest.py 的增强，增加了 SWA 写入通备份的测试，与本 PR 的测试扩展同属一个持续改进方向。