# PR #26302 完整报告

sgl-project/sglang

[UnifiedTree] gate load back pre-evict on full-attn availability only

合并时间：2026-05-29 00:30

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26302

## 执行摘要

- 一句话：限制 load-back 预驱逐仅使用 full attention pool 容量
- 推荐动作：值得精读，尤其是其设计权衡（避免污染基础接口的哲学）。建议 review 关注 full_available_size 在 SWA 和 HiSparse 分配器中的实现是否完整，以及未来是否有其他路径需要类似修复。

## 功能与动机

Load back in URT 原本基于 token_to_kv_pool_allocator.available_size() 进行预驱逐判断。对于 SWA-hybrid 分配器，该方法返回 min(full, swa)，即 full 和 SWA 子池的最小可用容量。当 SWA 子池是更紧张的资源时，会触发 tree.evict() 并级联执行 FullComponent.drive_eviction，释放不相关的叶子节点来腾出 SWA 容量。但 HiCacheController.load 实际只从 full attention 池分配全量 KV，这种预驱逐是不必要的。本 PR 修正此行为。

## 实现拆解

1. 基础分配器扩展：在 BaseTokenToKVPoolAllocator 中新增 full_available_size() 方法，默认返回 available_size()。SWA 和 HiSparse 分配器已覆盖此方法，返回对应 full 子池的可用容量。
2. URT 加载回传逻辑调整：在 unified_radix_cache.py 的 load_back() 方法中，将原来统一的 available_size() 调用改为分支判断。如果树支持 SWA（self.supports_swa()），则调用 full_available_size()；否则回退到 available_size()。从而确保预驱逐仅针对 full attention 池的容量进行。
3. 单元测试新增：在 test_unified_radix_cache_unittest.py 中新增 test_hicache_swa_load_back_uses_full_pool_capacity 方法。模拟 SWA 池紧张但 full 池充足的场景，通过 mock 验证 load_back 不会因 SWA 压力发起全节点驱逐（evict 调用参数仅包含 swa_num_tokens>0 而 num_tokens==0）。
4. 清理与后续：测试中完成加载和锁释放，并执行 sanity check。

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 load_back）：核心逻辑修改，在 load_back 中根据是否支持 SWA 使用 full_available_size() 作为容量判断，避免因 SWA 池紧张而错误触发 full attention 节点

的预驱逐。

- 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_swa_load_back_uses_full_pool_capacity）：新增完整单元测试，验证 load_back 在 SWA 混合分配器下使用 full pool 容量进行预驱逐判断，确保行为正确。

关键符号：load_back, test_hicache_swa_load_back_uses_full_pool_capacity

## 关键源码片段

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

核心逻辑修改，在 load_back 中根据是否支持 SWA 使用 full_available_size() 作为容量判断，避免因 SWA 池紧张而错误触发 full attention 节点的预驱逐。

```
# python/sglang/srt/mem_cache/unified_radix_cache.py (modified section in load_back)
# 在预驱逐前检查可用容量
if self.supports_swa():
    # 如果树支持 SWA，使用 full attention 池的可用容量
    avail = self.token_to_kv_pool_allocator.full_available_size()
else:
    # 否则使用默认 available_size()
    avail = self.token_to_kv_pool_allocator.available_size()
if avail < kv_tokens:
    needed = kv_tokens - avail
    result = self.evict(EvictParams(num_tokens=needed))
    if result.num_tokens_evicted < needed:
        self.dec_lock_ref(best_match_node, ancestor_lock_params)
        return False
```

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

新增完整单元测试，验证 load_back 在 SWA 混合分配器下使用 full pool 容量进行预驱逐判断，确保行为正确。

```
# test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py
def test_hicache_swa_load_back_uses_full_pool_capacity(self):
    """load_back should gate Full KV load on Full pool capacity only."""
    if not self.cfg.has_swa:
        self.skipTest("requires SWA")
    if self.cfg.page_size > 1:
        self.skipTest("page_size==1 for direct swa_attn_allocator access")

    tree, allocator, req_to_token_pool = self._build_hicache_fixture()
    sw = self.cfg.sliding_window_size
    kv_tokens = sw + 2
    chain = self._build_chain_pages(tree, allocator, req_to_token_pool, kv_tokens)
    leaf = chain[-1]
    self._backup_tree(tree)
    result = tree.evict(EvictParams(num_tokens=kv_tokens))
    self.assertIsNone(leaf.component_data[ComponentType.FULL].value)
```

```
    # 降低 SWA 可用容量至小于 full 需求
    target_swa_avail = sw - 1
    swa_avail = allocator.swa_attn_allocator.available_size()
    if swa_avail > target_swa_avail:
        external_swa = allocator.swa_attn_allocator.alloc(swa_avail - target_swa_avail)

    # 验证 full 池充足，SWA 池不足
    self.assertGreaterEqual(allocator.full_attn_allocator.available_size(), kv_tokens)
    self.assertLess(allocator.swa_attn_allocator.available_size(), kv_tokens)

    with mock.patch.object(tree, "evict", wraps=tree.evict) as evict_mock:
        self.assertTrue(tree.load_back(leaf))

    # 无 full 预驱逐 (num_tokens>0)
    full_evicts = [call for call in evict_mock.call_args_list
                   if call.args and call.args[0].num_tokens > 0]
    self.assertEqual(full_evicts, [])
    # 有 SWA 驱逐 (num_tokens==0, swa_num_tokens>0)
    self.assertTrue(any(call.args and call.args[0].num_tokens == 0
                        and call.args[0].swa_num_tokens > 0
                        for call in evict_mock.call_args_list))

    self._finish_pending_loads(tree)
    self._release_ongoing_load_back_locks(tree)
    tree.sanity_check()
```

## 评论区精华

主要讨论：ispobock 在 review 中建议使用分支代替添加 full_available_size 到基础分配器，因为该方法仅对混合池有意义，命名容易混淆。vladnosiv 接受建议，修改为分支方案（if self.supports_swa()）。最终方案避免了基础类引入不通用概念。

- 是否添加 full_available_size 到基础分配器还是使用分支 (design): vladnosiv 接受建议，修改为分支方案（if self.supports_swa()）。

## 风险与影响

- 风险：风险较低，改动范围小（仅 5 行源码），且分支明确。需要回归非 SWA 场景确保无影响。潜在风险：其他调用位置也可能误用 available_size() 导致类似问题，但本 PR 仅修复 load_back。另外需确认 SWA 和 HiSparse 分配器中 full_available_size() 的实现是否一致返回 full 子池容量（已有正确实现，但可作为 review 点）。
- 影响：主要影响使用 SWA 混合分配器的模型（如 DeepSeek 系列），避免因 SWA 池紧张而错误驱逐 full attention 节点，可能提升缓存命中率和推理延迟。对非 SWA 场景无影响。
- 风险标记：核心路径变更，影响 SWA 混合分配器

## 关联脉络

- 暂无明显关联 PR