PR #23316 完整报告

sgl-project/sglang

[UnifiedRadixTree]: Support HiCache Framework for UnifiedRadixTree

合并时间：2026-05-03 22:13

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23316

执行摘要

- 一句话：UnifiedRadixTree 集成 HiCache，统一缓存生命周期
- 推荐动作：本 PR 是 UnifiedRadixTree 集成 HiCache 的里程碑变更，涉及缓存核心架构、多组件协同和主机层管理，值得团队精读以理解设计决策，特别是 EvictLayer 枚举和组件化加载回接口的抽象方式。建议重点关注 unified_radix_cache.py 中 init_hicache 和驱逐链的调度，以及各组件中 build_hicache_transfers 的实现，这些是未来扩展 L3 的基础。

功能与动机

根据 PR 描述，该变更为 UnifiedRadixTree 添加 HiCache 支持，以统一 Full、Mamba 及相关混合组件的设备 / 主机缓存生命周期。它使 UnifiedRadixTree 能够为混合线性模型和 DeepSeek DSA 风格模型启用 HiCache，并提供了组件化的驱逐 / 加载回逻辑、显式 D-leaf/H-leaf 跟踪、辅助组件的主机端 LRU 管理，以及统一路径中的 HiCache 池 / 控制器集成。Issue 评论中 linjianyu233 和 icepoint666 验证了精度和多轮对话稳定性。

实现拆解

1. LRU 列表扩展与节点属性新增：在 unified_radix_cache.py 中，UnifiedTreeNode 新增 backuped 和 evicted 属性，用于指示 Full KV 是否已备份到主机或已被从设备驱逐。UnifiedLRUList 增加 use_host_ptr 参数，为主机 LRU 分配独立的指针槽位，避免与设备 LRU 指针冲突。
2. 驱逐层与传输阶段枚举：在 tree_component.py 中定义 EvictLayer（IntFlag：DEVICE、HOST、ALL）和 CacheTransferPhase（BACKUP_HOST、LOAD_BACK、BACKUP_STORAGE、PREFETCH），为分层驱逐和缓存迁移提供类型安全的基础。
3. 组件接口重构：将各组件（FullComponent、MambaComponent、SWAComponent）的 evict_component 签名从 (node, is_leaf) -> int 改为 (node, target: EvictLayer) -> tuple[int, int]，返回设备与主机的释放 token 数量。新增 build_hicache_transfers、commit_hicache_transfer、drive_host_eviction 等抽象方法，各组件按需实现主机 LRU 管理和传输构建。例如 FullComponent 在 evict_component 中同时处理设备层和主机层的释放，并在 redistribute_on_node_split 中克隆 host_value。
4. UnifiedRadixCache 主循环集成：在缓存主类中添加 init_hicache、register_hicache_anchor_kv_shared_indices_pool、_unevict_node_on_insert、_for_each_component_lru 和 evict_host 等方法，协调设备与主机间的数据传输。驱逐流程从单设备层扩展为设备层驱逐 + 主机层空间回收的顺序执行。

5. 调度器与池组装器适配：scheduler.py 和 hybrid_pool_assembler.py 调整导入及调用点，使其支持新的 HiCache 路径（如 Load-back 触发和主机池初始化）。
6. 测试覆盖：test_unified_radix_cache_unittest.py新增8个驱逐链测试（原子释放、级联、LRU 顺序等）；test_unified_radix_cache_kl.py 添加精度测试。CI 中 CUDA13 环境暂时跳过部分 HiCache 测试。

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 backuped, evicted, _reset_full, init_hicache）：主缓存类，新增 backuped/evicted 属性、扩展 LRU 列表为双指针层，并集成 HiCache 初始化与驱逐协调逻辑。
- python/sglang/srt/mem_cache/unified_cache_components/mamba_component.py（模块 Mamba 组件；类别 source；类型 core-logic；符号 evict_component, build_hicache_transfers, commit_hicache_transfer, drive_host_eviction）：Mamba 组件，在匹配验证器中增加主机值检查，在 finalize_match_result 中处理主机命中长度以触发加载回。
- python/sglang/srt/mem_cache/unified_cache_components/full_component.py（模块 全量组件；类别 source；类型 core-logic；符号 node_has_component_data, finalize_match_result, evict_component, drive_host_eviction）：全量组件，新增主机层驱逐支持，重构 redistribute_on_node_split 以复制 host_value。
- python/sglang/srt/mem_cache/unified_cache_components/tree_component.py（模块 树组件基类；类别 source；类型 core-logic；符号 EvictLayer, node_has_component_data, evict_component）：树组件基类，定义 EvictLayer 枚举和 CacheTransferPhase，重构 node_has_component_data 和 evict_component 接口以支持分层。
- 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_evict_leaf_frees_all_components, test_evict_cascade_parent_becomes_d_leaf, test_evict_iterative_tombstone_cleanup, test_evict_respects_lru_order）：新增 8 个驱逐链测试覆盖原子释放、级联、LRU 顺序等场景，确保 HiCache 驱逐逻辑正确。
- python/sglang/srt/mem_cache/unified_cache_components/swa_component.py（模块 SWA 组件；类别 source；类型 core-logic；符号 evict_component）：SWA 组件适配新的 evict_component 签名，暂不支持主机层（HOST 直接返回 0），并在 drive_eviction 中区分 D-leaf 和内部节点。

关键符号：backuped, evicted, init_hicache, register_hicache_anchor_kv_shared_indices_pool, _unevict_node_on_insert, _for_each_component_lru, evict_host, evict_component, build_hicache_transfers, commit_hicache_transfer, drive_host_eviction, node_has_component_data, finalize_match_result, EvictLayer, CacheTransferPhase, redistribute_on_node_split, create_match_validator, drive_eviction

关键源码片段

python/sglang/srt/mem_cache/unified_radix_cache.py

主缓存类，新增 backuped/evicted 属性、扩展 LRU 列表为双指针层，并集成 HiCache 初始化与驱逐协调逻辑。

```
from sglang.srt.mem_cache.unified_cache_components import (
    _NUM_COMPONENT_TYPES,
    ComponentData,
    ComponentType,
)
from sglang.srt.mem_cache.radix_cache import RadixKey
from typing import Optional

class UnifiedTreeNode:
    counter = 0

    def __init__(self, tree_components: tuple[ComponentType, ...]):
        self.parent: Optional['UnifiedTreeNode'] = None
        self.key: Optional[RadixKey] = None
        # 组件数据列表，按 ComponentType 索引
        self.component_data: list[ComponentData] = [
            ComponentData() for _ in range(_NUM_COMPONENT_TYPES)
        ]
        self.last_access_time = get_and_increase_time_counter()
        self.hash_value = None # 替换了旧的 host_value 字段
        self.hit_count = 0
        # LRU 指针：长度扩展为 _NUM_COMPONENT_TYPES * 2，
        # 前半段供设备 LRU 使用，后半段为主机 LRU 保留独立槽位
        self.lru_prev: list[Optional['UnifiedTreeNode']] = [None] * (
            _NUM_COMPONENT_TYPES * 2
        )
        self.lru_next: list[Optional['UnifiedTreeNode']] = [None] * (
            _NUM_COMPONENT_TYPES * 2
        )
        self.id = UnifiedTreeNode.counter
        UnifiedTreeNode.counter += 1

    @property
    def backuped(self) -> bool:
        """Full KV 数据已备份到主机（根据 host_value 是否非空判断）"""
        return self.component_data[ComponentType.FULL].host_value is not None

    @property
    def evicted(self) -> bool:
        """Full KV 已被从设备驱逐（非根节点且 value 为空）"""
        return (
            self.parent is not None
            and self.component_data[ComponentType.FULL].value is None
        )
```

```
class UnifiedLRUList:
    def __init__(
        self,
        component_type: ComponentType,
        tree_components: tuple[ComponentType, ...],
        use_host_ptr: bool = False,
    ):
        self.component_type = component_type
        # 指针槽位计算：主机 LRU 使用偏移 slot，避免与设备 LRU 指针冲突
        self._pt: int = component_type + (
            _NUM_COMPONENT_TYPES if use_host_ptr else 0
        )
        self.head = UnifiedTreeNode(tree_components)
        self.tail = UnifiedTreeNode(tree_components)
        self.head.lru_next[self._pt] = self.tail
        self.tail.lru_prev[self._pt] = self.head
        self.cache: dict[int, UnifiedTreeNode] = {}
```

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

Mamba 组件，在匹配验证器中增加主机值检查，在 finalize_match_result 中处理主机命中长度以触发加载回。

```
from sglang.srt.mem_cache.hicache_storage import PoolName, PoolTransfer
from sglang.srt.mem_cache.unified_cache_components.tree_component import (
    CacheTransferPhase,
    ComponentType,
    EvictLayer,
    TreeComponent,
    get_and_increase_time_counter,
)

class MambaComponent(TreeComponent):
    component_type = ComponentType.MAMBA

    def __init__(self, cache, params):
        super().__init__(cache, params)
        self.enable_mamba_extra_buffer = params.enable_mamba_extra_buffer
        self._mamba_pool_host = None # HiCache 启用时保存主机 mamba pool

    def create_match_validator(self):
        ct = self.component_type
        # HiCache：被驱逐但有主机备份的节点同样作为有效匹配边界
        return lambda node: (
            node.component_data[ct].value is not None
            or node.component_data[ct].host_value is not None
        )

    def finalize_match_result(self, result, params, value_chunks, best_value_len):
```

```
        # ... 省略 Copy-on-Write 部分 ...
        # HiCache：如果 mamba 值在设备上被驱逐但主机上有备份，
        # 则确保 host_hit_length >= 1 以触发后续加载回
        host_node = result.last_host_node
        cd = host_node.component_data[self.component_type]
        if cd.value is None and cd.host_value is not None:
            result = result._replace(
                host_hit_length=max(result.host_hit_length, 1)
            )
        return result._replace(mamba_branching_seqlen=branching_seqlen)
```

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

全量组件，新增主机层驱逐支持，重构 redistribute_on_node_split 以复制 host_value。

```
import heapq
from sglang.srt.mem_cache.hicache_storage import PoolName, PoolTransfer
from sglang.srt.mem_cache.unified_cache_components.tree_component import (
    CacheTransferPhase,
    ComponentType,
    EvictLayer,
    TreeComponent,
)

class FullComponent(TreeComponent):
    component_type = ComponentType.FULL

    def __init__(self, cache, params):
        super().__init__(cache, params)
        allocator = cache.token_to_kv_pool_allocator
        # 当存在 SWA 时，只释放 full-attention KV
        if ComponentType.SWA in cache.tree_components:
            self._free_full = allocator.full_attn_allocator.free
        else:
            self._free_full = allocator.free
        self._full_kv_pool_host = None # HiCache 启用时保存主机 KV pool

    def evict_component(self, node, target=EvictLayer.DEVICE):
        cd = node.component_data[self.component_type]
        freed = 0
        host_freed = 0

        # 设备层驱逐
        if EvictLayer.DEVICE in target and cd.value is not None:
            self._free_full(cd.value)
            freed = len(cd.value)
            self.cache.component_evictable_size_[self.component_type] -= freed
            # 注意：cd.value = None 延迟到 _cascade_evict，
            # 因为 SWA 的 free 仍需读取 Full.value
```

```
        # 主机层驱逐
        if EvictLayer.HOST in target and cd.host_value is not None:
            host_freed = len(cd.host_value)
            if self._full_kv_pool_host is not None:
                self._full_kv_pool_host.free(cd.host_value)
            cd.host_value = None
        return freed, host_freed

    def redistribute_on_node_split(self, new_parent, child):
        ct = self.component_type
        new_parent.component_data[ct].lock_ref = child.component_data[ct].lock_ref
        child_cd = child.component_data[ct]
        split_len = len(new_parent.key)
        # 分割设备值
        if child_cd.value is not None:
            new_parent.component_data[ct].value = child_cd.value[:split_len].clone()
            child_cd.value = child_cd.value[split_len:].clone()
        # 分割主机值（HiCache）
        if child_cd.host_value is not None:
            new_parent.component_data[ct].host_value = child_cd.host_value[
                :split_len
            ].clone()
            child_cd.host_value = child_cd.host_value[split_len:].clone()
```

评论区精华

核心讨论：

- Mooncake 后端兼容性：riZZZhik 指出 UnifiedTreeNode 缺少 get_last_hash_value 方法，且 UnifiedRadixCache 缺少预取相关方法（prefetch_from_storage、check_prefetch_progress），会在 Mooncake HiCache 后端下报错。作者 hzh0425 回应“本 PR 仅支持 L2 HiCache，L3 方法将在后续 PR 中添加”。
- CI 环境跳过测试：ispobock 在代码审查中建议暂时移除 CUDA13 环境下不兼容的 HiCache 测试，待修复后再恢复。作者同意并在注释中标记 TODO。
- 精度验证：linjianyu233 提交了 GSM8K 精度测试通过的结果；icepoint666 验证了 Qwen3.5-397B-A17B 模型在 L1+L2 HiCache 下的稳定性和正确性。
  - Mooncake 后端兼容性（缺少 get_last_hash_value 和 prefetch 方法） (correctness): hzh0425 回应本 PR 仅支持 L2 HiCache，L3 方法将在后续 PR 中添加。
  - CI CUDA13 环境兼容性 (testing): 作者同意，添加了 TODO 注释并跳过测试，待后续修复。

风险与影响

- 风险：
  1. Mooncake 后端兼容性风险：当前 PR 未实现 get_last_hash_value 和 prefetch 方法，与 Mooncake 后端的完整集成需要等待 L3 支持 PR。若在 L3 环境下运行会直接报错。

2. CI 环境覆盖不足：CUDA13 环境下的 HiCache 测试被跳过，可能导致回归问题未能及时发现。
3. SWA 组件功能缺失：swa_component.py 的 evict_component 在 target=HOST 时直接返回 (0, 0)，暂不支持主机层缓存，TODO 已标注，但未来可能成为性能瓶颈。
4. 核心驱逐路径变更：驱逐逻辑从单层扩展为双层，若主机层管理不当可能引入内存泄漏或死锁，新增的单元测试已经覆盖，仍需生产环境验证。

- 影响：
  - 用户：使用 UnifiedRadixTree 的混合模型用户可直接受益，减少设备内存占用，提升缓存命中率和吞吐量。需要启用 --enable-hierarchical-cache 等参数。
  - 系统：缓存子系统新增主机层生命周期管理，需要额外主机内存，但可通过 SGLANG_MM_FEATURE_CACHE_MB 等环境变量控制。调度器需要处理新引入的 Host 缓存状态，对现有调度流程有少量侵入。
  - 团队：后续需完成 SWA HiCache 支持（#23391）和 L3 预取功能。CI 环境需跟进 CUDA13 兼容性修复。
  - 风险标记：Mooncake 接口缺失，CUDA13 兼容跳过测试，SWA 无主机层，核心驱逐路径变更

关联脉络

- PR #23391 [SWA HiCache support]: PR body 中标记为已完成的后续 SWA HiCache 支持，与本 PR 紧密关联。
- PR #23216 Reclaim L2 host pool slots on flush_cache: 提交历史中显示为该 PR 的依赖修复，确保主机池槽位在缓存刷新时正确回收。