PR #35269 完整报告

sgl-project/sglang

[UnifiedTree] feat: support runtime attach/detach for hisotrage

合并时间：2026-08-19 22:49

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35269

执行摘要

- 一句话：统一 radix 树支持运行时存储 attach/detach
- 推荐动作：值得精读。重点关注 StorageAttachment 的组件化拆分、backfill_missing_hash_values 的哈希链一致性设计，以及 15 场景状态机测试的组织方式；建议后续跟进修复两个 P2 配置原子性问题。

功能与动机

UnifiedRadixCache 的 attach_storage_backend / detach_storage_backend 原本是 stub 总是返回 False，导致 /hicache/storage-backend 管理 API 只对 HiRadixCache 生效；在 #35081 将统一 radix 树设为默认后，运行时存储挂载成为刚需。PR body 明确说明是“Preceding fix for #35081”，并取代了并行开发的 #35240。

实现拆解

1. 新增 python/sglang/srt/mem_cache/unified_cache/storage_attachment.py：StorageAttachment 组件统一承载策略校验、extra-config 解析、controller attach、teardown 与 apply_runtime_config()，让启动、管理 API、atexit 三个入口共用一套实现；UnifiedRadixCache 从 2764 行降至 2520 行。
2. 改造 unified_radix_cache.py：将两个 stub 替换为对 StorageAttachment.attach/detach 的委托；新增 shutdown() 并注册 atexit；新增 drain_storage_control_queues_local() 供 teardown 清理；_revoke_pending_prefetch 公开为 revoke_pending_prefetch。
3. 哈希链回填：unified_tree_core.py 新增 backfill_missing_hash_values()，父先子后遍历，为 storage 未开启时构建的节点补齐页哈希链，避免 L3 key 只编码后缀导致错误命中；unified_tree_core_interface.py 同步声明抽象方法。
4. 修复缓存控制器：managers/cache_controller.py 将存储控制队列在 __init__ 初始化为 None，使未 attach 时的 detach 成为干净 no-op。
5. 测试配套：15 场景 attach/detach 状态机（单进程 HTTP、逐步骤断言）、TestUnifiedRadixCacheStorageAttachBackfill 固定哈希链契约、test_hicache_storage_runtime_attach_detach.py 通过子类复用 HiRadixCache 生命周期覆盖统一树。

关键文件：

- python/sglang/srt/mem_cache/unified_cache/storage_attachment.py（模块 存储挂载；类别 source；类型 core-logic；符号 StorageAttachment, attach, detach, shutdown）：PR 的核心新组件，集中实现存储后端 attach/detach 生命周期，是管理 API 在统一树上可用的关键。
- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 统一树缓存；类别 source；类型 dependency-wiring；符号 _revoke_pending_prefetch, revoke_pending_prefetch, shutdown, drain_storage_control_queues_local）：统一 radix 缓存主类，替换 stub、委托 StorageAttachment、注册 atexit shutdown，是功能真正接入的载体。
- python/sglang/srt/mem_cache/unified_cache/unified_tree_core.py（模块 树核心；类别 source；类型 core-logic；符号 backfill_missing_hash_values）：新增 backfill_missing_hash_values 哈希链回填，是本 PR 最核心的正确性修复，防止 L3 key 错误别名。
- python/sglang/srt/mem_cache/unified_cache/unified_tree_core_interface.py（模块 树接口；类别 source；类型 core-logic；符号 backfill_missing_hash_values）：在 TreeCore 接口抽象 backfill_missing_hash_values，确保所有树实现支持统一挂载语义。
- test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py（模块 缓存单测；类别 test；类型 test-coverage；符号 TestUnifiedRadixCacheStorageAttachBackfill, _build_two_level_tree, _hashes_by_token_ids, test_backfill_reproduces_hashing_from_the_start）：单元测试用两层树固定“回填链 = 自始哈希链”的契约，并通过变异验证测试有效性。
- test/registered/hicache/test_hicache_storage_runtime_attach_detach.py（模块 端到端测试；类别 test；类型 test-coverage；符号 _check_attach_detach_lifecycle, TestUnifiedRadixCacheStorageRuntimeAttachDetach, test_runtime_attach_detach）：端到端测试通过子类复用同一套 attach/detach 生命周期流程，保证统一树与 HiRadixCache 行为一致。
- python/sglang/srt/managers/cache_controller.py（模块 缓存控制器；类别 source；类型 bugfix）：修复从未 attach 时 detach 抛 AttributeError 的问题，属于基础健壮性修复。

关键符号：StorageAttachment.attach, StorageAttachment.detach, StorageAttachment.shutdown, StorageAttachment.clear, StorageAttachment.apply_runtime_config, UnifiedRadixCache.revoke_pending_prefetch, UnifiedRadixCache.shutdown, UnifiedRadixCache.drain_storage_control_queues_local, UnifiedTreeCore.backfill_missing_hash_values

关键源码片段

python/sglang/srt/mem_cache/unified_cache/storage_attachment.py

PR 的核心新组件，集中实现存储后端 attach/detach 生命周期，是管理 API 在统一树上可用的关键。

```
# python/sglang/srt/mem_cache/unified_cache/storage_attachment.py
class StorageAttachment:
    # Attach / detach the storage backend of one UnifiedRadixCache.
```

```
def __init__(self, cache: UnifiedRadixCache):
    self._cache = cache

def attach(self, storage_backend, storage_backend_extra_config_json=None, served_model_
name=None,
           hicache_storage_prefetch_policy=None, hicache_write_policy=None):
    # 启用存储后端：先校验策略，再启动 controller 的存储线程。
    cache = self._cache

    # 校验必须在任何副作用之前：拒绝的请求必须无痕
    invalid = self._validate_policies(hicache_storage_prefetch_policy, hicache_write_policy)
    if invalid is not None:
        return False, invalid

    controller = cache.cache_controller
    if controller is None:
        # 未启用 --enable-hierarchical-cache 时无法挂载
        return False, 'HiCache is not initialized; launch with --enable-hierarchical-cache'

    if cache.enable_storage:
        current_backend = controller.storage_backend_type
        if current_backend != storage_backend:
            # 不同后端必须先 detach，避免静默切换
            return False, f'HiCache storage backend is already enabled with {current_backend};
            detach first'
        # 同后端：请求退化为策略更新（注意 extra_config_json 在此分支被忽略）
        self._apply_policies(hicache_storage_prefetch_policy, hicache_write_policy)
        return True, 'HiCache storage backend already enabled with same backend; policies
        updated.'

    # 先应用策略，让存储线程启动时就能读到新值（失败路径会留下已生效的策略，见 Codex P2）
    self._apply_policies(hicache_storage_prefetch_policy, hicache_write_policy)

    try:
        (extra_config, prefetch_threshold, prefetch_timeout_base, prefetch_timeout_per_ki_
        token,
         hicache_storage_pass_prefix_keys) = HybridCacheController.parse_storage_backend_
         extra_config(
             storage_backend_extra_config_json)
    except Exception as e:
        return False, f'failed to parse storage_backend_extra_config_json: {e}'

    try:
        controller.attach_storage_backend(
            storage_backend=storage_backend,
            prefetch_threshold=prefetch_threshold,
            model_name=served_model_name,
            storage_backend_extra_config=extra_config,
            host_pools=controller.mem_pool_host.entries,
```

```
            )
        except Exception as e:
            return False, f'failed to attach storage backend: {e}'

        cache.enable_storage = True
        return True, 'storage backend attached'
```

python/sglang/srt/mem_cache/unified_cache/unified_tree_core.py

新增 backfill_missing_hash_values 哈希链回填，是本 PR 最核心的正确性修复，防止 L3 key 错误别名。

```
# python/sglang/srt/mem_cache/unified_cache/unified_tree_core.py
def backfill_missing_hash_values(self) -> int:
    # 为存储禁用期构建的节点补齐页哈希链。
    # 页哈希以父节点最后一个哈希为链头，若父节点无哈希，子节点会从半路重启
    # 链条，导致 L3 key 只编码真实前缀的后缀，进而错误别名无关请求。
    # 父先子后遍历保证每个节点都基于已填充的父节点计算。
    filled = 0
    stack = [self.root_node]
    while stack:
        node = stack.pop()
        # 根节点锚定所有链，本身不需要填充（预置空哈希链）
        if node is not self.root_node and node.hash_value is None:
            node.hash_value = compute_node_hash_values(node, self.page_size)
            filled += 1
        stack.extend(node.children.values())
    return filled
```

test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py

单元测试用两层树固定“回填链 = 自始哈希链”的契约，并通过变异验证测试有效性。

```
# test/registered/unit/mem_cache/test_unified_radix_cache_unittest.py
class TestUnifiedRadixCacheStorageAttachBackfill(CustomTestCase):
    # 回填必须让哈希链与“一开始就开启存储”完全一致。

    cfg = CacheConfig(page_size=4, components=(ComponentType.FULL,), kv_size=64, max_
    context_len=64)
    prefix_tokens = array('q', [1, 2, 3, 4, 5, 6, 7, 8])
    suffix_tokens = array('q', [9, 10, 11, 12])

    def _build_two_level_tree(self, *, storage_on_from_the_start):
        cache, allocator, _ = build_fixture(self.cfg)
        cache.enable_storage = storage_on_from_the_start
        for tokens in (self.prefix_tokens, self.prefix_tokens + self.suffix_tokens):
            value = allocator.alloc(len(tokens))
            self.assertIsNotNone(value)
            cache.insert(InsertParams(key=RadixKey(tokens), value=value))
        return cache

    @staticmethod
```

```
    def _hashes_by_token_ids(cache):
        root = cache.tree_core.root_node
        hashes = {}
        stack = [root]
        while stack:
            node = stack.pop()
            if node is not root:
                hashes[tuple(node.key.token_ids)] = node.hash_value
            stack.extend(node.children.values())
        return hashes

    def test_backfill_reproduces_hashing_from_the_start(self):
        expected = self._hashes_by_token_ids(
            self._build_two_level_tree(storage_on_from_the_start=True))
        self.assertEqual(len(expected), 2, 'expected a parent and a child node')
        late = self._build_two_level_tree(storage_on_from_the_start=False)
        self.assertTrue(all(h is None for h in self._hashes_by_token_ids(late).values()),
                        'nodes built while storage was disabled must start unhashed')
        self.assertEqual(late.tree_core.backfill_missing_hash_values(), len(expected))
        self.assertEqual(self._hashes_by_token_ids(late), expected,
                         'a backfilled chain must be identical to one hashed from the start')
```

评论区精华

> Codex P2（未解决）：_apply_policies 在 extra-config 解析与 attach 成功之前调用，失败后 controller 的策略、write-through 阈值与 tree_core.is_write_back 已被修改，后续请求会运行在被拒绝请求的策略下；建议延迟到解析和 attach 成功后变更，或恢复原值。Codex P2（未解决）：同后端 attach 时若 storage_backend_extra_config_json 不同，分支返回成功但不应用新配置，导致 GET /hicache/storage-backend 报告未生效的配置；建议拒绝或保留原配置。 两条评论均指向配置一致性问题，合并版本中未修复，属于已知边界行为。

- attach 失败路径会污染已生效策略 (correctness): 合并版本中 _apply_policies 仍在解析前调用，问题保留，属于已知边界行为。
- 同后端 attach 时 extra config 被静默忽略 (correctness): 合并版本中该分支仍直接返回成功并仅更新策略，extra config 未处理。

风险与影响

- 风险：
  1. 配置原子性：StorageAttachment.attach 中 _apply_policies 早于 parse_storage_backend_extra_config 与 controller.attach_storage_backend，失败后 controller 策略状态被污染，调度器与后续请求可能运行在错误策略下，影响 python/sglang/srt/mem_cache/unified_cache/storage_attachment.py。
  2. 同后端配置歧义：attach 已启用的同后端但不同 extra config 时返回成功却未应用，管理 API 展示与实际不一致（同一文件）。

3. 核心路径变更：unified_radix_cache.py、unified_tree_core.py 是缓存核心，回归影响面大；测试覆盖单进程与 --tp-size 2，PP/CP 混合规模未验证。
4. atexit 竞态：shutdown() 注册 atexit，进程异常退出时可能与进行中的 attach/detach 竞争，需要线上观察。 - 影响：对管理员：/hicache/storage-backend 管理 API 现可作用于统一 radix 树，实现 L3 存储热挂载 / 卸载，为 #35081 默认树铺路；对系统：UnifiedRadixCache 职责更聚焦，存储生命周期集中到 StorageAttachment；对团队：需要留意两个 P2 配置一致性边界，避免线上误用。 - 风险标记：核心缓存路径变更，失败路径策略污染，同后端配置歧义，atexit 竞态

关联脉络

- PR #35240 Support runtime attach for UnifiedCache: 并行开发的同一功能 PR，被本 PR 取代（body 声明 Supersedes #35240）。
- PR #35081 Using unified radix tree by default for all case: 本 PR 是其前置修复，为统一 radix 树默认化补齐运行时存储挂载能力。