# PR #37151 完整报告

sgl-project/sglang

[Unified Cache Linker][3/N]: Add backend-independent linker core

合并时间：2026-08-31 14:07

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37151

## 执行摘要

- 一句话：新增后端无关外部缓存链路核心并接入缓存树
- 推荐动作：值得精读。重点看三处设计：① _sync_restorable_prefix 用 0/1 mask + MIN all-reduce 求跨 rank 稀疏集合交集，避免“取局部最大值”落点不可恢复；② UnifiedCacheLinkerWrapper 的「排队 + 异步完成」传输协议与锁 pin/unpin 机制，兼顾按层加载与异常回滚；③ check_hicache_events 的双轨事件轮询与卸载结果二次归约（任一 rank 失败整体回滚）。建议后续 backend PR 合入前补充一次真实多卡端到端验证（含 DMA 失败注入）。

## 功能与动机

PR body 明确说明这是从 #35687 拆出的第三块，目标是 "Add the backend-independent external linker interface and tree-side wrapper"，并把 "direct load/offload lifecycle hooks" 集成进 UnifiedRadixCache。动机层面：给 UnifiedRadixCache 提供直连外部 KV 存储（L3）的能力——不经过 host tier，直接从设备池对接远端 KV 存储，以支持更大规模的 KV 复用与跨节点缓存；同时通过把巨型 PR 拆成多块（#37091、#37098 与本 PR）降低评审与回滚风险，让契约与后端实现解耦。

## 实现拆解

1. 新增传输契约与树侧包装层（unified_cache_linker.py，+571 行） - UnifiedCacheLinker(ABC) 定义 11 个抽象方法：lookup、load、start_layer_wise_loading、cancel_queued_load、num_completed_loads、pop_completed_load、offload、num_completed_offloads、pop_completed_offload、reset、close。 - 契约采用「排队 + 异步完成」模型：load 只入队，真正的传输由下一次 start_layer_wise_loading 触发并返回 layer-counter 消费者索引，这是为配合按层推进的 CUDA graph 或流水线约束而设计。 - UnifiedCacheLinkerWrapper 是树侧驱动，持有 hit_markers、pending_loads、pending_offloads 三个状态容器；构造时把 tree_core.enable_external_cache_linker 置 True、write_through_threshold 置 1，完成「挂载即切模式」。
2. 生命周期接入（unified_radix_cache.py，+52/-1） - 新增 self.linker: Optional[UnifiedCacheLinkerWrapper] = None 属性与 init_cache_linker() 入口。 - reset() 与 release_host_resources() 分别补充 linker.reset()、linker.close() 调用，保证后端先静默传输、再释放资源。

3. 热路径守卫钩子 - match_prefix() 在常规树匹配与 finalizer 之后追加 linker.match()，把外部可恢复长度合并进 host_hit_length 并暂存 hit_markers。 - init_load_back() 开头用 linker.has_hit(rid) 分流，命中则走 linker.load_back() 直接回灌并插入树。 - _apply_cache_action() 的 BackupKV 分支在挂载 linker 时改由 linker.offload_nodes() 异步持久化；ReplaceWriteThroughOnNodeSplit 分支同步把在途卸载重定向到拆分后的节点。 - release_aborted_request() 增加 linker.release_request(rid) 清理排队加载并解锁。
4. 事件轮询分流 - check_hicache_events() 在挂载 linker 时走独立分支：先 MIN all-reduce 归约各 rank 的 load/offload 完成数，再 drain_loads() 与 commit_completed_offloads()；卸载成功位图二次all-reduce，任一rank失败即整体回滚。该分支直接 return，绕开原有 HiCache 控制队列。 - ready_to_load_host_cache() 把启动权交给 linker.start_layer_wise_loading()。
5. 测试配套 - 新增 test_unified_cache_linker.py（+447 行）：_FakeLinker 用内存态实现全部抽象方法，覆盖挂载开关、稀疏集合交集、异步加载 / 卸载的锁 pin/unpin、失败卸载回滚拆分碎片、split action 重定向、reset 静默顺序、cancel 排队加载等 9 个用例。 - test_hiradix_pp_sync_drain.py 的 _make_cache 夹具补 cache.linker = None，适配新属性。

关键文件：

- python/sglang/srt/mem_cache/unified_cache/unified_cache_linker.py（模块 缓存链路；类别 source；类型 core-logic；符号 UnifiedCacheLinker, UnifiedCacheLinkerWrapper, ExternalCacheHitMarker, lookup）：本 PR 的核心新文件：定义后端无关的 UnifiedCacheLinker 传输契约（lookup/load/offload 等 11 个抽象方法）与树侧驱动 UnifiedCacheLinkerWrapper（match 探测、load_back 回灌、offload 原子持久化、锁 pin 管理）。所有后续外部 KV 存储后端都将基于此接口实现。
- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 缓存核心；类别 source；类型 core-logic；符号 init_cache_linker, match_prefix, init_load_back, check_hicache_events）：外部缓存链路与既有统一缓存树的唯一接入点：新增 linker 属性、init_cache_linker 入口，并在 match_prefix、init_load_back、_apply_cache_action（BackupKV / ReplaceWriteThroughOnNodeSplit）、release_aborted_request、check_hicache_events、ready_to_load_host_cache 六处热路径挂载守卫钩子。改动虽小但影响面广，决定外部缓存何时接管写穿与事件轮询。
- test/registered/unit/mem_cache/test_unified_cache_linker.py（模块 链路测试；类别 test；类型 test-coverage；符号 _FakeLinker, test_cache_linker_attachment_is_backend_independent, test_restorable_prefix_intersects_sparse_rank_results, test_async_offload_pins_node_until_completion）：用 _FakeLinker 在内存中模拟后端，覆盖挂载开关、稀疏集合交集、异步加载 / 卸载锁 pin/unpin、失败卸载回滚拆分碎片、split action 重定向、reset 静默顺序、cancel 排队加载等关键状态机，是接口契约唯一的行为级文档。
- test/registered/unit/mem_cache/test_hiradix_pp_sync_drain.py（模块 PP 同步；类别 test；类型 test-coverage；符号 _make_cache）：夹具补 cache.linker = None，适配 UnifiedRadixCache 新属性，防止既有 PP 同步测试因新属性缺失而失败。

关键符号：UnifiedCacheLinker.lookup, UnifiedCacheLinker.load, UnifiedCacheLinker.start_layer_wise_loading, UnifiedCacheLinker.offload, UnifiedCacheLinkerWrapper.match, UnifiedCacheLinkerWrapper._sync_restorable_prefix, UnifiedCacheLinkerWrapper.load_back, UnifiedCacheLinkerWrapper.offload_nodes, UnifiedCacheLinkerWrapper.release_request, UnifiedRadixCache.init_cache_linker, UnifiedRadixCache.check_hicache_events

## 关键源码片段

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

外部缓存链路与既有统一缓存树的唯一接入点：新增 linker 属性、init_cache_linker 入口，并在 match_prefix、init_load_back、_apply_cache_action（BackupKV / ReplaceWriteThroughOnNodeSplit）、release_aborted_request、check_hicache_events、ready_to_load_host_cache 六处热路径挂载守卫钩子。改动虽小但影响面广，决定外部缓存何时接管写穿与事件轮询。

```
# unified_radix_cache.py -- 外部缓存链路的树侧接入点
# 原则：所有 linker 钩子都以 self.linker is not None 守卫，
# 默认（未挂载）路径与原先完全一致，避免影响现有 HiCache 行为。

def init_cache_linker(self, cache_linker: UnifiedCacheLinker) -> None:
    """把外部 KV 存储直连到设备池上。"""
    # 树侧只保存包装层，主树文件不感知具体后端；
    # 包装层构造时会自动打开树侧开关并压低写穿阈值。
    self.linker = UnifiedCacheLinkerWrapper(self, cache_linker)

def match_prefix(self, params: MatchPrefixParams) -> MatchResult:
    result = self.tree_core.match_prefix(params)
    self._apply_cache_actions(result.cache_actions)
    for component in self._components_tuple:
        result = component.finalize_match_result_in_cache(params, result)
    # 常规 match 结束后，再交给 linker 探测外部存储的可恢复尾部
    if self.linker is not None and params.req is not None:
        result = self.linker.match(params.key, params.req, result)
    return result

def _apply_cache_action(self, action: CacheAction | ComponentAction) -> None:
    # ... 其余 action 分支保持不变 ...
    elif isinstance(action, BackupKV):
        # 挂载 linker 后，写穿语义由外部存储异步接管；
        # 未挂载时退回原有 HiCache D→H backup 路径
        if self.linker is not None:
            self.linker.offload_nodes(action.node_ids)
        else:
            self._execute_and_commit_kv_backup(action)

def check_hicache_events(self) -> None:
    """每个 scheduler step 轮询异步事件；linker 路径在此分流。"""
```

```
if self.linker is not None:
    # 先用 MIN 归约各 rank 的完成数，保证所有 rank 消费一致的
    # 批次数量，避免部分 rank 提前消费造成跨 rank 不一致
    finish_counts = torch.tensor(
        [
            self.linker.num_completed_loads(),
            self.linker.num_completed_offloads(),
        ],
        dtype=torch.int,
        device="cpu",
    )
    self._all_reduce_attn_groups(finish_counts, torch.distributed.ReduceOp.MIN)
    load_count, offload_count = map(int, finish_counts.tolist())
    self.linker.drain_loads(load_count)
    local_successes = self.linker.take_completed_offloads(offload_count)
    if local_successes:
        # 卸载结果再归约一次：只要有一个 rank 失败就整体回滚
        successes = torch.tensor(local_successes, dtype=torch.int, device="cpu")
        self._all_reduce_attn_groups(successes, torch.distributed.ReduceOp.MIN)
        self.linker.commit_completed_offloads(
            [bool(success) for success in successes.tolist()]
        )
    return # linker 路径完成后直接返回，不再走 HiCache 控制队列
# ... 原有 HiCache 事件处理逻辑 ...
```

## 评论区精华

本 PR 没有 review 评论，reviewer huangtingwei9988 直接 APPROVED，说明接口与树侧接入方案在内部评审中无争议。实际讨论集中在 issue 评论区的 CI 复跑：

- 作者两次 /rerun-test：test_unified_cache_linker.py 与 test_hiradix_pp_sync_drain.py 在 ubuntu-latest 均通过。
- 作者一次 /rerun-group radix_cache/unified_radix_tree：在 4-gpu-h100、4-gpu-b200、8-gpu-h200 上共 8 组多卡 unified radix cache 用例全部通过，验证了跨 rank MIN 归约在真实多卡环境下的正确性。
- 未决事项：PR body 中 AMD ROCm 7.2 CI 运行失败（Run #33325476754），评论中没有任何说明，无法确认是否与本次变更相关。
- CI 复跑验证单测与多卡回归 (testing): ubuntu-latest 单测与多卡 H100/B200/H200 回归全部通过，跨 rank MIN 归约逻辑在真实多卡环境验证正确。
- AMD ROCm 7.2 CI 失败未澄清 (other): 无法确认是否与本次变更相关，建议后续 PR 跟进澄清。
- Review 直接通过 (other): 已合入，无遗留 review 意见。

## 风险与影响

- 风险：

1. 热路径分支增加：match_prefix、init_load_back、check_hicache_events、_apply_cache_action、release_aborted_request 都新增了守卫分支。缺省 linker=None 时零开销，但 check_hicache_events 的 linker 分支直接 return，意味着挂载后原有 HiCache 事件轮询（ack_write_queue、ack_load_queue）整体停用，未来若出现「外部存储 + host tier」共存需求需显式校验互斥。
2. 锁生命周期依赖异常处理：pending_loads、pending_offloads 通过 inc_lock_ref、dec_lock_ref 机制 pin 住节点，任何异常路径漏调 dec_lock_ref 都会泄漏锁；测试覆盖了 cancel 与失败回滚，但真实后端 DMA 失败路径尚无验证。
3. 跨 rank 一致性假设：_sync_restorable_prefix 与 check_hicache_events 依赖 _all_reduce_attn_groups 的 MIN 归约并要求各 rank 同步调用，PP 长流水线或 CP 组合下的额外同步等待需后续性能评估。
4. 全局副作用：UnifiedCacheLinkerWrapper.__init__ 无条件把 write_through_threshold 改为 1，是对共享 cache 对象的全局改写。
5. CI 缺口：AMD ROCm 7.2 运行失败未澄清；本 PR 无真实后端接入，接口稳定性有待后续 backend PR 验证。 - 影响：对用户：无直接可见功能变化——linker 缺省为 None，所有钩子短路，现有 HiCache 路径行为不变。对系统：UnifiedRadixCache 核心热路径新增 8 处守卫分支，后续所有 cache 相关修改都需同时考虑“挂载 / 未挂载”两种模式；check_hicache_events 形成双轨事件轮询。对团队：这是外部缓存链路系列的关键基座，为后续真实后端落地提供稳定接口，_FakeLinker + 多卡回归构成可靠安全网，可支撑后续并行开发。影响程度：中，属于基础设施分层改动，非线上行为变更。 - 风险标记：热路径新增守卫分支，check_hicache_events 双轨互斥，锁生命周期依赖异常处理，跨 rank 一致性依赖 all-reduce，AMD ROCm CI 失败未澄清，缺真实后端端到端验证

## 关联脉络

- PR #35687 原始大 PR（本 PR 的拆分来源）：本 PR body 明确说明是 #35687 拆出的第 3 块，主题完全一致。后续 block 都将沿此主线合入。
- PR #37091 前序拆分（第 1 块）：PR body 标注为本 PR 的 previous split，同属 unified cache linker 系列。
- PR #37098 前序拆分（第 2 块）：PR body 标注为本 PR 的 previous split，同属 unified cache linker 系列。
- PR #36798 [HiCache] Align chunked CUDA host registrations: 同属 mem_cache / hicache 存储链路演进，共享 memory pool 与 host 注册基础设施，本 PR 的 linker 路径与其并存。
- PR #36958 [mem_cache] Keep req.kv non-optional and key KV ownership on req_pool_idx: 对 UnifiedRadixCache 与调度器所有权建模的重构，与本 PR 修改同一文件域，是 external linker 落地的依赖基础。