

PR #38350 完整报告

sgl-project/sglang

[HiCache] Fix side pools to use resolved host allocator

合并时间: 2026-09-10 14:28

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38350>

执行摘要

- 一句话: HiCache 侧池改用解析后的 host allocator 类型
- 推荐动作: 值得精读, 但重点不在 diff 本身 (只有两行), 而在于理解三个问题: (1) `_get_allocator_type()` 与 `get_memory().hicache_storage_backend` 的语义差别, 以及 dynamic backend 的解析契约; (2) 为什么 full/SWA host pool 通过 `build_kv_host_pool` 天然正确, 而 Mamba 与 INDEXER 池需要显式改参; (3) HiCache 中“锚点池 + 侧池”必须共享同一 host 分配器的隐含约束。建议后续跟进: 把 allocator 解析下沉到 `MambaPoolHost / MHATokenToKOnlyPoolHost` 的构造默认值, 或在 `build_pool_entry` 层做断言校验; 同时补一个不 `patch _get_allocator_type` 的集成用例, 真正验证 shm 分配路径, 并确认 Extra / AMD ROCm 7.2 两个失败任务与本改动无关。

功能与动机

PR body 给出的两个目标就是动机本身: route Mamba+SWA HiCache side-pool host allocation through the resolved allocator type, 以及 route MiniMax sparse index host allocation through the resolved allocator type。HiCache 允许把 `hicache_storage_backend` 设为 dynamic 这类“需要再解析”的 backend, 真实主机内存分配器写在 `hicache_storage_backend_extra_config` (例如 `'{"allocator": "shm"}'`) 里; 主路径 `build_kv_host_pool` 已经通过 `_get_allocator_type()` 完成解析, 但侧池构造仍是 `allocator_type=get_memory().hicache_storage_backend`, 等于把 backend 名当成分配器名使用。这会让 Mamba host pool、MiniMax INDEXER host pool 与其所属的主 KV host pool 采用不同的 host 内存分配器, 属于隐蔽的一致性缺陷。关联 Issue 列表为空, 问题由代码审查 / 自测发现。

实现拆解

1. 变更入口与范围: 唯一改动文件是 `python/sglang/srt/mem_cache/hybrid_cache/hybrid_pool_assembler.py`, 共 2 处、+2/-2, 全部落在 host pool 构造的 `allocator_type=` 实参上, 没有新增函数、没有签名变化、没有配置项增减。
2. 混合 Mamba + SWA 栈对齐: `build_hybrid_mamba_swa_stack` 中构造 `MambaPoolHost` 时, `allocator_type` 由 `get_memory().hicache_storage_backend` 改为 `_get_allocator_type()`。该栈同时存在 full KV、SWA、Mamba 三个 host pool, 其中 full/SWA 由 `build_kv_host_pool` 构建、已使用解析后的类型, 因此这次修改的直接效果是让 Mamba 侧池与同组的主池使用同一个 host 分配器; `hicache_ratio`、

`hicache_mem_layout` 等参数保持不变，容量切分逻辑 `_split_hicache_size` 也不受影响。

3. MiniMax 稀疏栈对齐：`build_minimax_sparse_hicache_stack` 中 `sparse_pool.index_k_pool is not None` 分支构造 `MHATokenToKOnlyPoolHost`（对应 `PoolName.INDEXER` 池）时，同样把 `allocator_type` 改成 `_get_allocator_type()`。该 `host pool` 本身复用 `kv_host_pool` 作为层缓冲载体，分配器类型不一致会在同一块布局内混用两种分配策略，修改后 `INDEXER` 池与其复用的 `KV host pool` 保持同源。
4. 解析语义与一致性契约：`_get_allocator_type()` 承担“backend 名 → 真实 host 分配器名”的解析职责（测试中以 `hicache_storage_backend="dynamic"` + extra config `{"allocator": "shm"}` 反推，解析结果为 `shm`）。修改后的契约是：凡是需要 `host` 分配器的池，都必须经过 `_get_allocator_type()`，不得再直接读 `get_memory().hicache_storage_backend`；这为后续新增侧池提供了明确范式。
5. 测试与 CI 配套：本次 PR 的最终 diff 不含测试文件，只做了 `python3 -m compileall -q` 与 pre-commit（并跳过 `check-registered-tests`，PR body 说明该检查当前会报仓库内已有的 taxonomy 违规）。review 上下文中的 `test/registered/unit/mem_cache/test_hybrid_pool_assembler.py` 与作者手动执行的 `/rerun-test test/registered/unit/mem_cache/test_hybrid_pool_assembler.py` 覆盖了这条路径，CI 返回 1 个测试通过；PR body 中的 CI 状态显示 PR Test (Base) 通过，Extra 与 AMD ROCm 7.2 两个任务失败。

关键文件：

- `python/sglang/srt/mem_cache/hybrid_cache/hybrid_pool_assembler.py`（模块 混合缓存池；类别 source；类型 core-logic；符号 `build_hybrid_mamba_swa_stack`, `build_minimax_sparse_hicache_stack`, `MambaPoolHost`, `MHATokenToKOnlyPoolHost`）：本 PR 唯一改动的源文件，承载 HiCache 混合栈的 `host pool` 装配逻辑；两处 `allocator_type` 实参替换决定了 Mamba / SWA / MiniMax `INDEXER` 侧池使用哪种 `host` 内存分配器，是问题与修复的共同落点。
- `test/registered/unit/mem_cache/test_hybrid_pool_assembler.py`（模块 单元测试；类别 test；类型 test-coverage；符号 `TestHybridHostAllocatorType`, `test_mamba_swa_side_pool_uses_resolved_allocator_type`）：PR 最终 diff 未包含该文件，但 Codex 的 P1 评论与作者手动执行的 `/rerun-test` 都指向这里的 `TestHybridHostAllocatorType`：它以 `hicache_storage_backend="dynamic"` + extra config `{"allocator": "shm"}` 的场景断言侧池走了 `_get_allocator_type()`，是本次修复的主要验证手段（据此判断其属于 PR 早前 revision 的改动）。

关键符号：`build_hybrid_mamba_swa_stack`, `build_minimax_sparse_hicache_stack`, `_get_allocator_type`, `MambaPoolHost`, `MHATokenToKOnlyPoolHost`, `test_mamba_swa_side_pool_uses_resolved_allocator_type`

关键源码片段

`python/sglang/srt/mem_cache/hybrid_cache/hybrid_pool_assembler.py`

本 PR 唯一改动的源文件，承载 HiCache 混合栈的 `host pool` 装配逻辑；两处 `allocator_type` 实参替换决定了 Mamba / SWA / MiniMax `INDEXER` 侧池使用哪种 `host` 内存

分配器，是问题与修复的共同落点。

混合 Mamba + SWA 栈中 Mamba 侧池的构造 (`build_hybrid_mamba_swa_stack`) :

```
def build_hybrid_mamba_swa_stack(
    # ... 省略与本次改动无关的参数
) -> tuple[HostPoolGroup, HybridCacheController]:
    # transfer_layer_num 取 full / SWA / Mamba 三类层映射的并集，控制器按该层数循环搬运 KV
    transfer_layer_num = len(
        full_layer_mapping | swa_layer_mapping | mamba_layer_mapping
    )
    swa_attn_allocator = params.token_to_kv_pool_allocator.swa_attn_allocator
    mamba_allocator = params.req_to_token_pool.mamba_allocator

    # 仅在显式给出 --hicache-size 时按设备池规模比例切分主机池容量，否则沿用默认推导
    kv_host_size, swa_host_size, mamba_host_size = None, None, 0
    if get_memory().hicache_size > 0:
        kv_host_size, swa_host_size, mamba_host_size = _split_hicache_size(
            get_memory().hicache_size, (full_kv_pool, swa_kv_pool, mamba_pool)
        )

    # 锚点池 (full KV) 与 SWA 池统一由 build_kv_host_pool 构建，
    # 该工厂内部已经调用解析逻辑得到真实 host allocator，因此天然正确
    kv_host_pool = build_kv_host_pool(
        kv_pool=full_kv_pool,
        page_size=page_size,
        use_mla=False,
        host_size=kv_host_size,
        pool_label="full",
    )
    swa_host_pool = build_kv_host_pool(
        kv_pool=swa_kv_pool,
        page_size=page_size,
        use_mla=False,
        host_size=swa_host_size,
        pool_label="swa",
    )

    # 修复点：Mamba 侧池此前直接透传 hicache_storage_backend 的原始值，
    # 当 backend 取值为 "dynamic" 这类需二次解析的名字时，拿到的是非法分配器名，
    # 会与上面的 kv_host_pool / swa_host_pool 使用不同的 host 分配器；
    # 现在改为统一走 _get_allocator_type()，与主池保持同源
    mamba_host_pool = MambaPoolHost(
        mamba_pool,
        get_memory().hicache_ratio,
        mamba_host_size,
        allocator_type=_get_allocator_type(),
        layout=get_memory().hicache_mem_layout,
    )
```

MiniMax 稀疏注意力栈中 INDEXER 侧池的构造 (`build_minimax_sparse_hicache_stack`) :

```
# 稀疏场景的主池同样作为锚点池, host 侧由 build_kv_host_pool 负责解析 allocator
main_pool = sparse_pool.main_pool
start_layer = main_pool.start_layer
transfer_layer_num = main_pool.layer_num
# Stage-local keys (0..transfer_layer_num) 对齐控制器的逐层加载循环,
# values 指向 host pool 的本地层缓冲
full_layer_mapping = {layer_id: layer_id for layer_id in range(transfer_layer_num)}

kv_host_pool = build_kv_host_pool(
    kv_pool=main_pool,
    page_size=params.page_size,
    use_mla=False,
)
entries = [
    build_pool_entry(
        name=PoolName.KV,
        host_pool=kv_host_pool,
        device_pool=main_pool,
        layer_mapping=full_layer_mapping,
        transfer_layer_num=transfer_layer_num,
        is_anchor=True,
    ),
]

index_k_pool = sparse_pool.index_k_pool
if index_k_pool is not None:
    # 修复点: INDEXER host pool 复用 kv_host_pool 的层缓冲,
    # 分配器类型必须与主池解析结果一致, 否则同一布局内会混用两种分配策略
    index_host_pool = MHATokenToKOnlyPoolHost(
        index_k_pool,
        kv_host_pool,
        get_memory().hicache_mem_layout,
        allocator_type=_get_allocator_type(),
    )
    entries.append(
        build_pool_entry(
            name=PoolName.INDEXER,
            host_pool=index_host_pool,
            device_pool=index_k_pool,
            layer_mapping={
                gid - start_layer: sub_id
                for gid, sub_id in sparse_pool.index_k_layer_id_mapping.items()
            },
            transfer_layer_num=transfer_layer_num,
        )
    )
)
```

评论区精华

review 讨论很轻量，但每条都指向同一个“对称性”主题。Codex 机器人针对 `test/registered/unit/mem_cache/test_hybrid_pool_assembler.py` 打了 P1 标签：当新注册的 CPU 测试走到 `build_minimax_sparse_hicache_stack` 时，`HybridCacheController` 会读取 `params.pp_cache_group`（对应 `hybrid_pool_assembler.py:1929`），而 fixture 若未定义该属性，测试会在 allocator 断言之前先抛 `AttributeError`，建议补 `pp_cache_group=None`；从评论携带的 hunk 看 fixture 已包含该字段，说明该意见大概率已在后续 revision 处理，也解释了最终 diff 为什么只剩源码两行。作者 `alphabetc1` 随后用 `/rerun-test test/registered/unit/mem_cache/test_hybrid_pool_assembler.py` 定向重跑，机器人回报 `ubuntu-latest` (1 test) 通过。维护者 `huangtingwei9988` 连续两次 APPROVED，评论只有 `lgm`，没有对 allocator 语义本身提出异议。此外不存在关于“是否应该把默认值收敛到 helper”或“是否需要真实 shm 分配的集成测试”的讨论，属于未展开的改进空间。

- 侧池 `allocator_type` 必须使用解析后的结果，而不是 `hicache_storage_backend` 原始值 (design): 两处全部改为 `_get_allocator_type()`，侧池与主池对齐；维护者 `huangtingwei9988` 两次 APPROVED (`lgm`)，无反对意见。
- 新增单元测试的 fixture 缺少 `pp_cache_group` 会导致 `AttributeError (testing)`: 评论携带的 hunk 中 fixture 已包含 `pp_cache_group=None`，且最终 PR diff 不含测试文件，判断该问题已在后续 revision 或独立提交中处理；作者重跑该测试用例并通过 (`ubuntu-latest`, 1 test)。
- CI 覆盖与 `bypass-fastfail` 合并 (other): 定向单测通过，失败任务与本改动的关系未在讨论中澄清，缺少明确结论。

风险与影响

- 风险：
 1. 分配器语义变更的隐性回归：两处改动决定了 Mamba / SWA / INDEXER host pool 实际走哪种 host 内存分配器。若 `_get_allocator_type()` 在非 dynamic 场景下的解析结果与旧代码直接读取 `hicache_storage_backend` 的值不同（例如做了别名归一化或默认值替换），这些侧池的内存类型就会随之变化，可能影响内存占用与跨进程共享语义；提供材料未展示 `_get_allocator_type()` 的实现体，这一点无法完全排除。
 2. 验证强度偏弱：最终 PR 不含测试文件，覆盖依赖既有 / 同期测试用例；而 Codex 的 hunk 显示该用例把 `_get_allocator_type` 直接 patch 成返回 "shm"，属于“验证调用链是否走了解析入口”，并不验证 real shm 分配是否成功，因此对真实配置的回归保护有限。
 3. CI 状态不一致：PR body 显示 Latest PR Test (Extra) 与 Latest PR Test (AMD ROCm 7.2) 均为失败 (🔴)，且 PR 带 `bypass-fastfail` 标签后被合并，无法从提供材料确认失败与本改动无关。
 4. 配置组合窄但语义关键：只有 HiCache + 混合 Mamba/SWA (Qwen3.5 GDN 类模型) 或 MiniMax 稀疏注意力，并且使用需要二次解析的 backend (如 dynamic) 时才会命中，命中概率低但一旦命中就是主机内存分配层面的错误，排查成本高。
 5. 安全面：若旧行为导致进错分配器（例如本该 shm 却走普通 host 内存），仅影响本进程内存分配，未见明显越权或信息泄露路径，安全风险低。 - 影响：用户侧：使用

--hcache-storage-backend dynamic (配合 extra config 指定 allocator) 并运行 Mamba/SWA 混合模型或 MiniMax 稀疏注意力模型的部署, 主机池分配行为改为与主 KV host pool 一致, 避免潜在的分配失败或类型不一致; 不使用这些配置的用户无感知。系统侧: hybrid_pool_assembler.py 中的 host 分配出口完成统一, 减少“主池解析、侧池直读”的不对称模式, 降低后续新增侧池时复现同类 bug 的概率; 对性能无正向或负向的直接影响, 属于正确性修复。团队侧: 这是一个典型的“局部修复掩盖系统性问题”的案例——真正值得沉淀的是把 allocator_type 的解析收敛到统一的注入点 (例如在 hostpool 构造函数内部解析或提供统一默认值), 否则每新增一个池都有再次漏改的风险。- 风险标记: HiCache 侧池分配路径变更, 缺少随 PR 落地的测试改动, 测试默认 patch _get_allocator_type, 真实验证偏弱, CI Extra / AMD 任务失败后 bypass-fastfail 合并, 仅覆盖 dynamic backend 等窄配置组合

关联脉络

- PR #37306 [Rust TreeCore] Support external cache linker: 同属 HiCache / unified cache 内存池演进线, 改动同样落在 python/sglang/srt/mem_cache/ 下的 host 池与 cache linker 抽象; 本 PR 修的“侧池必须与主池共享同一 host 分配器”正是该抽象层的一致性要求。
- PR #33922 Fix Qwen3.5 GDN multi-item scoring: Qwen3.5 GDN 是典型的 Mamba + SWA 混合模型, 正是 build_hybrid_mamba_swa_stack 服务的对象; 该 PR 改动 linear/gdn_backend.py 与 gdn_triton.py, 与本 PR 的 Mamba/SWA 侧池路径构成同一模型线的上下游依赖。
- PR #38844 [Cookbook] DeepSeek-V4.1: add the HiCache L2 knob to the Playground: 近期 HiCache 配置面 (hcache_storage_backend / extra config) 的文档化改动, 与本 PR 修复的 dynamic backend + extra config allocator 解析属于同一配置语义线, 可作为理解用户如何触发该 bug 的背景。