PR #36739 完整报告

sgl-project/sglang

[misc] Fold the allocator free-group flag into `free_group`

合并时间：2026-08-28 07:12

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36739

执行摘要

- 一句话：折叠 free group 标志，消除分配器状态漂移与双重否定
- 推荐动作：值得精读，尤其是 base.py 中 free_group_begin/free_group_end 的“容器即状态”设计：用 None 表示非激活、list 表示激活，并用一条赋值语句同时取走数据和复位状态，是消除双状态漂移的典型手法，可作为后续状态机重构的参考。建议在合入前确认 3 个 CI job 失败的原因，并考虑后续补充 free group 状态机专项测试。

功能与动机

PR body 指出：is_not_in_free_group 是负向命名的布尔值，导致调用点在需要延迟释放时都要读 not self.is_not_in_free_group，形成双重否定；而且该标志与 free_group 永远是同时写入的，即同一状态的两个编码，它们可能漂移——BaseTokenToKVPoolAllocator.free_group_end 清除了标志却保留了列表，重复调用 free_group_end() 会再次释放相同的索引。因此作者把 free_group 容器本身作为状态，从结构上杜绝双状态漂移。

实现拆解

1. 统一状态源（allocator/base.py）：BaseTokenToKVPoolAllocator 将 free_group 的语义定义为 None/list 二态，删除 is_not_in_free_group 字段；free_group_begin 置 free_group = []，free_group_end 用 pending，self.free_group = self.free_group，None 一次性交付暂存索引并复位状态，从结构上杜绝标志与列表漂移。
2. 替换读法（token.py / paged.py / swa.py / multi_ended_allocator.py / allocator_npu.py）：所有 is_not_in_free_group 的读取点改为 free_group is None / is not None，clear() 里的初始化也统一为 free_group = None；swa_free_group、free_page_reps_group 等纯数据容器保持 list 不变。
3. 去重 override：MultiEndedAllocator、UnifiedMambaTokenToKVPoolAllocator、UnifiedSWATokenToKVPoolAllocator 的 free_group_begin/free_group_end 与基类实现完全一致，改为继承；PureSWATokenToKVPoolAllocator 因父类 hooks 驱动 swa_free_group 而保留自己的实现，并加注释防止后续误删。
4. HiSparse 清理：HiSparseTokenToKVPoolAllocator 不支持延迟释放（group hooks 是 no-op），free() 中 else 分支不可达，删除；同时删除对子分配器 hisparse_attn_allocator 的冗余标志写入，因为只有顶层 token_to_kv_pool_allocator 才会进入 free group。

5. 测试与 CI：无新增测试文件，PR 说明依赖 test_paged_free_segment.py、test_swa_unittest.py、test_radix_cache_unit.py 覆盖该路径；PR CI 的 3 个 job 曾失败，作者 rerun 相关单元测试后 ubuntu-latest 上 4 个测试通过。

关键文件：

- python/sglang/srt/mem_cache/allocator/base.py（模块 分配基类；类别 source；类型 core-logic；符号 free_group_begin, free_group_end, free_group）：定义 free_group 的 None/list 单一状态源语义，并重构 begin/end 实现，是本次重构的核心基座。
- python/sglang/srt/mem_cache/multi_ended_allocator.py（模块 多端分配；类别 source；类型 core-logic；符号 free_group_begin, free_group_end, clear, free）：改动最大的文件，删除 MultiEndedAllocator 与 UnifiedMamba 自身的 begin/end 重写，并统一 clear/free 的状态判断。
- python/sglang/srt/mem_cache/allocator/hisparse.py（模块 稀疏分配；类别 source；类型 core-logic；符号 free, free_group_begin, free_group_end, free_hisparse_indices）：删除不可达的延迟释放分支与三处对子分配器的冗余标志写入，并保留 no-op hooks 的说明注释。
- python/sglang/srt/mem_cache/allocator/swa.py（模块 SWA 分配；类别 source；类型 core-logic；符号 free, free_swa, free_group_begin, free_group_end）：SWA 与 PureSWA 分配器的 free/free_swa 改为基于 free_group 状态判断，PureSWA 保留自己的 begin/end 并加注释说明原因。
- python/sglang/srt/mem_cache/allocator/paged.py（模块 分页分配；类别 source；类型 core-logic；符号 free, free_segment, clear）：Paged 分配器 free/free_segment 的状态判断切换，clear 统一重置。
- python/sglang/srt/mem_cache/allocator/token.py（模块 Token 分配；类别 source；类型 core-logic；符号 free, clear）：Token 分配器同样切换读法并统一 clear 状态。
- python/sglang/srt/hardware_backend/npu/allocator_npu.py（模块 NPU 分配；类别 source；类型 core-logic；符号 free）：NPU 后端分配器的一处状态读取同步切换，保持跨后端一致性。

关键符号：free_group_begin, free_group_end, free, free_swa, free_hisparse_indices, free_segment, clear

关键源码片段

python/sglang/srt/mem_cache/allocator/base.py

定义 free_group 的 None/list 单一状态源语义，并重构 begin/end 实现，是本次重构的核心基座。

```
# allocator/base.py 中 free group 的"容器即状态"设计
class BaseTokenToKVPoolAllocator(abc.ABC):
    def __init__(self, size, page_size, dtype, device, kvcache, need_sort):
        # ... 其他初始化 ...
        self.free_pages = None
        self.release_pages = None
        # None 表示立即释放；list 表示处于 free group 中，暂存待释放索引。
```

```
        # 之前这里还有一个 is_not_in_free_group 布尔标志，与本列表共同编码
        # 同一状态，两者可能漂移（如 free_group_end 清标志但不清列表导致重复释放）。
        self.free_group: list[torch.Tensor] | None = None

    def free_group_begin(self):
        # 开始一个 group：后续 free() 只把索引克隆进列表，不真正释放。
        self.free_group = []

    def free_group_end(self):
        # 一次性交付：取出全部暂存索引并把状态复位为 None。用一条赋值语句
        # 同时完成"读走数据"和"复位状态"，保证二者永远不会不一致。
        pending, self.free_group = self.free_group, None
        if pending:
            self.free(torch.cat(pending))
```

python/sglang/srt/mem_cache/multi_ended_allocator.py

改动最大的文件，删除 MultiEndedAllocator 与 UnifiedMamba 自身的 begin/end 重写，并统一 clear/free 的状态判断。

```
# multi_ended_allocator.py 中 free() 入口控制流
    def free(self, free_index: torch.Tensor) -> None:
        """释放虚拟 token 编号：恢复虚拟 page 编号、解映射 v2p/p2v，
        若持有 id 则回收 page 编号，最后触发 eager compaction。
        free_index 是 token 粒度，不要求页对齐；LAZY 模式把 compaction
        推迟到 _flush。
        """
        with record_function("MultiEndedAlloc.free"):
            if free_index is None or free_index.numel() == 0:
                return
            # free_group 非 None 即处于 group 中：克隆并暂存索引，
            # 避免调用方后续修改 tensor 视图影响延迟释放的正确性。
            if self.free_group is not None:
                self.free_group.append(self._copy_for_free_group(free_index))
                return
            if self.lazy_compaction:
                self._free_lazy(free_index)
                return
            # --- EAGER 路径 ---
            # 省略后续：v2p 查页、同步检查、页面回收与 compaction 移动。
```

python/sglang/srt/mem_cache/allocator/hisparse.py

删除不可达的延迟释放分支与三处对子分配器的冗余标志写入，并保留 no-op hooks 的说明注释。

```
# hisparse.py 中与 free group 相关的清理结果
    def free_hisparse_indices(self, buffer_indices: torch.Tensor):
        # 只有最顶层的 token_to_kv_pool_allocator 会被放入 free group，
        # 子分配器 hisparse_attn_allocator 的 free_group 恒为 None，
        # 因此这里无需再"禁用 free group 机制"，直接释放即可。
```

```
        self.hisparse_attn_allocator.free(buffer_indices[buffer_indices > 0])

    def free_group_begin(self):
        # HiSparse 不支持延迟释放：free() 必须立刻清掉 full_to_hisparse 映射，
        # 因此 group hooks 是空操作（保留注释避免后人误删）。
        return

    def free_group_end(self):
        return

    def free(self, free_index: torch.Tensor):
        if free_index.numel() == 0:
            return
        # 原先这里还有一个 else 分支把索引投进 free_group，但上面的 hooks
        # 都是 no-op，free_group 恒为 None，else 分支不可达，已删除。
        self.logical_attn_allocator.free(free_index)
        self.free_hisparse(free_index)
        assert (
            self.logical_attn_allocator.available_size()
            <= self.logical_attn_allocator.size
        )
```

评论区精华

PR 没有 review 评论。唯一讨论是作者触发 /rerun-test 重跑分配器相关单元测试，github-actions 返回：ubuntu-latest 上 test_multi_ended_allocator.py、test_paged_free_segment.py、test_unified_mla_views.py、test_unified_memory_move_gate.py 等 4 个测试通过，说明重构未破坏 free group 主路径。初始 CI 失败原因未在 PR 内说明。

- free group 重构后的测试覆盖与 CI 重跑 (testing): 未发现代码层面的讨论或未决问题；CI 初始失败原因未说明，rerun 覆盖了主要 free group 路径。

风险与影响

- 风险：本 PR 改动的是缓存分配的公共路径（free/free_group_begin/free_group_end），涉及多端分配器、SWA、Unified Mamba、HiSparse、Paged 与 Token 分配器，回归面较广；行为上唯一的显式变化是重复 free_group_end() 从 double-free 变为安全 no-op，任何依赖旧错误行为的调用方需要适配（按语义不应存在）。PR CI 初始 3 个 job（Base/Extra/AMD ROCm 7.2）均失败，虽经 rerun 部分测试通过，但失败原因未在 PR 中说明，不能完全排除与本次改动相关。此外没有新增测试文件，长期建议针对 free group 状态机补一个专门的单元测试。
- 影响：影响范围：所有使用 KV pool / token pool 分配器的推理路径（调度器、缓存管理、SWA、Mamba、HiSparse 等），但行为契约不变（仅消除错误的重入 double-free），对用户无感知。对团队而言，free_group 成为单一事实来源降低了状态维护成本，后续新增 allocator 只需继承基类实现即可。PR 整体是纯内部重构，不需要配置或部署变更。
- 风险标记：核心路径变更，无新增测试，CI 失败

关联脉络

- 暂无明显关联 PR