PR #35838 完整报告

sgl-project/sglang

refactor(disagg): remove unreferenced dead code

合并时间：2026-08-21 22:09

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35838

执行摘要

- 一句话：清理 disagg 模块 4 处无调用死代码，纯删除零行为变化
- 推荐动作：值得快速浏览，不是因为它改变了什么，而是因为它示范了“如何安全地删除代码”：每个删除项都给出来源 commit、存活替身、全仓库扫描证据、lint 验证，并明确区分死代码与扩展点。对维护者而言，真正可借鉴的是 page_indices_to_cp_rank_page_indices 被孤儿化的过程——重构时把调用方迁移到新实现后应顺手清理旧符号，避免正确性修复落在无人调用的代码上。若团队有 dead code 治理需求，本 PR 可作为模板。

功能与动机

PR body 明确说明删除依据：四个符号在 python/、test/、docs/、sgl-router/、benchmark/ 全仓库范围内均无调用方，且每个都有存活的替身或邻居。最典型的是 page_indices_to_cp_rank_page_indices——它在 #28718（Fix CP page filtering by request-local position）中被孤儿化：那次修复把唯一调用方 filter_kv_indices_for_cp_rank 重写到新 helper _get_cp_rank_page_bounds 上，却把旧函数留在原地，意味着 CP 正确性修复实际作用在无人调用的代码上。此外 get_transferred_rids 内部是 poll_and_all_reduce_attn_cp_tp_group 的 collect 通信，删除比保留更安全，可避免未来部分 rank 误调用导致整组挂起。

实现拆解

1. 全局静态验证：对 python/、test/、docs/、sgl-router/、benchmark/ 下所有 .py/.md/.sh/.rs 文件扫描四个目标符号，确认均无调用方；同时核查删除后不产生孤儿 import——utils.py 中 np 仍有 6 处使用，prefill.py 中 List[仍有 8 处、poll_and_all_reduce_attn_cp_tp_group 仍有 6 处使用。
2. utils.py（-46 行）：删除 page_indices_to_cp_rank_page_indices。该函数基于“请求页占用连续物理全局页 id 区间”的旧假设，而 #28718 引入的 _get_cp_rank_page_bounds + filter_kv_indices_for_cp_rank 已按“请求内页位置”重新实现并成为唯一存活路径，旧函数成为孤儿且有误导性假设，直接删除。
3. prefill.py（-18 行）：删除 get_transferred_rids。其 docstring 声称“Used by PP”，但实际 PP 路径直接调用 poll_and_all_reduce_attn_cp_tp_group，无人经过该包装函数；由于该函数内部是跨 rank 的 collect 通信，保留一个无调用方的入口反而埋下“部分 rank 误调用即挂起”的隐患。

4. staging_buffer.py 与 base/conn.py（-5 行）：删除 StagingAllocator.get_round（三个同类存取器中唯一无调用方，轮次信息仍可通过 get_watermark() 获取）与 KVArgs.ib_traffic_class（bare annotation，从未被赋值或读取；其邻居 ib_device 仍是活跃字段）。
5. 删除边界控制：明确保留 register_publisher（插件注册钩子）、空的后端 Server 子类、FastAPI 路由等"grep 无调用但语义上是扩展点"的符号，避免误删对外契约。
6. 测试与部署配套：无测试、文档、配置变更——纯删除无行为变化，作者未本地运行需多 GPU 的 disaggregation 集成测试，改用 /rerun-group 触发 CI 重跑确认。

关键文件：

- python/sglang/srt/disaggregation/utils.py（模块 分离部署；类别 source；类型 cleanup；符号 page_indices_to_cp_rank_page_indices）：删除 page_indices_to_cp_rank_page_indices（46 行），这是 #28718 重构后遗留的孤儿函数，其"连续全局页 id"假设已被新实现取代，是本次清理的核心项。
- python/sglang/srt/disaggregation/prefill.py（模块 预填充调度；类别 source；类型 cleanup；符号 get_transferred_rids）：删除 get_transferred_rids（18 行），该函数 docstring 声称供 PP 使用但实际无任何 PP 路径调用，且内部是 collect 通信，删除可避免未来部分 rank 误调用导致挂起。
- python/sglang/srt/disaggregation/common/staging_buffer.py（模块 暂存缓冲；类别 source；类型 cleanup；符号 StagingAllocator.get_round）：删除 StagingAllocator.get_round（4 行），三个同族存取器中唯一无调用方，轮次信息仍可通过 get_watermark() 获取。
- python/sglang/srt/disaggregation/base/conn.py（模块 传输连接；类别 source；类型 cleanup；符号 KVArgs.ib_traffic_class）：删除 KVArgs.ib_traffic_class 字段注解（1 行），该字段从未被赋值或读取，邻居 ib_device 才是活跃字段。

关键符号：page_indices_to_cp_rank_page_indices, get_transferred_rids, StagingAllocator.get_round

关键源码片段

python/sglang/srt/disaggregation/utils.py

删除 page_indices_to_cp_rank_page_indices（46 行），这是 #28718 重构后遗留的孤儿函数，其"连续全局页 id"假设已被新实现取代，是本次清理的核心项。

```
# python/sglang/srt/disaggregation/utils.py
# 本文件删除了 page_indices_to_cp_rank_page_indices（46 行）。
# 该函数在 #28718 修复后失去唯一调用方：filter_kv_indices_for_cp_rank
# 已改用下面按"请求内页位置"切分的 _get_cp_rank_page_bounds，
# 而旧函数仍按"物理全局页 id 连续区间"过滤，既无调用方，
# 其隐含假设也与新逻辑相悖，故直接删除。

def _get_cp_rank_page_bounds(
    total_pages: int, cp_rank: int, cp_size: int
) -> Tuple[int, int]:
```

```
    # 计算本 CP rank 在"请求内页位置"空间的页区间。
    # 这是 #28718 引入的存活实现，支持物理页 id 不连续、无序的情况。
    base = total_pages // cp_size
    rem = total_pages % cp_size
    local_start = cp_rank * base + min(cp_rank, rem)
    n_pages = base + (1 if cp_rank < rem else 0)
    return local_start, local_start + n_pages


def filter_kv_indices_for_cp_rank(
    kv_mgr: CommonKVManager,
    kv_indices: np.ndarray,
    index_slice: slice,
    total_pages: Optional[int] = None,
) -> Tuple[np.ndarray, slice]:
    # 唯一存活的 CP 页过滤入口：先求本 rank 的请求内页区间，
    # 再与当前传输 chunk 的 index_slice 求交，避免逐页全量过滤。
    if total_pages is None:
        total_pages = len(kv_indices)
    cp_rank = kv_mgr.attn_cp_rank
    cp_size = kv_mgr.attn_cp_size

    if cp_size <= 1:
        return kv_indices, index_slice

    rank_start, rank_end = _get_cp_rank_page_bounds(total_pages, cp_rank, cp_size)
    chunk_start = index_slice.start if index_slice.start is not None else 0
    chunk_end = index_slice.stop if index_slice.stop is not None else total_pages
    first_pos = max(rank_start, chunk_start) - chunk_start
    last_pos = min(rank_end, chunk_end) - chunk_start

    if last_pos <= first_pos:
        # 无交集时返回空切片，避免越界或错位。
        new_kv_indices = kv_indices[:0]
        new_index_slice = slice(chunk_start, chunk_start)
    else:
        new_kv_indices = kv_indices[first_pos:last_pos]
        new_index_slice = slice(
            chunk_start + first_pos,
            chunk_start + last_pos,
        )
    return new_kv_indices, new_index_slice
```

python/sglang/srt/disaggregation/prefill.py

删除 get_transferred_rids（18 行），该函数 docstring 声称供 PP 使用但实际无任何 PP 路径调用，且内部是 collect 通信，删除可避免未来部分 rank 误调用导致挂起。

```
# python/sglang/srt/disaggregation/prefill.py
# 本文件删除了 get_transferred_rids（18 行）。该函数 docstring 声称
```

```
# “Used by PP”，但全仓库没有 PP 路径调用它：PP 侧实际直接调用
# poll_and_all_reduce_attn_cp_tp_group。由于该函数内部是 collect
# 通信，若未来只有部分 rank 误调用，整组会永久挂起，删除比保留更安全。
# 下方为删除后紧邻的存活方法，可见同文件仍保留传输失败处理的完整逻辑。

class Scheduler:
    def handle_inflight_transfer_failure(
        self: Scheduler, req: Req
    ) -> Optional[Exception]:
        # 处理 KV 传输失败的收尾：记录异常、解锁 tree cache、
        # 按失败原因终止请求并上报指标。
        error_message = (
            f"Prefill transfer failed for request rank={self.ps.tp_rank} "
            f"{req.rid=} {req.bootstrap_room=}"
        )
        exc: Optional[Exception] = None
        try:
            req.disagg_kv_sender.failure_exception()
        except Exception as e:
            exc = e
            error_message += f" with exception {e}"
        # 对已由其他 rank 传播的异常降噪，避免重复告警。
        if getattr(exc, "is_from_another_rank", False):
            logger.debug(error_message)
        else:
            logger.warning(error_message)
        req.time_stats.trace_ctx.abort(abort_info={"reason": error_message})
        release_kv_cache(req, self.tree_cache)
        if not isinstance(req.finished_reason, FINISH_ABORT):
            prepare_abort(
                req, error_message, status_code=HTTPStatus.INTERNAL_SERVER_ERROR
            )
        if self.metrics_reporter.enable_metrics:
            self.metrics_collector.increment_transfer_failed_reqs()
        return exc
```

python/sglang/srt/disaggregation/common/staging_buffer.py

删除 StagingAllocator.get_round（4 行），三个同族存取器中唯一无调用方，轮次信息仍可通过 get_watermark() 获取。

```
# python/sglang/srt/disaggregation/common/staging_buffer.py
# StagingAllocator 原先对同一 allocation 元组提供 get_ptr / get_offset /
# get_round 三个存取器；前两者有调用方，get_round 没有，且轮次信息仍可
# 通过 get_watermark() 获取，因此删除 get_round，保留其余存取器。

class StagingAllocator:
    def get_ptr(self, alloc_id: int) -> int:
        # 返回分配起始处的绝对地址，供发送侧读取 KV 数据。
        offset, _, _ = self.allocations[alloc_id]
```

```
        return self.base_ptr + offset

    def get_offset(self, alloc_id: int) -> int:
        # 返回分配在暂存缓冲区内的相对偏移。
        offset, _, _ = self.allocations[alloc_id]
        return offset

    def get_base_ptr(self) -> int:
        return self.base_ptr

    def get_total_size(self) -> int:
        return self.total_size
```

评论区精华

该 PR 没有任何 review 评论，有价值的讨论体现在 PR body 的安全删除论证和 CI 重跑记录中。作者特别强调：> “The clearest case is page_indices_to_cp_rank_page_indices. It became dead in 1adb53f14 (#28718), which reimplemented its only caller filter_kv_indices_for_cp_rank on top of the new _get_cp_rank_page_bounds helper but left the old function behind. That means the CP correctness fix in #28718 was applied to code nothing calls.”

> “One note on get_transferred_rids: it calls poll_and_all_reduce_attn_cp_tp_group, which is a collective. An accidental caller reaching it on only some ranks would hang, so removing it is slightly safer than leaving it available.”

关于删除边界，作者明确列出 grep 也找不到调用方但刻意保留的符号：EventPublisherFactory.register_publisher（插件注册钩子）、空的 MooncakeKVBootstrapServer / MoriKVBootstrapServer / NixlKVBootstrapServer 与 AscendKV* 子类、以及经装饰器可达的 FastAPI 路由——这体现了“无静态调用”不等于“可删除”的判别标准。CI 侧作者两次请求 /rerun-group disaggregation，第一次因分支 diverged 被 bot 拒绝要求 rebase，rebase 后重跑成功，2-gpu-h100 与 4-gpu-gb300 等 disaggregation 用例全部通过。

- CI 重跑需先 rebase 到最新 main (other): rebase 到最新 main 后，disaggregation 组在 2-gpu-h100（5 个测试）与 4-gpu-gb300（1 个测试）上全部通过。
- 删除边界：无静态调用不等于可删除 (design): 删除范围严格限定在四个有存活替身或存活邻居的符号，扩展点全部保留。

风险与影响

- 风险：整体风险很低，但仍有几个值得注意的盲区： 1) 静态 grep 无法覆盖动态引用（getattr、monkey patch、字符串路由注册等），作者虽然跑过 Ruff F401/F821 并手动保留了扩展点，但跨语言扫描只覆盖 .py/.md/.sh/.rs，未覆盖 .cu/.cpp/.json 等文件，理论上仍可能遗漏极端情况下的动态引用； 2) KVArgs.ib_traffic_class 是 bare annotation，若外部代码通过 KVArgs.__annotations__检查该字段存在性会受影响，但仓库内部无此用法； 3) 作者未在本地运行需要多 GPU 的 disaggregation 集成测试，依赖 CI 验证

——PR body 中基础 CI 槽位曾标记为 :x:，最终以 /rerun-group 重跑通过为准，合并后仍需关注 main 上该模块是否有回归。

- 影响：影响范围严格限定在 python/sglang/srt/disaggregation/ 的 4 个文件，全部为删除操作，无任何执行路径变化，对用户、模型输出、性能均不可见。对团队的实际收益是降低维护认知负担：移除一个与存活实现并存的误导性 CP 过滤函数（其注释仍宣称"连续全局页区间"这一已被 #28718 否定的假设）、消除一个可能引发 collect 挂起的隐患入口，并清理类定义中的无效字段。对 disagg 模块后续重构（如近期 35847、35843、35844 等清理系列）提供了更干净的基座。
- 风险标记：静态扫描可能遗漏动态引用，基础 CI 槽位曾标记失败（重跑通过），无本地集成测试验证

关联脉络

- PR #28718 Fix CP page filtering by request-local position: 本 PR 删除的 page_indices_to_cp_rank_page_indices 正是在该 PR 中失去唯一调用方而成为死代码，是本次清理的直接触发源。
- PR #35847 refactor(disagg): collapse duplicated branches in get_kv_class: 同模块 utils.py 的同类重构清理，与本次死代码删除属于同一波 disagg 模块维护浪潮。
- PR #35843 refactor(disagg): remove dead build_and_send_encode_request: 同样以删除无调用死代码为主题的系列 PR，反映 disagg 模块正在系统性清理历史遗留代码。