PR #37166 完整报告

sgl-project/sglang

fix(staging): make empty staging rings reusable

合并时间：2026-08-31 11:59

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37166

执行摘要

- 一句话：修复空暂存环水位不前进导致的 PD 暂存转发卡死
- 推荐动作：值得精读。两个设计点值得学习：一是环形缓冲区水位的经典解法——用 round 的单调性抵消 tail 归零带来的回退，保证异步 readiness 检查始终可推进；二是作者用 " allocator-reset-only 仍失败 " 的对照实验把问题拆成两个正交缺陷，这种验证方法可复用到其他分布式状态同步类 bug。对维护 staging 传输层的同学，建议后续关注 _send_watermark 静默异常路径与订阅快照丢失时的补偿重试。

功能与动机

PR body 描述了完整失败链路：开启 staging 后，decode 侧分配器可能把空环留在上一次分配的 head 位置；后续分配 wrap 进新一轮并越过陈旧 head 时，prefill 侧 readiness 检查会等待一个永远无法推进的水位。例如 100 字节环中 [0, 60) 释放后旧状态把 watermark 留在 (0, 60)，后续 70 字节分配变成 (round=1, offset=0, end=70)，在 70 <= 60 上永远等待，即使环内已无存活分配。该问题在异构 attention-TP PD 环境（SGLANG_DISAGG_STAGING_BUFFER=1）中复现，warmup 阶段完成 4/11 请求后停止。

实现拆解

步骤 1：空环状态机重置（staging_buffer.py）

在 StagingAllocator.free() 的 if not self.allocations 分支中新增 self.round += 1 与 self.head = 0，并把 watermark 置为 (self.round, 0)。原实现把 watermark_tail 取为 self.head（即最后一次分配的末尾，例如 60），在空环场景下该值已无意义且会卡住后续 wrap 分配。推进 round 是为了在 tail 归零时仍保持水位单调递增，这样 prefill 侧 is_watermark_ready 的 prev_round < wm_round 分支就能放行；同时 head 归零后下一次 assign 直接从 offset 0 起步，无需再次 wrap。

步骤 2：订阅者快照 + 发送逻辑复用（staging_handler.py）

register_wm_subscriber() 改为：key 已存在时直接 return，注册成功后立刻调用 _send_watermark 发送 self.staging_allocator.get_watermark() 当前快照。原实现只把订阅者记录进_wm_subscribers，新会话会以启动值(0,0)等待，错过decode侧已经推进的水位。同时把 _free_and_send_watermark() 内联的发送代码抽成静态方法 _send_watermark(receiver, session_id, watermark)，释放后广播与订阅快照共用同一发送路径；_free_and_send_watermark 不再使用 decode_req（参数改名 _decode_req），改为

遍历 _wm_subscribers.values() 直接广播。

步骤 3：回归测试（test_disaggregation_wire.py）

新增 TestStagingWatermark 两个用例：test_empty_ring_restarts_at_zero 用 100 字节环分配 60 再释放，断言 watermark 为 (1, 0)，再分配 70 得到 (0, 1)（offset 0、round 1），精确复现 PR body 中的卡死场景；test_new_watermark_subscriber_receives_current_allocator_state mock 分配器返回 (3, 0)，注册新订阅者后断言发送 [b"WATERMARK", b"3", b"0", b"session-new"]。测试文件新增 StagingAllocator 与 DecodeStagingHandler 导入。

步骤 4：验证与 CI

作者在异构 attention-TP PD 环境（prefill --tp-size 4 --dp-size 4 --ep-size 4 --enable-dp-attention，decode --tp-size 8 --dp-size 1 --ep-size 1 --disable-attn-tp-gather）做了三臂对比：上游基线 warmup 停在 4/11（3 个请求在途）；只叠加 allocator 重置仍停在 4/11；两个生产文件同时叠加后 warmup 11/11、profile 846/846 完成、TTFT 与 ITL 覆盖 100%。CI 侧通过 /rerun-group disaggregation 重跑，2-gpu-h100 与 8-gpu-h20 共 10 个测试全部通过。

关键文件：

- python/sglang/srt/disaggregation/common/staging_buffer.py（模块 暂存分配；类别 source；类型 core-logic；符号 StagingAllocator.free, StagingAllocator.assign）：核心修复点之一：StagingAllocator.free() 在环空时推进 round 并把水位重置为 (round, 0)，是消除 prefill 侧永久等待的第一半修复。
- python/sglang/srt/disaggregation/common/staging_handler.py（模块 水位广播；类别 source；类型 entrypoint；符号 register_wm_subscriber, _free_and_send_watermark, _send_watermark）：水位协议入口：新 prefill 订阅者注册时立即收到当前水位快照，并抽出 _send_watermark 复用释放后广播与订阅快照两条发送路径，是第二半修复。
- test/registered/unit/disaggregation/test_disaggregation_wire.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestStagingWatermark, test_empty_ring_restarts_at_zero, test_new_watermark_subscriber_receives_current_allocator_state）：新增 TestStagingWatermark 两个回归测试，分别锁定空环重启状态机与新订阅者快照下发，直接覆盖本 PR 两个修复点。

关键符号：StagingAllocator.free, StagingAllocator.assign, StagingAllocator.get_watermark, DecodeStagingHandler.register_wm_subscriber, DecodeStagingHandler._send_watermark, DecodeStagingHandler._free_and_send_watermark, is_watermark_ready, handle_watermark_msg

关键源码片段

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

核心修复点之一：StagingAllocator.free() 在环空时推进 round 并把水位重置为 (round, 0)，是消除 prefill 侧永久等待的第一半修复。

```
# python/sglang/srt/disaggregation/common/staging_buffer.py
class StagingAllocator:
```

```
    """环状暂存分配器：分配必须连续，放不下剩余尾段时就整体跳到下一轮 offset 0。"""

    def assign(self, required_bytes: int):
        """分配一段连续区域，返回 (alloc_id, offset, round) 或 None。"""
        with self.lock:
            if required_bytes > self.total_size:
                return None

            space_at_end = self.total_size - self.head
            if required_bytes <= space_at_end:
                offset = self.head
                self.head += required_bytes
            else:
                # 尾段放不下，进入新一轮并从 offset 0 重新开始
                self.round += 1
                offset = 0
                self.head = required_bytes

            alloc_id = self.next_alloc_id
            self.next_alloc_id += 1
            self.allocations[alloc_id] = (offset, required_bytes, self.round)
            self.alloc_order.append(alloc_id)
            return (alloc_id, offset, self.round)

    def free(self, alloc_id: int):
        """释放分配，并把水位推进到最早存活分配之前。"""
        with self.lock:
            if alloc_id not in self.allocations:
                return
            self.allocations.pop(alloc_id)

            # 弹出队首已释放的分配，保证 alloc_order 队首是存活分配
            while self.alloc_order and self.alloc_order[0] not in self.allocations:
                self.alloc_order.pop(0)

            if not self.allocations:
                # 空环意味着整轮都可复用：必须推进 round 让水位保持单调，
                # 同时把 tail 归零，否则旧水位 (round, 60) 会让后续 wrap
                # 分配在 70 <= 60 上永远等待（100 字节环、先分配 60 的场景）
                self.round += 1
                self.head = 0
                self.watermark_round = self.round
                self.watermark_tail = 0
            elif self.alloc_order:
                off, _, rnd = self.allocations[self.alloc_order[0]]
                self.watermark_round = rnd
                self.watermark_tail = off
```

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

水位协议入口：新 prefill 订阅者注册时立即收到当前水位快照，并抽出 _send_watermark 复用释放后广播与订阅快照两条发送路径，是第二半修复。

```
# python/sglang/srt/disaggregation/common/staging_handler.py
class DecodeStagingHandler:
    """decode 侧暂存处理器：持有分配器，并向所有 prefill 会话广播水位。"""

    def register_wm_subscriber(self, receiver, session_id: str) -> None:
        """注册一个 prefill 水位订阅连接。"""
        if receiver is None or not receiver.bootstrap_infos:
            return
        key = tuple(str(bi) for bi in receiver.bootstrap_infos)
        if key in self._wm_subscribers:
            return

        self._wm_subscribers[key] = (receiver, session_id)
        # 分配器跨请求常驻，而每个 prefill session 独立学习水位；
        # 新订阅者必须立刻拿到当前快照，否则首个 wrap 分配会一直
        # 等在早已过去的旧水位 (0, 0) 上
        self._send_watermark(
            receiver,
            session_id,
            self.staging_allocator.get_watermark(),
        )

    @staticmethod
    def _send_watermark(receiver, session_id: str, watermark) -> None:
        """向一个 prefill 会话发送一条 WATERMARK 消息。"""
        wm_round, wm_tail = watermark
        wm_round_b = str(wm_round).encode("ascii")
        wm_tail_b = str(wm_tail).encode("ascii")
        sid_b = session_id.encode("ascii")
        for bootstrap_info in receiver.bootstrap_infos:
            try:
                sock, lock = receiver._connect_to_bootstrap_server(bootstrap_info)
                with lock:
                    sock.send_multipart([b"WATERMARK", wm_round_b, wm_tail_b, sid_b])
            except Exception:
                # 与既有行为一致：发送失败静默跳过，靠后续广播补偿
                pass
```

评论区精华

本 PR 没有正式的 Review 评论，唯一的外部交互是作者触发 /rerun-group disaggregation 重跑 CI。值得提炼的 " 讨论 " 其实沉淀在 PR body 的验证方案里：作者明确指出 " Advancing the round is necessary to keep the watermark monotonic while its tail returns to zero"，并设计了 allocator-reset-only 对照臂——该臂仍停在 4/11，从而证明 " 初始订阅者水位快照 " 与 " 空环重启 " 是两个正交缺陷，必须同时修复。这种把一个问题拆成两

个独立根因并用对照实验确认的做法，比单纯贴测试结果更有说服力。

- disaggregation CI 组重跑结果 (testing): disaggregation 组回归重跑通过，未发现本 PR 引入的功能回退。

风险与影响

- 风险：
  - 水位协议语义变更：空环时 watermark 会直接跳到新 round，依赖 prefill 端 is_watermark_ready 已有的 prev_round < wm_round 分支放行；该分支原本就存在，此次只是让 round 真正推进。但仍需关注 decode 侧广播与 prefill 侧处理之间的异步时序，若某个 prefill 仍持有旧 (0, 60) 且新消息因网络问题丢失，会继续卡住。
  - _send_watermark 静默吞异常：发送失败仅 pass，无重试；新订阅者快照若发送失败，该会话会以旧值等待。这是既有协议缺陷的延续，本 PR 未引入重试机制。
  - 快照竞态：register_wm_subscriber 中 get_watermark() 与后续 alloc/free 之间有时序窗口，但 prefill 端 handle_watermark_msg 只接受更大的 (round, tail)（单调推进），过期消息不会回退水位，因此竞态方向是安全的。
  - 覆盖范围：新增测试为单元级，未覆盖真实多 rank、异构 TP 的集成路径；E2E 为作者手工验证，仓库 CI 仅有 disaggregation 组的回归测试兜底。
  - 影响：用户影响：启用 staging 的 PD 部署（SGLANG_DISAGG_STAGING_BUFFER=1）不再在 warmup 或运行中因空环水位停滞而完全卡死，这是该特性上线前的阻塞性问题。系统影响：decode 侧分配器状态机与 prefill 订阅协议行为发生变化，但无公开 API、无配置参数变更、无模型数学改动。团队影响：改动仅 3 个文件、67 行新增，附带两个针对性单元测试，ci 重跑 disaggregation 组全部通过，回归风险可控。
  - 风险标记：水位协议语义变更，异步订阅发送无重试，快照时序依赖单调推进，集成路径仅手工 E2E 验证

关联脉络

- PR #36958 [mem_cache] Keep req.kv non-optional and key KV ownership on req_pool_idx: 同属 PD disaggregation / mem_cache 子系统，重构 decode 侧 KV 生命周期与所有权判定，与本 PR 的 staging 分配器状态管理属于同一演进线。
- PR #37164 [mem_cache] Move mamba state and retraction_backup into ReqKvInfo: 同属 mem_cache 与调度器状态归属重构，与 allocator/ 水位这类跨请求持久状态的组织方式调整方向一致。
- PR #36834 [HiCache] buffer mode: decide staged-fetch fate against the live tree: 同为调度与暂存推进逻辑的修复，但属于 HiCache buffer 模式的不同实现路径，可作为 staging 类状态推进问题的横向参照。