PR #30545 完整报告

sgl-project/sglang

[Disagg][StagingBuffer][2/2] Support radix cache

合并时间：2026-08-06 23:59

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30545

执行摘要

- 一句话：staging buffer 支持 radix cache，修复异构 TP 网格错位损坏
- 推荐动作：值得精读。核心看三点：compute_grid_segments 的网格几何统一、decode 侧到达驱动 scatter 的生命周期判定、以及 ' 失败单请求而非杀死 scheduler' 的错误处理哲学。同时建议跟进 reviewer 的 cleanup 意见，把 nixl/mooncake 重复的 outstanding/teardown 逻辑收敛为共享实现，并补充 mooncake 后端的 e2e 覆盖。

功能与动机

PR body 明确指出：staging 路径按位置识别 chunk（chunk_idx = start_page // full_chunk_pages），对统一的预取环形分配网格要求每个发送边界都落在网格上，否则协议会静默损坏。一旦 prefill 侧发生 radix cache 命中，第一次发送会被前缀膨胀到超过一个网格槽位，后续所有 chunk 索引全部错位，引发环形分配越界、decode-prefix scatter 偏移损坏以及从 chunk_infos[-1] 盲发尾部数据。因此此前 staging buffer 与 radix cache 无法组合使用。

实现拆解

1. 网格几何抽象（staging_buffer.py）：新增 staging_grid_tokens 与 compute_grid_segments 两个纯函数，统一预取端槽位划分与发送端对齐对网格的定义（槽宽 = chunked_prefill_size 向下取整到 page_size 的整数倍），并给出按网格边界切分 [start_idx, end_idx) 的通用算法，空范围返回一个空段以支持元数据 -only 的最后一个 chunk。
2. Prefill 发送路径调整（prefill.py）：finalize_bootstrap 记录 req.disagg_decode_prefix_len = decode_prefix_len 作为网格原点；send_kv_chunk 对非最后发送把 end_idx 向下取整到网格边界，余量并入下一次发送；maybe_send_cached_prefix_chunk 在 staging 下重新启用缓存前缀提前发送，并通过 early_send_prefix_end 在首个 batch 快照静态前缀，避免 overlap 调度下 prefix_indices 动态增长导致边界漂移。
3. Decode 侧 scatter 生命周期重构（staging_handler.py）：新增 _staging_all_success、_staging_failed、_staging_success_ts、_writer_counts 与 completion_timeout（复用 SGLANG_DISAGGREGATION_WAITING_TIMEOUT）；scatter 完全到达驱动，最后一个 chunk 也发送 CHUNK_READY；room 只有所有 rank Success、所有分配完成 scatter 且所有事件结束后才完成，超时以 KVPoll.Failed 浮出；decode 前缀非页对齐时标记该请

求失败而不是 raise。

4. NIXL / Mooncake 传输后端同步改造（nixl/conn.py、mooncake/conn.py）：两后端都引入 _staging_outstanding 按 room 计数，确保背压下被 deferred 重排的 chunk 不会在 room teardown 时被丢弃；teardown 只在无 outstanding chunk 且 room 已结论（Success 或 Failed 的最后一个 chunk）时执行；writer fan-in 计数从 conn 层移到 staging handler，便于 room 清理时清除部分计数。
5. 测试与基准配套：新增 TestDisaggregationStagingRadixPrefillLargerTP e2e 测试（prefill TP4 → decode TP2，chunked-prefill-size=256，decode 开启 radix cache），强制共享前缀跨越多个网格槽，精确覆盖此前损坏的 grid-split 与 scatter-offset 路径；同时提供 gsm8k 0.9802、128k needle-in-haystack 128/128 与 8xGB300 AgentX 吞吐基准。

关键文件：

- python/sglang/srt/disaggregation/common/staging_handler.py（模块 暂存调度；类别 source；类型 entrypoint；符号 num_writers_for, is_failed, _submit_last_scatter, register_decode_req）：decode 侧 staging scatter 生命周期核心文件：新增完成 / 失败 / 超时状态、writer fan-in 计数，并把页面不对齐从 raise 降级为标记请求失败。
- python/sglang/srt/disaggregation/common/staging_buffer.py（模块 暂存缓冲；类别 source；类型 core-logic；符号 staging_grid_tokens, compute_grid_segments）：新增网格几何基础函数 staging_grid_tokens 与 compute_grid_segments，统一预取端与发送端的网格划分，是本次修复的数学基础。
- python/sglang/srt/disaggregation/prefill.py（模块 预填充端；类别 source；类型 dependency-wiring）：prefill 侧核心改动：记录 disagg_decode_prefix_len 作为网格 base，send_kv_chunk 对非最后发送做网格对齐，并重新启用 staging 下提前发送缓存前缀。
- python/sglang/srt/disaggregation/nixl/conn.py（模块 传输后端；类别 source；类型 core-logic；符号 transfer_worker）：NIXL 后端传输 worker 同步改造：_staging_outstanding 计数防止 deferred chunk 在 room teardown 时被丢弃，并调整 teardown 触发条件。
- python/sglang/srt/disaggregation/mooncake/conn.py（模块 传输后端；类别 source；类型 core-logic；符号 _send_chunk_ready, transfer_worker）：Mooncake 后端同步改造：每个 chunk（含最后一个）都发送 CHUNK_READY，scatter 完全到达驱动；writer fan-in 计数移到 staging handler。
- test/registered/disaggregation/test_disaggregation_different_tp.py（模块 端到端测试；类别 test；类型 test-coverage；符号 TestDisaggregationStagingRadixPrefillLargerTP, setUpClass, start_prefill, start_decode）：新增 TestDisaggregationStagingRadixPrefillLargerTP 端到端测试，覆盖此前损坏的 ' 多网格槽共享前缀 ' 场景，作为回归护栏。
- python/sglang/srt/disaggregation/utils.py（模块 分发工具；类别 source；类型 core-logic）：为 staging 相关状态传递补充工具逻辑，支撑 handler 与 conn 之间的状态语义。
- python/sglang/srt/managers/schedule_batch.py（模块 调度批处理；类别 source；类型 core-logic）：为 Req 增加 early_send_prefix_end、disagg_decode_prefix_len 等

staging 相关字段声明。

- python/sglang/srt/disaggregation/common/utils.py（模块 分发工具；类别 source；类型 core-logic）：少量配置键 / 控制流调整，配合 staging 状态标记。
- test/registered/unit/disaggregation/test_nixl_backend_basic.py（模块 单元测试；类别 test；类型 test-coverage）：NIXL 单元测试配套改动，适配新的 staging 状态字段。

关键符号：staging_grid_tokens, compute_grid_segments, maybe_send_cached_prefix_chunk, send_kv_chunk, finalize_bootstrap, num_writers_for, register_decode_req, is_failed, submit_last_scatter_async, transfer_worker, _send_chunk_ready

关键源码片段

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

decode 侧 staging scatter 生命周期核心文件：新增完成 / 失败 / 超时状态、writer fan-in 计数，并把页面不对齐从 raise 降级为标记请求失败。

```
def register_decode_req(self, room: int, decode_req: DecodeRequest) -> None:
    """注册一个 decode room，初始化 staging 完成状态机。

    在 pop_preallocated 之后、send_metadata 之前调用，每个 room 只执行一次。
    """
    # 初始化成功 / 失败 / 超时三组状态：全部 rank 成功标记、成功时刻、失败标记，
    # 以及按 chunk 的事件列表（scatter 事件全部结束才算真正 done）。
    decode_req._staging_all_success = False
    decode_req._staging_success_ts = 0.0
    decode_req._staging_failed = False
    decode_req._staging_scatter_done = False
    decode_req._chunk_events = []
    self._room_to_decode_req[room] = decode_req
    self._room_to_receiver[room] = decode_req.kv_receiver

    # scatter 偏移会把 suffix-relative 的 page_start 前移 decode 前缀长度，
    # 但只有前缀按页对齐时该偏移才精确。这里选择只失败当前请求而不是 raise：
    # raise 会杀死整个 prefill scheduler，影响面远大于单请求失败。
    page_size = self.kv_buffer_info["page_size"]
    if decode_req.req.cache_protected_len % page_size != 0:
        logger.error(
            "[STAGING] decode prefix length %s is not page-aligned "
            "(page_size=%s); failing room=%s (staging scatter offsets "
            "would be wrong).",
            decode_req.req.cache_protected_len,
            page_size,
            room,
        )
        decode_req._staging_failed = True
```

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

新增网格几何基础函数 staging_grid_tokens 与 compute_grid_segments，统一预取端与发送端的网格划分，是本次修复的数学基础。

```
def staging_grid_tokens(
    chunked_prefill_size: Optional[int], page_size: int
) -> int:
    """计算一个 staging 网格槽覆盖的 token 宽度。

    预取端按该宽度在环上预分配 slot，发送端也按同一宽度对齐发送边界；
    槽宽 = chunked_prefill_size 向下取整到 page_size 的整数倍，
    保证槽边界必然落在物理页边界上，避免跨页拆分造成协议错位。
    """
    cps = chunked_prefill_size or 8192 # 未显式配置时默认 8192 tokens
    return max(1, cps // page_size) * page_size


def compute_grid_segments(
    start_idx: int, end_idx: int, base: int, grid_tokens: int
) -> List[Tuple[int, int]]:
    """把 [start_idx, end_idx) 按网格边界 base + k * grid_tokens 切分。

    每个返回的段恰好映射到一个 staging 槽位，发送端逐段构造 TransferKVChunk；
    空范围也返回一个空段，因为仅元数据的最后一个 chunk 仍需要一次发送来宣告完成。
    """
    segments: List[Tuple[int, int]] = []
    seg_start = start_idx
    while seg_start < end_idx:
        # 下一个网格边界：先算当前段起点落在第几个槽，再乘回边界坐标
        next_boundary = base + ((seg_start - base) // grid_tokens + 1) * grid_tokens
        seg_end = min(next_boundary, end_idx)
        segments.append((seg_start, seg_end))
        seg_start = seg_end
    if not segments:
        segments = [(start_idx, end_idx)]
    return segments
```

python/sglang/srt/disaggregation/prefill.py

prefill 侧核心改动：记录 disagg_decode_prefix_len 作为网格 base，send_kv_chunk 对非最后发送做网格对齐，并重新启用 staging 下提前发送缓存前缀。

```
def maybe_send_cached_prefix_chunk(self: Scheduler, req: Req) -> None:
    # 总开关：未开启提前发送时直接返回
    if not envs.SGLANG_DISAGG_PREFILL_EARLY_SEND_CACHED_PREFIX.get():
        return

    # staging 按位置向网格槽发送，提前发送的边界必须在请求的多个 batch
    # 之间保持稳定：在第一个 batch 上快照 at-rest 前缀。非 staging 路径
    # 仍然读取实时前缀。
    if self.enable_staging and req.early_send_prefix_end is None:
```

```
        req.early_send_prefix_end = max(
            0, len(req.prefix_indices) - req.host_hit_length
        )

    # 只有 bootstrap 完成的请求才能发送，避免在元数据分配前触碰 KV
    if req.pending_bootstrap:
        return

    # 只发送 device-resident 的缓存前缀，且页对齐保证 start_send_idx 精确
    cached_end = (
        req.early_send_prefix_end
        if self.enable_staging
        else len(req.prefix_indices) - req.host_hit_length
    )
    if cached_end <= req.start_send_idx:
        return
    if cached_end % self.token_to_kv_pool_allocator.page_size != 0:
        return
    # ... 后续按 page_start/page_end 构造 TransferKVChunk 并触发发送


# send_kv_chunk 中的网格对齐逻辑:
if self.enable_staging:
    # staging 按位置识别 chunk，非最后发送必须以网格边界结束:
    # 余量并入下一次发送，否则 chunk_idx 的按位置索引会错位
    grid_tokens = staging_grid_tokens(
        self.server_args.chunked_prefill_size, page_size
    )
    base = req.disagg_decode_prefix_len # 网格原点 = decode 侧前缀长度
    end_idx = base + ((end_idx - base) // grid_tokens) * grid_tokens
```

评论区精华

1. 页面不对齐的错误处理哲学（ShangmingCai）：'We should not raise and kill the prefill scheduler just because the decode side's req has a mismatched page size. Marked the req as failed（and maybe add this dst decode endpoint to a blacklist in the future）.' 该意见被采纳，后续 commit 02644c33 改为设置 _staging_failed 标记，避免单个请求拖垮整个 prefill 进程。
2. no-getattr-defensive 规则（ShangmingCai）：在 mooncake conn.py 中建议 'What about checking self.enable_staging first and then avoiding using getattr?' 并引用仓库规则文件；YAMY1234 回复 'Adjusted with the rule'，后续 commit 410affe6 把动态 _staging_counted 改为构造期声明的 TransferKVChunk.staging_counted 字段。
3. 超时配置复用（ShangmingCai 提问）：is_failed 是否应尊重 SGLANG_DISAGGREGATION_WAITING_TIMEOUT? YAMY1234 确认 completion_timeout 已在 __init__ 中从该环境变量初始化，语义一致。

4. 复杂度清理建议（ShangmingCai，APPROVE 时）：'Others look good. But the logic is becoming more and more complicated, we should do some cleanup in the near future.' 该意见保留了技术债，暂未在本 PR 解决。

- 页面不对齐时 raise 会杀死 prefill scheduler (correctness): 已改为失败当前请求，staging poll 路径以 KVPoll.Failed 暴露，避免拖垮整个 prefill 进程。
- 遵循 no-getattr-defensive 规则，用显式字段替代 getattr (style): 已按仓库规则重构，动态属性改为显式字段声明。
- is_failed 超时是否复用 SGLANG_DISAGGREGATION_WAITING_TIMEOUT (question): 已复用同一超时配置，无需额外改动。
- staging 逻辑复杂度与近期清理计划 (design): 未解决，作为技术债留待后续 PR 处理。

风险与影响

- 风险：
  1. 状态机与并发风险：staging_handler 新增成功 / 失败 / 超时三组状态与 _writer_counts，由 decode_thread、transfer_worker、scheduler 主线程共享；任一路径漏判 complete 条件都会导致 room 悬挂或提前销毁，回归面集中在 staging 协议核心。
  2. 双后端不一致风险：NIXL 与 Mooncake 各自维护一份近似但不同的 _staging_outstanding 与 teardown 逻辑，未来改动容易只改一半；本次 e2e 只覆盖单一 transfer_backend，mooncake 协议路径的测试覆盖不足。
  3. 页对齐约束：decode 前缀非页对齐（cache_protected_len % page_size != 0）时直接失败该请求；如果线上常见非页对齐前缀会抬高失败率，虽然这是有意降级，仍需要监控。
  4. 提前发送快照：early_send_prefix_end 在首个 batch 快照，若 overlap 调度下 prefix_indices 增长超前于快照，缓存前缀可能漏发；PR 声明快照覆盖已启动但未解析的 chunk，但缺少针对 overlap 场景的专项测试。
  5. 性能影响：非最后发送按网格向下取整会把尾部 token 推迟一个槽位，理论上增加少量时延；基准显示稳态总吞吐仅 -0.16%，可接受但需持续观察。 - 影响：用户侧：启用 heterogeneous-TP（如 DEP4 prefill → TP4 decode）加 decode radix cache 的部署从 ' 输出损坏 / 卡死 ' 变为正确可用；在 AgentX 256k 上下文负载下输出吞吐提升约 9.61%（281.02 vs 256.38 output TPS/user）。系统侧：staging 传输协议语义变化（每个 chunk 都发 CHUNK_READY、scatter 到达驱动、room 完成判定收紧），同时影响 NIXL 与 Mooncake 两套后端；未启用 staging 的常规 PD 路径不受影响。团队侧：维护复杂度明显上升，reviewer 已明确提出近期 cleanup 需求，后续需要统一两后端状态机并补充 mooncake 后端 e2e 测试。 - 风险标记：核心协议变更，双后端状态机同步，状态逻辑复杂，页对齐失败降级，提前发送快照语义

关联脉络

- PR #31217 [Disagg][StagingBuffer][1/2] Staging buffer robustness groundwork: PR body 明确说明本 PR 的首个 commit 与 #31217 相同，是 'Staging buffer + RadixCache' 两阶段工作的第一部分，必须先合并；本 PR 在其基础上叠加 radix cache 支

持。