PR #35198 完整报告

sgl-project/sglang

[Spec] Relay ngram accept tokens through the FutureMap

合并时间：2026-08-18 05:21

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35198

执行摘要

- 一句话：ngram 接受状态改走 FutureMap 中继，修复跨流竞态
- 推荐动作：值得精读，尤其是 overlap_utils.py 里 stash / resolve 的时序设计。它展示了一个典型的跨流竞态修复范式：让写者流负责读者流的数据准备，中间层只传索引。建议配合 #35059 一起阅读，理解 coarse / fine fence 的取舍，以及如何用 FutureMap 中继模式收敛多种 speculative 算法的状态传递。

功能与动机

PR body 明确描述了竞态成因：ngram 的 accept_tokens / accept_lens 由 forward 流在 verify 末尾写入，但当请求结束时 NgramVerifyInput.filter_batch 会在 schedule 流上 gather 这些张量，而 forward 可能仍在飞行中——这是对共享 buffer 的跨流先读后写。陈旧值到达 host 后会在下一轮 embedding gather 时崩溃。此前仅靠 #35059 引入的粗粒度 WAR fence 偶然兜底，作者希望通过 FutureMap 中继从根源修复，并顺带移除这个临时特例。

实现拆解

1. 中继 payload 扩展：overlap_utils.py 中 RelayPayload 新增 accept_tokens / accept_lens 可选字段，bonus_tokens 从必填降级为可选（ngram 不产生 bonus）。新增工厂方法 RelayPayload.from_ngram()，将 NgramVerifyInput 的扁平 accept_tokens 按 draft_token_num 重塑为 (batch, draft_token_num) 二维布局，便于按请求行 scatter 到 pool 索引 buffer。
2. FutureMap 增加 ngram 中继 buffer：FutureMap.__init__ 增加 accept_tokens_buf / accept_lens_buf；_maybe_init_ngram_bufs() 按首个 payload 形状惰性初始化，特意用 torch.zeros 而非 empty，保证未 stash 的行 resolve 为 accept_len == 0 的空 splice 而非垃圾长度。stash() 中原先的 ngram 直接 return 改为真实写入两个 buffer。
3. resolve 阶段接管消费：_resolve_spec_extras() 的 ngram 分支不再直接返回，而是按 draft_input.future_indices 从 buffer gather，回填 draft_input.accept_tokens / accept_lens。该 resolve 在 forward 流上按序执行，天然排在上一轮 stash 之后，跨流读写被消除。
4. filter/merge 只碰索引：ngram_info.py 的 filter_batch() / merge_batch() 在 future_indices 非 None 时仅切片 future_indices 并立即返回，不再触碰 accept_tokens / accept_lens，确保 schedule 流不读 forward 流仍在写的数据。

5. 调度入口与 fence 收尾：scheduler.py 的 _relay_forward_payload() 在 ngram 分支构造 RelayPayload.from_ngram() 并调用 future_map.stash()；ngram_worker.py 删除 shared_read_done_event = None 粗粒度 fence 特例，ngram 回归细粒度 shared-read fence。测试无新增文件，依赖现有 test_spec_ngram.py（18 个用例）与 test_spec_ngram_extra.py 回归，作者用 stall 放大法在改动前 100% 复现了竞态。

关键文件：

- python/sglang/srt/managers/overlap_utils.py（模块 重叠执行；类别 source；类型 core-logic；符号 RelayPayload.from_ngram, FutureMap._maybe_init_ngram_bufs, FutureMap._resolve_spec_extras, FutureMap.stash）：承载 ngram 中继的全部核心机制：RelayPayload.from_ngram 工厂方法、FutureMap 新增 accept_tokens_buf / accept_lens_buf、_maybe_init_ngram_bufs 惰性初始化、stash 的 ngram 写入分支、_resolve_spec_extras 的 ngram gather 分支。这是本次修复的主战场。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 Scheduler._relay_forward_payload）：调度入口：_relay_forward_payload 的 ngram 分支从 " 直接跳过 " 改为构造 RelayPayload.from_ngram 并调用 future_map.stash，是 ngram 状态真正进入 FutureMap 中继管线的触发点。
- python/sglang/srt/speculative/ngram_info.py（模块 投机解码；类别 source；类型 core-logic；符号 NgramVerifyInput.filter_batch, NgramVerifyInput.merge_batch）：filter_batch / merge_batch 的行为变化是修复正确性的关键：future_indices 非 None 时只切片索引并立即返回，不再触碰 accept_tokens / accept_lens，避免 schedule 流读取 forward 流仍在写的数据。
- python/sglang/srt/speculative/ngram_worker.py（模块 投机解码；类别 source；类型 core-logic）：删除 shared_read_done_event = None 的粗粒度 fence 特例，ngram verify 回归框架的细粒度 shared-read fence，是本次修复的收尾动作。

关键符号：RelayPayload.from_ngram, FutureMap._maybe_init_ngram_bufs, FutureMap._resolve_spec_extras, FutureMap.stash, Scheduler._relay_forward_payload, NgramVerifyInput.filter_batch, NgramVerifyInput.merge_batch

关键源码片段

python/sglang/srt/managers/overlap_utils.py

承载 ngram 中继的全部核心机制：RelayPayload.from_ngram 工厂方法、FutureMap 新增 accept_tokens_buf / accept_lens_buf、_maybe_init_ngram_bufs 惰性初始化、stash 的 ngram 写入分支、_resolve_spec_extras 的 ngram gather 分支。这是本次修复的主战场。

```
# ---- 生产侧：把 ngram verify 的接受结果打包成中继 payload ----
@classmethod
def from_ngram(cls, draft_input: NgramVerifyInput) -> RelayPayload:
    # accept_tokens 是一维连续 buffer，按 draft_token_num 重塑成
    # (batch, draft_token_num)，后续可以按 req 行号直接 scatter 进 pool buf
    return cls(
        bonus_tokens=None,
        accept_tokens=draft_input.accept_tokens.reshape(
```

```
            -1, draft_input.draft_token_num
        ),
        accept_lens=draft_input.accept_lens,
    )


# ---- 存储侧：pool 索引的 ngram 中继 buffer，zeros 兜底 ----
def _maybe_init_ngram_bufs(self, payload: RelayPayload) -> None:
    if self.accept_tokens_buf is not None:
        return
    # 用 zeros 而不是 empty：未 stash 的行 resolve 出来 accept_len == 0，
    # 在 draft prep 里是 " 空 splice"，不会拿到垃圾长度
    self.accept_tokens_buf = torch.zeros(
        (self.req_pool_size, payload.accept_tokens.shape[1]),
        dtype=payload.accept_tokens.dtype,
        device=self.device,
    )
    self.accept_lens_buf = torch.zeros(
        (self.req_pool_size,),
        dtype=payload.accept_lens.dtype,
        device=self.device,
    )


# ---- 中继主路径：forward 流上先写后读，schedule 流只碰索引 ----
def stash(self, future_indices: torch.Tensor, payload: RelayPayload) -> None:
    indices = future_indices
    if indices.shape[0] == 0:
        return # DP idle：payload 是空桩，提前退出避免 shape peek 越界
    if self.spec_algo.is_ngram():
        # ngram 分支只写 accept 相关 buffer，直接落 pool 索引行
        self._maybe_init_ngram_bufs(payload)
        self.accept_tokens_buf[indices] = payload.accept_tokens
        self.accept_lens_buf[indices] = payload.accept_lens
        return
    # eagle 等其他 spec 的 topk / hidden_states 中继保持不变……


# ---- 消费侧：下一轮迭代在 forward 流上按序 gather ----
def _resolve_spec_extras(self, batch: ScheduleBatch) -> None:
    if self.spec_algo.is_ngram():
        draft_input = batch.spec_info
        if draft_input is None or draft_input.future_indices is None:
            # FIXME：当前只支持连续 verify；混合 prefill/verify 模式未覆盖
            return
        indices = draft_input.future_indices
        if indices.shape[0] == 0:
            return
        # 在 forward 流上按序执行，stash 必然先于本 resolve 完成
```

```
        draft_input.accept_tokens = self.accept_tokens_buf[indices].flatten()
        draft_input.accept_lens = self.accept_lens_buf[indices]
        return
    #…… eagle 路径保持不变……
```

python/sglang/srt/managers/scheduler.py

调度入口：_relay_forward_payload 的 ngram 分支从 " 直接跳过 " 改为构造 RelayPayload.from_ngram 并调用 future_map.stash，是 ngram 状态真正进入 FutureMap 中继管线的触发点。

```
def _relay_forward_payload(
    self, future_indices: torch.Tensor, batch_result: GenerationBatchResult
) -> None:
    """Stash 本轮的中继 payload，供下一轮 resolve_forward_inputs 使用。"""
    if self.spec_algorithm.is_ngram():
        # ngram 之前直接跳过（状态走 batch.spec_info），现在也走 FutureMap：
        # 避免 schedule 流在 forward 还在飞行时读取共享 buffer
        if batch_result.next_draft_input is not None:
            payload = RelayPayload.from_ngram(batch_result.next_draft_input)
            self.future_map.stash(future_indices, payload)
        return
    # eagle / 非 spec 路径保持不变：
    # 有 draft 则从 draft_input 构造，否则用采样的 bonus token
    if batch_result.next_draft_input is not None:
        payload = RelayPayload.from_draft_input(batch_result.next_draft_input)
    elif batch_result.has_sampled_token_ids:
        payload = RelayPayload(bonus_tokens=batch_result.next_token_ids)
    else:
        return
    self.future_map.stash(future_indices, payload)
```

python/sglang/srt/speculative/ngram_info.py

filter_batch / merge_batch 的行为变化是修复正确性的关键：future_indices 非 None 时只切片索引并立即返回，不再触碰 accept_tokens / accept_lens，避免 schedule 流读取 forward 流仍在写的数据。

```
def filter_batch(
    self,
    new_indices: torch.Tensor,
    new_indices_cpu: Optional[List[int]] = None,
):
    if self.future_indices is not None:
        # 走 FutureMap 中继时，accept_tokens / accept_lens 不再在这里切片：
        # 这些 buffer 由 forward 流写入，schedule 流上读取会触发跨流竞态；
        # 值会在下一轮 verify 的 resolve 阶段按新索引重新 gather
        self.future_indices = self.future_indices[new_indices]
        return

    # 非 overlap 的旧路径：本地直接维护 accept 状态
```

```
        if self.new_seq_lens is not None:
            self.new_seq_lens = self.new_seq_lens[new_indices]
        self.accept_tokens = self.accept_tokens.reshape(-1, self.draft_token_num)[
            new_indices, :
        ]
        self.accept_tokens = self.accept_tokens.flatten()
        self.accept_lens = self.accept_lens[new_indices]


    def merge_batch(self, spec_info: NgramVerifyInput):
        if self.future_indices is not None:
            # 同上：合并时只拼接索引，accept 数据由 FutureMap 统一管理
            assert spec_info.future_indices is not None
            self.future_indices = torch.cat(
                (self.future_indices, spec_info.future_indices), dim=0
            )
            return

        # 非 overlap 的旧路径保持不变
        if self.new_seq_lens is not None:
            assert spec_info.new_seq_lens is not None
            self.new_seq_lens = torch.cat(
                (self.new_seq_lens, spec_info.new_seq_lens), dim=0
            )
        self.accept_tokens = torch.cat(
            (self.accept_tokens, spec_info.accept_tokens), dim=0
        )
        self.accept_lens = torch.cat((self.accept_lens, spec_info.accept_lens), dim=0)
```

评论区精华

PR 没有 Review 评论线程。Issue 评论区仅有作者触发的 /rerun-test 命令与 CI 结果回传：test_spec_ngram.py 与 test_spec_ngram_extra.py 均在 1-gpu-h100 上通过。PR body 中作者说明了验证手段：test_spec_ngram.py 18 个用例通过，其中包含一个 stall 放大的复现用例，改动前 100% 复现竞态崩溃。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. filter/merge 行为变化：ngram_info.py 在 future_indices 非 None 时跳过 accept 字段切片，任何在 filter 之后、resolve 之前读取 accept_tokens / accept_lens 的新增路径都会拿到上一轮旧值。当前流程中下游读取都在下一轮 forward 流的 resolve 之后，顺序安全，但后续迭代间新增读取点需要警惕。
  2. buffer 未初始化风险：_resolve_spec_extras 的 ngram 分支直接索引 accept_tokens_buf，若出现 future_indices 非空但从未 stash 的路径（如混合 prefill/verify 模式），会触发 AttributeError。代码中的 FIXME 注释已标注 mixed

mode 尚未支持，这与 eagle 路径的既有限制一致。

3. fence 语义变化：移除粗粒度 shared_read_done_event = None 后，ngram verify 的共享状态读取改由框架细粒度 fence 保护。grammar 路径走 host tree 同步读取（注释确认无 pending 异步），但其他共享设备状态若未被 fence 覆盖，可能暴露新竞态。
4. 测试覆盖：没有新增回归测试文件，stall 放大复现用例未固化进测试套件，后续改动可能重新引入同类竞态。
5. 平台差异：FutureMap 的 pinned host copy 与私有 D2H 流是 CUDA-only 的，非 CUDA 平台走 plain .cpu() 路径，本改动未对 NPU/XPU 单独验证。 - 影响：启用 overlap + ngram 组合的用户将不再遇到偶发的 embedding gather 崩溃或错误 token；ngram 从粗粒度 WAR fence 回到细粒度 shared-read fence，理论上减少 schedule 流与 forward 流之间的同步点，恢复部分 overlap 收益，但 PR body 未给出量化数据。架构层面，ngram 与 eagle 的状态中继统一到 FutureMap，为代码中 FIXME 标注的 precomputed draft 支持铺路；团队层面，这是 speculative 路径持续演进的一环，与 #35059 的 shared-read fence 重构、#34478 的 DSpark spec 支持属于同一功能线。 - 风险标记：跨流同步竞态修复，无新增回归测试，依赖 resolve 时序，移除粗粒度 fence，混合模式未覆盖

关联脉络

- PR #35059 [Spec] Resolve shared-read ends from the backend declaration alone: 本 PR 的目标正是移除 #35059 为 ngram 添加的粗粒度 shared-read fence 特例（ngram_worker.py 中 shared_read_done_event = None）。#35059 用 coarse fence 偶然排序了 ngram 的跨流竞态，本 PR 用 FutureMap 中继从根源修复后将其删除，二者构成 " 临时绕过 → 根治 " 的演进对。
- PR #34478 [Spec] Support output logprobs with DSpark: 同属 speculative 路径的近期改动，涉及共享投机处理器与 DSpark（ngram 的兄弟算法），与本 PR 在 speculative 状态传递机制的演进上有共同脉络。