PR #35071 完整报告

sgl-project/sglang

[PD] Overlap prefill DP-rank bootstrap queries

合并时间：2026-08-19 17:25

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35071

执行摘要

- 一句话：DP-rank 查询提前异步提交，*decode* 调度吞吐提升 1.36%
- 推荐动作：值得精读，尤其是对关注 PD 调度性能的工程师。核心看点是：如何在不改变权威结果语义的前提下，利用调度周期内已知信息与在途计算重叠；以及如何用测量数据（prefetch lead 时间、A/B 收益）驱动设计决策——删掉低收益的第二次 prefetch。建议结合 commit 历史（两个 commit 展示了从双 prefetch 到单 prefetch 的收敛）与 review 讨论一起阅读。

功能与动机

PR body 明确指出：原先 *decode* 调度器在结果消费点同步查询 prefill bootstrap server 获取 DP-rank 路由，这个 HTTP 往返处于调度器关键路径上，即使 pending 的 bootstrap rooms 在请求 intake 之前就已知，且上一轮 *decode* graph 可能仍在执行。作者意图利用这两个时间窗重叠查询，将延迟从关键路径中移除，且不改变消费点权威结果、重试或 receiver 初始化路径。

实现拆解

1. 状态容器：在 DecodePreallocQueue.__init__中新增 _prefill_dp_rank_queries 字典，键为 bootstrap 地址，值为 (rooms 元组 , Future)，并导入 concurrent.futures.Future。
2. 预取入口：新增 prefetch_prefill_dp_rank_queries()，先按 bootstrap_addr 聚合 pending_reqs，取消已无 pending 请求的陈旧 Future；对每个地址跳过已在途查询、prefill_info_table 未发布且 _resolve_prefill_dp_rank 已能命中的房间，再通过 kv_manager._ensure_prefill_recompute_executor().submit 提交 CommonKVReceiver.query_prefill_dp_ranks 为异步 Future。
3. 消费点改造：_resolve_pending_reqs() 中，需要查询的地址优先从 _prefill_dp_rank_queries 弹出预取条目，若当前 rooms 与预取房间集合一致则使用 future.result()，否则走原同步查询兜底，保持原有重试与 error fallback 语义不变；空 pending_reqs 分支调用 _cancel_prefill_dp_rank_queries 清理。
4. 生命周期清理：新增 _cancel_prefill_dp_rank_queries()，遍历取消所有在途 Future 并清空字典，在 release_memory_occupation() 中调用，避免队列清理后 Future 悬空；
5. 调度循环接入：在 event_loop_normal_disagg_decode 与 overlap 的 pop_and_process 循环顶部各添加一次 prefetch 调用，review 后根据测量数据（顶部调用平均提前 17.35

ms，低位调用仅 0.8 ms）删除了低位调用，只保留每个迭代顶部的单次预取；

6. 测试配套：test_decode_queue_cleanup.py 新增 test_prefetches_prefill_dp_rank_query，验证 executor.submit 只提交预取地址与房间、消费点优先使用 Future 结果、新进入的请求走同步兜底查询、receiver init 按 DP-rank 顺序正确调用。

关键文件：

- python/sglang/srt/disaggregation/decode.py（模块 解码队列；类别 source；类型 core-logic；符号 prefetch_prefill_dp_rank_queries, _cancel_prefill_dp_rank_queries）：核心实现文件：新增预取状态容器、prefetch_prefill_dp_rank_queries 与 _cancel_prefill_dp_rank_queries，改造 _resolve_pending_reqs 消费 Future，并接入两个调度主循环顶部；改动集中在 DecodePreallocQueue（+81/-3）。
- test/registered/unit/disaggregation/test_decode_queue_cleanup.py（模块 队列清理；类别 test；类型 test-coverage；符号 test_prefetches_prefill_dp_rank_query, decode_req）：新增 test_prefetches_prefill_dp_rank_query 单元测试，验证预取只提交一次、消费点正确使用 Future 结果、新请求走同步兜底且 receiver 初始化顺序正确（+46/-0）。

关键符号：prefetch_prefill_dp_rank_queries, _cancel_prefill_dp_rank_queries, _resolve_pending_reqs, test_prefetches_prefill_dp_rank_query

关键源码片段

python/sglang/srt/disaggregation/decode.py

核心实现文件：新增预取状态容器、prefetch_prefill_dp_rank_queries 与 _cancel_prefill_dp_rank_queries，改造 _resolve_pending_reqs 消费 Future，并接入两个调度主循环顶部；改动集中在 DecodePreallocQueue（+81/-3）。

```
# python/sglang/srt/disaggregation/decode.py
# DecodePreallocQueue 内部：将原本在消费点同步执行的 DP-rank 查询
# 提前到每个调度迭代开头提交，使 HTTP 往返与请求接收、上一轮 decode graph 重叠。

def prefetch_prefill_dp_rank_queries(self) -> None:
    """提前启动 DP-rank 查询，使其与请求接收和上一轮 decode graph 重叠执行。"""
    if not self.pending_reqs:
        return

    queries = self._prefill_dp_rank_queries

    # 按 bootstrap 地址聚合本轮待查询的请求，保证同一地址的 rooms 批量提交一次。
    addr_to_reqs: Dict[str, List[DecodeRequest]] = {}
    for decode_req in self.pending_reqs:
        addr = _bootstrap_addr(decode_req.req)
        addr_to_reqs.setdefault(addr, []).append(decode_req)

    # 清理已无 pending 请求的陈旧查询，避免孤儿 Future 长期占住 executor。
    for bootstrap_addr in set(queries) - set(addr_to_reqs):
        _, stale_future = queries.pop(bootstrap_addr)
        stale_future.cancel()
```

```
    for bootstrap_addr, decode_reqs in addr_to_reqs.items():
        if bootstrap_addr in queries:
            continue # 该地址已有在途查询，保持 FIFO，不重复提交
        if self.kv_manager.prefill_info_table.get(bootstrap_addr) is None:
            continue # prefill 侧尚未发布并行信息，查询必然失败，留给消费点兜底

        # 只查询尚未缓存的房间；_resolve_prefill_dp_rank 命中缓存时可直接使用。
        rooms = tuple(
            decode_req.req.bootstrap_room
            for decode_req in decode_reqs
            if self._resolve_prefill_dp_rank(decode_req.req) is None
        )
        if not rooms:
            continue

        # 通过 KV-manager 的 executor 异步提交，查询与后续请求 intake 并行。
        future = self.kv_manager._ensure_prefill_recompute_executor().submit(
            CommonKVReceiver.query_prefill_dp_ranks,
            bootstrap_addr,
            list(rooms),
        )
        queries[bootstrap_addr] = (rooms, future)


def _resolve_pending_reqs(self) -> None:
    """批量解析 DP rank 并初始化 receiver（原同步消费点，这里展示核心分支）。"""
    # ... 既有分组与 need_query 判定逻辑保持不变 ...
    if need_query:
        rooms = [decode_req.req.bootstrap_room for decode_req in need_query]
        # 优先消费预取的 Future；房间集合变化时回退同步查询，保证权威结果与重试语义。
        prefetched = self._prefill_dp_rank_queries.pop(bootstrap_addr, None)
        prefetched_rooms = prefetched[0] if prefetched is not None else None
        if prefetched is not None and tuple(rooms) == prefetched_rooms:
            room_to_rank = prefetched[1].result()
        else:
            room_to_rank = CommonKVReceiver.query_prefill_dp_ranks(
                bootstrap_addr, rooms
            )
```

评论区精华

核心争议点是双重 prefetch 是否造成重复执行。ShangmingCai 在 event_loop_normal_disagg_decode 的 diff 上指出："Should we only keep one prefetch_prefill_dp_rank_queries? The logic here seems a little bit confusing after checking if not self._engine_paused: and then also adding a prefetch_prefill_dp_rank_queries after if self._engine_paused:, it looks like this prefetch will be done twice when engine is not paused?"，并在 overlap 循环处回复 "ditto"。YAMY1234 承认并解释："I removed the second call and now prefetch only once

at the top of each scheduler iteration. Instrumentation showed about 17.35 ms of leadfor the top call versus about 0.8 ms for the lower call, and the clean top-only AgentXA/B remained positive at +1.36% Output TPS/User."，最终保留单次预取，两个 review 线程均以设计修正解决。

- 双重 prefetch 是否在非暂停状态重复执行 (performance): YAMY1234 接受建议，移除低位调用，只保留每个调度迭代顶部的单次预取，并以测量数据佐证：顶部调用平均提前 17.35 ms，低位调用仅 0.8 ms 且多已由顶部提交。
- overlap 循环中的相同双 prefetch 问题 (design): YAMY1234 同步更新 overlap 循环，仅保留顶部 prefetch；低位调用可重叠的时间窗口太短，收益可忽略。

风险与影响

- 风险：
  1. 调度器关键路径变更：prefetch 调用位于 decode 调度主循环（event_loop_normal_disagg_decode 与 overlap 循环）顶部，虽移除了 HTTP 同步等待，但提交逻辑本身（遍历 pending_reqs、构建字典、过滤房间）仍在迭代内同步执行，复杂度增加可能引入回归；
  2. Future 生命周期：Future.cancel() 对已启动任务无效，取消后底层 HTTP 请求可能仍在执行，产生无效查询但结果被丢弃，资源浪费有限；已在 release_memory_occupation 与空队列分支处理，但未覆盖极端时序；
  3. 房间集合一致性：消费时若请求集合与预取时不同（如请求被 abort 或新请求加入），会回退同步查询，正确性有保障但增加了控制流分支，需关注 FIFO 语义是否被破坏；
  4. 缺少专门取消路径测试：_cancel_prefill_dp_rank_queries 的取消行为没有独立测试用例，回归风险集中在队列清理场景；
  5. 验证范围：性能数据来自单一工作负载（Qwen3.5-397B-A17B NVFP4，24 GB300 GPUs，4P+1D），对不同模型规模和并发形态的普适性未证明。 - 影响：影响范围限定在 PD（prefill-decode disaggregation）模式的 decode 调度器。对生产用户，期望获得约 1.36% 的 Output TPS/User 提升，且精度测试（GSM8K 0.995）与零 retraction、零 error marker 表明无正确性回退。对系统，decode 调度循环每迭代多一次字典构建与查询开销，但换取同步等待从 1.140 ms 降至 0.265 ms 的收益。对团队，该模式（在已知信息可用时提前异步提交、消费时复用）可作为后续调度器关键路径优化的参考范式，但需注意其仅覆盖预取受益场景。 - 风险标记：调度器关键路径变更，并发 Future 生命周期管理，缺少取消路径专项测试，单工作负载验证

关联脉络

- PR #35070 （PR body 提及的上游 PREBUILT 预取相关 PR）：本 PR body 明确说明功能上独立于 #35070，但生产验证是在 #35070 启用状态下进行的，性能数据为该基线之上的增量；两者属于同一 PD 调度优化功能线。
- PR #36219 [Performance] Tune FlashInfer EXTEND for DP prefill: 同为 DP prefill/decode 调度器关键路径性能优化，虽侧重 FlashInfer 内核调优，但目标都是缩短 PD 模式下 prefill 侧关键路径耗时，属于同一性能优化脉络。