PR #49206 完整报告

vllm-project/vllm

fix: resolve silent request skipping in PRIORITY scheduling

合并时间：2026-08-07 06:21

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/49206

执行摘要

- 一句话：修复 PRIORITY 调度抢占后请求被静默跳过
- 推荐动作：值得精读。虽然源码改动仅 12 行，但这是典型的“遍历中删除元素导致游标错位”的经典陷阱，且该 bug 为静默错误（不报错、不崩溃，只表现为请求被跳过），复现与根因分析过程很有教学价值；回归测试用三个不同优先级请求精确构造触发路径，适合作为调度器单元测试的模板。关注点：req_index 与 self.running 删除位置的关系，以及 FCFS/PRIORITY 两条抢占分支的对比。

功能与动机

issue #49097 指出：在 SchedulingPolicy.PRIORITY 下，preempted_req 通过 max(self.running, key=lambda r: (r.priority, r.arrival_time)) 选出，与列表位置无关；而 req_index 只在 victim 已在本轮被调度（在 scheduled_running_reqs 中）时才递减。当 victim 位于游标之前但本轮被 DP prefill 节流、pipeline 资格门控或 encoder 预算耗尽等 continue 路径跳过时，self.running.remove() 使后续元素左移而游标不动，下一次迭代就会越过本应处理的请求，造成整个 step 的静默跳过。PR body 也明确说明这是 req_index 记账未正确调整导致的逻辑 bug。

实现拆解

1. 根因定位：issue #49097 结合 base 版本 scheduler.py 第 591-614 行确认，PRIORITY 抢占分支仅在 victim 属于 scheduled_running_reqs 时执行 req_index -= 1；对“位于游标之前但本轮被跳过”的 victim 缺少处理，remove() 造成的左移使外层循环访问错位。
2. 核心修复：在 vllm/v1/core/sched/scheduler.py 的 PRIORITY 分支中，先用 self.running.index(preempted_req) 记录 victim_index，再用 del self.running[victim_index] 删除；当 victim_index < req_index 时递减 req_index。同时把原来 scheduled_running_reqs 分支内的 req_index -= 1 移除——因为已调度 victim 必在游标之前，统一由新条件覆盖，逻辑等价且更完整。
3. 回归测试：新增 tests/v1/core/test_priority_preemption_bug.py，构造 A（priority=9、200 token、最早到达）、B（priority=0、15 token）、C（priority=1、1 token）三个请求，在 throttle_prefills=True 触发抢占时断言 C 仍出现在 num_scheduled_tokens 中且保持 RequestStatus.RUNNING。
4. Review 调整：按 njhill 建议用 del 替代 remove 避免二次线性查找，并清理多余空行与注释；随后通过多次 merge main 同步后合入。

关键文件：

- vllm/v1/core/sched/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 Scheduler.schedule）：修复点所在：PRIORITY 抢占分支中记录 victim 下标并按位置条件递减 req_index，消除请求静默跳过；同时把 req_index 调整从 scheduled_running_reqs 分支统一外提，逻辑更完整。
- tests/v1/core/test_priority_preemption_bug.py（模块 调度器；类别 test；类型 test-coverage；符号 build_scheduler, make_request, mock_output, test_priority_scheduler_preempt_skipped_request）：新增回归测试，用三个不同优先级请求复现抢占路径下的游标错位，断言中等优先级请求 C 在节流抢占后仍被调度并保持 RUNNING。

关键符号：Scheduler.schedule, test_priority_scheduler_preempt_skipped_request, build_scheduler, make_request, mock_output

关键源码片段

vllm/v1/core/sched/scheduler.py

修复点所在：PRIORITY 抢占分支中记录 victim 下标并按位置条件递减 req_index，消除请求静默跳过；同时把 req_index 调整从 scheduled_running_reqs 分支统一外提，逻辑更完整。

```
# 请求无法获得 KV 块时，进入抢占循环
while True:
    new_blocks = self.kv_cache_manager.allocate_slots(
        request,
        num_new_tokens,
        num_lookahead_tokens=self.num_lookahead_tokens,
    )
    if new_blocks is not None:
        break # 分配成功，正常调度

    # PRIORITY 策略：抢占全局最低优先级请求作为 victim
    if self.policy == SchedulingPolicy.PRIORITY:
        preempted_req = max(
            self.running,
            key=lambda r: (r.priority, r.arrival_time),
        )
        # 先记录 victim 在 running 列表中的下标，再删除
        victim_index = self.running.index(preempted_req)
        del self.running[victim_index]
        # 关键修复：若 victim 位于当前迭代游标之前（即便本轮
        # 只是被 continue 跳过、尚未调度），删除后列表左移，
        # 必须同步递减 req_index，否则下一请求会被静默跳过
        if victim_index < req_index:
            req_index -= 1

        if preempted_req in scheduled_running_reqs:
            # victim 本轮已调度过，回收其 token 预算与 KV 分配
```

```
            preempted_req_id = preempted_req.request_id
            scheduled_running_reqs.remove(preempted_req)
            token_budget += num_scheduled_tokens.pop(preempted_req_id)
            req_to_new_blocks.pop(preempted_req_id)
            scheduled_spec_decode_tokens.pop(preempted_req_id, None)
            preempted_encoder_inputs = scheduled_encoder_inputs.pop(
                preempted_req_id, None
            )
            if preempted_encoder_inputs:
                # 恢复被抢占请求占用的 encoder 计算预算
                num_embeds_to_restore = sum(
                    preempted_req.get_num_encoder_embeds(i)
                    for i in preempted_encoder_inputs
                )
                encoder_compute_budget += num_embeds_to_restore
        else:
            # FCFS 策略：直接弹出队尾请求
            preempted_req = self.running.pop()

        self._preempt_request(
            preempted_req,
            scheduled_timestamp,
            drop_stale_output=self.requires_kv_delivery,
        )
        preempted_reqs.append(preempted_req)
        if preempted_req == request:
            # 已无其他请求可抢占，当前请求本轮无法调度
            break
```

tests/v1/core/test_priority_preemption_bug.py

新增回归测试，用三个不同优先级请求复现抢占路径下的游标错位，断言中等优先级请求 C 在节流抢占后仍被调度并保持 RUNNING。

```
# 前置：build_scheduler() 内部以 SchedulerConfig(policy="priority", ...) 构建，
# 且 KV 池仅 6 块（NUM_BLOCKS = 6），保证后续抢占真实发生
def test_priority_scheduler_preempt_skipped_request():
    scheduler = build_scheduler()

    # A：最差优先级且最早到达，会被选为抢占 victim；
    # B：最好优先级；C：中等优先级，位于 A 与 B 之间
    A = make_request("A", num_tokens=200, priority=9, arrival_time=1.0)
    scheduler.add_request(A)
    out = scheduler.schedule()
    scheduler.update_from_output(out, mock_output(out))

    B = make_request("B", num_tokens=15, priority=0, arrival_time=2.0)
    scheduler.add_request(B)
    out = scheduler.schedule()
    scheduler.update_from_output(out, mock_output(out))
```

```
C = make_request("C", num_tokens=1, priority=1, arrival_time=3.0)
scheduler.add_request(C)
out = scheduler.schedule()
scheduler.update_from_output(out, mock_output(out))

# 触发 prefill 节流，迫使 C 抢占 A；旧代码因游标错位
# 会让 C 本轮静默跳过，修复后 C 应正常出现在调度输出中
out = scheduler.schedule(throttle_prefills=True)
is_c_scheduled = "C" in out.num_scheduled_tokens
assert is_c_scheduled, (
    "Bug present: Request C was silently skipped during scheduling "
    "because the req_index was not decremented after preempting A."
)
assert C.status == RequestStatus.RUNNING
```

评论区精华

> njhill：should be slightly more efficient —— 既然已经用 index() 拿到了下标，直接 del self.running[victim_index] 即可，避免 remove() 再做一次线性查找。作者回复 @njhill Done，最终版本采纳。 njhill（nit）：两处多余的空行与注释行建议删除，保持补丁最小化。作者均已落实。 njhill 最终给出 APPROVED，CI 由维护者触发（Buildkite CI #82707）。

- 用 del 替代 remove 提升删除效率 (performance): 作者回复 @njhill Done，最终提交采用 del 写法。
- 删除多余空行与注释的 nit 建议 (style): 作者已按要求清理，最终版本保留 4 行必要注释。

风险与影响

- 风险：
  - 核心路径变更：Scheduler.schedule() 是 vLLM v1 每步执行的热点路径，PRIORITY 分支的游标逻辑影响所有启用 PRIORITY 策略的请求的调度正确性；虽然修复仅 10 行，但属于核心调度语义变更。
  - 逻辑等价性验证：req_index -= 1 从 scheduled_running_reqs 分支外提后，需确认已调度 victim 必满足 victim_index < req_index（按循环顺序遍历成立），新条件覆盖原逻辑且补上了缺失分支。
  - 复杂度：index() 与 del 均为 O(n)，与原 remove() 同级，无性能退化；但每次抢占多一次线性扫描，在超大规模 running 列表下可忽略。
  - 测试覆盖局限：回归测试仅覆盖 PRIORITY + 单抢占场景，DP 多请求、encoder 预算回收等分支未有专门用例。
- 影响：
  - 用户影响：PRIORITY 策略下，KV 压力触发抢占时不再出现请求整步静默饥饿，延迟与吞吐更可预期；FCFS 等其他策略走 pop() 分支，行为完全不变。
  - 系统影响：调度器输出更稳定，preempted_reqs、scheduled_running_reqs 与 token 预算回收逻辑保持一致。
  - 团队影响：新增回归测试为后续调度器改动提供保护网，特别是对索引记账类改动。

- 风险标记：核心路径变更，索引游标易回归，仅 PRIORITY 策略受影响

关联脉络

- PR #51113 [Bugfix] Keep mamba align prefill chunks block-aligned past last_cache_position: 同样修改 vllm/v1/core/sched/scheduler.py，同属调度器预填充 / 抢占链路的边界 bugfix，与本 PR 共享同一调度循环逻辑，改动时需互相回归。
- PR #50613 [Attention][MLA] Per-request scheduling for MLA chunked context: 同属请求级调度正确性与公平性演进线，per-request 调度让 prefill/ 抢占路径更加细分，与本 PR 的游标校正逻辑相关。