PR #6090 完整报告

verl-project/verl

[fully_async] fix: allow drain loop to resume early on parameter sync when partial_rollout is enabled

合并时间：2026-04-21 20:30

原文链接：http://prhub.com.cn/verl-project/verl/pull/6090

执行摘要

- 一句话：修复完全异步 rollouter 在部分 rollout 恢复时因 drain loop 阻塞导致无法及时分发新样本的问题。
- 推荐动作：该 PR 值得精读，特别是 _processor_worker 方法中的 drain 循环重写展示了如何优雅处理异步任务管理和中断信号。关注点包括： 1) asyncio.Event 与 Condition 的选择权衡； 2) 部分 rollout 启用 / 禁用时的不同处理策略； 3) 锁粒度优化以避免性能瓶颈。

功能与动机

根据 PR body 描述，当启用 partial_rollout 时，一旦 _processor_worker 进入 drain 循环（while self.active_tasks: ...），就会一直阻塞直到所有进行中任务完成。这导致新样本无法及时分发到空闲副本，影响了训练效率。修复目标是让 drain 循环可中断，以便在参数同步时能提前恢复样本分发。

实现拆解

1. 同步原语重构：在 _init_async_objects 方法中，将原有的 asyncio.Condition 和其内部锁替换为独立的 asyncio.Lock 和 asyncio.Event。lock 保护共享状态（paused/active_tasks/staleness_samples 等），_resume_event 作为运行状态信号。
2. 状态重置逻辑更新：在 reset_staleness 方法中，移除 condition.notify_all() 调用，改为设置 _resume_event.set() 来唤醒 drain 循环。
3. drain 循环重写：在 _processor_worker 方法中，根据 partial_rollout 配置采用不同策略：
   - 启用时：创建 resume_future 监听恢复事件，使用 asyncio.wait 同时等待任务完成和恢复信号，收到信号后立即退出循环。
   - 禁用时：保持原有逻辑但修复锁竞争问题。
4. 无测试配套：本次变更未包含测试文件修改，属于核心逻辑修复。

关键文件：

- verl/experimental/fully_async_policy/fully_async_rollouter.py（模块 完全异步；类别 source；类型 core-logic；符号 _init_async_objects, reset_staleness, _processor_worker）：这是本次 PR 唯一修改的文件，包含了完全异步 rollouter 的核心逻辑变更，特别是 drain 循环的重写和同步原语重构。

关键符号：_init_async_objects, reset_staleness, _processor_worker

关键源码片段

verl/experimental/fully_async_policy/fully_async_rollouter.py

这是本次 PR 唯一修改的文件，包含了完全异步 rollouter 的核心逻辑变更，特别是 drain 循环的重写和同步原语重构。

```
def _init_async_objects(self):
    # Initialize asyncio synchronization primitives.
    # `lock` protects shared state: paused / active_tasks / staleness_samples / timing fields.
    self.lock = asyncio.Lock()
    # `_resume_event` signals that the rollouter is currently running (paused == False).
    self._resume_event = asyncio.Event()
    self._resume_event.set() # 初始设置为运行状态

async def reset_staleness(self):
    """
    Reset staleness samples after parameter update.
    Returns timing_raw dictionary for metrics.
    """
    async with self.lock:
        self.paused = False
        # 唤醒 _processor_worker 中的 drain 循环，使其能提前退出并恢复向空闲副本提交新样本
        # 而不是等待所有进行中的长尾任务完成
        self._resume_event.set()
        # 每次参数更新后，重置陈旧样本计数
        self.staleness_samples = len(self.active_tasks) + await self.message_queue_client.get_
        queue_size()
        # ... 后续计时和日志逻辑保持不变
        return timing_raw

async def _processor_worker(self):
    """
    Streaming worker coroutines, a sample is submitted for processing without waiting for
    batches
    """
    partial_rollout_enabled = self.config.async_training.get("partial_rollout", False)
    while True:
        if self.paused or await self._should_pause_generation():
            print("[FullyAsyncRollouter][Processor] Received pause signal, waiting for remaining
            tasks...")
            async with self.lock:
                self.paused = True
                self._resume_event.clear() # 清除事件，准备等待恢复信号

            resume_future = asyncio.ensure_future(self._resume_event.wait())
            try:
                # Drain 循环：等待 (a) 至少一个活跃任务完成，或 (b) 恢复信号（reset_staleness/
```

```
            monitor 设置 paused=False)
            # 以提前中断 drain，从而能向空闲副本提交新样本。
            # 在等待期间不持有锁，因此发布者可以并发获取锁来更新 paused/staleness_samples。
            while self.active_tasks and not resume_future.done():
                wait_set = set(self.active_tasks) | {resume_future}
                done, _pending = await asyncio.wait(wait_set, return_when=asyncio.FIRST_
                COMPLETED)
                actual_done = done - {resume_future}
                if actual_done:
                    async with self.lock:
                        for task in actual_done:
                            self.active_tasks.discard(task)
                            await task # 处理已完成任务
                if resume_future in done:
                    print("[FullyAsyncRollouter][Processor] Resume event fired, breaking drain
                    loop")
                    break
        finally:
            resume_future.cancel() # 确保清理 future
    # ... 正常处理逻辑
```

评论区精华

gemini-code-assist[bot] 在 review 中指出了几个关键问题：

1. 锁竞争问题：在 partial_rollout 禁用时的 else 块中，self.lock 在 await asyncio.wait 期间被持有，会阻塞 reset_staleness 等方法的调用，导致训练管道性能停滞。
2. 集合管理不一致：else 块中直接替换 self.active_tasks 集合引用，可能导致内存泄漏和任务丢失。
3. 错误处理缺陷：对已完成任务的 await 操作可能抛出异常。 建议采用统一的循环结构来提升健壮性和性能。ArronHZG 随后批准了 PR，表明问题已得到解决或接受。

- drain 循环实现中的锁竞争和集合管理问题 (design)：建议采用统一的循环结构来提升健壮性和性能，ArronHZG 的批准表明问题已解决或接受。

风险与影响

- 风险：
  1. 并发逻辑风险：新的 drain 循环逻辑涉及复杂的异步等待和锁管理，如果实现有误可能导致死锁或竞态条件。
  2. 向后兼容性：同步原语从 Condition 改为 Lock+Event，可能影响依赖原有 Condition 通知机制的其他代码路径。
  3. 部分 rollout 配置依赖：修复仅针对 partial_rollout 启用时生效，如果配置不一致可能无法解决原始问题。
  4. 测试覆盖不足：变更未包含测试更新，回归风险较高。
- 影响：

1. 性能提升：解决了 drain 循环阻塞问题，使空闲副本能更快接收新样本，提高完全异步训练的资源利用率。
2. 用户体验：训练流程更流畅，减少因等待长尾任务完成造成的人为停顿。
3. 系统影响：仅影响完全异步 rollouter 模块，不涉及其他训练器或 rollout 后端。
4. 团队影响：需要确保所有使用 partial_rollout 的团队了解此修复，并验证其训练脚本的兼容性。 - 风险标记：并发逻辑变更，缺少测试覆盖，配置依赖

关联脉络

- PR #6069 [fully_async] fix: fix rollouter/idle compute in async-mode: 同样修改了 fully_async_rollouter.py 文件，修复了空闲比计算逻辑，属于同一模块的连续修复。
- PR #6052 [fully_async] fix: avoid blocking ray.get inside async actor methods: 都涉及完全异步训练器的并发问题修复，关注异步环境下的阻塞和性能优化。
- PR #6046 [fully_async] fix: preserve per-iteration routed_experts on partial rollout resume: 都处理 partial_rollout 恢复时的逻辑问题，本 PR 修复样本分发，6046 修复专家路由。