PR #6544 完整报告

verl-project/verl

[reward] fix: release worker count when NaiveRouter request fails after retries

合并时间：2026-06-02 11:06

原文链接：http://prhub.com.cn/verl-project/verl/pull/6544

执行摘要

- 一句话：修复 NaiveRouter 请求失败后 worker 计数泄漏
- 推荐动作：建议审批前要求作者修复 body 读取的泄漏问题，并补充针对此修复的单元测试。代码整体方向正确，设计思路值得学习。

功能与动机

原本 _release_worker() 只在请求成功路径上调用，当所有 max_attempts 失败并抛出 RuntimeError 或重新抛出的异常时，worker 的计数永远不会递减，导致泄漏。由于 _select_worker() 使用 min(request_counts) 选择 worker，不断失败的 worker 会累积虚假负载，永久破坏负载均衡。

实现拆解

1. 将 retry 循环移入 try 块：在 verl/experimental/reward_loop/router/naive_router.py 的 _make_async_request 方法中，把原来的 for 循环整体包裹在一个 try 块内部。
2. 添加 finally 块确保释放：在 try 块的 finally 中，无条件调用 self._release_worker(worker_url)，确保无论是成功返回、重试耗尽、还是异常抛出，都能平衡 _select_worker() 的递增。
3. 删除成功路径中的冗余释放：原本成功时在 return 前调用的 self._release_worker(worker_url) 被移除，避免重复释放。

关键文件：

- verl/experimental/reward_loop/router/naive_router.py（模块 路由器；类别 source；类型 core-logic；符号 _make_async_request）：核心修改文件，修复 worker 计数泄漏问题

关键符号：_make_async_request

关键源码片段

verl/experimental/reward_loop/router/naive_router.py

核心修改文件，修复 worker 计数泄漏问题

```
async def _make_async_request(self, request: Request, endpoint: str):
    """Proxy single request to a worker URL."""
    if not self.worker_urls:
```

```
        return JSONResponse(status_code=503, content={"error": "No available workers"})

    worker_url = self._select_worker() # 递增该 worker 的请求计数
    target_url = f"{worker_url}/{endpoint}"

    if self.verbose:
        logger.debug(f"[router] Forwarding request → {target_url}")

    # Copy request data
    body = await request.body() # 注意：此处异常也会跳过 finally!
    headers = dict(request.headers)

    try:
        for attempt in range(self.max_attempts):
            # Send request to worker
            try:
                async with self.client.request(
                    request.method, target_url, data=body, headers=headers
                ) as response:
                    response.raise_for_status()
                    output = await _read_async_response(response)
                    return output # 成功时通过 finally 释放
            except asyncio.TimeoutError:
                logger.warning(f"Async request to {endpoint} timed out (attempt {attempt + 1})")
            except aiohttp.ClientConnectorError:
                logger.warning(f"Connection error for {endpoint} (attempt {attempt + 1})")
            except aiohttp.ClientResponseError as e:
                logger.error(f"HTTP error for {endpoint}: {e}")
                raise # 重新抛出，由外层 finally 释放
            except Exception as e:
                logger.error(f"Unexpected error for {endpoint}: {e}")
                if attempt == self.max_attempts - 1:
                    raise # 最后一次失败，由外层 finally 释放

            if attempt < self.max_attempts - 1:
                await asyncio.sleep(self.retry_delay * (2**attempt))

        raise RuntimeError(
            f"Failed to complete async request to {endpoint} after {self.max_attempts} attempts"
        )
    finally:
        # 总是平衡 _select_worker() 的递增，即使失败或重试耗尽
        self._release_worker(worker_url)
```

评论区精华

gemini-code-assist[bot] 指出，如果 await request.body() 在进入 try 块之前抛出异常（如客户端断开），_select_worker() 已经递增了计数，但 finally 块不会执行，仍然存在泄漏。建议将 try 块扩展到包含 await request.body()。该评论尚未被解决，属于未解决疑虑。

- await request.body() 异常路径的泄漏 (correctness): 未解决，当前 PR 未处理此边界情况。

风险与影响

- 风险：主变更逻辑正确，但存在边界漏洞：await request.body() 若在 try 块外抛出异常，仍可能泄漏计数。此外，由于没有配套的单元测试，回归风险无法被自动化覆盖。
- 影响：影响范围限于 verl/experimental/reward_loop/router/naive_router.py 的 _make_async_request 方法，属于 reward 路由模块的内部实现。修正后，负载均衡行为在请求失败场景下更加准确，多 worker 场景下的假性过载问题得到缓解。
- 风险标记：缺少测试覆盖，边界漏洞未修复

关联脉络

- 暂无明显关联 PR