

PR #7851 完整报告

verl-project/verl

[worker] fix: sync ranks before resume kv_cache to avoid OOM on non-leader ranks

合并时间: 2026-09-14 22:55

原文链接: <http://prhub.com.cn/verl-project/verl/pull/7851>

执行摘要

- 一句话: 权重同步后加跨 rank barrier, 避免非 leader 恢复 KV cache 时 OOM
- 推荐动作: 值得精读 (约 5 行, 10 分钟以内), 特别是正在维护权重同步、rollout 显存管理或遇到过“非 0 号 rank OOM”的工程师: 它是理解“leader 抢先分配资源 → 其它 rank OOM”这类跨 rank 竞态的典型样本, 也是 `asyncio.to_thread` 包装阻塞式集合通信的现成写法。阅读时重点确认三点: 一是 barrier 的位置是否确实覆盖了所有会持有显存的重 rank (本例放在 `offload` 与 `aggressive_empty_cache` 之后、`resume` 之前, 逻辑正确); 二是是否应加 `torch.distributed.is_initialized()` 守卫或改用带超时的 barrier, 以降低挂起 / 异常风险; 三是其它未覆盖的同步路径 (`disaggregated`、TRT-LLM 之外的专用后端) 是否需要同样处理。此外建议补一个最小的多 rank / 异步单测或在 PR 描述中说明手工验证方法, 弥补当前评审与测试证据不足的问题。

功能与动机

PR body 明确指出: leader rank 可能先于其它 rank 完成本地权重更新与清理, 随后就触发 KV cache 分配, 而此时其它 rank 仍持有训练参数或临时缓冲区, 从而出现非 leader rank OOM; local device synchronization (即本路径已有的 `aggressive_empty_cache(force_sync=True)`) 无法阻止这种跨 rank 竞态。作者同时在关联讨论中说明, 这与 #5208 为 TensorRT-LLM 路径做的同步修复同类, 本 PR 是把同样的保护补到共享 worker 的 naive 权重同步路径上, 从而同时覆盖 vLLM 与 SGLang 后端。

实现拆解

1) 导入改造: 在 `verl/workers/engine_workers.py` 头部新增 `import asyncio`, 为后续把阻塞式集合通信移出事件循环做准备; 该文件此前并未导入 `asyncio`。2) 同步点插入: 在 `update_weights` (async 方法, `@register(dispatch_mode=Dispatch.ONE_TO_ALL, blocking=False)`) 的末尾, 把 `if self.config.rollout.free_cache_engine:` 分支改为先 `await asyncio.to_thread(torch.distributed.barrier)`, 再 `await self.rollout.resume(tags=["kv_cache"])`。位置选择在 `actor.engine.to("cpu", model=True, optimizer=False, grad=False)` 与 `aggressive_empty_cache(force_sync=True)` 之后, 保证 barrier 返回时各 rank 的卸载与显存清理都已完成。3) 事件循环友好: barrier 是同步阻塞的集合通信, 若直接在协程里调用会卡住 worker 的 `asyncio` 事件循环, 因此用 `asyncio.to_thread` 放进线程池执行, `await` 期间事件循环仍可处理其它任务。4) 作用域收敛: barrier 只在 `free_cache_engine=True` 的分支内生效, `free_cache_engine=False` 与

disaggregated（分离式）权重同步路径完全不变，也不新增任何 API 或配置项。5) 测试与配套：本次未新增单测、e2e 或 CI 工作流改动；PR body 只写“Test all passed”，没有说明是本地手工多卡验证还是仓库 CI，属于该 PR 最明显的配套缺口。

变更入口与影响面：调用链是 trainer 侧 ONE_TO_ALL 派发 → 各训练 rank 进入 `update_weights` → 权重同步 / 卸载 / 清理 → （新增）跨 rank barrier → leader 触发 `rollout.resume`。新增的 barrier 位于 leader 唤醒 rollout worker 之前，因此它约束的是“训练 rank 之间”的时序，不改变 rollout worker 的唤醒协议本身。

关键文件：

- `verl/workers/engine_workers.py`（模块 权重同步；类别 source；类型 core-logic；符号 `update_weights`）：唯一改动文件，也是问题所在处：共享 worker 的权重同步 / 卸载 / 恢复 KV cache 主路径都集中在这里的 `update_weights` 方法，本 PR 的 5 行改动（新增 `asyncio` 导入 + barrier）全部落在此文件。

关键符号：`update_weights`

关键源码片段

`verl/workers/engine_workers.py`

唯一改动文件，也是问题所在处：共享 worker 的权重同步 / 卸载 / 恢复 KV cache 主路径都集中在这里的 `update_weights` 方法，本 PR 的 5 行改动（新增 `asyncio` 导入 + barrier）全部落在此文件。

```
# verl/workers/engine_workers.py: 模块头部新增标准库导入
import asyncio # 本 PR 新增: barrier 是阻塞式集合通信, 需要 asyncio.to_thread
                把它挪出事件循环
import functools
import logging

# ... (其它既有导入保持不变)

async def update_weights(self, global_steps: int = None, mode: str = "auto"):
    # ... (省略前序: summon 参数、LoRA base sync 等步骤)

    per_tensor_param, peft_config = self.actor.engine.get_per_tensor_param(
        layered_summon=self.layered_summon, base_sync_done=True
    )

    do_lora_base_sync = False
    if not self.peft_merge and peft_config is not None:
        self.rollout.sleep_level = 1
        do_lora_base_sync = not self.base_sync_done

    # 3. sync weights: For SGLang, we need base first (when needed), then adapter/merged
    if do_lora_base_sync:
        per_tensor_param_base, _base_peft_config = self.actor.engine.get_per_tensor_param(
            layered_summon=self.layered_summon, base_sync_done=False
```

```

    )
    await self.rollout.update_weights(
        per_tensor_param_base, peft_config=_base_peft_config, base_sync_done=False,
        global_steps=global_steps
    )

    await self.rollout.update_weights(
        per_tensor_param, peft_config=peft_config, base_sync_done=True, global_steps=global_steps
    )

    log_gpu_memory_usage("After update_weights", logger=logger)

    # 3. offload model to cpu
    # 先把训练侧参数搬到 CPU 并清空显存缓存，让本 rank 的显存占用降到最低
    if self.actor.engine.is_param_offload_enabled:
        self.actor.engine.to("cpu", model=True, optimizer=False, grad=False)
    aggressive_empty_cache(force_sync=True)

    # 4. resume kv_cache
    if self.config.rollout.free_cache_engine:
        # 本 PR 新增：所有 trainer rank 都经 ONE_TO_ALL 进入这条 naive 同步路径，
        # 因此在 leader 唤醒 rollout worker 之前，先跨 rank 对齐一次，
        # 确保各 rank 的参数卸载与显存清理都已结束；
        # 仅靠上面的本地设备同步（aggressive_empty_cache）无法约束其它 rank，
        # 否则 leader 提前 resume 会在仍持有参数 / 临时缓冲区的非 leader rank 上触发 OOM。
        # 用 asyncio.to_thread 包装，避免阻塞 worker 的 asyncio 事件循环。
        await asyncio.to_thread(torch.distributed.barrier)
        await self.rollout.resume(tags=["kv_cache"])
    log_gpu_memory_usage("After resume kv_cache", logger=logger)

    self.base_sync_done = True
    set_expandable_segments(True)

```

评论区精华

本 PR 的 review 过程非常轻量：只有 1 条 issue 评论（作者 HT-Yuan 主动 @eric-haibin-lin、@ArronHZG、@wuxibin89 请求评审并说明与 #5208 的关系），以及 wuxibin89 的一条无正文 APPROVED，没有任何 inline review 评论或后续修改。因此值得记录的“讨论”主要是作者在描述中自述的设计取舍：为什么仅做本地设备同步不够（local device synchronization does not prevent this cross-rank race）、为什么必须放在 offload/cleanup 之后而 resume 之前、以及为什么用 asyncio.to_thread 而不是直接调用 barrier。这些关键论证没有被 reviewer 进一步质询，也没有形成关于“barrier 是否可能挂起”“是否该改用 dispatch 语义保证顺序”的公开讨论，属于评审深度上的空白。

- 为什么必须做跨 rank 同步，本地设备同步为何不够 (correctness): wuxibin89 直接 APPROVED，未提出反对意见，方案被接受并合并。

- barrier 的位置与 `asyncio.to_thread` 的使用 (design): 设计被默认接受, 但缺少 review 层面的对抗性验证, 属于可继续加固的点。
- 测试覆盖缺失 (testing): 无讨论结论, 测试缺口保留在合入版本中。

风险与影响

- 风险: 1) 集合通信挂起 / 失败风险 (最高): `torch.distributed.barrier()` 依赖默认 process group 已初始化。该文件通过 `initialize_global_process_group_ray` 建组, 若某些部署形态 (单进程调试、PG 未初始化或使用非默认 PG) 走到这一行, 会直接抛 `RuntimeError`; 同时代码没有 `torch.distributed.is_initialized()` 守卫。反之, 只要某个 rank 的 CPU offload 或显存清理异常缓慢, 其余 rank 会全部阻塞在 barrier 上, 训练 step 的尾延迟被最慢 rank 决定, 若默认 PG 配置了超时还可能直接超时失败。2) 分支不一致导致死锁的可能性: barrier 只包在 `self.config.rollout.free_cache_engine` 为真、且走到 naive 同步路径的 rank 内。正常情况下这是全局一致的配置 (ONE_TO_ALL 保证所有训练 rank 进入同一方法), 但若未来出现混合配置或某些 rank 走 disaggregated 路径, 就会出现部分 rank 进 barrier、部分不进的不对称, 进而挂死。3) 缺少测试覆盖: 本 PR 未新增任何单测 / e2e/CI 用例, 而它修复的恰恰是竞态型 OOM——这类问题只有在真实多卡、大模型、`free_cache_engine=True` 的场景下才可能复现, “Test all passed” 无法证明竞态被消除、也无法防止后续回归。4) 性能与延迟: 每次权重同步会多一次全局 barrier, 量级为同步开销, 对吞吐影响可忽略; 但 `asyncio.to_thread` 占用默认线程池线程, 若同进程内已有大量阻塞任务, 理论上存在排队延迟 (低风险)。5) 兼容性: 无 CLI/ 配置 / API 变更, 默认行为只在 `free_cache_engine=True` 的 naive 路径上多一次同步, 向后兼容; 但这也意味着凡是走 disaggregated 路径的用户仍暴露在同类竞态下, 需要确认那条路径是否已有等价保护 (TRT-LLM 侧 #5208 已处理, 其它后端未在本 PR 范围内)。
- 影响: 用户侧: 使用 vLLM 或 SGLang 做 colocated RL 训练、且开启 `free_cache_engine` (默认权重同步后释放 KV cache 以省显存) 的用户, 将不再出现“非 leader rank 在 resume KV cache 阶段 OOM”的随机失败, 训练稳定性提升; 代价仅是每个权重同步周期增加一次全局同步, 对端到端吞吐影响很小。系统侧: 改动落在 `verl/workers/engine_workers.py` 的权重同步热路径, 属于核心控制流, 但作用域被严格限制在 `free_cache_engine` 分支内, 不改变 rollouts 唤醒协议、不改变数据流与显存布局, 回归面窄。团队侧: 这属于 verl 近期一组“异步 / 竞态类修复”的延续 (V1 异步 warmup、Mooncake 完成槽位、profiler 窗口重叠等), 为后续在其它权重同步路径 (disaggregated、各后端专用路径) 补齐跨 rank 同步提供了可直接复用的范式: 先完成本地 offload/cleanup, 再跨 rank 对齐, 最后唤醒对端资源分配。
- 风险标记: 核心路径变更, 缺少测试覆盖, 分布式 barrier 挂起 / 超时风险, 无 process group 初始化守卫, 仅覆盖 naive 同步路径

关联脉络

- PR #5208 (上下文未提供标题) TensorRT-LLM 路径的同类同步修复: PR body 直接引用: 该 PR 为 TensorRT-LLM 的权重同步路径解决了同类的跨 rank 同步问题, 本 PR 是把同样的保护补到 vLLM/SGLang 共用的 naive 同步路径上。

- PR #7762 [trainer, ckpt] fix: skip V1 async warmup after inflight reissue: 同属 Trainer V1 异步 / 竞态类修复：都是处理“某个 rank/ 任务提前推进导致其它方状态错位”的时序问题，反映仓库近期在系统性收敛 async 路径的稳定性。
- PR #7764 [ckpt] fix: drain Mooncake completion slots across weight versions: 同样是权重 / 检查点跨版本传递中的资源释放时序修复，与本 PR 的“先对齐所有权与释放状态、再触发下一步分配”思路一致，可对照阅读。
- PR #7722 [perf, tool] fix: Avoid overlapping rollout and training profilers in Trainer V1: 同样涉及 rollout 与训练侧资源窗口重叠问题（profiler 叠加导致数据丢失），与本 PR 关注 rollout/training 交错阶段的资源冲突属于同一类工程问题。