PR #1967 完整报告

radixark/miles

[Bug Fix] Always release rollout-engine broadcast lock on failure

合并时间：2026-08-09 14:42

原文链接：http://prhub.com.cn/radixark/miles/pull/1967

执行摘要

- 一句话：广播异常不再泄漏 rollout-engine 锁，训练不会静默挂起
- 推荐动作：值得精读。这是分布式锁生命周期 bug 的教科书式修复：try/finally 的最小改动解决级联挂起，测试设计（用进程内 _LockState 模拟 Ray actor、哨兵注入失败、失败后重试）非常有参考价值。对于分布式训练框架的开发者，可重点关注"轮询型锁 + 异常路径"的组合为何必须用 finally 兜底，以及如何在 mock 层完整模拟远端 actor 语义。

功能与动机

PR body 明确指出：_update_weight_implementation 在开始分布式权重广播前获取共享 rollout-engine 锁，之前广播设置或等待 rollout-engine 完成期间的异常会跳过释放；因为 Lock.acquire() 被轮询直到成功，泄漏的锁会使下一次权重同步永远等待，并隐藏原始失败——这是典型的级联挂起故障。

实现拆解

变更入口为 miles/backends/megatron_utils/update_weight/update_weight_from_distributed/broadcast.py 的 _update_weight_implementation 方法，按以下步骤完成：

1. 根因定位：旧实现中 acquire、update_weights_from_distributed、ray.get(refs)、clear()、release 线性执行，任一步抛异常都会跳过 release；而 acquire 是无限轮询，泄漏的锁无法被后续调用感知，表现为训练静默挂起。
2. 核心修复：将广播提交（update_weights_from_distributed）、完成等待（ray.get(refs)）与张量清理（converted_named_tensors.clear()）整体移入 try 块，finally 中无条件执行 ray.get(self.rollout_engine_lock.release.remote())，使 setup 阶段异常（如 NCCL 初始化失败）与引擎侧异常（ray.get 抛错）都先释放锁再向上传播。
3. 保持成功路径语义：clear() 仍在广播成功后执行，失败时转换后的张量被保留（测试断言 len(tensors) == 1），便于调用方排查；pbar.update(1) 仍在 finally 之外，只有完整同步成功后才推进进度条。
4. 测试配套：新增 tests/fast/backends/megatron_utils/test_update_weight_from_distributed_lock.py（175 行），用 _LockState 进程内替身模拟 Ray Lock actor（acquire 返回 False 表示被占用、release 断言锁已持有），用 _passthrough_ray_get 模拟 ray.get 遇哨兵抛错。5个用例覆盖：成功路径（验证锁释放、张量清空、pbar推进、全部参数转发）、锁竞争轮询（验证 0.1s 重试）、广播 setup 失败、引擎侧失败、失败后重试成功（

release_calls == 2）。

5. 无配置与部署改动：变更只涉及源码与测试两个文件，无命令行参数、schema 或 Docker 变化。

关键文件：

- miles/backends/megatron_utils/update_weight/update_weight_from_distributed/broadcast.py（模块 权重同步；类别 source；类型 core-logic；符号 _update_weight_implementation）：核心修复文件：在 _update_weight_implementation 中用 try/finally 包裹广播提交与完成等待，保证 rollout-engine 锁在成功与两类失败路径都释放，是消除训练挂起的关键变更。
- tests/fast/backends/megatron_utils/test_update_weight_from_distributed_lock.py（模块 权重同步；类别 test；类型 test-coverage；符号 _LockState, _make_updater, _passthrough_ray_get, _named_tensors）：新增 175 行 mock 测试，完整锁定锁生命周期：覆盖成功路径、锁竞争轮询、广播 setup 失败、引擎侧失败与失败后重试，并验证传给 update_weights_from_distributed 的全部参数。

关键符号：_update_weight_implementation

关键源码片段

miles/backends/megatron_utils/update_weight/update_weight_from_distributed/broadcast.py

核心修复文件：在 _update_weight_implementation 中用 try/finally 包裹广播提交与完成等待，保证 rollout-engine 锁在成功与两类失败路径都释放，是消除训练挂起的关键变更。

```
def _update_weight_implementation(
    self, converted_named_tensors: list[tuple[str, torch.Tensor]], pbar: tqdm | None = None
) -> None:
    """串行化 NCCL 广播，并保证任何路径都释放 rollout 锁。"""
    # Lock.acquire() 被轮询直到返回 True（每 0.1 秒重试一次），
    # 因此一旦泄漏，下一次权重同步会永久阻塞在这里。
    while not ray.get(self.rollout_engine_lock.acquire.remote()):
        time.sleep(0.1)
    try:
        # 向各 rollout engine 提交广播任务并获得 ObjectRef 列表
        refs = update_weights_from_distributed(
            self._group_name,
            self._model_update_groups,
            self.weight_version,
            self.rollout_engines,
            converted_named_tensors,
            selector=self._weight_update_selector,
        )
        # 阻塞等待广播完成；引擎侧异常会在此处抛出
        ray.get(refs)
        # 广播成功后才清理转换后的张量；失败时保留张量以便排查
        converted_named_tensors.clear()
```

```
    finally:
        # 无论 setup 还是引擎侧失败都必须释放锁，
        # 否则下一次权重同步会因 acquire 轮询而永久挂起，
        # 并掩盖这里真正的失败原因。
        ray.get(self.rollout_engine_lock.release.remote())
    # 进度条只在同步完全成功后推进
    if pbar:
        pbar.update(1)
```

tests/fast/backends/megatron_utils/test_update_weight_from_distributed_lock.py

新增 175 行 mock 测试，完整锁定锁生命周期：覆盖成功路径、锁竞争轮询、广播 setup 失败、引擎侧失败与失败后重试，并验证传给 update_weights_from_distributed 的全部参数。

```
class _LockState:
    """进程内替身，模拟 miles.ray.utils.Lock actor 的 acquire/release 语义。"""

    def __init__(self):
        self.locked = False
        self.release_calls = 0

    def acquire(self):
        # 与真实 actor 一致：锁被占用时返回 False，由调用方轮询重试
        if self.locked:
            return False
        self.locked = True
        return True

    def release(self):
        # 真实 actor 同样把释放未持有的锁视为 bug
        assert self.locked, "Lock is not acquired, cannot release."
        self.release_calls += 1
        self.locked = False


def _passthrough_ray_get(mock_ray, fail_on=None):
    """让 mock 的 ray.get 原样返回参数，或在遇到指定哨兵对象时抛异常。"""

    def _get(ref):
        if fail_on is not None and ref is fail_on:
            raise RuntimeError("engine died during broadcast")
        return ref

    mock_ray.get.side_effect = _get
```

评论区精华

两位 reviewer（Shi-Dong、maocheng23）均 APPROVED，Shi-Dong 简短评价 “Nice fix. LGTM.”。无实质代码讨论；PR 内两条 comment 来自 gemini-code-assist[bot] 的停服通知，

无技术内容。

- 暂无高价值评论线程

风险与影响

- 风险：风险点集中在 finally 语义与失败路径行为：

1. finally 内 release 失败的异常覆盖：若 ray.get(self.rollout_engine_lock.release.remote()) 自身抛错（如 Ray 集群故障），会覆盖原始广播异常。但此时锁状态已不可知，向上暴露 release 错误反而更利于定位。
2. 失败路径张量保留：广播失败时 converted_named_tensors 不再清空，调用方需能容忍张量残留。这是有意设计（测试断言 len(tensors) == 1），便于排查失败现场，但若调用方误以为张量已被消费，可能重复处理。
3. 兼容性：成功路径的顺序完全未变（clear 仍发生在广播完成后、pbar 仍在同步完成后），对正常训练无行为变化；H200 集群完整训练配方多次权重同步实测通过，回归风险低。 - 影响：影响所有走 update_weight_from_distributed 的权重同步路径（Megatron 后端、fully-async rollout），消除"一次广播失败 → 后续所有权重同步永久挂起"的级联故障模式。在 fully-async 训练中权重广播是高频操作，该修复显著提升训练任务的可靠性与可排查性，且不改变成功路径行为，对已有任务透明。 - 风险标记：核心训练路径变更，finally 异常覆盖风险，失败路径张量保留

关联脉络

- PR #2244 pass the engine weight version from the trainer instead of polling the router: 同一模块演进：重构 update_weight_from_distributed 调用链（mixin.py / p2p.py / actor.py），本 PR 为其锁生命周期补强。
- PR #2223 Remove --disable-weights-backuper; default eligible colocate launchers to rematerialize: 同属权重更新链路，改动 update_weight/common.py，与本次广播路径修复共同提升权重同步可靠性。
- PR #1572 [optim]--rematerialize-param-from-master-weight: save the bf16 weight backup in colocate: 权重同步链路的早期优化，引入从主权重重建 bf16 备份的机制，与本次修复同属该链路的健壮性演进。