

PR #3129 完整报告

radixark/miles

fix: guard connection allocations during offloaded weight sync

合并时间: 2026-09-15 08:42

原文链接: <http://prhub.com.cn/radixark/miles/pull/3129>

执行摘要

- 一句话: 补上 offload 权重同步连接阶段的内存分配守卫
- 推荐动作: 值得精读。改动虽小, 但准确定位了 offload 显存管理中一个隐蔽问题——连接建立过程 (而非仅数据传输) 同样会分配 CUDA 张量, 且把状态标记与 barrier 一并纳入守卫的决策体现了对“分配窗口”的完整思考。建议与 #3128 一起阅读, 理解 offload 下 actor 权重同步的完整安全边界; 同时留意 guapisolo 对守卫放置位置的提醒, 后续重构 `disable()` 调用时应保持“覆盖所有可能分配 CUDA 张量的阶段”这一原则。

功能与动机

PR body 明确说明: "Creating or reconnecting a weight-sync connection can allocate CUDA tensors, for example during NCCL object collectives. With trainer offloading enabled, those allocations can otherwise reuse memory-saver blocks whose GPU storage has already been unmapped, causing invalid memory accesses during connection setup. The existing `torch_memory_saver.disable()` guard protects the subsequent weight transfer, but not the earlier connection setup. Protecting only the transfer is therefore insufficient." 即连接建立阶段同样会分配 CUDA 张量, 只保护传输阶段不够, 需要把连接建立也纳入守卫。

实现拆解

1. 变更入口: `miles/backends/megatron_utils/actor.py` 中的 `WeightUpdateActor.update_weights` 方法, 具体位于 `needs_reconnect` 分支。
2. 核心逻辑改造: 将原本裸执行的 `connect_rollout_engines`、`conn_status.mark_reconnected` 和 `dist.barrier(group=get_gloo_group())` 整体移入 `with torch_memory_saver.disable() if self.args.offload_train else nullcontext():` 守卫。这样连接建立期间的临时 CUDA 分配不会复用已被卸载的 memory-saver block; 同时把状态标记和 barrier 也留在守卫内, 避免它们与后续传输之间出现新的分配窗口。
3. 非 offload 路径不变: `offload_train=False` 时使用 `nullcontext()`, 行为与原先完全一致。
4. 测试配套: `tests/fast/backends/megatron_utils/test_shared_ppo_lifecycle.py` 新增参数化测试 `test_connection_uses_safe_allocations_when_offloading`, 用 `saver.disable.side_effect` 模拟安全区域, 断言连接回调在守卫内执行、异常后守卫正确退出、`disable` 调用次数符合预期 (offload 且连接成功时连接 + 传输共 2 次, 失败时仅 1 次), 且失败时不进入传输阶段。

5. 验证：PR body 提供 H200 A/B 对照——main 在首次 update_weights 即崩溃于 connect_rollout_engines → dist.all_gather_object；本修复后连接成功，配合 #3128 后整条 offload 权重同步链路通过，引擎报告所有同步张量相等。CI 上 tests/e2e/megatron/test_qwen3_4B_offload_disaggregated.py rerun 通过。

关键文件：

- miles/backends/megatron_utils/actor.py（模块 权重同步；类别 source；类型 core-logic；符号 update_weights）：核心修复文件：在 update_weights 的 needs_reconnect 分支中加入 torch_memory_saver.disable() 守卫，覆盖连接建立、状态标记与同步 barrier，直接修复 offload 下非法内存访问。
- tests/fast/backends/megatron_utils/test_shared_ppo_lifecycle.py（模块 生命周期；类别 test；类型 test-coverage；符号 test_connection_uses_safe_allocations_when_offloading, safe_region, check_connection）：新增参数化测试，验证 offload 开关与连接成败组合下守卫的进入、退出与异常恢复，防止该修复回归。

关键符号：update_weights, test_connection_uses_safe_allocations_when_offloading

关键源码片段

miles/backends/megatron_utils/actor.py

核心修复文件：在 update_weights 的 needs_reconnect 分支中加入 torch_memory_saver.disable() 守卫，覆盖连接建立、状态标记与同步 barrier，直接修复 offload 下非法内存访问。

```
# 连接建立同样会分配 CUDA 张量（例如 NCCL object collective）。
# 如果训练器已 offload 且内存块被 unmap，复用这些块会导致非法访问，
# 因此把“连接 + 状态标记 + 同步 barrier”整体放进分配守卫。
needs_reconnect = self.weight_updater.conn_status.needs_reconnect(snapshot_cell_id_to_hashes)
if needs_reconnect:
    # 只有 offload 训练才需要守卫；普通路径保持原样（nullcontext 无副作用）
    with torch_memory_saver.disable() if self.args.offload_train else nullcontext():
        self.weight_updater.connect_rollout_engines(
            rollout_engines,
            engine_gpu_counts=engine_gpu_counts,
            engine_gpu_offsets=engine_gpu_offsets,
        )
    # 状态标记与 barrier 也留在守卫内，避免二者之间出现新的分配窗口
    self.weight_updater.conn_status.mark_reconnected(snapshot_cell_id_to_hashes)
    dist.barrier(group=get_gloo_group())

# 后续真正的权重传输阶段早已有守卫，这里保持原样
with torch_memory_saver.disable() if self.args.offload_train else nullcontext():
    print_memory("before update_weights")
    self.weight_updater.update_weights()
    print_memory("after update_weights")
```

tests/fast/backends/megatron_utils/test_shared_ppo_lifecycle.py

新增参数化测试，验证 offload 开关与连接成败组合下守卫的进入、退出与异常恢复，防止该修复回归。

```
@pytest.mark.parametrize("offload_train", [False, True])
@pytest.mark.parametrize("connect_fails", [False, True])
def test_connection_uses_safe_allocations_when_offloading(
    actor_module: Any, monkeypatch: pytest.MonkeyPatch, offload_train: bool, connect_fails:
    bool
) -> None:
    worker = _weight_update_worker(actor_module, monkeypatch)
    worker.args.offload_train = offload_train
    worker._asleep = offload_train # offload 训练时 actor 处于睡眠态，matches 真实场景
    inside_safe_region = False

    # 模拟 torch_memory_saver.disable() 的守卫行为：进入置位、退出复位
    @contextmanager
    def safe_region() -> Iterator[None]:
        nonlocal inside_safe_region
        assert not inside_safe_region
        inside_safe_region = True
        try:
            yield
        finally:
            inside_safe_region = False

    def check_connection(*_args: Any, **_kwargs: Any) -> None:
        # 连接回调必须在安全区内（offload 时）或区外（非 offload 时）执行
        assert inside_safe_region == offload_train
        if connect_fails:
            raise RuntimeError("connection failed")

    saver = Mock()
    saver.disable.side_effect = safe_region
    monkeypatch.setattr(actor_module, "torch_memory_saver", saver)
    monkeypatch.setattr(actor_module, "reload_process_groups", Mock())
    monkeypatch.setattr(actor_module, "destroy_process_groups", Mock())
    worker.weight_updater.connect_rollout_engines = check_connection
    engines = _updatable_engines([], {"cell-0": "hash-a"}, gpu_count=8)

    if connect_fails:
        with pytest.raises(RuntimeError, match="connection failed"):
            worker.update_weights(engines)
    else:
        worker.update_weights(engines)

    # 异常路径也要求守卫被正确退出
    assert not inside_safe_region
    # offload 且连接成功时进入两次守卫（连接 + 传输）；失败时仅一次；非 offload 不进入
    assert saver.disable.call_count == ((1 if connect_fails else 2) if offload_train else 0)
```

```
assert worker.weight_updater.update_weights_calls == int(not connect_fails)
```

评论区精华

- guapisolo 在批准时提出: "Make sense. good fix. But the code placement of the `disable()` func need code owner double check." 即对 `disable()` 守卫的放置位置是否完全正确持保留意见, 希望 code owner 复核。
- claude[bot] 在三次 review 中均报告 "Code review found no issues"。
- Zhichenzzz 直接批准, 无额外评论。
- yueming-yuan 发起了两次 `/rerun-test tests/e2e/megatron/test_qwen3_4B_offload_disaggregated.py`, 第一次 workflow 被取消, 第二次在 `stage-c-8-gpu-h200` 通过 (12m58s), 验证了 offload 权重同步场景。
- `disable()` 守卫的放置位置 (design): 未引发进一步代码改动; PR 最终被批准合并, 守卫范围确定为“连接建立 + 状态标记 + 同步 barrier”, 同时保留原有传输阶段守卫。
- e2e 验证与 CI rerun (testing): 第二次 rerun 通过, 确认 offload 路径在真实 GPU 环境下的行为正确, CI 纳入该场景作为回归保障。

风险与影响

- 风险:
 - 行为变更面窄: 仅在 `offload_train=True` 且 `needs_reconnect=True` 分支生效, 非 offload 路径通过 `nullcontext()` 保持原行为, 回归风险小。
 - 把 `dist.barrier` 纳入守卫: 若 barrier 因进程组异常长时间阻塞, 会延长 `torch_memory_saver.disable()` 的持续时间, 导致显存占用上升; 正常路径下 barrier 是同组快速同步, 风险较低。
 - 异常路径依赖 `contextmanager` 语义恢复分配上下文: 测试用 mock 验证了该恢复逻辑, 但真实 NCCL 失败下的 memory-saver 行为仍缺少 e2e 直接覆盖 (依赖 #3128 引入的 disaggregated e2e 间接覆盖)。
 - 对 `torch_memory_saver` 的行为有隐式依赖: 后续该库的升级 (如 #3192 修复 `free-while-paused`) 可能影响此路径, 需要回归 offload 权重同步测试。
- 影响:
 - 用户影响: 修复启用 `--offload-train` 的 PPO 训练在权重同步首次连接 / 重连时的崩溃, 尤其是“训练器 offload 到 CPU、actor 睡眠”与 rollout 分离部署的组合; 此前该组合在第一次 `update_weights` 即挂死。
 - 系统影响: 无 API/ 配置变更, 非 offload 用户零感知; offload 用户在连接建立阶段会短暂禁用 memory-saver, 显存占用有所回升, 但换取正确性。
 - 团队影响: 与 #3128 构成 offload 权重同步可靠性修复系列, CI 新增并验证了 `test_qwen3_4B_offload_disaggregated.py` 这一覆盖场景。
 - 风险标记: offload 专用分支变更, barrier 纳入守卫, 依赖 `torch_memory_saver` 语义, e2e 依赖 #3128

关联脉络

- PR #3128 fix: use host weights when syncing an offloaded actor: 同一 offload 权重同步链路的配套修复: 本 PR 修复连接阶段的分配问题, #3128 修复传输阶段权重来源问题; PR body 明确要求先合并本 PR 再合并 #3128, 且 #3128 补充了覆盖此配置的 disaggregated e2e 测试。
- PR #3192 fix(docker): bump torch_memory_saver to pick up the free-while-paused fix: 同为 torch_memory_saver 相关修复: 升级 CUDA 镜像中的库版本, 修复暂停态释放崩溃; 与本 PR 构成 memory-saver 使用可靠性的修复系列, 后续升级该库时需要回归本路径。