PR #6919 完整报告

verl-project/verl

[vllm, rollout] fix: handle non-contiguous weights in bucketed transfer

合并时间：2026-07-06 11:07

原文链接：http://prhub.com.cn/verl-project/verl/pull/6919

执行摘要

- 一句话：修复非连续张量在分桶权重传输中的崩溃
- 推荐动作：值得所有使用 Megatron-Bridge 或自定义权重导出器的用户更新。代码变更简洁，设计决策清晰（利用 PyTorch 原生能力而非手动压缩），可作为小范围 bugfix 的范例。

功能与动机

PR body 指出 weight.view(-1).view(torch.uint8) 在张量非连续时会崩溃。Mecoli1219 在评论中确认该问题来自 megatron-bridge v0.5.0 的 gate tensors 返回 strided 视图。

实现拆解

1. 核心修复：在 verl/workers/rollout/vllm_rollout/bucketed_weight_transfer.py 中，将原来的 self.buffer[offset : offset + weight.nbytes].copy_(weight.view(-1).view(torch.uint8), non_blocking=True) 替换为 self.buffer[offset : offset + weight.nbytes].view(dtype=weight.dtype).view(weight.shape).copy_(weight, non_blocking=True)。此举直接利用 PyTorch copy_ 对非连续源张量的原生支持，避免了显式 contiguous() 调用的临时内存分配和额外拷贝。
2. 新增单元测试：在 tests/utils/test_bucketed_weight_transfer.py 中新增 test_sender_accepts_strided_tensor 测试。该测试构造一个非连续张量（通过切片获得），验证 async_send_weights 在不抛出异常的情况下正确将数据拷贝到缓冲区，并通过恢复比对确保内容一致。测试中使用 _FakeSocket 和 _FakeTorchDevice 模拟 ZMQ 通信和 GPU 同步，避免依赖真实硬件。
3. 第二提交：根据 Gemini Code Assist 的 review 建议，将原本的 _weight_to_bytes(weight) 辅助函数（含 contiguous()）优化为当前更高效的视图 +copy_ 方案。

关键文件：

- verl/workers/rollout/vllm_rollout/bucketed_weight_transfer.py（模块 分桶传输；类别 source；类型 core-logic；符号 async_send_weights）：核心源码：一行关键修复，将缓冲拷贝从 weight.view(-1).view(torch.uint8) 改为 buffer 视图 +copy_，兼容非连续张量。
- tests/utils/test_bucketed_weight_transfer.py（模块 测试；类别 test；类型 test-coverage；符号 _FakeSocket, _FakeTorchDevice, test_sender_accepts_strided_tensor）：新增单元测试

test_sender_accepts_strided_tensor，使用 FakeSocket 和 FakeTorchDevice 模拟环境，验证非连续张量发送的正确性。

关键符号：async_send_weights, test_sender_accepts_strided_tensor

关键源码片段

verl/workers/rollout/vllm_rollout/bucketed_weight_transfer.py

核心源码：一行关键修复，将缓冲拷贝从 weight.view(-1).view(torch.uint8) 改为 buffer 视图 +copy_，兼容非连续张量。

```
# verl/workers/rollout/vllm_rollout/bucketed_weight_transfer.py (line ~151)
# 原代码：self.buffer[offset:offset+weight.nbytes].copy_(weight.view(-1).view(torch.uint8), non_
blocking=True)
# 问题：weight 非连续时 .view(-1) 抛出 RuntimeError
#
# 新代码：将目标 buffer 切片先 .view() 为与 weight 相同的 dtype 和 shape，
# 然后直接 copy_。PyTorch 的 copy_ 原生支持非连续源张量，无需手动 .contiguous()。
self.buffer[offset : offset + weight.nbytes].view(dtype=weight.dtype).view(weight.shape).copy_(
    weight, non_blocking=True
)
```

tests/utils/test_bucketed_weight_transfer.py

新增单元测试 test_sender_accepts_strided_tensor，使用 FakeSocket 和 FakeTorchDevice 模拟环境，验证非连续张量发送的正确性。

```
# tests/utils/test_bucketed_weight_transfer.py（新增）
class _FakeSocket:
    """模拟 ZMQ socket，记录发送的消息"""
    def __init__(self):
        self.messages = []

    def send_pyobj(self, message):
        self.messages.append(message)

    def recv(self):
        return b""

class _FakeTorchDevice:
    """模拟 torch device，避免真实 CUDA 同步"""
    def synchronize(self):
        pass

def test_sender_accepts_strided_tensor(monkeypatch):
    from verl.workers.rollout.vllm_rollout import bucketed_weight_transfer

    # 构造一个非连续张量：(2,3,4) -> 取第 0 列 -> shape (2,4) 但 stride 不是 (4,1)
    base = torch.arange(2 * 3 * 4, dtype=torch.float32).reshape(2, 3, 4)
    weight = base[:, 0, :] # strided view
```

```
    buffer = torch.empty(weight.nbytes, dtype=torch.uint8)
    socket = _FakeSocket()

    # 创建 sender 并注入 mock
    sender = bucketed_weight_transfer.BucketedWeightSender(
        zmq_handle="ipc:///tmp/test-bwt-unused.sock",
        bucket_size_mb=1,
        use_shm=True,
    )
    monkeypatch.setattr(sender, "_init_socket", lambda: setattr(sender, "socket", socket))
    monkeypatch.setattr(sender, "_init_buffer", lambda: setattr(sender, "buffer", buffer))
    monkeypatch.setattr(sender, "_cleanup", lambda: None)
    monkeypatch.setattr(bucketed_weight_transfer, "get_torch_device", lambda: _
    FakeTorchDevice())

    # 执行发送
    asyncio.run(sender.async_send_weights(iter([("strided", weight)])))

    # 从 buffer 恢复并验证
    recovered = buffer.view(dtype=weight.dtype).view(weight.shape)
    assert torch.equal(recovered, weight)
```

评论区精华

Gemini Code Assist (grok) 在 review 中提出关键优化建议：避免使用 weight.contiguous() 创建临时张量，而是将目标缓冲区视图重塑为与 weight 相同的 dtype 和 shape，然后直接 copy_。理由是 PyTorch 的 copy_ 原生支持从非连续源张量拷贝到连续目标张量，无需中间内存分配。作者采纳了这一建议，并在第二个提交中实现了该优化方案。

- 避免 weight.contiguous() 临时分配替代方案 (performance): 作者采纳建议，在第二个提交中实现了视图 +copy_ 方案。

风险与影响

- 风险：本 PR 改动范围极小（3 行源码 + 单元测试），且直接替换为 PyTorch 标准语义的 copy_ 操作，回归风险低。但需注意：对目标缓冲区的两次 .view() 要求 buffer 切片的总字节数与 weight 的 nbytes 严格一致，否则会因 shape 不匹配报错。现有逻辑已经通过 offset 和 weight.nbytes 保证这一点，风险可控。
- 影响：影响用户：使用 Megatron-Bridge 或其他导出器可能返回非连续张量的用户，此前会遇到崩溃，修复后可正常运行。影响系统：无性能回退，对连续张量仅是增加了额外的 .view() 开销（可忽略）。影响团队：无架构或接口变更，无需迁移。
- 风险标记：核心路径变更

关联脉络

- PR #5599 [megatron] fix: Qwen3.5 LoRA & MTP support (with Megatron-Bridge): PR#5599 也涉及 bucketed_weight_transfer 模块，并且本 PR 的 bug 正是在使用

Megatron-Bridge 时发现，两个 PR 同属 Megatron-Bridge 集成路线。

- PR #7139 [sglang] fix: use _base guard in _compact_for_bucket to prevent NCCL buffer race: 同为 rollout 权重同步的修复，但针对 SGLang 后端的不同问题。