PR #7083 完整报告

verl-project/verl

[ckpt] fix: added cuda stream synchronization in NCCL broadcast wait for complete

合并时间：2026-07-20 10:45

原文链接：http://prhub.com.cn/verl-project/verl/pull/7083

执行摘要

- 一句话：修复 NCCL broadcast 缺少 CUDA stream 同步的竞争条件
- 推荐动作：该 PR 修了一个重要的数据竞争 bug，值得所有使用 NCCL checkpoint engine 的用户关注。建议合并后密切关注性能回归，特别是多 bucket 传输场景。

功能与动机

ray.util.collective.broadcast() 仅将 NCCL kernel 入队到 Ray 内部流池，不等 GPU 执行完成就返回，且未调用 record_stream()。wait_for_complete() 只等待入队完成，导致 buffer 可能被提前释放或重用，以及 send_weights 的 time cost 日志测量的是入队时间而非实际传输时间。详见 PR body 和关键讨论。

实现拆解

1. 修正 send_weights：在最后一个 bucket 的 broadcast_op.wait_for_complete() 之后、日志输出之前，添加 torch.cuda.synchronize()，确保广播 kernel 完成后再释放 buffer 和记录时间。
2. 修正 receive_weights（内部 _receive_weight_chunks）：在 first bucket 的 wait_for_complete() 之后、yield 之前，添加 torch.cuda.synchronize()，确保接收方在消费 buffer 前广播已完成，避免非阻塞流拷贝读到部分数据。
3. 文档更新：更新 wait_for_complete 的 docstring，明确说明其不保证 NCCL kernel 完成，仅保证入队完成。

关键文件：

- verl/checkpoint_engine/nccl_checkpoint_engine.py（模块 检查点引擎；类别 source；类型 core-logic；符号 BroadcastOperation._run, BroadcastOperation.wait_for_complete, NCCLCheckpointEngine.send_weights, NCCLCheckpointEngine._receive_weight_chunks）：核心变更文件：在 send_weights 和 receive_weights 中添加 torch.cuda.synchronize() 确保 NCCL broadcast 真正完成。

关键符号：BroadcastOperation._run, BroadcastOperation.wait_for_complete, NCCLCheckpointEngine.send_weights, NCCLCheckpointEngine._receive_weight_chunks

关键源码片段

verl/checkpoint_engine/nccl_checkpoint_engine.py

核心变更文件：在 send_weights 和 receive_weights 中添加 torch.cuda.synchronize() 确保 NCCL broadcast 真正完成。

```
# verl/checkpoint_engine/nccl_checkpoint_engine.py (head)

async def send_weights(
    self,
    named_params: Iterable[tuple[str, nn.Parameter]],
    global_steps: int | None = None,
) -> None:
    """Send weights to all ranks via NCCL broadcast with bucketization."""
    # ... 前面的 bucket 循环 ...

    # broadcast last bucket
    torch.cuda.synchronize() # 确保最后一个 bucket 的 copy-in 完成
    if broadcast_op is not None:
        await broadcast_op.wait_for_complete()

    broadcast_op = BroadcastOperation(
        rank=self.rank,
        group_name=self.group_name,
        bucket=send_buf,
        metadata={"bucket_meta": bucket_meta, "is_last": True},
        socket=self.socket,
        topic=self.topic,
    )
    await broadcast_op.wait_for_complete()

    # 等待 NCCL broadcast kernel 真正在 GPU 上完成，
    # 然后才释放 buffer 并记录时间。
    # 如果不做此同步，buffer 可能在 broadcast 还在后台运行时就被释放。
    torch.cuda.synchronize()

    logger.info(f"Rank {self.rank} send weights done, time cost: {time.time() - start_time:.2f}s")


async def _receive_weight_chunks(self) -> AsyncGenerator[tuple[str, torch.Tensor], None]:
    """Receive weight chunks, ensuring broadcast completion before yield."""
    assert self.rank > 0, "Rank 0 should not receive weights."
    send_buf, recv_buf = self.send_buf, self.recv_buf

    # receive first bucket
    start_time = time.time()
    broadcast_op = BroadcastOperation(
        rank=self.rank,
        group_name=self.group_name,
        bucket=recv_buf,
```

```
            metadata=None,
            socket=self.socket,
            topic=self.topic,
        )
        metadata = await broadcast_op.wait_for_complete()

        # 等待第一个 bucket 的 NCCL broadcast 完成后再 yield。
        # 如果外层使用非默认 CUDA 流拷贝，可能会读到未完成的数据。
        torch.cuda.synchronize()

        yield from self._process_bucket(metadata, recv_buf)
        # ... 继续接收剩余 bucket ...
```

评论区精华

Review 评论中，gemini-code-assist[bot] 指出在 _run 的线程中调用 torch.cuda.synchronize() 不带设备参数，将同步默认设备（device 0），而非当前 bucket 所在的设备。在多 GPU 环境下可能导致同步错误。但最终提交的版本将 synchronize 移到了 send_weights 和 receive_weights 的主线程上下文中，避免了多线程设备问题。此外，wuxibin89 在 Issue 评论中提出了 CUDA stream 同步的隐式同步可能足够，但 parinayc20 回应说无法保证所有 rollout 都使用默认流，因此需要显式同步。

- 多线程中 torch.cuda.synchronize() 的设备参数 (correctness): 最终提交将 synchronize 移出 _run，放在主线程的 send_weights 和 receive_weights 中，避免了线程设备问题。
- 显式同步 vs 隐式默认流同步 (design): PR 保留了显式同步，但移除了所有 bucket 的过度同步，仅保留在发送方最后和接收方第一个 bucket 后。

风险与影响

- 风险：添加 torch.cuda.synchronize() 会引入额外的 GPU 同步开销，可能降低性能。但 PR 将其限制在发送方最后一个 bucket 后和接收方第一个 bucket 后，影响范围有限。但如果存在多 bucket 传输，对于每个 bucket 都会有一次同步（发送方在迭代中已有一个同步，接收方在循环中也有同步），可能带来累积延迟。对单 bucket 传输影响最小。
- 影响：影响 NCCLCheckpointEngine 的 send_weights 和 receive_weights 方法，修复了潜在的权重损坏 bug。所有使用 NCCL checkpoint engine 的分布式训练任务都会受益，尤其是非默认 stream 拷贝权重的 rollout 引擎（如独立推理后端）。日志中的 time cost 现在反映真实传输时间。
- 风险标记：核心路径变更，性能退化风险（添加同步）

关联脉络

- PR #6974 [ckpt,rollout] feat: sharded delta weight sync over NCCL for disaggregated rollout: 同一模块（NCCLCheckpointEngine）的增强，共享权重同步路径，本 PR 修复了其中可能存在的同步问题。