PR #6738 完整报告

verl-project/verl

[rollout] fix: skip redundant clone in get_named_tensor_buckets to avoid OOM during SGLang weight sync

合并时间：2026-06-16 09:25

原文链接：http://prhub.com.cn/verl-project/verl/pull/6738

执行摘要

- 一句话：修复 SGLang 权重同步时冗余 clone 导致的 OOM
- 推荐动作：值得精读。这是一个典型的“一行改动解决大问题”的 PR，展示了如何通过理解张量存储布局来避免冗余分配。_compact_for_bucket 函数可作为工具函数在类似场景复用。建议合并并回测 Megatron 路径（作者提到无法测试），确保不引入回归。

功能与动机

Issue #6733 报告：SGLang 权重同步时，get_named_tensor_buckets 无条件 clone() 每个张量。对于多 GiB 的融合 MoE 权重，瞬态内存翻倍导致 CUDA OOM；而实际只需对视图（view）张量进行 clone 以压缩存储。PR body 也给出了详细的正确性论证。

实现拆解

1. 提取辅助函数 _compact_for_bucket(于 verl/workers/rollout/sglang_rollout/utils.py): 新增一个纯函数，判断张量是否同时满足 is_contiguous() 且 untyped_storage().nbytes() == numel * element_size()。若满足，说明张量拥有紧致自有存储，直接返回原张量；否则返回 tensor.clone()。
2. 修改 get_named_tensor_buckets(同一文件): 将内部两处 tensor.clone() 替换为 _compact_for_bucket(tensor)，其他逻辑不变。
3. 新增单元测试(于 tests/workers/rollout/test_sglang_rollout_sharding_manager.py): 导入 _compact_for_bucket，新增四个测试用例：
   - test_compact_for_bucket_skips_clone_for_owned_contiguous_tensor: 验证紧致自有张量返回原对象（is 断言）。
   - test_compact_for_bucket_clones_view_into_larger_buffer: 验证视图被 clone 且值不变。
   - test_compact_for_bucket_clones_non_contiguous_tensor: 验证非连续张量被 clone。
   - test_get_named_tensor_buckets_preserves_values: 端到端验证 bucket 后所有张量值不变。
4. 导入调整: 测试文件新增对 _compact_for_bucket 的导入，同时保留 get_named_tensor_buckets 的导入。

关键文件：

- verl/workers/rollout/sglang_rollout/utils.py（模块 权重同步；类别 source；类型 core-logic；符号 _compact_for_bucket）：核心修改文件，新增 _compact_for_bucket 函数并修改 get_named_tensor_buckets 以条件化 clone，直接修复 OOM。
- tests/workers/rollout/test_sglang_rollout_sharding_manager.py（模块 测试；类别 test；类型 test-coverage；符号 test_compact_for_bucket_skips_clone_for_owned_contiguous_tensor, test_compact_for_bucket_clones_view_into_larger_buffer, test_compact_for_bucket_clones_non_contiguous_tensor, test_get_named_tensor_buckets_preserves_values）：新增 4 个单元测试，覆盖 _compact_for_bucket 的三种分支以及端到端值校验，提高变更安全性。

关键符号：_compact_for_bucket, get_named_tensor_buckets

关键源码片段

verl/workers/rollout/sglang_rollout/utils.py

核心修改文件，新增 _compact_for_bucket 函数并修改 get_named_tensor_buckets 以条件化 clone，直接修复 OOM。

```
def _compact_for_bucket(tensor: torch.Tensor) -> torch.Tensor:
    """
    Return a tensor safe to retain in a weight-sync bucket without pinning
    extra memory.

    ``get_named_tensor_buckets`` keeps every tensor alive until its bucket is
    flushed.  A tensor that is a *view* into a larger backing buffer would
    therefore keep that whole buffer resident (and ship the whole buffer
    downstream), so such views must be compacted with ``clone()``.

    However the weights synced here come from ``DTensor.full_tensor()``
    (a fresh all-gather) and already own tight, contiguous storage.  Cloning
    those allocates a second full-size buffer and transiently doubles the
    tensor's footprint — which OOMs on multi-GiB fused MoE weights
    (e.g. ``[num_experts, ...]`` ``gate_up_proj``/``qkv``) while the actor
    params and rollout weights are both already resident.  Skip the clone
    when the tensor already owns its storage.
    """
    # 如果张量连续且自有存储大小恰好等于其元素所需大小，
    # 说明它拥有紧致自有存储，直接返回原张量以避免瞬态翻倍。
    if tensor.is_contiguous() and tensor.untyped_storage().nbytes() == (
        tensor.numel() * tensor.element_size()
    ):
        return tensor
    # 否则（视图或非连续），clone 以压缩到独立存储。
    return tensor.clone()


async def get_named_tensor_buckets(
    iterable: Iterator[tuple[str, torch.Tensor]], bucket_bytes: int
```

```
) -> Iterator[list[tuple[str, torch.Tensor]]]:
    # ... 省略 docstring 和边界检查 ...
    current_bucket = []
    current_size = 0
    async for name, tensor in ensure_async_iterator(iterable):
        tensor_size = tensor.element_size() * tensor.numel()
        if current_size + tensor_size > bucket_bytes:
            if current_bucket:
                yield current_bucket
            # 原为 tensor.clone()，现替换为条件 clone
            current_bucket = [(name, _compact_for_bucket(tensor))]
            current_size = tensor_size
        else:
            current_bucket.append((name, _compact_for_bucket(tensor)))
            current_size += tensor_size
    if current_bucket:
        yield current_bucket
```

tests/workers/rollout/test_sglang_rollout_sharding_manager.py

新增 4 个单元测试，覆盖 _compact_for_bucket 的三种分支以及端到端值校验，提高变更安全性。

```
def test_compact_for_bucket_skips_clone_for_owned_contiguous_tensor():
    # 一个刚分配的连续张量拥有紧致自有存储（对应 DTensor.full_tensor() 场景）。
    # 它必须被原样返回，这样 bucketing 就不会瞬态翻倍其峰值内存。
    tensor = torch.randn(128, 64)
    assert _compact_for_bucket(tensor) is tensor # 同一对象，无 clone


def test_compact_for_bucket_clones_view_into_larger_buffer():
    # 一个只占大 backing buffer 一部分的连续视图，必须被 compact，
    # 否则整个 backing buffer 会保持常驻 / 被传输。
    base = torch.randn(256, 64)
    view = base[:128]
    assert view.is_contiguous()
    out = _compact_for_bucket(view)
    assert out is not view # 新分配，不是原视图
    assert out.untyped_storage().nbytes() == out.numel() * out.element_size()
    assert torch.equal(out, view)


def test_compact_for_bucket_clones_non_contiguous_tensor():
    # 转置产生非连续张量，必须 clone 到自己的独立存储。
    tensor = torch.randn(64, 128).t()
    assert not tensor.is_contiguous()
    out = _compact_for_bucket(tensor)
    assert out is not tensor
    assert out.data_ptr() != tensor.data_ptr() # 不同存储
    assert torch.equal(out, tensor)
```

```
@pytest.mark.asyncio
async def test_get_named_tensor_buckets_preserves_values():
    # 条件 clone 不能改变最终进入 bucket 的数据。
    named_tensors = [("a", torch.randn(64, 64)),
                     ("b", torch.randn(64, 64)),
                     ("c", torch.randn(64, 64))]
    expected = {name: tensor.clone() for name, tensor in named_tensors}
    flat = {}
    async for group in get_named_tensor_buckets(iter(named_tensors), 0.5 * _BYTES_1MB):
        for name, tensor in group:
            flat[name] = tensor
    assert set(flat) == set(expected)
    for name, tensor in expected.items():
        assert torch.equal(flat[name], tensor)
```

评论区精华

无实质性 review 讨论。PR 作者在 body 中已做出详尽的正确性分析：FSDP 路径（DTensor.full_tensor()）产出紧致自有张量，跳过 clone 安全；Megatron 路径的大融合权重是视图，仍会被 clone（行为不变）；其余小张量来自 fresh all-gather buffer 或冻结参数，跳过 clone 也安全。作者还恳请维护者验证没有 rollout/checkpoint 路径会向此函数传入“可重用、紧致存储的 buffer”。

- 暂无高价值评论线程

风险与影响

- 风险：低风险。变更本质上是对 clone 的条件化，而非移除。对于紧致自有存储的张量跳过 clone，不会改变语义（clone 的语义是 deep copy，但此处不需要 copy）。对于视图和非连续张量，行为与变更前一致。唯一潜在风险是：若未来传入一个紧致自有但后续被复用的 buffer，跳过 clone 可能导致权重数据在 sync 完成前被修改；但从当前调用链（FSDP all-gather 或 Megatron 冻结参数）看，这种情况不存在。PR 作者也明确说明了这一点。
- 影响：直接影响 SGLang 后端的 actor→rollout 权重同步路径，主要受益方为使用大 MoE 模型（如 多专家融合权重）的用户。修复后，在 OOM 临界场景下可节省数 GiB 显存，使训练能够继续。对非大模型或无 OOM 的用户无负面影响。影响范围限定在 SGLang rollout 后端，不影响 vLLM 或其他后端。
- 风险标记：核心路径变更 ，缺少 Megatron 端到端验证

关联脉络

- 暂无明显关联 PR