PR #7553 完整报告

verl-project/verl

[trainer] fix: enforce strict dynamic micro-batch token limits

合并时间：2026-08-27 17:10

原文链接：http://prhub.com.cn/verl-project/verl/pull/7553

执行摘要

- 一句话：动态 micro-batch 强制 max_token_len 上限
- 推荐动作：值得精读。这是对训练显存控制核心逻辑的修正，关键设计是“用循环提高 micro-batch 数并重切分”与“先 all_reduce 再抛错防止死锁”，以及把约束错误从 assert 升级为显式 ValueError。建议关注 while 循环在最坏情况下的收敛性，以及 DP 间同步条件下强制组与 micro-batch 数的兼容性边界。

功能与动机

PR body 指出动态 micro-batch packing 原先用 ceil(total_sequence_length / max_token_len) 估算 micro-batch 数量，但这个值只是理论下界，因为单个样本不可拆分，Karmarkar-Karp 启发式主要用于均衡计算负载，可能产生超过 max_token_len 的 micro-batch。例如 max_token_len=8、序列长度为 [7,7,7,7,7,7,7,7] 时，初始计算得到 7 个 micro-batch，但其中一个必须包含两个样本，导致 14 个 token 违反配置上限。因此需要让 max_token_len 成为严格上界，否则用户的显存预算可能被意外突破。

实现拆解

1. 引入同步标志与分组 token 统计：在 *rearrange_micro_batches* 中将原判断 dist.is_initialized() and same_micro_num_in_dp and dp_group is not None 提取为布尔变量 sync_micro_batch_count；当 force_group_size > 1 时新增 group_token_lens（每组 token 总数），否则取单样本 seq_len_effective，为后续硬约束检查做准备。
2. DP 间 fatal 约束集体同步：若 sync_micro_batch_count 为真，先对 max_group_token_len 和 -num_groups 做 all_reduce(MAX)，确保所有 rank 在任何一个 rank 抛错前对“组是否超限”和“最少组数”达成一致，否则后续集合通信会挂死。
3. 强制组超限直接抛错：若 max_group_token_len > max_token_len，则 raise ValueError，说明强制分组不可拆分且超过 token 上限。
4. 循环重切分直至硬约束满足：在 while True 循环中，若 num_micro_batches 超过 min_num_groups（即每组一个 micro-batch 仍不够），抛 ValueError 说明无法同时满足 max_token_len 与 DP 间相同 micro-batch 数；否则调用 get_seqlen_balanced_partitions 重新切分，并检查每个 partition 的 token 总和是否 <= max_token_len；若 sync_micro_batch_count 为真还需对 within_limit 做 all_reduce(MIN) 后再决定是否 break，否则 num_micro_batches 按 step（num_batches_divided_by 或 1）递增。

5. 测试配套：tests/utils/test_seqlen_balancing.py 新增 test_micro_batches_respect_max_token_len（8 条序列长度 7、max_token_len=8，期望 8 个 micro-batch 且每个不超过 8）与分布式异常场景 _constraint_error_worker/test_seqlen_balancing_distributed_constraint_errors；tests/utils/test_prepare_micro_batches_with_group_size.py 补充 test_force_group_exceeding_token_limit_raises，并上调多个既有用例的 max_token_len_per_gpu 以适配更严格的新预算。

关键文件：

- verl/utils/seqlen_balancing.py（模块 序列切分；类别 source；类型 core-logic；符号 rearrange_micro_batches）：核心实现文件，将所有 micro-batch 生成的 token 限制从软约束改为硬约束，并新增 DP 间约束同步与 ValueError 保护。
- tests/utils/test_seqlen_balancing.py（模块 序列切分；类别 test；类型 test-coverage；符号 test_micro_batches_respect_max_token_len, _constraint_error_worker, test_seqlen_balancing_distributed_constraint_errors）：新增单机硬约束测试与分布式约束错误测试，验证循环切分和 DP 间同步逻辑，是防止回归的关键。
- tests/utils/test_prepare_micro_batches_with_group_size.py（模块 批切分；类别 test；类型 test-coverage；符号 test_force_group_exceeding_token_limit_raises）：验证 force_group_size 场景下超限组直接抛 ValueError，并上调既有用例的 token 预算以适配新硬约束。

关键符号：rearrange_micro_batches

关键源码片段

verl/utils/seqlen_balancing.py

核心实现文件，将所有 micro-batch 生成的 token 限制从软约束改为硬约束，并新增 DP 间约束同步与 ValueError 保护。

```
# verl/utils/seqlen_balancing.py 中 rearrange_micro_batches 的核心改动部分

    total_seqlen = seq_len_effective.sum().item()
    num_groups = batch_size // force_group_size
    num_micro_batches = min(num_groups, ceildiv(total_seqlen, max_token_len))
    if min_num_micro_batch is not None:
        num_micro_batches = max(min_num_micro_batch, num_micro_batches)

    # 是否需要 DP 间保持相同 micro-batch 数量（训练同 DP 组内时要求一致）
    sync_micro_batch_count = dist.is_initialized() and same_micro_num_in_dp and dp_group is
    not None
    if sync_micro_batch_count:
        num_micro_batches = torch.tensor([num_micro_batches], device=get_device_name())
        dist.all_reduce(num_micro_batches, op=dist.ReduceOp.MAX, group=dp_group)
        num_micro_batches = num_micro_batches.cpu().item()
    if num_batches_divided_by is not None:
        num_micro_batches = roundup_divisible(num_micro_batches, num_batches_divided_by)
```

```
seq_len_effective = seq_len_effective.long()

# 无论 force_group_size 是否 > 1，都准备“不可拆分组”的 token 总数，用于硬约束检查
if force_group_size > 1:
    workloads_per_sample = calculate_workload(seq_len_effective)
    group_workloads = workloads_per_sample.view(num_groups, force_group_size).sum(dim=1)
    .cpu().tolist()
    group_token_lens = seq_len_effective.view(num_groups, force_group_size).sum(dim=1).
    cpu().tolist()
    workloads = group_workloads
else:
    workloads = calculate_workload(seq_len_effective).cpu().tolist()
    group_token_lens = seq_len_effective.cpu().tolist()

max_group_token_len = max(group_token_lens)
min_num_groups = num_groups
if sync_micro_batch_count:
    # 关键：在任何一个 rank 抛错前，先把 fatal 约束通过 all_reduce 达成一致，
    # 否则其他 rank 仍会进入后续 collective，导致 peer 挂起（死锁）。
    constraints = torch.tensor([max_group_token_len, -num_groups], dtype=torch.long, device=
    get_device_name())
    dist.all_reduce(constraints, op=dist.ReduceOp.MAX, group=dp_group)
    max_group_token_len = int(constraints[0].item())
    min_num_groups = -int(constraints[1].item())

if max_group_token_len > max_token_len:
    raise ValueError(
        "A forced group exceeds max_token_len and cannot be split. "
        f"Got max_group_token_len={max_group_token_len} and max_token_len={max_token_len}
        ."
    )

# ceildiv(total_seqlen, max_token_len) 只是下界；样本不可拆分，
# 需要循环增加 micro-batch 数，直到均衡切分结果也满足硬上限。
step = num_batches_divided_by or 1
while True:
    if num_micro_batches > min_num_groups:
        raise ValueError(
            "Cannot satisfy max_token_len while keeping forced groups atomic and using the
            same "
            "micro-batch count across DP ranks. "
            f"Requested {num_micro_batches} non-empty micro-batches, but a rank has only "
            f"{min_num_groups} forced groups."
        )
    micro_bsz_group_idx = get_seqlen_balanced_partitions(workloads, num_micro_batches,
    equal_size=False)
    within_limit = all(
        sum(group_token_lens[idx] for idx in partition) <= max_token_len for partition in micro_
        bsz_group_idx
```

```
        )
        if sync_micro_batch_count:
            # DP 间任何一个 rank 未满足上限就继续增加 micro-batch 数，保持整体一致。
            within_limit_tensor = torch.tensor([int(within_limit)], device=get_device_name())
            dist.all_reduce(within_limit_tensor, op=dist.ReduceOp.MIN, group=dp_group)
            within_limit = bool(within_limit_tensor.item())
        if within_limit:
            break
        num_micro_batches += step
```

tests/utils/test_seqlen_balancing.py

新增单机硬约束测试与分布式约束错误测试，验证循环切分和 DP 间同步逻辑，是防止回归的关键。

```
# tests/utils/test_seqlen_balancing.py 中新增与修改的关键测试

def test_micro_batches_respect_max_token_len():
    # 复现 PR 描述的核心场景：8 条长度都为 7 的序列，max_token_len=8。
    # 原先 ceil(56/8)=7 会导致某个 micro-batch 含 14 个 token；修复后必须 8 个 micro-batch。
    input_ids = torch.zeros((8, 7), dtype=torch.long)
    attention_mask = torch.ones_like(input_ids)
    dataproto = DataProto.from_single_dict({"input_ids": input_ids, "attention_mask": attention_
    mask})

    micro_batches, _ = rearrange_micro_batches(dataproto.batch, max_token_len=8)

    assert len(micro_batches) == 8
    # 硬约束：每个 micro-batch 的 token 数均不得超过 max_token_len
    assert all(micro_batch["attention_mask"].sum().item() <= 8 for micro_batch in micro_batches)

# 分布式 worker 中，原来的计数断言改为同时校验上限与 DP 一致性
# minimum = min(len(seq_len_effective), ceildiv(total_seqlen, max_token_len))
# assert len(micros) >= minimum
# assert all(micro["attention_mask"].sum().item() <= max_token_len for micro in micros)
# ...
# if use_same_dp:
# counts[rank].fill_(len(micros))
# dist.all_gather(counts, counts[rank])
# assert len({int(count.item()) for count in counts}) == 1
```

tests/utils/test_prepare_micro_batches_with_group_size.py

验证 force_group_size 场景下超限组直接抛 ValueError，并上调既有用例的 token 预算以适配新硬约束。

```
# tests/utils/test_prepare_micro_batches_with_group_size.py 新增测试

def test_force_group_exceeding_token_limit_raises():
    # force_group_size=2：两个样本长度 200 和 210，组 token 总数为 410，
    # 超过 max_token_len_per_gpu=300，且组不可拆分，必须直接报错。
```

```
    batch = _make_batch(seq_lens=[200, 210], force_group_size=2, max_token_len_per_gpu=300)

    with pytest.raises(ValueError, match="forced group exceeds max_token_len"):
        prepare_micro_batches(batch)
```

评论区精华

PR 无 review 评论与讨论线程，仅有一条 APPROVED 审核。提交历史中第二条 commit 标题为 "prevent deadlock"，结合代码中多处 all_reduce 同步与“Fatal constraints must be agreed on before any rank raises; otherwise peers can hang in the next collective”注释，可推断审核过程中关注了分布式场景下异常路径导致的死锁风险，并以集体通信先行同步作为解决方案。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 性能风险：while 循环可能在极端长度分布下需要多次重切分，理论上存在迭代次数不可控的可能；不过每次增加 micro-batch 数量会快速收敛，且 PR 实测步耗时几乎不变。
  2. 分布式死锁风险：新增 all_reduce 同步点位于 collectives 之前，若某 rank 提前抛错或路径不一致仍可能挂起；代码已通过先同步约束条件缓解，但异常路径覆盖仍依赖新增的分布式测试。
  3. 行为变更风险：原来允许超限执行的场景，现在可能直接 ValueError。对依赖旧行为的配置（如 TP 场景下 max_token_len 设置偏小且样本不可拆）可能导致训练启动失败，需要用户上调 max_token_len 或关闭同 DP 数约束。
  4. 回归风险：forced_group 场景下预算收紧，测试中 max_token_len_per_gpu 从 200/500/300/400 上调到 220/850/420/650，说明新逻辑会让旧测试用例失败，实际集群中的既有配置可能需要相应调整。 - 影响：对用户而言，max_token_len 从“理论下界”变为“严格上界”，显存峰值更可预期，可避免 OOM；代价是可能增加 micro-batch 数量、略微增加调度开销。对系统而言，涉及所有使用 dynamic bsz / micro-batch packing 的训练路径（SFT、PPO 等），影响面较广但改动集中在 seqlen_balancing 单文件。对团队而言，新增了分布式约束错误测试，提升了对异常路径的回归保障。 - 风险标记：核心路径变更，分布式异常路径需测试覆盖，配置行为收紧可能影响既有用户，循环重切分收敛性未做最坏情况分析

关联脉络

- PR #7539 [ray] fix: skip unused TensorDict consolidation in NumPy DataProto serialization: 同为训练路径上的性能与稳定性修复，涉及 DataProto 与 rollout 的 token 处理，与本 PR 都关注数据传输 / 切分环节的隐性开销与边界问题。
- PR #7518 [rollout, ci] fix: make agent-loop tests fully deterministic: 同样是训练链路中保证确定性、消除隐式边界问题的修复，与本 PR 在‘让配置约束真正生效’与‘测试确定性’的思路上一致。