PR #6612 完整报告

verl-project/verl

[veomni] feat: reduce extra memory from O(E) to O(E/ep_size) during weight refit

合并时间：2026-06-05 10:38

原文链接：http://prhub.com.cn/verl-project/verl/pull/6612

执行摘要

- 一句话：VeOmni 专家重配内存从 O(E) 优化至 O(E/ep_size)
- 推荐动作：推荐阅读，特别是 transformer_impl.py 中的广播循环设计，以及 utils.py 中如何通过生成器模式灵活处理不同 MoE 配置下的参数映射。同时关注 review 中的两个争议点（参数名与 buffer 共享），提醒在类似模式中注意数据隔离。

功能与动机

在 VeOmni 权重重配阶段，原先使用 all_gather 一次性收集所有专家参数，导致单 rank 峰值内存与总专家数 E 成正比。当模型参数量大或 ep 较小时，O(E) 的内存占用成为瓶颈。本优化将 all_gather 替换为 broadcast 循环，各 rank 依次广播本地分片，使峰值内存降至 O(E/ep_size)。

实现拆解

1. 重构参数映射函数：在 verl/workers/engine/veomni/utils.py 中新增 _map_moe_params_common，接受 ep_rank 参数并计算全局专家索引；修改 _map_moe_params_qwen3_moe 拆解 gate_up_proj 并调用 _map_moe_params_common。同步更新 MOE_PARAM_HANDERS 字典，为 qwen3_5_moe 类型注册新版映射函数。
2. 核心通信模式变更：在 verl/workers/engine/veomni/transformer_impl.py 的 VeOmniEngine.get_per_tensor_param 中，将 all_gather_into_tensor 替换为 broadcast 循环。每个 rank 在 ps.ep_group 内依次广播自己的分片 tensor，其他 rank 接收并调用 process_func 产生参数条目。
3. 调整调试日志：在 verl/plugin/platform/platform_manager.py 中将平台检测相关的 logger.info 降为 logger.debug，避免在 ray 环境中因 torch.distributed 初始化前的 INFO 日志导致意外行为。
4. 新增端到端测试：创建 tests/utils/veomni/test_special_export_unfused_experts.py，利用 Qwen3.5MoeDecoderLayer 构建单元测试，验证专家参数在 eps=4 的 EP 网格上经 broadcast 映射后正确还原。
5. 补齐 CI 与 Code Owners：在 .github/workflows/e2e_ppo_trainer_veomni_vllm.yml 中添加 torchrun 执行新增测试的步骤；在 .github/CODEOWNERS 中添加 /verl/workers/engine/veomni 的维护者 @wuxibin89 @Luosuu @FoolPlayer。

关键文件：

- verl/workers/engine/veomni/transformer_impl.py（模块 VeOmni 引擎；类别 source；类型 core-logic；符号 get_per_tensor_param）：核心内存优化逻辑实现：将 all_gather 替换为 broadcast 循环，逐 rank 收集专家参数，大幅降低峰值内存。
- verl/workers/engine/veomni/utils.py（模块 VeOmni 工具；类别 source；类型 core-logic；符号 _map_moe_params_common, _map_moe_params_qwen3_moe）：参数映射函数重构：新增 _map_moe_params_common 并修改 _map_moe_params_qwen3_moe 以支持按 ep_rank 分配专家索引，注册新模型类型处理器。
- tests/utils/veomni/test_special_export_unfused_experts.py（模块 专家导出测试；类别 test；类型 test-coverage；符号 get_by_path, set_by_path, get_per_tensor_param, test_veomni_export_unfused_experts）：新增单元测试，验证 broadcast 映射后专家参数的正确性，覆盖 EP 网格下的数据一致性。
- verl/plugin/platform/platform_manager.py（模块 平台管理；类别 source；类型 core-logic；符号 _detect_platform_name, get_platform）：降低平台探测日志级别，避免在 ray 环境中因过早的 INFO 日志引发 torch.distributed 初始化问题。
- .github/workflows/e2e_ppo_trainer_veomni_vllm.yml（模块 CI 配置；类别 infra；类型 infrastructure）：在 CI 中添加 export unfused experts 测试步骤，确保回归捕获。
- .github/CODEOWNERS（模块 仓库配置；类别 infra；类型 infrastructure）：明确 VeOmni 模块的维护者，便于后续变更通知和 Review。

关键符号：VeOmniEngine.get_per_tensor_param, _map_moe_params_common, _map_moe_params_qwen3_moe, test_veomni_export_unfused_experts, get_by_path, set_by_path

关键源码片段

verl/workers/engine/veomni/transformer_impl.py

核心内存优化逻辑实现：将 all_gather 替换为 broadcast 循环，逐 rank 收集专家参数，大幅降低峰值内存。

```
def get_per_tensor_param(self, **kwargs):
    load_veomni_model_to_gpu(self.module)
    params = self.module.state_dict()
    params = convert_weight_keys(params, getattr(self.module, "_fsdp_wrapped_module", self.
    module))
    if self._is_offload_param:
        offload_veomni_model_to_cpu(self.module)

    ps = parallel_state.get_parallel_state()
    model_type = getattr(self.module.config, "model_type", "default")
    # process_func 现在接受 ep_rank 参数，用于区分不同 rank 的专家
    process_func = MOE_PARAM_HANDERS.get(model_type, lambda n, t, ep_rank: iter([(n, t)]))

    def param_generator():
        for name, param in params.items():
```

```
                unsharded_tensor = param.full_tensor() if isinstance(param, DTensor) else param
                is_expert_layer = "mlp.experts." in name
                is_proj = any(p in name for p in ["down_proj", "gate_proj", "up_proj", "gate_up_proj"])

                if is_expert_layer and is_proj and ps.ep_enabled:
                    ep_rank, ep_size = ps.ep_rank, ps.ep_size
                    # 使用 buffer 复用内存，确保广播过程中不会分配额外空间
                    buffer = torch.empty_like(unsharded_tensor) # [num_experts/ep_size, H, I]
                    for src_ep_rank in range(ep_size):
                        # 广播源使用自己的 tensor，其他 rank 使用 buffer 接收
                        tensor = unsharded_tensor if src_ep_rank == ep_rank else buffer
                        torch.distributed.broadcast(tensor, src=src_ep_rank, group=ps.ep_group)
                        yield from process_func(name, tensor, ep_rank=src_ep_rank)
                else:
                    if is_expert_layer:
                        # 非 EP 启用时仍传入 ep_rank=0 保持接口一致
                        yield from process_func(name, unsharded_tensor, ep_rank=0)
                    else:
                        yield name, unsharded_tensor

        return param_generator(), None
```

verl/workers/engine/veomni/utils.py

参数映射函数重构：新增 _map_moe_params_common 并修改 _map_moe_params_qwen3_moe 以支持按 ep_rank 分配专家索引，注册新模型类型处理器。

```
def_map_moe_params_common(name, tensor, ep_rank): """通用专家参数映射函数，根据 ep_rank 和分片数量计算全局专家索引。"""  num_experts_per_rank = tensor.size(0)  for i in range(num_experts_per_rank):  # 全局专家索引 = ep_rank * 每 rank 专家数 + 当前切片位置  idx = ep_rank * num_experts_per_rank + i  new_key = name.replace("mlp.experts.", f"mlp.experts.{idx}.") + ".weight"  # 注意：如果 tensor 是共享 buffer 的视图，必须 clone 以确保后续广播不覆盖  yield new_key, tensor[i].to(get_device_id(), non_blocking=True)#.clone() # 建议添加 clone()
def_map_moe_params_qwen3_moe(name, tensor, ep_rank): """Qwen3.5-MoE 专属映射，将 fused gate_up_proj 拆分为 gate_proj 和 up_proj。"""  if "gate_up_proj" in name:  gate, up = tensor.chunk(2, dim=1)  params = { name.replace("gate_up_proj", "gate_proj"): gate,  name.replace("gate_up_proj", "up_proj"): up,  }  else:  params = {name: tensor}  for key, value in params.items():  yield from _map_moe_params_common(key, value, ep_rank)
```

注意：MOE_PARAM_HANDERS 注册表也相应更新：qwen3_moe 和 deepseek_v3 指向通用函数，新增 qwen3_5_moe 指向拆分函数。

评论区精华

1. broadcast 参数名争议：Review 机器人指出 torch.distributed.broadcast 应使用 src 而非 group_src，否则运行时可能引发 TypeError。作者 @wuxibin89 回应 group_src 可用于子组内的广播，表明仓库可能采用了包装器或下游定制版本。该点未在代码中修改，风险

待确认。

2. buffer 共享导致数据覆盖：Review 机器人指出，_map_moe_params_common 中 yield 的 tensor 是 buffer 的视图，后续 broadcast 会覆盖先前 emit 的数据，建议加上 .clone()。PR 合并时未见显式 clone，若未修复可能导致静默错误。

- torch.distributed.broadcast 参数名 group_src 合法性 (correctness): 未修改代码，PR 合并。若环境为标准 PyTorch 则存在兼容风险。
- 广播循环中 buffer 共享导致数据覆盖 (correctness): 代码未见显式 clone，若未修复则存在静默错误风险。PR 已合并。

风险与影响

- 风险:
  - 通信 API 兼容性：torch.distributed.broadcast 的标准参数名为 src，但代码使用了 group_src。若环境未提供对应包装，将直接崩溃。该 PR 已合入，建议追踪确认是否存在自定义 broadcast 函数。
  - Buffer 共享导致数据错误：广播循环中公用 buffer，_map_moe_params_common 若直接 yield 未 clone 的 tensor，后续 broadcast 会覆盖已 emit 的数据。测试可能通过克隆避免，但源代码中未见显式 clone，存在静默数据损坏风险。
  - 日志级别降级影响可观测性：平台探测日志从 INFO 降为 DEBUG，在默认日志级别下不再输出，可能影响异地问题排查。但用户可通过 VERL_LOG_LEVEL 恢复。
- 影响:
  - 用户：VeOmni 用户直接受益于更低的峰值内存，特别是总专家数 E 较大时效果显著。无需修改配置，完全透明。
  - 系统：broadcast 循环增加了通信步数（E/ep_size 步串行），但单步 buffer 变小，整体网络负载不变。在 EP 组内依次广播可能引入额外的同步开销，但可以通过流水线或异步掩蔽。
  - 团队：Code Owners 的添加明确了 VeOmni 模块的维护责任，便于后续代码 Review 和变更通知。
  - 风险标记：通信 API 兼容性风险，Buffer 共享数据覆盖风险，日志可观测性下降

关联脉络

- PR #6086 [hardware] feat: add platform abstraction layer and plugin-based engine override system: 该 PR 引入了平台抽象层和 platform_manager.py，本 PR 对该文件进行了日志级别调整，属于同一模块的持续改进。