# PR #7306 完整报告

verl-project/verl

[fsdp] fix: add FSDP1 restore handler for Decoupled PPO

合并时间：2026-08-07 12:17

原文链接：http://prhub.com.cn/verl-project/verl/pull/7306

## 执行摘要

- 一句话：为 FSDP1 新增分片保存 / 恢复 handler，修复 Fully Async 崩溃
- 推荐动作：值得精读。该 PR 是一个小而清晰的 bugfix，展示了按「参数表示形态」拆分 handler 的设计思路：FSDP1 扁平 Tensor 与 FSDP2 DTensor 不能共用 CPU 保存 / 恢复工具。建议重点关注 fsdp_utils 中两个新函数的实现细节（barrier、跳过缺失参数、copy_ 语义）以及 engine_workers 中 strategy 分支的改动。但需注意测试缺失，后续应补一个 FSDP1 + fully_async + bypass_mode=False 的回归测试。

## 功能与动机

Issue #7249 明确给出了崩溃路径：当 actor 使用 FSDP1 且 algorithm.rollout_correction.bypass_mode=False 时，FullyAsyncTrainer._compute_old_log_prob 会调用 DetachActorWorker.save_model_to_cpu，进而调用 fsdp2_sharded_save_to_cpu。该函数面向 FSDP2 的 DTensor 参数设计，FSDP1 参数是扁平 Tensor，没有任何参数初始化 global_spec，因此抛出 'No DTensor-type parameters found in the model. FSDP2 sharding may not be enabled.' 的 AssertionError；恢复路径同样假设保存状态是 (cpu_sharded_state, global_spec) 元组，对 FSDP1 同样不成立。

## 实现拆解

1. 在 verl/utils/fsdp_utils.py 新增 FSDP1 专用保存 / 恢复函数：fsdp1_sharded_save_to_cpu 遍历 named_parameters，把每个 rank 的本地参数分片 detach 后拷贝到 CPU dict，分片布局隐含在本地形状中，无需像 FSDP2 那样记录 global_spec；fsdp1_sharded_load_from_cpu 在 torch.no_grad 下将 CPU 分片逐参数 copy_ 回 param.data，跳过保存状态中不存在的参数，并在末尾调用 dist.barrier() 保证各 rank 写回完成。
2. 修改 verl/experimental/separation/engine_workers.py 的 DetachActorWorker._get_strategy_handlers：原先 strategy in ['fsdp', 'fsdp2', 'veomni'] 统一走 fsdp2 工具，现在拆分为 strategy == 'fsdp' 映射到 fsdp1 新函数，strategy in ['fsdp2', 'veomni'] 仍走 fsdp2 工具，megatron 分支不变。这样从根上消除 FSDP1 误用 DTensor 保存逻辑的隐患。
3. 同步调整 restore_model_from_cpu 的拆包条件：只有 fsdp2/veomni 保存的是 (cpu_sharded_state, global_spec) 元组需要拆包，FSDP1 保存的是纯 dict，直接透传给

restore_handler，与 Megatron 分支一致。

4. 测试与配置配套：本 PR 未新增测试文件，也没有配置项变更；现有 Fully Async 默认配置 bypass_mode=True 不会走该路径，因此需要通过手动覆盖参数验证修复。

关键文件：

- verl/utils/fsdp_utils.py（模块 工具层；类别 source；类型 core-logic；符号 fsdp1_sharded_save_to_cpu, fsdp1_sharded_load_from_cpu）：新增 FSDP1 专用 CPU 分片保存 / 恢复函数，是本次修复的核心实现，直接解决 FSDP1 误用 DTensor 工具导致的断言崩溃。
- verl/experimental/separation/engine_workers.py（模块 解耦执行；类别 source；类型 dependency-wiring；符号 _get_strategy_handlers, restore_model_from_cpu）：DetachActorWorker 的 strategy handler 分流在此文件，是触发错误的入口链路；本次按 strategy 拆分 FSDP1 与 FSDP2/VeOmni，并修正 restore 拆包逻辑。

关键符号：fsdp1_sharded_save_to_cpu, fsdp1_sharded_load_from_cpu, _get_strategy_handlers, restore_model_from_cpu

## 关键源码片段

### verl/utils/fsdp_utils.py

新增 FSDP1 专用 CPU 分片保存 / 恢复函数，是本次修复的核心实现，直接解决 FSDP1 误用 DTensor 工具导致的断言崩溃。

```
def fsdp1_sharded_save_to_cpu(model: torch.nn.Module) -> dict[str, torch.Tensor]:
    """
    Sharded Save for FSDP1: each rank copies its own local parameter shards to CPU memory.

    FSDP1 暴露的是扁平、已分片的 ``torch.nn.Parameter`` 张量（use_orig_params=True 时是它的视图），
    而不是 DTensor，所以分片布局隐含在本地形状中，无需记录 global_spec。
    """
    cpu_sharded_state = {}
    for param_name, param in model.named_parameters():
        # detach 后拷贝到 CPU，避免保存引用导致后续训练修改本地参数
        cpu_sharded_state[param_name] = param.data.detach().to('cpu', copy=True)
    return cpu_sharded_state


def fsdp1_sharded_load_from_cpu(model: torch.nn.Module, cpu_sharded_state: dict[str, torch.Tensor]) -> None:
    """
    Sharded Load for FSDP1: each rank copies its own CPU shards back into the live parameters.

    模型必须与 save 时保持相同的分片状态，本地形状才能匹配。
    """
    with torch.no_grad():
        for param_name, param in model.named_parameters():
```

```
            # 跳过保存状态中不存在的参数（例如 save 后新增的参数），避免 KeyError
            if param_name not in cpu_sharded_state:
                continue
            # 将 CPU 分片拷回参数，.to(param.device) 保证与当前设备一致
            param.data.copy_(cpu_sharded_state[param_name].to(param.device))

    # 等待所有 rank 完成写回，确保 restore 后各 rank 观察到一致的状态
    dist.barrier()
```

## verl/experimental/separation/engine_workers.py

DetachActorWorker 的 strategy handler 分流在此文件，是触发错误的入口链路；本次按 strategy 拆分 FSDP1 与 FSDP2/VeOmni，并修正 restore 拆包逻辑。

```
# _get_strategy_handlers 核心分支：按 strategy 选择保存 / 恢复工具
# NOTE: FSDP1 分片是扁平 Tensor，FSDP2 分片是 DTensor，必须使用不同的工具
if strategy == 'fsdp':
    from verl.utils.fsdp_utils import (
        fsdp1_sharded_load_from_cpu,
        fsdp1_sharded_save_to_cpu,
    )
    self._strategy_handlers = (fsdp1_sharded_save_to_cpu, fsdp1_sharded_load_from_cpu)
elif strategy in ['fsdp2', 'veomni']:
    # VeOmni 内部数据并行实际走 FSDP2（参数为 DTensor），与 FSDP2 共用一套工具
    from verl.utils.fsdp_utils import (
        fsdp2_sharded_load_from_cpu,
        fsdp2_sharded_save_to_cpu,
    )
    self._strategy_handlers = (fsdp2_sharded_save_to_cpu, fsdp2_sharded_load_from_cpu)
elif strategy == 'megatron':
    from verl.utils.megatron_utils import (
        copy_megatron_model_to_cpu,
        restore_megatron_model_from_cpu,
    )
    self._strategy_handlers = (copy_megatron_model_to_cpu, restore_megatron_model_from_cpu)
else:
    raise NotImplementedError(f'Unsupported strategy: {strategy}')

# restore_model_from_cpu 的拆包逻辑
if n in self.cpu_saved_models:
    strategy = self.config.actor.strategy

    # FSDP2/VeOmni 的保存结果是 (cpu_sharded_state, global_spec) 元组，需要拆包；
    # FSDP1 与 Megatron 保存的是可直接传给 restore_handler 的 dict。
    if strategy in ['fsdp2', 'veomni']:
        cpu_sharded_state, global_spec = self.cpu_saved_models[n]
        self.restore_handler(self.actor.engine.module, cpu_sharded_state, global_spec)
    else:
        self.restore_handler(self.actor.engine.module, self.cpu_saved_models[n])
```

## 评论区精华

本 PR 无 review 评论与评论线程。核心结论来自关联 Issue #7249 的根因分析：DetachActorWorker._get_strategy_handlers 将 FSDP1、FSDP2、VeOmni 归并为同一组 handler，而 FSDP1 参数不是 DTensor，无法满足 fsdp2_sharded_save_to_cpu 的断言；修复思路即按参数表示形态拆分 handler，该思路与本 PR 实现一致。

- 暂无高价值评论线程

## 风险与影响

- 风险：
  1. 缺少测试覆盖：本次改动没有配套单测或集成测试，FSDP1 的 save/restore 路径回归风险只能靠手工验证。
  2. restore 形状依赖：fsdp1_sharded_load_from_cpu 要求模型处于与 save 时相同的分片状态，若 use_orig_params 或分片配置在保存后发生变化，copy_ 会因本地形状不匹配报错；若新增参数则被静默跳过，可能留下不一致的模型状态。
  3. CPU 往返同步开销：保存到 CPU、恢复时再 copy 回 GPU 会产生同步与传输开销，但在反复切换策略版本的 fully_async 路径上属于可接受成本。
  4. param_offload 场景：若 FSDP1 开启参数 offload 且参数在 CPU 上，直接调用 sharded save/load 会假设参数在 GPU，仍可能出错，注释已明确要求调用方先确保参数在 GPU。
  5. 保存格式变化：FSDP1 的 cpu_saved_models 条目从 (state, spec) 元组变为纯 dict，任何依赖旧格式的扩展代码需要同步适配。 - 影响：影响范围限定在 Decoupled PPO/Fully Async 训练：actor 使用 FSDP1 且 rollout_correction.bypass_mode=False 的组合此前必现崩溃，修复后该路径可正常重算 old_log_prob。FSDP2、VeOmni、Megatron 的保存 / 恢复行为保持不变，默认 Fully Async 配置（bypass_mode=True）不受影响。对使用该配置组合的团队是阻断性 bug 修复，影响面窄但价值明确。 - 风险标记：缺少测试覆盖，核心路径变更，保存格式变化，restore 依赖分片状态一致

## 关联脉络

- PR #7283 [fsdp,veomni] fix: backfill missing state for DSD optimizer checkpoint: 同为 FSDP1/FSDP2 状态保存恢复链路的修复，涉及 CPU 侧状态处理，格局一致。
- PR #7117 [ckpt] fix: save base model's code, not the PeftModel wrapper's, in FSDP checkpoints: FSDP checkpoint 保存相关的 bugfix，同属 FSDP 参数状态序列化 / 保存正确性问题。