PR #7144 完整报告

verl-project/verl

[ckpt, fsdp] feat: sharded delta block placements + backend-owned HF export (engine core + FSDP)

合并时间：2026-07-25 12:15

原文链接：http://prhub.com.cn/verl-project/verl/pull/7144

执行摘要

- 一句话：扩展分片 delta 同步：块放置 + 后端 HF 导出
- 推荐动作：值得精读，尤其是对分布式训练权重同步优化感兴趣的工程师。块放置设计思路清晰，契约分离架构有助于可扩展性。建议阅读 spec.py 中的 BlockPlacement 和 translate_flat_indices，以及 utils.py 中的 hf_delta_export 和 prime_delta_snapshots。

功能与动机

本 PR 是 #7085 拆分的首部分，也是 #7060 路线图的第一项。它为之前 #6974 的 Shard(0) 分片 delta 扩展了块放置支持以处理更通用的分片（如专家并行的 Shard(k)），并将导出契约转移到后端，为后续 veomni 集成铺平道路。

实现拆解

1. 核心契约重构：在 verl/workers/engine/spec.py 中，ShardSpec 新增 BlockPlacement 数据类，描述本地 shard 在全张量中的超矩形块。新增 translate_flat_indices 函数，支持从 shard 局部扁平索引映射到全张量扁平索引（混合进制分解 + 逐维偏移 + 重组）。derive_placement 函数扩展为支持 Shard(k) 和多 Shard 维，返回相应 BlockPlacement 或整数偏移。不支持的方案（_StridedShard 等）被明确拒绝。
2. 后端导出工具：在 verl/workers/engine/utils.py 中新增 hf_delta_export 和 prime_delta_snapshots 函数。前者接收原始 shard 生成器，将当前 tensor 与 pinned CPU 快照比较得到变化，并调用 entry_fn（FSDP 用身份映射，veomni 用 EP 转换器）将 shard 本地 delta 转换为最终 HF 坐标条目 (slots, dtype, counts, idx, val, gather_group)。后者在 seed 同步后立即将各 rank 当前 shard 固定到 CPU 作为稳态 diff 基。
3. 引擎核心重构：verl/checkpoint_engine/delta_checkpoint_engine.py 重写发送端。引入 _FlushPiece 和 _ValuesPiece 数据结构，分别表示索引式 flush 和 values-only flush 的单参数片段。_FlushBucket 实现一步前置流水线：片段累积至 cap 字节后组装为 flush。engine 现在消费后端产出的 HF 坐标条目，经过 _GatherQueue（调用 gather_slot_entries_to_rank0）收集到 rank 0，再经 _bucket_sliced 等方法打包为 flush 发出。seed 路径使用 values-only wire（get_per_tensor_param() 完整导出），稳态路径使用索引 + 值 wire。
4. 稀疏收集重构：verl/checkpoint_engine/delta_sync/sparse_gather.py 中 gather_v_batched_to_rank0 重命名为 gather_slot_entries_to_rank0，弃用 grouped 模

式，新增 max_round_bytes 参数支持确定性子轮次分割以避免单轮过大的 padded blob。移除了 gather_dense_to_rank0 等旧函数。

5. SGLang delta 加载器优化：verl/workers/rollout/sglang_rollout/delta_loader.py 中的 _masked_copy 使用 torch.where 代替之前的 bool(mask.all())+ 布尔索引，避免了逐参数 device→host 同步，提升了 MoE 等大量细粒度 flush 的性能。
6. FSDP 后端实现：verl/workers/engine/fsdp/transformer_impl.py 实现 _hf_delta_entry（身份映射配置）和 get_per_tensor_param_delta_shard 方法，与新的导出契约对接。verl/workers/engine/veomni/transformer_impl.py 中 delta_sharded 被标记为 NotImplementedError，直到后续 PR #7085。
7. 测试配套：新增 tests/checkpoint_engine/test_block_placement.py 全面测试 translate_flat_indices 正确性（快路径、全坐标、稀疏子集、NaN 重建）。test_sharded_delta.py 更新覆盖 prime→delta roundtrip、seed-required fail-loud 等。分布式 gather 测试相应调整。

关键文件：

- verl/checkpoint_engine/delta_checkpoint_engine.py（模块 delta 引擎；类别 source；类型 core-logic；符号 _FlushPiece, _ValuesPiece, _FlushBucket, init）：核心引擎重写，引入 FlushPiece/FlushBucket 数据结构，改变发送端流程，支持两种 flush 类型
- verl/workers/engine/spec.py（模块 引擎规约；类别 source；类型 dependency-wiring；符号 _prod, derive_placement, _row_major_strides, BlockPlacement）：ShardSpec 增加 BlockPlacement、place、gather_group 等字段，新增 translate_flat_indices 和 derive_placement 扩展，定义引擎与后端的导出契约
- verl/checkpoint_engine/delta_sync/sparse_gather.py（模块 稀疏收集；类别 source；类型 core-logic；符号 gather_v_batched_to_rank0, gather_slot_entries_to_rank0, gather_dense_to_rank0, gather_v_grouped_to_rank0）：重构 gather 函数，重命名为 gather_slot_entries_to_rank0，增加 max_round_bytes 子轮次支持，移除旧代码
- verl/workers/engine/utils.py（模块 后端工具；类别 source；类型 core-logic；符号 _prodshape, _hf_entry_identity, hf_delta_export, prime_delta_snapshots）：新增 hf_delta_export 和 prime_delta_snapshots 函数，实现后端侧 HF 导出工具，定义 delta 条目生成流水线
- verl/workers/engine/base.py（模块 引擎基类；类别 source；类型 dependency-wiring；符号 prime_delta_snapshots, get_per_tensor_param_delta_shard）：在引擎基类中新增 prime_delta_snapshots 和 get_per_tensor_param_delta_shard 接口方法
- verl/workers/engine/fsdp/transformer_impl.py（模块 FSDP 后端；类别 source；类型 dependency-wiring；符号 _hf_delta_entry, get_per_tensor_param_delta_shard）：FSDP 后端实现 _hf_delta_entry 和新的导出方法，与 delta 引擎契约对接
- verl/workers/rollout/sglang_rollout/delta_loader.py（模块 SGLang 加载；类别 source；类型 core-logic；符号 apply_delta, apply_dense, _masked_copy, masked_copy）：优化 masked_copy_ 使用 torch.where 避免逐参数 host sync，提升细粒度 flush 性能
- verl/workers/engine/veomni/transformer_impl.py（模块 VeOmni 后端；类别 source；类型 core-logic；符号 get_per_tensor_param_shard）：标记 delta_sharded 为 NotImplementedError，提醒 veomni 用户等待 #7085

- tests/checkpoint_engine/test_block_placement.py（模块 块放置测试；类别 test；类型 test-coverage；符号 _blocks_for_grid, test_translate_int_fast_path, test_translate_matches_full_coordinates, test_translate_sparse_subset）：新增全面 CPU 测试覆盖 BlockPlacement 和 translate_flat_indices 的各种场景
- tests/checkpoint_engine/test_sharded_delta.py（模块 delta 导出测试；类别 test；类型 test-coverage；符号 test_spec_to_hf_pure_permutation, test_spec_to_hf_chunk_preserves_nan_sentinels, to_hf_chunk, to_hf）：更新测试覆盖新的 hf_delta_export/prime_delta_snapshots 契约，seed-required 验证等
- tests/special_distributed/test_sharded_delta_gather.py（模块 分布式收集测试；类别 test；类型 test-coverage）：调整分布式 gather 测试适配新的函数签名

关键符号：BlockPlacement, translate_flat_indices, derive_placement, hf_delta_export, prime_delta_snapshots, hf_entry_identity, _hf_delta_entry, get_per_tensor_param_delta_shard, gather_slot_entries_to_rank0, masked_copy, _FlushBucket, _GatherQueue

关键源码片段

verl/workers/engine/utils.py

新增 hf_delta_export 和 prime_delta_snapshots 函数，实现后端侧 HF 导出工具，定义 delta 条目生成流水线

```
# verl/workers/engine/utils.py - 新增的后端侧 HF delta 导出工具


def _hf_entry_identity(name, spec, place, lidx, lval):
    '''身份映射：参数名称就是 HF 名称，将 shard 本地 delta
    转换为全局扁平坐标的 int32 索引和值。'''
    from .spec import translate_flat_indices

    gidx = (translate_flat_indices(lidx, place) if lidx.numel() else lidx).to(torch.int32)
    counts = torch.zeros(1, dtype=torch.int64)
    counts[0] = int(gidx.numel())
    return [(name, tuple(spec.full_shape))], str(lval.dtype).replace('torch.', ''), counts, gidx, lval


def hf_delta_export(gen, snaps: dict, entry_fn):
    '''稳态导出：将原始 (name, local_shard, spec) 生成器包装为
    HF 坐标 delta 条目生成器。每个参数与 pinned CPU 快照 diff，
    刷新快照，然后通过 entry_fn 转换成最终条目。'''
    from verl.checkpoint_engine.delta_sync.sparse_gather import shard_delta_indices
    from .spec import derive_placement

    for name, local, spec in gen:
        local = local.detach().contiguous().view(-1)
        snap = snaps.get(name)
        assert snap is not None and snap.numel() == local.numel(), (
```

```
                f'{name}: no seed snapshot for this shard; run the seed export first'
            )
            place, contributes, pg = derive_placement(spec)
            if contributes:
                base = snap.to(local.device, non_blocking=True)
                lidx, lval = shard_delta_indices(local, base, 0)
            else:
                # 复制参数由其他 rank 拥有，发送空 delta 维持锁步
                lidx = torch.empty(0, dtype=torch.int64, device=local.device)
                lval = torch.empty(0, dtype=local.dtype, device=local.device)
            snap.copy_(local, non_blocking=True)
            yield (*entry_fn(name, spec, place, lidx, lval), pg)


    def prime_delta_snapshots(gen, snaps: dict) -> None:
        '''seed 同步后立即将各 rank 当前 shard pin 到 CPU，
        作为后续 diff 的基线。'''
        from verl.utils.device import is_cuda_available

        for name, local, _spec in gen:
            local = local.detach().contiguous().view(-1)
            snap = snaps.get(name)
            if snap is None or snap.numel() != local.numel():
                snap = torch.empty_like(local, device='cpu', pin_memory=is_cuda_available)
                snaps[name] = snap
            snap.copy_(local, non_blocking=True)
```

verl/workers/rollout/sglang_rollout/delta_loader.py

优化 masked_copy_ 使用 torch.where 避免逐参数 host sync，提升细粒度 flush 性能

```
    # verl/workers/rollout/sglang_rollout/delta_loader.py - 优化的 masked copy

    @contextmanager
    def _masked_copy() -> Iterator[None]:
        '''临时替换 Tensor.copy_ 为无同步的带掩码版本。
        对于每个浮点参数，使用 torch.where 将 NaN 位置替换为自身，
        避免原本的 bool(mask.all()) + 布尔索引导致的 host sync。'''
        orig_copy = torch.Tensor.copy_

        def masked_copy_(self: torch.Tensor, src: torch.Tensor, *args, **kwargs) -> torch.Tensor:
            # 如果 src 是浮点且形状匹配，用 torch.where 实现带掩码拷贝
            if isinstance(src, torch.Tensor) and src.is_floating_point() and self.shape == src.shape:
                cast = src.to(self.dtype)
                # torch.where 保持所有操作在 CUDA stream 上，无同步点
                return orig_copy(self, torch.where(torch.isnan(cast), self, cast))
            return orig_copy(self, src, *args, **kwargs)

        torch.Tensor.copy_ = masked_copy_
        try:
```

```
        yield
    finally:
        torch.Tensor.copy_ = orig_copy
```

评论区精华

该 PR 是 #7085 review 要求拆分的结果，并由 #6974 review 要求支持 Shard(1) 块放置。审核者 wuxibin89 审批通过，无实质讨论。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 兼容性风险：新 delta 引擎契约改变，旧版 delta_sharded 可能无法直接升级，但 seed 路径兼容全量导出，且全量导出路径不变。
  - 功能风险：veomni delta_sharded 被标记为 NotImplementedError，使用 veomni+delta_sharded 的用户会得到错误提醒，需等待 #7085。
  - 性能风险：新路径在稀疏场景下性能更优，但引入的 _GatherQueue 和子轮次逻辑可能增加延迟窗口，但测试显示吞吐量提升。
  - 回归风险：核心路径变更大，但等价性测试覆盖了 400 步 GRPO 比较，零接收端校验和失败。
- 影响：
  - 用户：使用 FSDP + delta_sharded 的用户将获得明显的性能提升（1.3-2.3x），且无需修改代码（seed 路径自动处理 resume）。
  - 系统：新的导出契约使后端职责清晰，便于将来添加新后端（如 veomni）。
  - 团队：veomni 支持将在后续 PR #7085 完成，当前团队需注意 veomni 用户可能暂时无法使用 delta_sharded。
  - 风险标记：核心路径变更，缺乏 veomni 支持，契约变更，大量新代码

关联脉络

- PR #7085 veomni delta sharded consumer (follow-up): 本 PR 是第一部分的拆分，veomni+EP 支持将在 #7085 中完成
- PR #7060 delta weight sync roadmap: 本 PR 是 #7060 路线图的第一项
- PR #6974 sharded delta weight sync initial PR: 之前的 Shard(0) 分片 delta，本 PR 扩展了其块放置支持