PR #6974 完整报告

verl-project/verl

[ckpt,rollout] feat: sharded delta weight sync over NCCL for disaggregated rollout

合并时间：2026-07-16 12:57

原文链接：http://prhub.com.cn/verl-project/verl/pull/6974

执行摘要

- 一句话：新增分片 delta 权重同步引擎，提升分离式训练速度
- 推荐动作：该 PR 值得精读，特别是 ShardSpec 的声明式设计和 gather_v_batched_to_rank0 的批量收集优化。与 SGLang 的集成方式也是一个很好的插件式接口案例。

功能与动机

在分离式异步训练中，每步只有约 1-3% 的参数字节发生变化，因此无需每次都广播完整权重。delta_sharded 引擎通过分片本地 diff 和稀疏收集，大幅降低通信开销，同时保持位精确性。

实现拆解

1. 定义分片导出合约（verl/workers/engine/spec.py）：创建 ShardSpec 数据类，包含 full_shape、DeviceMesh 和 Placement，通过 derive_placement 计算当前 rank 的全局偏移和贡献属性。在 BaseEngine 上声明 get_per_tensor_param_shard 抽象方法。
2. 实现 FSDP 分片导出（verl/workers/engine/fsdp/transformer_impl.py）：在 FSDP 引擎中实现 get_per_tensor_param_shard，为每个本地参数生成 (name, local_shard, ShardSpec)。分片仅在首次同步时从 GPU 转移到 CPU 快照，后续仅对比快照。
3. 稀疏 diff 与批量收集（verl/checkpoint_engine/delta_sync/sparse_gather.py）：shard_delta_indices 对本地分片进行字节级对比，返回全局位置和改变的值。gather_v_batched_to_rank0 将多个参数的 delta 拼接后通过单轮集体通信收集到 rank 0，避免多次 collectives。
4. 编码与线路格式（verl/checkpoint_engine/delta_sync/encode.py）：定义 DeltaParam（参数清单）和 DeltaFlush（一个 flush 的 positions/values/checksum）数据结构。checksum 使用 torch.hash_tensor 防线路损坏。
5. 流式发送与接收（verl/checkpoint_engine/delta_checkpoint_engine.py）：DeltaShardedCheckpointEngine 继承 NCCLCheckpointEngine，发送端 send_weights 中逐 bucket 组装并发布 flush（首个为全量 dense flush），接收端 receive_weights 作为生成器逐 flush 产出稀疏 payload。广播通过 ray.util.collective 的 NCCL 进行，并用 cupy 缓冲区避免 use-after-free。
6. SGLang 原地应用（verl/workers/rollout/sglang_rollout/delta_loader.py）：通过 SGLang 的 --custom-weight-loader 钩子注册 apply_delta 函数，该函数解析 DeltaFlush

中的 manifest、positions 和 values，分批进行 masked copy 原地修改模型权重，峰值内存仅为单个 decode 块。

7. 集成与配置：SGLang ServerAdapter 通过 wire_format 参数分发 delta flush，vLLM 等其他后端若使用 delta 引擎会主动抛出 NotImplementedError。V1 trainer 中加入同步指标记录（changed ratio、payload 大小等）。
8. 测试与文档：新增 CPU 单元测试验证位精确性（test_sglang_loader.py、test_sharded_delta.py）和多 GPU 分布式收集测试（test_sharded_delta_gather.py）。新增 docs/advance/delta_weight_sync.md 设计文档。

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 _prodshape, DeltaShardedCheckpointEngine, prepare, _publish_flush）：核心引擎实现：继承 NCCLCheckpointEngine，实现流式发送（send_weights）和接收（receive_weights），处理 dense 首同步和稀疏后续同步。
- verl/checkpoint_engine/delta_sync/sparse_gather.py（模块 稀疏收集；类别 source；类型 core-logic；符号 shard_delta_indices, gather_v_batched_to_rank0, gather_dense_to_rank0, gather_v_grouped_to_rank0）：稀疏收集核心：提供 shard_delta_indices 进行本地 byte-diff，gather_v_batched_to_rank0 实现批量稀疏收集，大幅减少集体通信次数。
- verl/workers/engine/spec.py（模块 分片规范；类别 source；类型 data-contract；符号 ShardSpec, from_param, _prod, derive_placement）：ShardSpec 合约定义：声明式描述参数分片信息，被 delta 引擎和未来 nccl-m2n 引擎共用。derive_placement 派生出偏移和 gather group。
- verl/checkpoint_engine/delta_sync/encode.py（模块 线路编码；类别 source；类型 data-contract；符号 DeltaParam, checksum, DeltaFlush, nnz）：线路编码格式：定义 DeltaParam、DeltaFlush 和 checksum，是发送和接收的共同约定。
- verl/workers/rollout/sglang_rollout/delta_loader.py（模块 SGLang 加载器；类别 source；类型 core-logic；符号 apply_delta, _apply_dense, _decode_one, _masked_copy）：SGLang 消费端：通过 custom-weight-loader 钩子实现原地 delta 应用，无需修改 SGLang 源码。
- tests/checkpoint_engine/test_sglang_loader.py（模块 测试覆盖；类别 test；类型 test-coverage；符号 _FakeModel, init, load_weights, _make_named）：CPU 单元测试：验证 delta loader 的位精确性和原位应用不破坏未变化位置。
- verl/workers/engine/fsdp/transformer_impl.py（模块 FSDP 引擎；类别 source；类型 core-logic；符号 get_per_tensor_param_shard, _gen）：FSDP 引擎分片导出实现：实现 get_per_tensor_param_shard，为 delta 引擎提供本地分片和 ShardSpec。
- tests/checkpoint_engine/test_sharded_delta.py（模块 测试覆盖；类别 test；类型 test-coverage；符号 test_shard_delta_indices_matches_bytewise_diff, test_shard_delta_indices_no_change_is_empty, test_derive_placement_unsharded, test_spec_to_hf_pure_permutation）：CPU 单元测试：验证 shard_delta_indices 与全局 diff 的一致性。

- tests/special_distributed/test_sharded_delta_gather.py（模块 测试覆盖；类别 test；类型 test-coverage；符号 _run_case, main）：分布式多 GPU 测试：验证稀疏收集在多 rank 环境下的正确性。

关键符号：shard_delta_indices, gather_v_batched_to_rank0, gather_dense_to_rank0, gather_shards_to_rank0, DeltaShardedCheckpointEngine.prepare, DeltaShardedCheckpointEngine.receive_weights, DeltaShardedCheckpointEngine.publish_flush, DeltaShardedCheckpointEngine._publish_dense_flush, derive_placement, ShardSpec.from_param, checksum, DeltaFlush, apply_delta, masked_copy

关键源码片段

verl/checkpoint_engine/delta_sync/sparse_gather.py

稀疏收集核心：提供 shard_delta_indices 进行本地 byte-diff，gather_v_batched_to_rank0 实现批量稀疏收集，大幅减少集体通信次数。

```
# 对本地分片进行字节级 diff，返回全局位置和改变的值
def shard_delta_indices(
    local_new: torch.Tensor,
    local_snap: torch.Tensor,
    offset: int,
) -> tuple[torch.Tensor, torch.Tensor]:
    # 获取元素大小，选择对应整数类型
    es = local_new.element_size()
    int_dtype = _DTYPE_INT.get(es)
    if int_dtype is None:
        raise ValueError(f"unsupported element size {es}")
    # 转为整数视图进行字节级比较
    mask = local_new.view(int_dtype) != local_snap.view(int_dtype)
    local_idx = mask.nonzero(as_tuple=False).view(-1)
    values = local_new[local_idx]
    # 将本地索引偏移到全局参数中的位置
    global_idx = local_idx.to(torch.int64) + offset
    return global_idx, values


# 批量稀疏收集：将 K 个参数的 delta 拼接后通过一次 all_gather 和两次 gather 完成
def gather_v_batched_to_rank0(
    idx_concat: torch.Tensor,
    val_concat: torch.Tensor,
    counts: torch.Tensor,
    group=None,
    grouped: bool = False,
) -> list | None:
    """Variable-length sparse gather, batched: one collective round for K parameters."""
    rank = dist.get_rank(group)
    world = dist.get_world_size(group)
```

```
    dst = dist.get_global_rank(group, 0) if group is not None else 0
    dev = idx_concat.device
    k = int(counts.numel())

    # 交换各 rank 的参数长度矩阵：单次 all_gather
    counts_all = [torch.zeros_like(counts) for _ in range(world)]
    dist.all_gather(counts_all, counts.to(dev), group=group)
    # 单次 D2H 传输，减少 host 同步
    counts_cpu = torch.stack(counts_all).cpu().tolist()
    totals = [sum(c) for c in counts_cpu]
    max_n = max(totals) if totals else 0
    if max_n == 0:
        if rank != 0:
            return None
        # 返回空结构
        ...
        return [(empty_i, empty_v) for _ in range(k)]

    # 填充到统一长度后通过 gather 收集到 rank 0
    idx_pad = torch.zeros(max_n, dtype=idx_concat.dtype, device=dev)
    val_pad = torch.zeros(max_n, dtype=val_concat.dtype, device=dev)
    n = int(idx_concat.numel())
    idx_pad[:n] = idx_concat
    val_pad[:n] = val_concat

    idx_list = [torch.zeros(max_n, dtype=idx_pad.dtype, device=dev) for _ in range(world)] if rank
    == 0 else None
    val_list = [torch.zeros(max_n, dtype=val_concat.dtype, device=dev) for _ in range(world)] if
    rank == 0 else None
    dist.gather(idx_pad, idx_list, dst=dst, group=group)
    dist.gather(val_pad, val_list, dst=dst, group=group)
    if rank != 0:
        return None

    # rank 0 根据 counts 拆分各 rank 各参数的部分
    out = []
    for i in range(k):
        # 收集所有 rank 对该参数的 idx/val
        ...
    return out
```

verl/workers/engine/spec.py

ShardSpec 合约定义：声明式描述参数分片信息，被 delta 引擎和未来 nccl-m2n 引擎共用。derive_placement 派生出偏移和 gather group。

```
@dataclass
class ShardSpec:
    """声明式分片描述符：描述参数在 mesh 上的分布方式。"""
```

```
    # 完整参数形状
    full_shape: tuple
    # DeviceMesh，None 表示未分片
    mesh: Optional[object] = None
    # 每个 mesh 维度对应的 Placement（如 Shard(0)）
    placements: Optional[tuple] = None
    # 预留：Megatron 等需要从 gather 后的 shards 转换为 HF 格式的纯置换函数
    to_hf: Optional[Callable[[list[torch.Tensor]], list[tuple[str, torch.Tensor]]]] = None

    @classmethod
    def from_param(cls, param: torch.Tensor) -> ShardSpec:
        if isinstance(param, DTensor):
            # 从 DTensor 直接导出 DeviceMesh 和 Placements
            return cls(full_shape=tuple(param.shape), mesh=param.device_mesh, placements=
            tuple(param.placements))
        # 非 DTensor（如 replicated 参数）视为完整参数
        return cls(full_shape=tuple(param.shape))


def derive_placement(spec: ShardSpec):
    """根据 ShardSpec 推断本 rank 的贡献属性：
    返回 (flat_offset, contributes, gather_group)。"""
    if spec.mesh is None:
        # 未分片：只有 rank 0 贡献，对应整个参数
        return 0, (dist.get_rank() == 0 if dist.is_initialized() else True), None

    placements = spec.placements
    shard_dims = [d for d, p in enumerate(placements) if p.is_shard()]
    for d in shard_dims:
        # 仅支持 Shard(0)，其他维度（如 Shard(1)）暂未实现
        if placements[d].dim != 0:
            raise NotImplementedError(
                f"sharded delta only supports Shard(0) (FSDP2 default); got placements=
                {placements}"
            )
    # ... 计算 flat_offset 和 gather_group
    # 利用 torch.distributed 的 compute_local_shape_and_global_offset
    _, global_offset = compute_local_shape_and_global_offset(spec.full_shape, spec.mesh,
    list(placements))
    inner = _prod(spec.full_shape[1:])
    offset = int(global_offset[0]) * inner
    group = spec.mesh.get_group(mesh_dim=shard_dims[0])
    return offset, contributes, group
```

verl/checkpoint_engine/delta_sync/encode.py

线路编码格式：定义 DeltaParam、DeltaFlush 和 checksum，是发送和接收的共同约定。

```
@dataclass
class DeltaParam:
```

```
    """单个参数在一个 bucket 中的 delta 描述。"""
    name: str
    dtype: str # 参数 dtype 的字符串表示
    shape: list[int] # 参数完整形状
    pos_start: int # 在 positions 字节块中的起始偏移
    pos_end: int # 结束偏移
    pos_width: int # 位置编码宽度（2 或 4 字节）
    val_start: int # 在 values 张量中的元素起始索引
    val_end: int # 元素结束索引


def checksum(positions: torch.Tensor, values: torch.Tensor) -> int:
    """线路完整性校验：发送前和接收后分别计算并比对。"""
    # 使用 torch.hash_tensor 计算哈希， XOR 混合后返回 int
    p = int(torch.hash_tensor(positions).item()) if positions.numel() else 0
    v = int(torch.hash_tensor(values).item()) if values.numel() else 0
    return p ^ (v << 1)


@dataclass
class DeltaFlush:
    """一个可发送的 delta flush：包含位置字节块、值张量和参数清单。"""
    encoding: DeltaEncodingName # 编码方式，当前仅支持 "indices"
    params: list[DeltaParam]
    positions_cpu: torch.Tensor # uint8 位置字节块
    values_gpu: torch.Tensor # 改变的值张量
    checksum: int # 发送前计算的校验和

    @property
    def nnz(self) -> int:
        return self.values_gpu.numel()

    @property
    def wire_bytes(self) -> int:
        return self.positions_cpu.numel() + self.values_gpu.numel() * self.values_gpu.element_
        size()
```

评论区精华

- 移除 full-gather 变体：wuxibin89 提问是否还需要 all-gather 变体 delta，鉴于 delta_sharded 始终更优，最终在 commit 7c192394 中移除，只保留 delta_sharded。
- 抽象分层与解耦：wuxibin89 指出 DeltaCheckpointEngine 不应绑定 SGLang，应提供抽象接口。最终通过 wire_format 和 ServerAdapter.update_weights 分发 delta flush，SGLang 特定逻辑仅在 sglang_rollout 包中实现。
- ShardSpec 形式化：wuxibin89 建议使用 DeviceMesh 和 Placement 形式化 ShardSpec，以便 DTensor 后端和 Megatron 后端统一使用。作者在 commit bef12972 中重写 ShardSpec，利用 compute_local_shape_and_global_offset 派生出所有必要信息。

- 性能优化：减少 host 同步：gemini-code-assist 指出 .item() 循环导致大量 host-device 同步，建议使用单次 .cpu().tolist() 传输。作者在 commit 0398b0ef 中采纳。
- 组织转移：wuxibin89 建议将 delta_loader 移到 verl/workers/rollout/sglang_rollout，将 spec.py 移到 verl/workers/engine，将稀疏收集核心与 FSDP 特定代码分离。作者均一一响应。
  - 移除 full-gather delta 变体 (design): 在 commit 7c192394 中移除 delta 后端，仅保留 delta_sharded。
  - 抽象 SGLang 解耦 (design): 通过 wire_format 和 ServerAdapter.update_weights 分发 delta flush；SGLang 特定逻辑仅在 sglang_rollout 包中。分离后非 SGLang 后端在初始化时主动抛出 NotImplementedError。
  - ShardSpec 形式化与移动 (design): 在 commit bef12972 中重写 ShardSpec，删除 closure 字段，利用 compute_local_shape_and_global_offset 派生偏移。移至 verl/workers/engine/spec.py。
  - 性能优化：减少 host 同步 (performance): 在 commit 0398b0ef 中改为 stacked .cpu().tolist() 单次 D2H 传输。
  - 文件组织：delta_loader 和 spec 迁移 (style): 作者在后续 commits 中依次移动，最终结构清晰。
  - 支持 Shard(1) 分片 (design): 当前引擎仅支持 Shard(0)（FSDP2 默认），已显式 raise NotImplementedError，计划未来支持。

风险与影响

- 风险：
  - 仅 SGLang 兼容：当前 delta 引擎仅支持 SGLang rollout，vLLM 等其他后端会抛出 NotImplementedError。计划在后续 PR 中添加。
  - int32 位置溢出边界：线路上位置使用 int32，对于元素数超过 2^31 的参数（如超大规模嵌入）会静默回绕。代码中已有 fail loud 检查，但尚未全面覆盖所有路径。
  - NCCL 广播 use-after-free 风险：使用 cp.asarray 视图时若 torch 释放内存可能导致竞态。代码已通过 cupy 副本 staging 解决，但依赖细节易被后续修改破坏。
  - 分片维度仅支持 Shard(0)：derive_placement 仅支持 Shard(0)，对于 veomni 等需要 EP+FSDP 的专家分片（Shard(1)）无法直接使用。
  - 跨节点大规模测试覆盖不足：多 GPU 测试仅覆盖单机 8 GPU 场景，跨节点和大规模场景未持续集成。
  - 配置兼容性提醒：之前使用 delta 后端的配置需要改为 delta_sharded，且需设置 custom_weight_loader。
- 影响：
  - 用户：分离式训练用户可通过设置 checkpoint_engine=delta_sharded 获得显著性能提升（2-3x），需要确保 rollout 为 SGLang 并配置 custom_weight_loader。
  - 系统：新引擎增加了代码库的复杂度，但通过 ShardSpec 设计保持了可扩展性，未来可支持更多后端。

- 团队：维护者需同时支持新旧两种同步方案，但 delta_sharded 有望成为默认推荐。
- 风险标记：仅 SGLang 兼容，int32 位置溢出边界，NCCL 广播竞态，分片维度仅 Shard(0)，跨节点测试不足，配置迁移注意

关联脉络

- PR #7014 [fsdp] fix: sync merged LoRA weights before context exit: 同为 FSDP 权重同步相关修复，涉及同一文件 verl/workers/engine/fsdp/transformer_impl.py 的修改，两者都处理分片参数导出。
- PR #7061 [ckpt] feat: add save_lora_only checkpoint support: 同为 checkpoint 引擎功能扩展，虽功能不同但涉及 checkpoint 管理器的抽象与注册，可视为同一演进方向。