PR #1313 完整报告

radixark/miles

RDT weight sync: GPU->GPU zero copy transfer through SGLang Ray actor backend

合并时间：2026-08-23 08:00

原文链接：http://prhub.com.cn/radixark/miles/pull/1313

执行摘要

- 一句话：RDT/NIXL 零拷贝权重同步：GPU 直传较 NCCL 提速 2 倍
- 推荐动作：值得精读。这不仅是功能新增，更是一份「零拷贝跨进程 / 跨节点 GPU 传输」的工程教材：VMM expandable_segments 与 CUDA-IPC 导出的冲突、NIXL 注册生命周期与远端缓存失效的关联、EFA 小 MR 走 host bounce pool 的假阳性探测、以及 Ray concurrency group 导致的主线程外执行语义，每一个都是只有实测才能发现的坑。建议重点研读 update_weight_from_rdt.py 的桶分配 / 钉扎 / 分轮逻辑、actor_factory.py 的 max_concurrency 分析（配合 guapisolo 对 Ray 2.56 的源码追踪评论），以及 sglang#34621 的 actor 命名稳定化设计。

功能与动机

PR body 明确指出设计动机："RDT weight sync shares P2P and nccl distributed weight sync bucketed all-gather + HF conversion pipeline, moving the bucket payload over NIXL (ray.put(_tensor_transport="nixl") + RDMA pull) instead of holding a full GPU replica of the rollout model. We have measured faster weight sync speed across models than P2P RDMA and NCCL." 核心诉求有三：一是消除 NCCL broadcast 的串行 lock-step 瓶颈；二是避免 Mooncake RDMA P2P 在每个 trainer rank 上保留完整 CPU 模型副本；三是实现 per-rank 零拷贝 GPU→GPU 拉取，且 2→4 节点扩展零额外开销，为更大模型的多节点 rollout 扩展扫清路径。

实现拆解

在 miles/utils/arguments.py 中把 --update-weight-transfer-mode 的 choices 扩展为 broadcast / p2p / disk-delta / rdt，并在 miles_validate_args 中增加 RDT 专属断言：仅支持 --train-backend megatron、拒绝 Shared Actor/Critic PPO（RDT 每个 trainer rank 独占整张 GPU，PPO 的 critic 复用 actor bundle 必然无法调度）、与 p2p 共用 --colocate / PD 分离 / LoRA 的不兼容检查。miles/backends/megatron_utils/actor.py 的 init 按 update_weight_transfer_mode == "rdt" 优先选择新类 UpdateWeightFromRDT。

新增miles/backends/megatron_utils/update_weight/update_weight_from_rdt.py（+430行）。UpdateWeightFromRDT 继承 DistBucketedWeightUpdateMixin，完全复用其 bucketed TP/EP all-gather 与 HF 转换流程；差异化部分在于：_EngineRankBucket 用固定大小 GPU staging bucket 替代完整 rollout 模型副本，stage() 按 _staging_span 对齐算法把每个 param 的 .data 重指向桶内连续 view；connect_rollout_engines 按 engine rank 分组

target，为每个 rank 建立本地 GPU replica（dummy load 后释放存储、只保留 weight_loader 元数据）并分配桶，期间完成两个关键修复——临时关闭 expandable_segments 再分配桶（VMM 段无法导出 CUDA-IPC handle，否则 NIXL 退化为 TCP 软件模拟约 0.3 GB/s）以及 register_nixl_memory 进程级钉扎（避免每次 flush 重注册导致远端重握手）；桶大小取 max(配置值, 最大参数 nbytes) 动态扩容，超限 flush 拆分为顺序 NIXL rounds。_get_engine_scheduler_actors 缓存目标 SchedulerActor 并调用 register_weight_for_rdt 保持远端注册存活。

新增 miles/ray/rollout/sglang_server_actor.py：SGLangServerActor 在同一个 Ray job 内调用 sglang 的 launch_engine 创建全部 SchedulerActor，再在 daemon 线程运行 uvicorn，start() 直接返回 SchedulerActor handle。miles/backends/sglang_utils/sglang_engine.py 的 _init_normal 在 RDT 模式设置 use_ray=True 与 enable_rdt_weight_sync=True，经 _launch_sglang_server 启动 Actor 托管服务；多节点 engine 仅 node-0 启动（RayEngine 跨节点放置全部 rank），新增 get_scheduler_actors 暴露 handle，shutdown 走 ray.kill 清理。

miles/ray/placement_group.py 的 _create_placement_group 在 RDT 模式按节点密集度重排 bundle（先填满 trainer 节点，避免 RayEngine STRICT_PACK 死锁），allocate_train_group 把 num_gpus_per_actor 从 0.4 提为 1。miles/ray/rollout/server_group.py 复用 miles 的 rollout PG（对应 MILES_RDT_REUSE_PG=1 语义），coordinator actor 不再占 GPU，node-0 facade 拿到整套 bundle 列表。miles/ray/train/actor_factory.py 为训练 Actor 注入 RAY_EXPERIMENTAL_NOSET_CUDA_VISIBLE_DEVICES、PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True（回收碎片）与 MILES_RDT_NIXL_VALIDATE_BYTES（按真实桶大小探测 EFA GPUDirect），并把 max_concurrency 提升到 1 + rollout_num_gpus_per_engine，避免阻塞在 ray.get() 的 update_weights 饿死 NIXL 传输线程。

新增 e2e tests/e2e/megatron/test_qwen3_30B_A3B_rdt.py：8 卡 H100、actor 与 rollout 池分离（6+2）、显式 --update-weight-transfer-mode rdt，并注册 5 个 metric gate；tests/fast/utils/test_arguments.py 新增 TestRdtValidation 覆盖合法 megatron+grpo 与非法 fsdp/ppo 组合；修复 tests/fast/ray/rollout/conftest.py 与 test_actor_factory.py 缺失的 update_weight_transfer_mode 字段；requirements.txt 钉 sglang 版本以匹配 sglang-miles 分支。

关键文件：

- miles/backends/megatron_utils/update_weight/update_weight_from_rdt.py（模块 权重同步；类别 source；类型 core-logic；符号 _staging_span, _EngineRankBucket, stage, UpdateWeightFromRDT）：RDT 传输核心实现，新增 430 行：固定 GPU staging bucket、VMM/expandable 修复、NIXL 生命周期钉扎、动态扩容与分轮传输、SchedulerActor 发现与缓存。
- miles/ray/rollout/sglang_server_actor.py（模块 服务器托管；类别 source；类型 core-logic；符号 SGLangServerActor, init, start, is_alive）：新增的 SGLangServerActor 用同一 Ray job 的 Actor 托管 SGLang HTTP 服务与 SchedulerActor，替代 mp.Process，是 RDT 路径能直接拿回 SchedulerActor handle 的

关键。

- miles/backends/sglang_utils/sglang_engine.py（模块 引擎启动；类别 source；类型 core-logic；符号 _launch_sglang_server, _init_normal, get_scheduler_actors, shutdown）：RDT 模式下 SGLangEngine 从 mp.Process 切到同 job Actor 托管：_init_normal 设置 use_ray/enable_rdt_weight_sync、多节点仅 node-0 启动、shutdown 兼容 ray.kill，并暴露 get_scheduler_actors。
- miles/ray/placement_group.py（模块 资源分配；类别 source；类型 core-logic；符号 _create_placement_group, create_placement_groups, allocate_train_group）：RDT 模式下按节点密集度重排 placement group bundle，避免 RayEngine STRICT_PACK 死锁；同时把每个训练 actor 的 GPU 占用从 0.4 提为 1，是资源语义的关键变更。
- miles/ray/train/actor_factory.py（模块 角色工厂；类别 source；类型 core-logic；符号 allocate_gpus_for_actor）：为 RDT 训练 Actor 注入关键环境变量（保留 Ray GPU 掩码、expandable_segments、NIXL 真实大小探测）并提升 max_concurrency，防止阻塞的 update_weights 饿死 NIXL 服务线程。
- miles/ray/rollout/server_group.py（模块 引擎组管理；类别 source；类型 core-logic；符号 start_engines）：RDT 模式复用 miles 已预留的 rollout placement group：coordinator actor 不再占 GPU，node-0 facade 拿到整套跨节点 bundle 列表传给 SGLangEngine。
- miles/utils/arguments.py（模块 参数校验；类别 source；类型 core-logic；符号 add_rollout_arguments, miles_validate_args）：transfer mode 新增 rdt 选项并接入校验：仅 megatron、拒绝 shared PPO/colocate/PD/LoRA，是防止用户误用导致训练静默失败的第一道防线。
- miles/backends/megatron_utils/actor.py（模块 依赖装配；类别 source；类型 dependency-wiring；符号 init）：Megatron actor 初始化时按 transfer mode 选择 UpdateWeightFromRDT，是 RDT 路径接入训练前向 / 后向的依赖装配点。
- tests/e2e/megatron/test_qwen3_30B_A3B_rdt.py（模块 端到端测试；类别 test；类型 test-coverage；符号 prepare, execute）：RDT 的 e2e CI 验证：8 卡 H100、actor/rollout 池分离（6+2）、显式 rdt 模式，并注册 grad_norm/ppo_kl 等 5 个 metric gate。
- tests/fast/utils/test_arguments.py（模块 参数校验；类别 test；类型 test-coverage；符号 TestRdtValidation, _validate, test_accepts_megatron_without_critic, test_rejects_unsupported_configuration）：新增 TestRdtValidation 覆盖合法（megatron+grpo）与非法（fsdp、shared PPO）组合，防止校验回归。
- miles/backends/megatron_utils/update_weight/update_weight_from_distributed/p2p_transfer_utils.py（模块 依赖装配；类别 source；类型 dependency-wiring）：RDT 复用了 P2P 的 RemoteTransferPlan 与 create_server_args_from_dict，此文件的小改动是共享依赖装配的一部分。
- miles/backends/training_utils/ci_utils.py（模块 CI 工具；类别 source；类型 core-logic）：CI 工具增加 4 行控制流调整，配合 RDT e2e 的 metric gate 注册流程。
- tests/fast/ray/rollout/conftest.py（模块 测试配置；类别 test；类型 test-coverage）：修复 make_args() 缺失 update_weight_transfer_mode 字段导致的 AttributeError，保证既

有非 RDT 快测仍走原路径。

- tests/fast/ray/test_actor_factory.py（模块 测试配置；类别 test；类型 test-coverage）：同步修复 SimpleNamespace 缺 update_weight_transfer_mode 字段，保证 actor_factory 快测可运行。
- requirements.txt（模块 依赖配置；类别 docs；类型 configuration）：钉 sglang 版本以匹配 sglang-miles 分支，是跨仓依赖能过 CI 的最终保障。

关键符号：_staging_span, _EngineRankBucket.stage, UpdateWeightFromRDT.connect_rollout_engines, UpdateWeightFromRDT.create_gpu_replica, UpdateWeightFromRDT._get_engine_scheduler_actors, UpdateWeightFromRDT.is_rollout_engines_fresh, UpdateWeightFromRDT.mark_engine_connection_stale, SGLangServerActor.start, SGLangServerActor.is_alive, _launch_sglang_server, SGLangEngine._init_normal, SGLangEngine.get_scheduler_actors, SGLangEngine.shutdown, _create_placement_group, allocate_gpus_for_actor, ServerGroup.start_engines

关键源码片段

miles/backends/megatron_utils/update_weight/update_weight_from_rdt.py

RDT 传输核心实现，新增 430 行：固定 GPU staging bucket、VMM/expandable 修复、NIXL 生命周期钉扎、动态扩容与分轮传输、SchedulerActor 发现与缓存。

```
def _staging_span(offset: int, dtype: torch.dtype, nbytes: int) -> tuple[int, int]:
    # 在 uint8 桶内按 dtype 对齐放置 nbytes 字节，返回（起始，结束）偏移。
    # 桶是全模型共用的连续显存，stage 时按各参数 shape 切 view，避免每次重建。
    start = (offset + dtype.itemsize - 1) // dtype.itemsize * dtype.itemsize
    return start, start + nbytes


class _EngineRankBucket:
    """单个 engine rank 的传输上下文：GPU 模型副本、固定大小显存桶、
    参数规格（shape/dtype/nbytes）以及负责拉取该分片的 SchedulerActor 列表。"""

    def __init__(
        self,
        model_replica: torch.nn.Module,
        params_dict: dict[str, torch.nn.Parameter],
        param_specs: dict[str, tuple[torch.Size, torch.dtype, int]],
        gpu_bucket: torch.Tensor,
        actors: list[ActorHandle],
    ) -> None:
        self.model_replica = model_replica
        self.params_dict = params_dict
        self.param_specs = param_specs
        self.gpu_bucket = gpu_bucket
        self.actors = actors

    def stage(self, names: list[str]) -> list[torch.Tensor]:
```

```
# 把 params_dict[name].data 重新指向桶内的连续 view：load_weights 写入的
# 显存，就是随后 ray.put(views, _tensor_transport="nixl") 导出并被
# SchedulerActor pull 到 param.data 的那块显存，全程零拷贝。
offset = 0
views: list[torch.Tensor] = []
capacity = self.gpu_bucket.numel()
for name in names:
    shape, dtype, nbytes = self.param_specs[name]
    offset, end = _staging_span(offset, dtype, nbytes)
    assert end <= capacity, (
        f"[RDT] Bucket overflow while staging '{name}': "
        f"need {end} bytes but bucket is {capacity} bytes. "
        f"Increase --update-weight-buffer-size."
    )
    view = self.gpu_bucket[offset:end].view(dtype).reshape(shape)
    self.params_dict[name].data = view
    views.append(view)
    offset = end
return views
```

```
# 关键修复（throughput fix）：expandable (VMM / cuMemCreate) 段无法导出
# CUDA-IPC handle，UCX 会静默丢弃 cuda_ipc 通道，NIXL 退化为 TCP 软件模拟，
# 带宽从 ~150 GB/s (NVLink) 掉到 ~0.3 GB/s。因此临时关掉
# expandable_segments 再分配桶。
expandable = "expandable_segments:True" in os.environ.get("PYTORCH_CUDA_ALLOC_
CONF", "")
if expandable:
    torch._C._accelerator_setAllocatorSettings("expandable_segments:False")
try:
    # 桶大小 = max( 用户配置值，该 rank 最大目标参数 )，避免超大参数
    #  （如 TP=1 时 Qwen3-30B-A3B 的 embed_tokens 约 0.58 GiB）触发断言。
    max_param_nbytes = max((spec[2] for spec in param_specs.values()), default=0)
    if max_param_nbytes > self.args.update_weight_buffer_size:
        logger.warning(
            f"[RDT] Largest destination parameter needs {max_param_nbytes} bytes, above "
            f"--update-weight-buffer-size ({self.args.update_weight_buffer_size}); "
            f"growing bucket to fit."
        )
    gpu_bucket = torch.empty(
        max(self.args.update_weight_buffer_size, max_param_nbytes),
        dtype=torch.uint8,
        device=torch.cuda.current_device(),
    )
finally:
    if expandable:
        torch._C._accelerator_setAllocatorSettings("expandable_segments:True")
```

```
# 进程生命周期内钉住注册：否则每次 ray.put 重新注册桶，ObjectRef 释放还会
# 抬高 NIXL agent 元数据版本，迫使远端引擎每次 flush 都重新握手。
```

```
        from ray.experimental import register_nixl_memory

        register_nixl_memory(gpu_bucket)
```

miles/ray/rollout/sglang_server_actor.py

新增的 SGLangServerActor 用同一 Ray job 的 Actor 托管 SGLang HTTP 服务与 SchedulerActor，替代 mp.Process，是 RDT 路径能直接拿回 SchedulerActor handle 的关键。

```
class SGLangServerActor:
    """Miles 托管的进程，承载 SGLang HTTP 服务（RDT / use_ray 模式）。

    与 SGLangEngine facade 处于同一个 Ray job：start() 通过 launch_engine
    创建 SchedulerActor，再在守护线程里跑阻塞式 uvicorn，让 RPC 调用能
    直接拿回这些 handle。杀掉该 Actor 即拆除所有 scheduler。
    """

    def __init__(self):
        self._serve_thread: threading.Thread | None = None

    def start(self, server_args: ServerArgs, bundle_indices: list[int]) -> list:
        from sglang.srt.ray.http_server import launch_engine, serve_http

        # 必须在这里设置：父 Actor 的 os.environ 修改不会传到子进程。
        envs.SGLANG_RAY_BUNDLE_INDICES.set(",".join(str(i) for i in bundle_indices))
        placement_group = ray.util.get_current_placement_group()
        assert placement_group is not None
        engine = launch_engine(server_args, placement_group=placement_group)
        _, _, _, scheduler_init_result, _ = engine
        # uvicorn 是阻塞式的，放进 daemon 线程后 start() 才能返回 SchedulerActor。
        self._serve_thread = threading.Thread(
            target=serve_http,
            args=(engine, server_args),
            daemon=True,
            name="sglang-uvicorn",
        )
        self._serve_thread.start()
        return list(getattr(scheduler_init_result, "scheduler_actors", None) or [])

    def is_alive(self) -> bool:
        return self._serve_thread is not None and self._serve_thread.is_alive()
```

miles/ray/placement_group.py

RDT 模式下按节点密集度重排 placement group bundle，避免 RayEngine STRICT_PACK 死锁；同时把每个训练 actor 的 GPU 占用从 0.4 提为 1，是资源语义的关键变更。

```
    bundle_infos = [(i, gpu_ids[i][0], gpu_ids[i][1]) for i in range(num_bundles)]
    if is_rdt:
        # 让 trainer 先占满节点（node PACK 填满），rollout bundle 才能落到仍有空闲
```

```
        # GPU 的节点：RayEngine 会 STRICT_PACK 到 engine actor 所在节点，若该节点
        # 没有未预留资源就会死锁等待。
        node_bundle_counts: dict = {}
        for _, node_identifier, _ in bundle_infos:
            node_bundle_counts[node_identifier] = node_bundle_counts.get(node_identifier, 0) + 1
        sorted_bundle_infos = sorted(
            bundle_infos,
            key=lambda info: (-node_bundle_counts[info[1]], *sort_key(info)),
        )
    else:
        sorted_bundle_infos = sorted(bundle_infos, key=sort_key)

def allocate_train_group(args, num_nodes, num_gpus_per_node, pg, role, with_ref, rollout_
manager, with_opd_teacher=False):
    # RDT 每物理 GPU 钉一个 NIXL/NCCL rank，不能像 0.4 分数预留那样与 colocated
    # rollout 共享设备；因此 rdt 模式下每个训练 actor 独占一张 GPU。
    num_gpus_per_actor = 1 if args.update_weight_transfer_mode == "rdt" else 0.4
    train_group_cls = _select_train_group_class()
    return train_group_cls(
        args=args,
        num_nodes=num_nodes,
        num_gpus_per_node=num_gpus_per_node,
        pg=pg,
        num_gpus_per_actor=num_gpus_per_actor,
        role=role,
        with_ref=with_ref,
        rollout_manager=rollout_manager,
        with_opd_teacher=with_opd_teacher,
    )
```

评论区精华

评审中最有价值的交锋集中在三处。第一处是架构层面：yueming-yuan 指出用同 job SGLangServerActor 替代 mp.Process 会破坏 miles 以进程控制引擎生命周期的契约（涉及 cleanup 与 fault tolerance）；xyuzh 以「非 RDT 路径下非零 node rank 本就返回 None 进程」回应，并补齐 shutdown 对 Actor 的 ray.kill 清理，最终保留 Actor 托管方案。第二处是资源语义：maocheng23 问 RDT 每 GPU 独占一 rank 是否与 PPO 冲突，guapisolo 确认这会在 Shared Actor/Critic PPO 下确定性死锁，并直接推送参数校验拒绝组合（commit e2f87822）。第三处是并发调优：guapisolo 对 max_concurrency 改动先提 P1 质疑，经追踪 Ray 2.56 concurrency group 实现后撤回——命名 concurrency group 会为 default 单独建 executor，FT 路径的 update_weights 已跑在主线程外，非 FT 路径必须借 max_concurrency>1 达到同样效果；同时确认 NIXL 是 one-sided 传输，目标侧并发提升由 sglang#27723 承担。另有 gemini-code-assist 提出的桶溢出风险（TP=1 时 Qwen3-30B-A3B 的 embed_tokens 约 0.58 GiB 超过默认 512 MiB buffer），由动态扩容与分轮传输闭环解决。

- pull_weights 的定义位置与跨仓依赖 (question): 确认该 PR 依赖 sglang 侧配套分支，需在 requirements 中钉版本才能稳定。

- 双 flag 并存（--use-rdt-weight-sync 与 --update-weight-transfer-mode）(design): 移除 --use-rdt-weight-sync，收敛到 --update-weight-transfer-mode rdt。
- 同 job Actor 托管是否破坏进程生命周期契约 (design): 保留 Actor 托管方案，但补齐 shutdown 分支处理。
- 固定桶在超大参数下溢出（默认 512 MiB vs embed_tokens 0.58 GiB）(correctness): 动态扩容 + 分轮传输，e2e 强制覆盖扩容路径。
- 异构 engine GPU 数导致 transfer plan 错位 (correctness): 先拒绝异构布局，per-engine plan 延迟实现。
- max_concurrency 与 NIXL 传输服务线程饿死 (performance): 保留 max_concurrency = 1 + rollout_num_gpus_per_engine 的设定。
- RDT 每 GPU 单 rank 与 Shared Actor/Critic PPO 冲突 (correctness): 参数校验阶段拒绝 rdt + PPO，并补充 TestRdtValidation 测试。
- sglang_engine.py 引入 ray 与 placement group 超出文件职责 (style): 职责收敛 + 消除硬编码。

风险与影响

- 风险：
  1. 跨仓强依赖：RDT 依赖 sglang-miles 分支（sglang#27723、#34621）与 ray>=2.55.1，PR 末尾仍需钉 sglang 版本才能过 CI；若 sglang 侧接口（pull_weights、get_scheduler_actor_name、enable_engine_info_bootstrap）不稳定，本 PR 的传输路径会整体失效。
  2. 核心传输路径为全新代码：update_weight_from_rdt.py 单文件 +430 行，含显存生命周期、对齐、分轮等精细逻辑；_shared_param_mapper 为 None 的分支与 bucket 溢出断言都曾被 review 指出为潜在崩溃点。
  3. 兼容性缺口：rdt 与 --colocate、Shared PPO、PD 分离、LoRA 均不兼容（参数校验会拒绝），异构 engine GPU 布局被显式拒绝而非支持。
  4. 资源占用变化：num_gpus_per_actor 从 0.4 提到 1，且每个映射训练端桶的 SchedulerActor 额外消耗约 520 MiB CUDA context，对显存紧张的训练配置是新的约束。
  5. 共享代码回归面：mixin._finalize_and_resume_engines 新增的 needs_post_process 门控会同时影响 p2p/broadcast 路径的 post_process_weights 调用行为。
  6. 测试覆盖：e2e 仅覆盖单节点 8 卡、TP1•CP2•PP3 的 Qwen3-30B-A3B；TP8/EP8 跨 5 节点的 GLM-4.5-Air 与跨节点 EFA 场景靠人工验证，未进入 CI。 - 影响：对用户：新增 --update-weight-transfer-mode rdt 后，RL 训练在 H100/EFA 集群可获得最快的权重同步（基准中 4/5 模型领先，最高约 2.2 倍于 NCCL），且不再需要每个 trainer rank 保留 rollout 模型的 CPU 副本，显著降低显存 / 内存压力；代价是必须配套 sglang-miles 分支与最新 Ray，并对资源布局有额外约束。对系统：权重同步从「广播 / 捎带完整副本」走向「per-rank GPU 直传」，实测 2→4 节点同步开销为零增长，为大模型多节点 rollout 扩展扫清路径。对团队：miles 与 sglang 两个仓库需同步演进，验收时须统一钉版本；该 PR 沉淀了 NIXL 注册生命周期、VMM/CUDA-IPC、EFA

GPUDirect 探测等多条可复用的工程经验，对后续 zero-copy 通道（如 disk-delta、Mooncake 演进）有直接参考价值。 - 风险标记：暂无

关联脉络

- PR #2682 fix: resume from the checkpoint step in bridge mode: 同改 miles/utils/arguments.py 的校验路径，与本次 RDT 参数断言在同一函数内叠加演进，同属 weight-sync / checkpoint 配置语义的邻近改动。
- PR #2576 fix: skip rollout construction for debug replay: 同处 rollout 生命周期与 tests/fast/ray/rollout 测试域，本次 PR 也修改了同目录 conftest.py，两者共享 rollout 启动 / 回放语义的回归风险面。