PR #2088 完整报告

THUDM/slime

Add rollout_data_transport nixl

合并时间：2026-06-16 14:05

原文链接：http://prhub.com.cn/THUDM/slime/pull/2088

执行摘要

- 一句话：新增 rollout 数据 NIXL 传输支持
- 推荐动作：值得精读的实现包括：_tensorize_rollout_data_for_training 中对多模态数据的递归处理、actor.py 中 non_blocking 传输的改造以及 ray.put 条件分支。建议在合并后一段时间内监控 object-store 路径的稳定性，并在后续 PR 中补齐 nixl 的测试覆盖率。

功能与动机

减少大规模训练中 rollout 数据从 Ray object store 到 trainer GPU 的传输延迟，特别是在 tokens、log_probs 等大 tensor 上。NIXL 是 Ray 的高效 tensor 传输通道，可避免序列化开销。

实现拆解

1. 在 slime/utils/arguments.py 新增 --rollout-data-transport 参数，支持 object-store 和 nixl 两种模式，默认 object-store。
2. 在 slime/ray/rollout.py 新增 _ROLLOUT_DATA_TENSOR_DTYPES 字典定义各字段的 dtype，新增 _cpu_tensor 和 _tensorize_rollout_data_for_training 函数将 rollout_data 中的 tokens、loss_masks、log_probs 等字段转换为 contiguous CPU tensor；在 _split_train_data_by_dp 中按 DP 分组后先调用 _tensorize_rollout_data_for_training，然后根据配置走 ray.put(object-store) 或 ray.put(..., _tensor_transport='nixl')。
3. 在 slime/ray/placement_group.py 的 create_rollout_manager 中，当配置为 nixl 时，为 RolloutManager 设置 enable_tensor_transport=True。
4. 在 slime/ray/actor_group.py 的 _allocate_gpus_for_actor 中，当配置为 nixl 时，为 TrainRayActor 设置 enable_tensor_transport=True。
5. 在 slime/backends/megatron_utils/actor.py 的 _get_rollout_data 中，将原来用 torch.tensor 创建新张量的方式改为直接调用预制 tensor 的 .to(device, non_blocking=True)，并优化 multimodal 输入的处理，同时移除 numpy 依赖和 rollout_routed_experts 的预处理。
6. 测试配套：为 test_qwen3_30B_A3B_r3.py、test_qwen3_30B_A3B.py、test_qwen3.5_0.8B_gsm8k_short.py、test_qwen3_4B_ppo_disaggregate.py 添加 --rollout-data-transport nixl 参数以覆盖新路径，其中 test_qwen3_30B_A3B_r3.py 还调整了 --sglang-cuda-graph-max-bs 从 16 到 32（可能为适配更大 batch）。

关键文件：

- slime/ray/rollout.py（模块 rollout 模块；类别 source；类型 core-logic；符号 _cpu_tensor, _tensorize_rollout_data_for_training）：核心变更文件，新增 _cpu_tensor 和 _tensorize_rollout_data_for_training 函数，负责将 rollout 各字段转换为 CPU tensor，并根据传输方式选择 ray.put 路径。
- slime/backends/megatron_utils/actor.py（模块 训练后端；类别 source；类型 dependency-wiring；符号 _get_rollout_data）：修改 trainer 端 _get_rollout_data 方法，采用 non_blocking 传输并简化 multimodal 处理，移除 numpy 依赖和 rollout_routed_experts 转换。
- slime/ray/placement_group.py（模块 资源组；类别 source；类型 core-logic；符号 create_rollout_manager）：在创建 RolloutManager 时根据配置启用 enable_tensor_transport 选项。
- slime/ray/actor_group.py（模块 actor 组；类别 source；类型 core-logic；符号 _allocate_gpus_for_actor）：在创建 TrainRayActor 时根据配置启用 enable_tensor_transport 选项。
- slime/utils/arguments.py（模块 参数配置；类别 source；类型 configuration；符号 add_rollout_arguments）：新增 --rollout-data-transport 配置参数定义。

关键符号：_cpu_tensor, _tensorize_rollout_data_for_training, _get_rollout_data, create_rollout_manager, _allocate_gpus_for_actor

关键源码片段

slime/ray/rollout.py

核心变更文件，新增 _cpu_tensor 和 _tensorize_rollout_data_for_training 函数，负责将 rollout 各字段转换为 CPU tensor，并根据传输方式选择 ray.put 路径。

```
# 定义每个 rollout 字段的预期 dtype，None 表示不进行 tensor 化
_ROLLOUT_DATA_TENSOR_DTYPES = {
    "tokens": torch.long,
    "loss_masks": torch.int,
    "rollout_log_probs": torch.float32,
    "teacher_log_probs": torch.float32,
    "rollout_routed_experts": None, # 保持原样，下游按需处理
}

def _cpu_tensor(value, dtype: torch.dtype | None = None) -> torch.Tensor:
    """将输入转换为 contiguous CPU tensor，若输入为只读 ndarray 则先复制。"""
    if isinstance(value, np.ndarray) and not value.flags.writeable:
        value = value.copy()
    tensor = torch.as_tensor(value, dtype=dtype) if dtype is not None else torch.as_tensor(value)
    return tensor.detach().cpu().contiguous()

def _tensorize_rollout_data_for_training(rollout_data: dict[str, Any]) -> None:
    """修改 rollout_data 原地，将各字段替换为 CPU tensor 列表。
```

```
    对于 multimodal_train_inputs，递归处理内部的 tensor 或 ndarray。
    rollout_mask_sums 在 DP 拆分前已经聚合，直接转换为单个 float32 tensor。
    """
    for key, dtype in _ROLLOUT_DATA_TENSOR_DTYPES.items():
        if key in rollout_data:
            rollout_data[key] = [_cpu_tensor(value, dtype=dtype) for value in rollout_data[key]]
    if "multimodal_train_inputs" in rollout_data:
        rollout_data["multimodal_train_inputs"] = [
            {
                key: _cpu_tensor(value) if isinstance(value, (np.ndarray, torch.Tensor)) else value
                for key, value in mm_dict.items()
            } if mm_dict is not None else None
            for mm_dict in rollout_data["multimodal_train_inputs"]
        ]
    if "rollout_mask_sums" in rollout_data:
        rollout_data["rollout_mask_sums"] = _cpu_tensor(
            rollout_data["rollout_mask_sums"], dtype=torch.float32
        )

# 在 _split_train_data_by_dp 中按 DP 分组后调用
rollout_data = {"partition": partition, ...} # 构造过程略
_tensorize_rollout_data_for_training(rollout_data)
transport = getattr(self.args, "rollout_data_transport", "object-store")
if transport == "nixl":
    rollout_data_refs.append(Box(ray.put(rollout_data, _tensor_transport="nixl")))
elif transport == "object-store":
    rollout_data_refs.append(Box(ray.put(rollout_data)))
else:
    raise ValueError(f"Unsupported rollout data transport: {transport!r}")
```

slime/backends/megatron_utils/actor.py

修改 trainer 端 _get_rollout_data 方法，采用 non_blocking 传输并简化 multimodal 处理，移除 numpy 依赖和 rollout_routed_experts 转换。

```
def _get_rollout_data(self, rollout_data_ref: Box) -> RolloutBatch:
    rollout_data = process_rollout_data(self.args, rollout_data_ref, ...)
    device = torch.cuda.current_device()
    # 直接使用预制 tensor，通过 non_blocking 异步传输到 GPU
    rollout_data["tokens"] = [
        t.to(device=device, dtype=torch.long, non_blocking=True) for t in rollout_data["tokens"]
    ]
    rollout_data["loss_masks"] = [
        t.to(device=device, dtype=torch.int, non_blocking=True) for t in rollout_data["loss_masks"]
    ]
    if "rollout_mask_sums" in rollout_data:
        rollout_data["rollout_mask_sums"] = rollout_data["rollout_mask_sums"].to(
            device=device, dtype=torch.float32, non_blocking=True
        )
    if "multimodal_train_inputs" in rollout_data:
```

```
        rollout_data["multimodal_train_inputs"] = [
            {
                key: value.to(device=device, non_blocking=True) if isinstance(value, torch.Tensor)
                else value
                for key, value in mm_dict.items()
            } if mm_dict is not None else None
            for mm_dict in rollout_data["multimodal_train_inputs"]
        ]
    # ... 后续处理保持不变
    # 注意：rollout_routed_experts 不再显式转换为 tensor，保持传来形式
    return rollout_data
```

评论区精华

无 review 讨论。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 兼容性风险：默认 object-store 路径保持不变，但新增的 nixl 路径依赖 Ray 版本支持 enable_tensor_transport，若使用了不支持的 Ray 版本会导致 Actor 创建失败。
  2. 回归风险：actor.py 中移除了 rollout_routed_experts 字段的预处理（原先用 torch.from_numpy 转换为 tensor），若其他下游代码仍依赖该字段为 numpy 数组则可能中断。但 patch 中该字段在 _get_rollout_data 中原有处理被移除，而 rollout_data 中该字段在 rollout 侧 tensorize 时 dtype 设为 None 保持原样，actor 不再处理，若下游期望 tensor 则需检查。
  3. 测试覆盖：只有一部分测试启用了 nixl，其他测试仍使用默认 object-store 路径，nixl 路径的测试覆盖不足。
  4. 性能风险：tensorize 步骤增加了 CPU 开销，但预期被传输加速抵消。 - 影响：用户可通过 --rollout-data-transport nixl 启用 NIXL 传输，预期在大规模训练（如 Qwen3-30B）中减少数据传输开销，提升训练吞吐。变更向后兼容，默认行为不变。对 Ray 集群需确保版本支持 NIXL。团队需关注 rollout_routed_experts 字段的处理变更，确保下游兼容。 - 风险标记：依赖 Ray NIXL 版本，废弃 rollout_routed_experts 处理，新增路径测试不足

关联脉络

- PR #2082 Overlapping data loading and sglang initialization: 同为优化 rollout 数据传输与训练性能的 PR，两者在 slime/ray/rollout.py 和启动流程上有重叠关注。