PR #21887 完整报告

sgl-project/sglang

[Ray] Add data parallel (DP) and DP attention support to RayEngine

合并时间：2026-04-16 06:00

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/21887

执行摘要

- 一句话：为 RayEngine 添加数据并行和 DP 注意力支持，扩展多 GPU 推理能力。
- 推荐动作：建议精读 RayDataParallelController 类的实现，了解如何将 Ray actors 集成到现有数据并行框架中，并覆盖基类方法。同时关注提交历史中的调整点，如返回类型修复和安全绑定，这些是重要的设计决策和陷阱规避。

功能与动机

根据 PR body，动机是支持在 Ray 环境中使用数据并行，当 dp_size > 1 或 enable_dp_attention 设置时，之前会抛出 NotImplementedError，这限制了 RayEngine 在多 GPU 场景下的扩展性。

实现拆解

1. 新增 RayDataParallelController 类：在 python/sglang/srt/ray/data_parallel_controller.py 中定义，继承自 DataParallelController，覆盖 launch_dp_schedulers 和 launch_dp_attention_schedulers 方法，使用 Ray actors（SchedulerActor）替代多进程，处理端口绑定和 ZMQ 路由。
2. 修改 RayEngine 初始化逻辑：在 python/sglang/srt/ray/engine.py 中，移除对 dp_size > 1 的 NotImplementedError，根据 dp_size 分支处理：dp_size == 1 时沿用原有张量并行逻辑，否则调用 RayDataParallelController；添加 import threading 支持并发控制。
3. 更新测试覆盖：在 test/manual/test_ray_engine.py 中添加测试类 TestRayEngineOfflineDP2 和 TestRayEngineOfflineDPAttention，扩展 _create_engine_on_pg 函数以支持 dp_size 参数，验证 DP 和 DP-attention 功能。
4. 安全修复：在提交历史中，修复 ZMQ 套接字绑定，使用 get_zmq_socket_on_host 绑定到 rank0_node_ip，避免暴露未认证套接字（CVE-2026-3060）。
5. 代码结构调整：通过多个提交调整方法顺序和返回类型（如修复 _launch_scheduler_processes 返回 tuple[SchedulerInitResult, None]），以保持与基础引擎的一致性。

关键文件：

- python/sglang/srt/ray/data_parallel_controller.py（模块 Ray 控制器；类别 source；类型 entrypoint；符号 RayDataParallelController, init, launch_dp_schedulers, launch_dp_attention_schedulers）：新增 Ray 专用的数据并行控制器，核心实现 DP 支持

，覆盖基类方法以使用 Ray actors 替代多进程。

- python/sglang/srt/ray/engine.py（模块 Ray 引擎；类别 source；类型 core-logic；符号 wait_for_completion, _launch_dp_scheduler_processes）：修改 RayEngine 以支持 DP，移除 NotImplementedError 并集成新控制器，是功能启用的入口点。
- test/manual/test_ray_engine.py（模块 测试套件；类别 test；类型 test-coverage；符号 _create_engine_on_pg, TestRayEngineErrors, TestRayEngineOfflineDP2, test_dp_greater_than_1_raises）：添加手动测试验证 DP 和 DP-attention 功能，确保变更的正确性。

关键符号：RayDataParallelController.init, RayDataParallelController.launch_dp_schedulers, RayDataParallelController.launch_dp_attention_schedulers, RayEngine._launch_scheduler_processes

关键源码片段

python/sglang/srt/ray/data_parallel_controller.py

新增 Ray 专用的数据并行控制器，核心实现 DP 支持，覆盖基类方法以使用 Ray actors 替代多进程。

```
class RayDataParallelController(DataParallelController):
    """DataParallelController that uses Ray actors for scheduler processes.

    覆盖进程生成方法以创建 SchedulerActor Ray actors，而非 mp.Process。
    在进程中运行（非独立 mp.Process），复用父级的事件循环、分发和 ZMQ 路由。
    """

    def __init__(
        self,
        server_args: ServerArgs,
        port_args: PortArgs,
        placement_group,
        bundle_for_node: List[int],
        rank0_node_ip: str,
    ):
        # 在调用 super().__init__ 之前设置 Ray 特定属性，因为父构造函数会调用我们覆盖的 launch 方法
        self.pg = placement_group # Ray placement group，用于调度 actor
        self.bundle_for_node = bundle_for_node # 节点与 bundle 的映射
        self.rank0_node_ip = rank0_node_ip # 主节点 IP，用于安全绑定 ZMQ 套接字
        self.scheduler_actors: List = [] # 存储创建的 Ray actor 引用
        self.event_loop_refs: List = [] # 事件循环引用，用于异步控制

        # super().__init__ 会通过方法解析顺序（MRO）调用我们覆盖的方法
        # 传递 run_scheduler_process_func=None，因为不生成 mp.Process
        super().__init__(server_args, port_args, run_scheduler_process_func=None)
```

python/sglang/srt/ray/engine.py

修改 RayEngine 以支持 DP，移除 NotImplementedError 并集成新控制器，是功能启用的入口点。

```
@classmethod
def _launch_scheduler_processes(
    cls,
    server_args: ServerArgs,
    port_args: PortArgs,
    run_scheduler_process_func: Callable,
) -> tuple[SchedulerInitResult, None]:
    """启动调度器为 Ray actors。

    返回:
        (RaySchedulerInitResult, None) 元组。
        scheduler_procs 为 None，因为 Ray 使用 actors 而非 mp.Process。
    """
    pg = ray.util.get_current_placement_group()
    if pg is None:
        raise RuntimeError(
            "use_ray=True 需要 placement group，但未检测到。"
            "请将 Engine actor 调度到 placement group 上。"
        )

    nnodes = server_args.nnodes
    engine_bundle, engine_ip = _find_engine_bundle(pg, nnodes)
    bundle_for_node = [engine_bundle] + [i for i in range(nnodes) if i != engine_bundle]
    rank0_node_ip = engine_ip

    if server_args.dp_size == 1:
        # dp_size == 1 时，启动张量并行调度器 actors
        world_size = server_args.tp_size * server_args.pp_size
        gpus_per_node = world_size // nnodes
        logger.info(f"Ray 集群: {nnodes} 节点，使用 {gpus_per_node} GPU/节点，world_size={world_
        size}")
        # 后续代码创建 SchedulerActor 并返回结果
        scheduler_actors = []
        # ... 省略具体 actor 创建逻辑
        return RaySchedulerInitResult(scheduler_actors=scheduler_actors), None
    else:
        # dp_size > 1 或 enable_dp_attention 设置时，使用 RayDataParallelController
        # 控制器会处理多 DP rank 的 actor 启动和路由
        controller = RayDataParallelController(
            server_args, port_args, pg, bundle_for_node, rank0_node_ip
        )
        return controller.wait_for_completion(), None
```

评论区精华

Review 中没有具体评论，但提交历史显示多次迭代：例如修复返回类型以匹配基类期望，避免运行时 ValueError；以及绑定 ZMQ 套接字到特定 IP 以提高安全性。这表明在实现过程中关注了正确性和安全细节，但未在 review 中形成显式讨论。

- 暂无高价值评论线程

风险与影响

- 风险：技术风险包括：新控制器 RayDataParallelController 可能引入并发或通信问题，特别是在多 DP rank 下的 actor 生命周期管理；ZMQ 套接字绑定变化可能影响网络配置，需确保跨节点通信稳定；DP 支持增加系统复杂性，可能带来性能开销，需监控推理延迟。具体到文件，data_parallel_controller.py 中的端口分配和 actor 启动逻辑，以及 engine.py 中的分支处理，是潜在错误点。
- 影响：对用户影响：Ray 用户现在可以在多 GPU 上使用数据并行和 DP 注意力，提高推理吞吐量和模型扩展性。对系统影响：扩展了 RayEngine 的功能范围，增加了代码库复杂性，需要更多测试和维护。对团队影响：开发团队需熟悉新控制器设计，运维团队需关注部署时的资源分配和网络配置。
- 风险标记：新控制器引入，ZMQ 安全变更，并发通信风险

关联脉络

- PR #22898 [Ray] Auto-create placement group in RayEngine when none is detected: 同属 Ray 模块改进，扩展 RayEngine 功能，涉及 placement group 自动创建，与本 PR 的 DP 支持相辅相成。