

PR #1888 完整报告

radixark/miles

Start the rollout engines without blocking the caller

合并时间: 2026-09-04 13:17

原文链接: <http://prhub.com.cn/radixark/miles/pull/1888>

执行摘要

- 一句话: 将 rollout 引擎启动改为异步非阻塞, 提升并发启动能力。
- 推荐动作: 这个 PR 是重构异步启动流程的关键一步, 值得精读。它展示了如何将同步阻塞代码转换为异步协程, 使用 `asyncio.gather` 实现并发启动。设计决策上, 将初始化逻辑从构造函数移到显式异步方法, 有助于清晰分离关注点。但需注意 Issue 评论中提到的端口分配阻塞问题, 可能在后续 PR 中解决。

功能与动机

根据 commit 消息, `start_rollout_servers` 之前通过 `async_utils.run` 运行每个模型的启动, 导致 orchestrator 进程在嵌套事件循环上阻塞, 模型一个接一个启动。为了非阻塞地启动, 需将此函数改为协程, 并调整 InferenceController 的初始化流程。此外, 为避免并发模型共享端口分配器时的冲突, 将 PortAllocator 提升为共享实例。

实现拆解

1. 修改 `miles/ray/rollout/inference_controller.py`: 将 `InferenceController.__init__` 中的引擎启动逻辑移至新的异步 `init()` 方法。构造函数现在只初始化空服务器字典、锁等; `init()` 方法检查 `debug_train_only` 标志, 否则调用异步 `start_rollout_servers` 并注册路由器。
2. 修改 `miles/ray/rollout/rollout_server.py`: 将 `start_rollout_servers` 从同步函数改为异步协程。将 `PortAllocator` 的创建提升到函数级别, 确保所有模型共享同一个实例。启动每个模型的服务器时, 不再使用 `async_utils.run` 顺序执行, 而是使用 `asyncio.gather` 并发启动所有服务器的 `start_all_cells` 方法。
3. 修改 `miles/ray/placement_group.py`: 在 `create_rollout_components` 函数中, 创建 `InferenceController` 实例后, 立即调用 `await inference_controller.init()` 以启动引擎。
4. 更新测试文件: 适配新的异步初始化模式。移除了 `test_inference_controller.py` 中的 `_make_controller` 辅助函数, 改为直接创建 `InferenceController` 并调用 `init()`。在 `test_placement_group.py` 中, 更新了 mock 以模拟异步 `init()` 方法。

关键文件:

- `miles/ray/rollout/inference_controller.py` (模块 推理控制器; 类别 source; 类型 entrypoint; 符号 `init`): 核心控制器类, 初始化流程从同步改为异步, 是 PR 的入口变更。
- `miles/ray/rollout/rollout_server.py` (模块 启动服务器; 类别 source; 类型 core-logic; 符号 `start_rollout_servers`): 核心启动函数, 从同步改为异步, 实现并发启动。

关键符号: InferenceController.init, start_rollout_servers

关键源码片段

miles/ray/rollout/inference_controller.py

核心控制器类，初始化流程从同步改为异步，是 PR 的入口变更。

```
class InferenceController:
    def __init__(self, args, pg):
        self.pg = pg
        self.args = args
        # 初始化空服务器字典，不启动引擎，避免阻塞
        self.servers: dict[str, RolloutServer] = {}
        self.rollout_engine_lock = Lock.options(num_cpus=1, num_gpus=0).remote()
        self.rollout_id = -1
        self.eval_fleet = EvalFleet(args, srv=self.servers["eval"]) if args.eval_num_gpus > 0 else
        None

    async def init(self) -> None:
        """异步初始化：启动所有 rollout 引擎。"""
        if self.args.debug_train_only:
            return
        # 调用异步函数启动服务器，避免阻塞调用者
        self.servers = await start_rollout_servers(self.args, self.pg)
        dashboard_hooks.register_router(self.args)
        start_session_server(self.args)
```

miles/ray/rollout/rollout_server.py

核心启动函数，从同步改为异步，实现并发启动。

```
async def start_rollout_servers(args, pg) -> dict[str, "RolloutServer"]:
    """启动 rollout 服务器：每个模型一个，带各自的路由器。
    返回映射模型名 -> RolloutServer 的字典。"""
    config = _resolve_sglang_config(args)
    servers: dict[str, RolloutServer] = {}
    gpu_offset = 0
    engine_offset = 0
    # 提升 PortAllocator 为共享实例，避免并发模型端口冲突
    port_allocator = PortAllocator()

    # ... 循环创建服务器单元（省略部分逻辑） ...

    # 使用 asyncio.gather 并发启动所有服务器的 cells
    await asyncio.gather(*[srv.start_all_cells(port_allocator) for srv in servers.values()])
    args.sglang_model_routers = {name: (srv.router_ip, srv.router_port) for name, srv in servers.
    items()}
    return servers
```

评论区精华

从 Issue 评论中, guapisolo 指出: 'Non-blocking: #1889 made the node probes concurrent, but dynamic port allocation still calls blocking `ray.get()` from `async _BaseActorManager.alloc_ports()` through `PortAllocator.alloc()`; this remains present at #2176 and the checked #2649 tip. Could we make port allocation awaitable, or explicitly offload the blocking call, so the `RayWorkerManager` event loop cannot be stalled?' 这表明尽管这个 PR 解决了引擎启动的阻塞问题, 但端口分配中的阻塞调用仍然存在, 可能需要后续优化。未解决疑虑: `PortAllocator.alloc()` 中的阻塞 `ray.get()` 可能阻塞事件循环。

- 端口分配阻塞问题 (performance): 此问题未在本 PR 中解决, 可能需要在后续 PR 中处理。

风险与影响

- 风险: 主要技术风险:
 - 异步初始化变更可能引入竞态条件, 但通过共享 `PortAllocator` 和测试覆盖缓解。
 - 依赖 `asyncio` 的正确使用, 但代码使用了标准的 `asyncio.gather`, 风险较低。
 - 测试覆盖全面, 包括端到端 smoke 测试和单元测试, 降低了回归风险。
 - 对现有调用方的兼容性: 所有调用方都需要更新以调用 `init()`, 但 PR 已更新了主要调用方。
 - 影响: 对用户: 无直接影响, 因为这是内部重构。对系统: 启动速度提升, 因为引擎现在并发启动; 内存使用可能优化, 因为共享 `PortAllocator`。对团队: 代码更符合异步编程模式, 便于后续维护和扩展; 测试套件更新, 确保正确性。
- 风险标记: 异步初始化变更, 并发启动依赖, 共享资源分配

关联脉络

- PR #1843 Wire up InferenceController and RolloutExecutor and make the controller API async: 直接相关, 该 PR 使控制器 API 异步, 为本 PR 的异步初始化铺路。
- PR #1989 Add a skeleton RayWorkerManager and a wiring entry point: 引入 `RayWorkerManager`, 为工作器管理奠基, 与本 PR 的异步启动重构属于同一模块。
- PR #1987 Move PortAllocator into miles.utils.workers: 移动 `PortAllocator` 到通用模块, 影响端口分配逻辑, 与本 PR 的共享 `PortAllocator` 相关。