

PR #1841 完整报告

radixark/miles

Make the CI fault injection wait asynchronously to avoid long blocking

合并时间: 2026-09-04 12:19

原文链接: <http://prhub.com.cn/radixark/miles/pull/1841>

执行摘要

- 一句话: 将 CI 故障注入等待从同步 `sleep` 改为异步 `await`, 避免阻塞主线程。
- 推荐动作: 这是一个小规模、目标明确的重构 PR, 值得精读。其核心价值不在于改动的复杂性, 而在于它清晰展示了如何逐步、安全地将一个阻塞操作异步化。对于关注系统异步化重构或 CI 测试性能优化的工程师, 此 PR 提供了一个具体的实施范例。

功能与动机

该变更源于对代码质量和测试基础设施性能的持续优化。在 CI 测试环境中, `_try_ci_fault_injection` 方法中的同步 `time.sleep` 会阻塞事件循环, 可能影响系统响应或导致不必要的延迟。将其异步化是迈向全系统非阻塞 I/O 的一个小步骤, 也是整体异步架构重构 (PR#1837 及其子项) 的前置工作。

实现拆解

1. 核心逻辑异步化: 在 `miles/ray/rollout/rollout_manager.py` 中, 将 `prepare_rollout` 方法及其调用的 `_try_ci_fault_injection` 方法从同步函数 (`def`) 重构为异步协程 (`async def`)。这改变了方法的调用语义。
2. 阻塞等待替换: 在 `_try_ci_fault_injection` 方法内部, 将用于等待健康监控器检测故障的 `time.sleep(wait_time)` 替换为 `await asyncio.sleep(wait_time)`。这是本次变更的核心技术点, 将阻塞 I/O 操作转变为非阻塞操作。
3. 调用点适配: 在 `prepare_rollout` 方法内部, 对 `_try_ci_fault_injection` 的调用增加了 `await` 关键字, 以匹配其新的协程签名。
4. 测试用例同步更新: 在测试文件 `tests/fast/ray/rollout/real_ray/test_rollout_manager.py` 中, 所有对 `manager.prepare_rollout(...)` 的直接调用均更新为 `await manager.prepare_rollout(...)`, 以适配其异步化后的接口, 确保测试能正常运行。

关键文件:

- `miles/ray/rollout/rollout_manager.py` (模块 Rollout 管理器; 类别 source; 类型 core-logic; 符号 `prepare_rollout`, `_try_ci_fault_injection`): 核心源码变更文件, 将故障注入的同步等待改为异步等待, 并将相关方法签名异步化, 是本次重构的核心。
- `tests/fast/ray/rollout/real_ray/test_rollout_manager.py` (模块 Rollout 管理器; 类别 test; 类型 test-coverage): 测试配套文件, 更新测试用例以适配 `prepare_rollout` 方法异步化后的调用方式。

关键符号：prepare_rollout, _try_ci_fault_injection

关键源码片段

miles/ray/rollout/rollout_manager.py

核心源码变更文件，将故障注入的同步等待改为异步等待，并将相关方法签名异步化，是本次重构的核心。

miles/ray/rollout/rollout_manager.py 中的关键变更片段

变更前：同步方法

```
# def prepare_rollout(self, rollout_id):
```

```
# ...
```

```
# if self.args.ci_test and self._rollout_ft_enabled and rollout_id >= 2:
```

```
# self._try_ci_fault_injection()
```

```
# ...
```

变更后：异步方法

```
async def prepare_rollout(self, rollout_id):
```

```
    self.rollout_id = rollout_id
```

```
    self._health_monitoring_resume()
```

```
    # 在 CI 测试且启用了容错功能，且从第二个 rollout 开始注入故障
```

```
    if self.args.ci_test and self._rollout_ft_enabled and rollout_id >= 2:
```

```
        await self._try_ci_fault_injection() # 调用变为 await
```

```
    dashboard_hooks.register_engines(self.servers)
```

变更前：同步方法，内部使用 time.sleep

```
# def _try_ci_fault_injection(self):
```

```
# ...
```

```
# time.sleep(wait_time)
```

```
# ...
```

变更后：异步方法，内部使用 asyncio.sleep

```
async def _try_ci_fault_injection(self):
```

```
    """Try to inject fault during generate (when health monitor is running)."""
```

```
    if not self._ci_fault_injection_pending:
```

```
        return
```

```
    # Only inject fault once
```

```
    self._ci_fault_injection_pending = False
```

```
    if (
```

```
        self._server
```

```
        and self._server.server_groups[0].all_engines
```

```
        and self._server.server_groups[0].all_engines[0].is_allocated
```

```
    ):
```

```
        logger.info("CI Fault Injection: Simulating crash on engine 0 during generate")
```

```
        try:
```

```
            # This will cause the ray actor to exit
```

```
self._server.server_groups[0].all_engines[0].actor_handle.simulate_crash.remote()
# Wait for health monitor to detect the crash and mark engine as None
# health_check_interval + health_check_timeout + buffer
wait_time = self.args.rollout_health_check_interval + self.args.rollout_health_check_timeout + 5
logger.info(f"CI Fault Injection: Waiting {wait_time}s for health monitor to detect crash")
await asyncio.sleep(wait_time) # 核心变更：从 time.sleep 改为 await asyncio.sleep
except Exception as e:
    logger.warning(f"CI Fault Injection failed: {e}")
```

评论区精华

在本次提供的材料中，review 记录显示仅有一名 reviewer (guapisolo) 进行了批准 (APPROVED)，且无具体评论内容。因此，本次 PR 的技术决策未引发公开讨论，其修改方案被视为直接且符合预期的重构步骤。

- 暂无高价值评论线程

风险与影响

- 风险：

1. 接口兼容性风险：prepare_rollout 方法从同步变为异步，是一个 API 破坏性变更。所有调用此方法的代码都必须更新为使用 await。本次 PR 仅更新了其调用的 _try_ci_fault_injection 和测试中的调用点。虽然当前上下文未显示其他调用者，但在更大的代码库合并时，必须确认所有调用方都已适配，否则将导致运行时错误。
2. CI 测试可靠性：await asyncio.sleep(wait_time) 的等待时间计算逻辑未变，其可靠性依赖于 args.rollout_health_check_interval 和 args.rollout_health_check_timeout 配置的准确性。异步化本身未改变超时逻辑，风险可控。
3. 生产环境影响：_try_ci_fault_injection 方法仅在 self.args.ci_test 为真时被调用，属于 CI 测试专用路径，因此对正常的生产训练流程没有直接影响。- 影响：影响范围：影响 CI 测试流程中执行故障注入测试的路径；为 RolloutManager 类的 API 契约引入了一个异步方法。影响程度：对于 CI 流程，将阻塞等待改为非阻塞等待，可以减少测试执行时间并避免潜在的死锁或性能降级问题。对于代码库架构，这是推动核心组件 (RolloutManager) 异步化的小步改进，有助于后续实现更灵活、非阻塞的控制器 - 执行器交互模式（如 PR#1842、#1843、#1844 所规划）。

- 风险标记：API 破坏性变更（同步→异步），CI 路径变更

关联脉络

- PR #1842 Split RolloutManager into InferenceController and RolloutExecutor: 属于同一重构链 (#1837)，为将 RolloutManager 拆分为异步控制器 (InferenceController) 和同步执行器 (RolloutExecutor) 做准备。本 PR 将 prepare_rollout 异步化是该拆分的一个必要前置步骤。

- PR #1843 Wire up InferenceController and RolloutExecutor and make the controller API async: 直接接续本 PR 的异步化工作，将 InferenceController 的 API 全面异步化，并完成控制器与执行器的接线。本 PR 的 prepare_rollout 异步化是其中的一个具体实现点。
- PR #1844 Rename RolloutExecutor.generate to get for future async and multi trainer scenarios: 同一重构链中的后续优化，将 RolloutExecutor 的 generate 方法重命名为 get，以更准确地反映其在异步多训练器场景下的职责。