

# PR #1873 完整报告

radixark/miles

Serve HTTP from the mock engine on its allocated port

合并时间: 2026-09-04 13:06

原文链接: <http://prhub.com.cn/radixark/miles/pull/1873>

## 执行摘要

- 一句话: 重构测试 Mock 引擎, 使其在自身分配的端口上提供 HTTP 服务, 提升测试保真度。
- 推荐动作: 此 PR 是有价值的测试基础设施改进, 它遵循了“测试应尽可能接近生产环境”的最佳实践。建议精读, 特别是 MockSGLangHttpServer 中连接跟踪和端口绑定的实现细节, 以及测试夹具简化后的模式。设计决策上, 将 HTTP 服务器生命周期与 Actor 绑定是正确的方向, 但 必须关注并跟进 review 中提出的 IP 地址硬编码问题, 这是一个在后续测试扩展中必须解决的技术债。

## 功能与动机

当前的 real\_ray 测试套件通过 stub 掉地址分配器函数并预绑定一个由驱动进程管理的 HTTP 服务器池来模拟引擎。这导致测试与分配器的存在和伪造的 rank->port 映射耦合, 且测试中的 URL 没有真正的 HTTP 服务器在背后提供服务。此次重构旨在让 Mock 引擎模拟真实的 Ray Actor 行为: 在分配的端口上启动 HTTP 服务器, 并将请求日志记录在 Actor 内部。这使得测试能够验证生产代码从真实分配器推导出的 URL 是否确实可达, 并且请求是否被正确记录。

## 实现拆解

1. 核心源码改造 (MockSGLangHttpServer): 在 miles/utils/test\_utils/mock\_sglang\_http\_server.py 中, 为 MockSGLangHttpServer 的 \_\_init\_\_ 方法增加可选的 port 参数 (默认 0 表示随机端口), 使其能绑定到指定端口。同时增加了连接跟踪机制 ( \_register\_connection, \_unregister\_connection, \_drain\_connections ), 确保在 close() 时能主动断开已建立的 keep-alive 连接, 模拟引擎崩溃的场景。MockSGLangHttpServerPool 类被移除。
2. 核心源码改造 (MockSGLangEngine): 在 miles/utils/test\_utils/mock\_sglang\_engine.py 中, MockSGLangEngine 在 init() 时根据传入的 port 参数创建一个 MockSGLangHttpServer 实例, 并在 shutdown() 时关闭它。新增了 get\_http\_paths() 和 get\_http\_payloads\_of(path) 方法, 允许测试通过 Ray Actor 远程调用获取请求日志。 \_get\_current\_node\_ip\_and\_free\_port 方法改为调用真实的 get\_free\_port 函数。
3. 测试夹具简化: 在 tests/fast/ray/rollout/real\_ray/conftest.py 中, 移除了 mock\_engine\_http\_servers fixture 和 patched\_sglang\_engine 中的 \_fake\_alloc 程序。patched\_sglang\_engine fixture 现在只替换引擎类, 保留真实的地址分配器。原先需要 mock HTTP 服务器列表来验证请求的测试, 现在改为通过 ray.get(actor\_handle.get\_http\_paths.remote()) 等方式从 Actor 获取日志。

4. 测试断言迁移：多个测试文件（如 `test_inference_controller.py`, `test_server_group.py`, `test_rollout_server.py`, `test_fault_tolerance.py`）中的断言逻辑从直接查询 `MockSGLangHttpServerPool` 中的服务器对象，迁移为调用 `Actor` 的远程方法获取日志，使测试逻辑更贴近生产代码路径。
5. 新增 Mock HTTP 服务器单元测试：在 `tests/fast/utils/test_utils/` 下新增了 `test_mock_sglang_http_server.py` 和 `conftest.py`，为重构后的 `MockSGLangHttpServer` 提供了全面的单元测试，覆盖端口绑定、连接管理、请求记录和错误处理。

关键文件：

- `miles/utils/test_utils/mock_sglang_http_server.py`（模块 测试工具；类别 `source`；类型 `dependency-wiring`；符号 `init`, `_register_connection`, `_unregister_connection`, `_drain_connections`）：核心变更之一。为 Mock HTTP 服务器增加了指定端口绑定和连接跟踪能力，使其能模拟引擎崩溃时断开连接的行为，为 Mock 引擎在 `Actor` 内部提供服务奠定了基础。
- `miles/utils/test_utils/mock_sglang_engine.py`（模块 测试工具；类别 `source`；类型 `core-logic`；符号 `get_http_paths`, `get_http_payloads_of`, `init`, `init`）：核心变更之二。将 Mock 引擎从一个“哑”记录器改为一个在 `init` 时启动真实 HTTP 服务器的 `Actor`，并提供了获取请求日志的远程方法。
- `tests/fast/ray/rollout/real_ray/conftest.py`（模块 测试夹具；类别 `test`；类型 `test-coverage`；符号 `mock_engine_http_servers`, `patched_sglang_engine`, `_fake_alloc`, `patched_sglang_engine_real_allocator`）：展示了测试夹具的简化过程：移除了 Mock HTTP 服务器池和地址分配 `stub`，使测试能使用真实的地址分配流程。

关键符号：`MockSGLangHttpServer.init`, `MockSGLangHttpServer.close`, `MockSGLangHttpServer._register_connection`, `MockSGLangHttpServer._unregister_connection`, `MockSGLangHttpServer._drain_connections`, `MockSGLangEngine.init`, `MockSGLangEngine.shutdown`, `MockSGLangEngine.get_http_paths`, `MockSGLangEngine.get_http_payloads_of`, `MockSGLangEngine._get_current_node_ip_and_free_port`

## 关键源码片段

### `miles/utils/test_utils/mock_sglang_http_server.py`

核心变更之一。为 Mock HTTP 服务器增加了指定端口绑定和连接跟踪能力，使其能模拟引擎崩溃时断开连接的行为，为 Mock 引擎在 `Actor` 内部提供服务奠定了基础。

```
from __future__ import annotations

import dataclasses
import json
import logging
import socket
import threading
from http.server import BaseHTTPRequestHandler, ThreadingHTTPServer
```

```

from typing import Any

logger = logging.getLogger(__name__)

@dataclasses.dataclass(frozen=True)
class RecordedRequest:
    method: str
    path: str
    payload: dict[str, Any] | None

class MockSGLangHttpServer:
    # 核心变更: __init__ 新增 port 参数, 允许绑定到指定端口; 新增 _connections
    # 集合用于跟踪活跃连接
    def __init__(self, response_payload: dict[str, Any] | None = None, port: int = 0):
        self._response_payload = response_payload if response_payload is not None else {"mock":
        True}
        self._requests: list[RecordedRequest] = []
        self._lock = threading.Lock()
        self._connections: set[socket.socket] = set() # 跟踪所有活跃的连接 socket

        # 绑定到指定端口 (port=0 为随机端口)
        self._server = ThreadingHTTPServer(("127.0.0.1", port), self._make_handler())
        self._thread = threading.Thread(target=self._server.serve_forever, daemon=True)
        self._thread.start()

    # ... (host, port, url, requests, paths, payloads_of 属性不变) ...

    # 核心变更: close 方法现在会主动断开所有已建立的连接, 模拟引擎崩溃
    def close(self) -> None:
        self._server.shutdown()
        self._server.server_close()
        # 主动关闭所有活跃连接, 模拟引擎宕机时客户端立即感知
        for connection in self._drain_connections():
            try:
                connection.shutdown(socket.SHUT_RDWR)
            except OSError:
                logger.debug("mock sglang http server: connection was already down", exc_info=
                True)
            connection.close()
        self._thread.join(timeout=5)

    def _record(self, method: str, path: str, payload: dict[str, Any] | None) -> None:
        with self._lock:
            self._requests.append(RecordedRequest(method=method, path=path.split("?")[0],
            payload=payload))

    # 新增方法: 注册、注销和排空连接

```

```

def _register_connection(self, connection: socket.socket) -> None:
    with self._lock:
        self._connections.add(connection)

def _unregister_connection(self, connection: socket.socket) -> None:
    with self._lock:
        self._connections.discard(connection)

def _drain_connections(self) -> list[socket.socket]:
    with self._lock:
        connections = list(self._connections)
        self._connections.clear()
    return connections

def _make_handler(self) -> type[BaseHTTPRequestHandler]:
    server = self

    class _Handler(BaseHTTPRequestHandler):
        protocol_version = "HTTP/1.1"

        # 核心变更：在 setup 和 finish 中跟踪连接生命周期
        def setup(self):
            super().setup()
            server._register_connection(self.connection) # 连接建立时注册

        def finish(self):
            server._unregister_connection(self.connection) # 连接结束时注销
            super().finish()

        # ... (do_GET, do_POST, log_message, _respond 不变) ...

    return _Handler

```

### miles/utils/test\_utils/mock\_sglang\_engine.py

核心变更之二。将 Mock 引擎从一个“哑”记录器改为一个在 init 时启动真实 HTTP 服务器的 Actor，并提供了获取请求日志的远程方法。

```

# ... ( 文件头部导入 ) ...
from miles.utils.misc import get_free_port
from miles.utils.test_utils.mock_sglang_http_server import MockSGLangHttpServer

class MockSGLangEngine:
    def __init__(self, args, rank: int, ...):
        # ... ( 其他属性初始化 ) ...
        # 移除了 _port_seq 和 _lock, 改为持有 http_server 实例
        self._http_server: MockSGLangHttpServer | None = None

    # ... (set_fault, get_calls, get_init_kwargs 不变) ...

```

```

# 新增远程方法：供测试通过 Ray 调用获取请求日志
def get_http_paths(self) -> list[str]:
    return self._http_server.paths if self._http_server is not None else []

def get_http_payloads_of(self, path: str) -> list[dict | None]:
    return self._http_server.payloads_of(path) if self._http_server is not None else []

# 核心变更：init 方法现在会在分配的端口上启动 HTTP 服务器
def init(self, **kwargs):
    self._record("init", (), kwargs)
    self._maybe_fault("init")
    # 使用 init 传入的 port 参数创建 HTTP 服务器，实现“在分配端口上服务”
    self._http_server = MockSGLangHttpServer(port=kwargs["port"])
    self.initialized = True
    return None

# 核心变更：shutdown 方法现在会关闭 HTTP 服务器
def shutdown(self):
    self._record("shutdown", (), {})
    self._maybe_fault("shutdown")
    if self._http_server is not None:
        self._http_server.close()
    self.initialized = False
    return True

# ... (simulate_crash 不变) ...

# 核心变更：使用真实的 get_free_port 函数获取端口，而非自己维护序列
def _get_current_node_ip_and_free_port(self, start_port: int = 15000, consecutive: int = 1):
    self._record("_get_current_node_ip_and_free_port", (), {"start_port": start_port,
        "consecutive": consecutive})
    return ("127.0.0.1", get_free_port(start_port=start_port, consecutive=consecutive))

# ... (_record, _maybe_fault 不变) ...

```

## tests/fast/ray/rollout/real\_ray/conftest.py

展示了测试夹具的简化过程：移除了 Mock HTTP 服务器池和地址分配 stub，使测试能使用真实的地址分配流程。

```

# ... ( 文件头部导入和 placement_group_factory fixture 不变 ) ...

# 核心变更：移除了 mock_engine_http_servers fixture
# 核心变更：简化 patched_sglang_engine，不再依赖 mock_engine_http_servers 和 _fake_alloc
@pytest.fixture
def patched_sglang_engine(monkeypatch, mock_engine_class):
    """Replace SGLangEngine with the mock; the real addr allocator runs, and
    each mock engine serves HTTP on the port it is allocated, so the urls
    ServerGroup derives from the allocator actually serve requests."""
    import miles.ray.rollout.server_cell as cell_mod

```

```
monkeypatch.setattr(cell_mod, "SGLangEngine", mock_engine_class)
# 移除了对 allocate_rollout_engine_addr_and_ports_normal 的 monkeypatch

# 核心变更：移除了 patched_sglang_engine_real_allocator fixture
# 因为 patched_sglang_engine 现在已经使用真实的分配器
```

## 评论区精华

reviewer guapisolo 指出了—个关键的设计问题（P1 级别）：重构后的 `MockSGLangEngine._get_current_node_ip_and_free_port` 方法仍然硬编码返回 `127.0.0.1`，而新的 `MockSGLangHttpServer` 也只绑定到回环地址。这在测试运行于真实多节点集群或隔离的网络命名空间时（如通过 `RAY_ADDRESS` 连接），会导致驱动端解析到的引擎 URL 无法访问，引发 `ConnectionRefusedError`。reviewer 要求返回真实的节点 IP 并绑定到该地址或 `0.0.0.0`，但指出此问题在后续 PR 中未修复。

- 硬编码的 `127.0.0.1` 地址在多节点 / 隔离网络场景下的兼容性问题 (correctness): 问题被识别并标记为 P1，但在此 PR 中未修复。reviewer 在后续评论中确认该问题在后续 PR 中仍未被修复。

## 风险与影响

- 风险：
  1. 兼容性 / 正确性风险：如 review 指出，Mock 引擎返回的 `127.0.0.1` 地址在非本地单机测试场景（如多节点、容器网络）下会导致测试失败，因为驱动进程和引擎 Actor 可能处于不同的网络命名空间。当前修复不完整，存在回归风险。
  2. 测试稳定性风险：端口分配从预计算改为运行时真实探测（`get_free_port`），理论上在并发测试或资源紧张环境下可能存在端口冲突的微小概率，但通过监听随机端口（`port=0`）来缓解。
  3. 代码依赖变更：移除了 `MockSGLangHttpServerPool`，可能影响其他未在此 PR 中更新、但依赖该类的测试代码（需确认已无其他依赖）。
- 影响：
  1. 对系统的影响：无直接影响。变更完全限于测试基础设施（`miles/utils/test_utils`）和测试代码（`tests/fast`）。
  2. 对用户的影响：无直接影响。
  3. 对团队的影响：正面：提升了测试的保真度，使测试环境更接近生产环境，有助于发现因地址 / 端口处理不一致导致的潜在 bug。简化了测试夹具，减少了测试间的耦合。负面：引入了一个已知但未修复的兼容性问题（硬编码 IP），可能在后续扩展测试覆盖（如多节点测试）时造成阻碍。 - 风险标记：硬编码 IP 地址，测试环境假设

## 关联脉络

- PR #2050 Stop checking actor liveness while waiting for a tcp port: 此 PR 移除了 `wait_tcp_ready` 的 actor 存活检查参数。与本 PR 相关联，因为本 PR 的重构使测试中的 HTTP 端口就绪性与真实的 Actor 初始化绑定，移除额外的存活检查可能使测试更简洁。两

者都是对测试 / 部署基础设施的简化和规范化。

- PR #1989 Add a skeleton RayWorkerManager and a wiring entry point: 此 PR 引入了 RayWorkerManager 骨架，是统一工作器生命周期管理的开端。本 PR 对测试 Mock 引擎的改造，使其生命周期（HTTP 服务器的 start/stop）更贴近真实工作器，为后续基于 RayWorkerManager 的更复杂测试奠定了更可靠的模拟基础。
- PR #1987 Move PortAllocator into miles.utils.workers: 此 PR 将端口分配器模块化。本 PR 的核心变更之一是让 Mock 引擎使用真实的端口分配（通过 get\_free\_port），而不是测试夹具中的伪造序列。这使得测试能够验证与真实 PortAllocator 的交互，两者共同推动了端口管理逻辑的规范化。