PR #1904 完整报告

radixark/miles

Move the shell exec helpers next to their only consumers

合并时间：2026-08-09 18:45

原文链接：http://prhub.com.cn/radixark/miles/pull/1904

执行摘要

- 一句话：shell 执行辅助函数迁出 misc.py 独立成模块
- 推荐动作：作为 #1837 重构链上的纯搬迁 PR，不必精读函数细节，但值得快速了解它建立的模块边界和测试 patch 方式的收窄，便于理解后续 #1905、#1909、#1910 等更有价值的 PR。重点关注 *exec_command_multi_node* 的 Ray 调度逻辑，它会在后续多节点启动场景中继续演进。

功能与动机

1837 跟踪 issue 明确要系统性重构命令执行与启动脚本相关代码。此前 *exec_command_** 系列函数被放在通用工具模块 misc.py 中，与端口、节点等杂项工具混在一起，职责不清晰；本 PR 将函数就近移动到唯一消费者所在目录下的 *external_utils/exec_command.py*，降低 misc.py 的耦合面，也为 #1905、#1909、#1910 等后续改造提供更干净的落点。

实现拆解

1. 新增独立模块 miles/utils/*external_utils/exec_command.py*：将 *exec_command_gpu*、*exec_command_cpu*、*_exec_command*、*_exec_command_on_node*、*exec_command_multi_node* 原样搬入，并保留 re、subprocess、ray 与 NodeAffinitySchedulingStrategy 等依赖。其中 *exec_command_multi_node* 内含按节点替换 {{node_rank}} 等占位符并并行调度的核心逻辑。
2. 从 miles/utils/misc.py 删除上述函数：同时清理不再需要的 re、subprocess 等导入，misc.py 只保留 *get_current_node_ip*、*get_free_port* 等杂项工具。
3. 更新全部调用方导入：*command_utils.py*、*debug_utils/run_megatron/cli/commands/compare.py*、*debug_utils/run_megatron/cli/commands/run.py*、*examples/experimental/formal_math/single_round/kimina_wrapper.py*、*examples/geo3k_vlm/multi_turn/run_geo3k_vlm_multi_turn.py*、*scripts/run_qwen3_4b_npu.py* 等统一改为从 miles.utils.*external_utils.exec_command* 导入。

4. 调整测试配套：tests/fast/utils/command_recorder.py 不再对 misc 模块打 patch（该模块已无这些属性），只 patch command_utils；tests/e2e/short/test_run_megatron.py 同步更新导入路径。
5. 合并主干：最终提交把 main 合并回分支，解决与同期 PR 的冲突，保证分支可继续演进。

关键文件：

- miles/utils/external_utils/exec_command.py（模块 命令执行；类别 source；类型 core-logic；符号 exec_command_gpu, exec_command_cpu, _exec_command, _exec_command_on_node）：新增的独立模块，集中承载全部 shell 命令执行辅助函数，是本 PR 的核心交付物。
- miles/utils/misc.py（模块 工具库；类别 source；类型 dependency-wiring；符号 exec_command_gpu, exec_command_cpu, _exec_command, _exec_command_on_node）：从通用工具库中剥离命令执行职责，减少模块耦合，是本 PR 的源侧主要调整对象。
- miles/utils/external_utils/command_utils.py（模块 命令构建；类别 source；类型 dependency-wiring）：命令构建的核心消费者，其 import 更新与命令录制测试直接关联。
- miles/utils/debug_utils/run_megatron/cli/commands/run.py（模块 CLI 命令；类别 source；类型 dependency-wiring）：run_megatron CLI 的 run 命令使用 exec_command_cpu / exec_command_gpu，是本 PR 的调用方之一。
- miles/utils/debug_utils/run_megatron/cli/commands/compare.py（模块 CLI 命令；类别 source；类型 dependency-wiring）：run_megatron CLI 的 compare 命令使用 exec_command_cpu，是本 PR 的调用方之一。
- tests/fast/utils/command_recorder.py（模块 测试录制；类别 test；类型 test-coverage）：测试辅助代码从对 misc 和 command_utils 双模块打 patch 收窄为只 patch command_utils，体现搬迁后依赖面的变化。
- tests/e2e/short/test_run_megatron.py（模块 端到端测试；类别 test；类型 test-coverage）：端到端测试同步更新导入路径。

关键符号：exec_command_gpu, exec_command_cpu, _exec_command, _exec_command_on_node, exec_command_multi_node

关键源码片段

miles/utils/external_utils/exec_command.py

新增的独立模块，集中承载全部 shell 命令执行辅助函数，是本 PR 的核心交付物。

```
import re
import subprocess

import ray
from ray.util.scheduling_strategies import NodeAffinitySchedulingStrategy

from miles.utils.misc import get_current_node_ip


def exec_command_gpu(cmd: str, capture_output: bool = False) -> str | None:
```

```
    # CPU / GPU 共用同一执行入口，仅消费方命名不同，便于表达调用意图
    return _exec_command(cmd, capture_output=capture_output)


def exec_command_cpu(cmd: str, capture_output: bool = False) -> str | None:
    # 同上，保留独立命名以表达调用方对资源类型的意图
    return _exec_command(cmd, capture_output=capture_output)


def _exec_command(cmd: str, capture_output: bool = False) -> str | None:
    # 打印将要执行的命令，方便排障与测试断言
    print(f"EXEC: {cmd}", flush=True)

    try:
        result = subprocess.run(
            ["bash", "-c", cmd],
            shell=False,
            check=True,
            capture_output=capture_output,
            **(dict(text=True) if capture_output else {}),
        )
    except subprocess.CalledProcessError as e:
        if capture_output:
            print(f"{e.stdout=} {e.stderr=}")
        raise

    if capture_output:
        # 仅在需要捕获输出时打印，避免正常路径刷屏
        print(f"Captured stdout={result.stdout} stderr={result.stderr}")
        return result.stdout


@ray.remote(num_cpus=0.001)
def _exec_command_on_node(cmd: str, capture_output: bool) -> str | None:
    # 远端执行前先清掉 CUDA_VISIBLE_DEVICES，避免把 GPU 环境误带到其他节点
    return _exec_command(f"unset CUDA_VISIBLE_DEVICES; {cmd}", capture_output=capture_
    output)


def exec_command_multi_node(cmd: str, capture_output: bool = False, num_nodes: int | None =
None) -> list[str | None]:
    """Execute a shell command on every alive Ray node in parallel.

    Supported placeholders in `cmd` (replaced per-node before execution):
        {{node_rank}}   - 0-based index of the node
        {{nnodes}}      - total number of alive nodes (or num_nodes if specified)
        {{master_addr}} - NodeManagerAddress of the first node
        {{node_ip}}     - NodeManagerAddress of the current node
```

```
    Args:
        num_nodes: If set, only use the first `num_nodes` nodes instead of all alive nodes.
    """
    ray.init(address="auto")
    try:
        current_ip = get_current_node_ip()
        # 当前节点排最前，其余按地址排序，保证 master 选择稳定
        nodes = sorted(
            [n for n in ray.nodes() if n.get("Alive")],
            key=lambda n: (n["NodeManagerAddress"] != current_ip, n["NodeManagerAddress"]),
        )
        assert len(nodes) > 0

        if num_nodes is not None:
            assert num_nodes <= len(nodes), f"Requested {num_nodes} nodes but only {len(nodes)}
            alive nodes available."
            nodes = nodes[:num_nodes]

        master_addr = nodes[0]["NodeManagerAddress"]
        nnodes = str(len(nodes))

        placeholder_pattern = re.compile(
            "|".join(map(re.escape, ["{{node_rank}}", "{{nnodes}}", "{{master_addr}}", "{{node_ip}}"]))
        )

        refs = []
        for rank, node in enumerate(nodes):
            substitutions = {
                "{{node_rank}}": str(rank),
                "{{nnodes}}": nnodes,
                "{{master_addr}}": master_addr,
                "{{node_ip}}": node["NodeManagerAddress"],
            }
            node_cmd = placeholder_pattern.sub(lambda m, s=substitutions: s[m.group(0)], cmd)
            refs.append(
                _exec_command_on_node.options(
                    scheduling_strategy=NodeAffinitySchedulingStrategy(node_id=node["NodeID"],
                    soft=False),
                ).remote(node_cmd, capture_output=capture_output)
            )
        return ray.get(refs)
    finally:
        ray.shutdown()
```

评论区精华

本 PR 无任何 review 评论。提交人 fzyzcjy 自行合并，reviewer yueming-yuan 两次直接批准，说明这是一次无争议的纯搬迁重构；真正的设计讨论都发生在跟踪 issue #1837 与同链条的相邻 PR 中。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 若存在未更新的第三方插件或隐藏脚本仍从 miles.utils.misc 导入 *exec_command_*，会触发 ImportError；需要全局搜索确认无遗漏。
  2. *exec_command_multi_node* 涉及 ray.init / ray.shutdown 生命周期，在并发或嵌套调用场景下需注意；本 PR 未改变该逻辑，风险与之前相同。
  3. 测试 patch 面收窄：command_recorder.py 现在只 patch command_utils，若未来有别的模块直接复用 *exec_command_*，快照测试将不会覆盖到，需要后续补测。
  4. 对用户无行为影响，所有模型脚本、launcher、CLI 变更仅涉及导入语句，功能不变。
     - 影响：影响范围集中在 miles 代码库的开发组织层面：misc.py 卸载了命令执行职责，*external_utils/exec_command.py* 成为 shell 命令执行的唯一入口。对用户与运行中的训练 / 推理流程无行为影响；对团队而言，后续命令构建、模型配置加载、launcher 相关改动都将在 *external_utils* 下进行，模块边界更清晰。外部插件若曾依赖 misc.*exec_command_*，需要跟随迁移到新模块。 - 风险标记：纯代码搬迁，导入面调整，多模块影响

关联脉络

- PR #1903 Rename *exec_command* by the resource its command needs: 将 *exec_command* 按资源重命名，本 PR 是其后续代码移动，二者共同完成命令执行函数的组织调整。
- PR #1905 Remove non-reproducible file arguments by supporting inline base64 payloads: 同属 #1837 链，修改 command_utils.py 与启动脚本依赖，依赖本 PR 提供的新模块。
- PR #1906 Snapshot the launchers that build their own command line: 依赖命令记录器与 *exec_command* 新路径，与本 PR 共享 command_utils 相关测试设施。
- PR #1909 Expand the model args in python before building the command: 涉及 command_utils 与模型脚本启动路径，与本 PR 同属命令构建重构链。
- PR #1910 Replace the model config shell scripts with python: 基于本 PR 建立的 *exec_command* 模块继续改造模型配置加载。
- PR #1911 Quote the model args miles inlines into the launch command: 同样聚焦命令组装安全，与本 PR 在 launch 命令生成路径上直接相关。