

PR #1997 完整报告

radixark/miles

Add worker specs for the sglang and miles routers

合并时间: 2026-09-04 13:44

原文链接: <http://prhub.com.cn/radixark/miles/pull/1997>

执行摘要

- 一句话: 为路由器引入 Worker Spec, 将启动命令生成逻辑从路由器管理器迁移到统一的规范描述中。
- 推荐动作: 此 PR 值得精读。它不是一个简单的功能添加, 而是对核心基础设施模式的重大改进。重点关注 `miles/ray/specs/inference.py` 中新的规范实现模式, 以及 `worker_spec.py` 中数据结构的演化。理解从“静态字符串命令”到“上下文相关的命令生成器”的转变, 对于掌握后续相关 PR 的演进方向至关重要。需要特别检查 `start_router` 接口变更的兼容性影响。

功能与动机

根据关联 Issue #1837 的描述, 该 PR 是大型重构计划的一部分 ("Part of #1837")。从历史 PR 链 (如 #1842、#1843、#2006) 和本次变更内容看, 动机是实施一个“工作者规范” (worker spec) 抽象层, 以标准化所有后台进程 (路由器、会话服务器、推理引擎) 的描述、端口分配和启动逻辑。本次具体聚焦于路由器, 将其从内联逻辑转变为符合规范的可复用组件, 从而简化 `RouterManager`, 并支持未来的多模型和多端口场景。

实现拆解

1. 扩展 Worker Spec 数据模型: 修改 `miles/utils/workers/worker_spec.py`。为 `PortInfo` 添加默认值 (`mode="per_worker", allow_dynamic=False`) 以简化构造。为 `SchedulingSpec` 添加了 `single()` 类方法。最关键的是, 新增 `LaunchCommandContext` 类作为参数, 并将 `CommandWorkerSpec.launch_command` 的类型从固定的字符串 `str` 改为可调对象 `Callable[[LaunchCommandContext], str]`。这使得启动命令能根据运行时上下文 (主机、端口) 动态生成。
2. 实现路由器 Worker Spec: 在 `miles/ray/specs/inference.py` 中, 将原本返回 `None` 的 `specs_router` 函数实现为真实的规范生成器。新增核心函数 `_compute_spec_router`, 它根据模型配置构建一个 `CommandWorkerSpec`。该规范内嵌了一个闭包 `_compute_launch_command`, 该闭包根据 `LaunchCommandContext` 中的主机和端口信息, 动态决定是使用 Miles 路由器还是 SGLang 路由器, 并生成对应的启动命令字符串。同时新增了 `compute_router_pool_id` 和 `_compute_router_primary_port_info` 辅助函数。
3. 重构路由器启动入口: 更新 `miles/ray/rollout/router_manager.py` 中的 `start_router` 函数签名, 从 `(args, *, has_pd_disaggregation)` 改为 `(args, model_idx, model_cfg)`。函数内部不再包含具体的路由器命令构建逻辑, 而是调用新实现的 `_compute_spec_router` 获取规

范，然后使用该规范和动态端口构建 LaunchCommandContext，最终调用 spec.launch_command(ctx) 获取启动命令字符串。_launch_command_on_head 的参数也从列表 list[str] 简化为字符串 str。

4. 更新调用方与测试：修改 miles/ray/rollout/rollout_server.py 中对 start_router 的调用以匹配新签名。全面更新相关单元测试，包括 tests/fast/ray/rollout/test_router_manager.py 和 tests/fast/ray/train/test_group.py 等，以反映接口变化并验证新逻辑。
5. 配套重命名与清理：将测试辅助类和函数中的“group”术语统一重命名为“controller”（例如 FTTestActionGroupExecutor -> FTTestActionControllerExecutor），这与历史 PR #1842/#1843 中将 RayTrainGroup 重命名为 RayTrainController 的方向一致，确保命名一致性。

关键文件：

- miles/ray/specs/inference.py（模块 工作器规范；类别 source；类型 core-logic；符号 compute_router_pool_id, _compute_spec_router, _compute_launch_command, _compute_router_primary_port_info）：核心变更文件，实现了路由器的 Worker Spec，包含了新的规范计算和命令生成逻辑。
- miles/ray/rollout/router_manager.py（模块 路由器管理；类别 source；类型 entrypoint；符号 start_router, _launch_command_on_head）：重构了路由器启动的核心入口函数，将具体逻辑委托给新的 Worker Spec。
- miles/utils/workers/worker_spec.py（模块 工作者规范；类别 source；类型 data-contract；符号 single, LaunchCommandContext）：扩展了 Worker Spec 的基础数据模型，引入了 LaunchCommandContext 并使 CommandWorkerSpec 支持动态命令生成。

关键符号：compute_router_pool_id, _compute_spec_router, _compute_router_primary_port_info, start_router, _launch_command_on_head, SchedulingSpec.single

关键源码片段

miles/ray/specs/inference.py

核心变更文件，实现了路由器的 Worker Spec，包含了新的规范计算和命令生成逻辑。

```
# miles/ray/specs/inference.py
```

```
def _compute_spec_router(args, model_idx: int, model_cfg: ModelConfig) -> CommandWorkerSpec:
```

```
    """为单个路由器模型构建完整的工作者规范。
```

```
    返回的 CommandWorkerSpec 包含了一个 launch_command 可调用对象。
    该对象会在运行时接收 LaunchCommandContext（包含主机和端口），
    并根据 args.use_miles_router 和模型配置决定生成哪种路由器的启动命令。
    """
```

```
def _compute_launch_command(ctx: LaunchCommandContext) -> str:
    # 从上下文中获取动态分配的端口
    router_port = ctx.ports["primary"]
```

```

if args.use_miles_router:
    # 如果使用 Miles 路由器，构建其配置并生成命令
    assert not model_cfg.has_pd_disaggregation, "miles router does not support PD
    disaggregation."
    router_config = compute_miles_router_config(args, host=ctx.host, port=router_port)
    launch_argv = [sys.executable, "-m", "miles.router.router", *config_to_argv(router_
    config)]
else:
    # 否则，构建 SGLang 路由器的参数并生成命令
    router_args = compute_sglang_router_args(
        args,
        host=ctx.host,
        port=router_port,
        prometheus_port=ctx.ports["prometheus"],
        has_pd_disaggregation=model_cfg.has_pd_disaggregation,
    )
    logger.info(f"Launch router with args: {router_args}")
    launch_argv = [sys.executable, "-m", "sglang.router.launch_router", *router_args_to_
    argv(router_args)]

# 将参数列表连接成一个完整的命令字符串
return shlex.join(launch_argv)

# 返回配置好的 CommandWorkerSpec 对象
return CommandWorkerSpec(
    name=compute_router_pool_id(model_idx),
    port_infos=[
        _compute_router_primary_port_info(args, model_idx=model_idx),
        PortInfo(name="prometheus", static_port=9000, allow_dynamic=True),
    ],
    env_var=lambda: {},
    scheduling=SchedulingSpec.single(num_gpus_per_worker=0),
    launch_command=_compute_launch_command, # 存储可调用对象，而非字符串
)

```

miles/ray/rollout/router_manager.py

重构了路由器启动的核心入口函数，将具体逻辑委托给新的 Worker Spec。

miles/ray/rollout/router_manager.py

```

def start_router(args, model_idx: int, model_cfg: ModelConfig) -> tuple[str, int]:
    """启动路由器并返回 (router_ip, router_port)。

```

此函数现在是一个轻量级协调器：

1. 获取主机信息并分配主端口和 prometheus 端口。
2. 通过 `_compute_spec_router` 获取路由器的工作器规范。
3. 从规范中获取启动命令生成器 (`launch_command`)，并传入运行时上下文。
4. 将生成的命令字符串交给 `_launch_command_on_head` 执行。

```

"""

```

```

router_ip = _wrap_ipv6(get_host_info()[1])
router_port = find_available_port(random.randint(3000, 4000))
prometheus_port = find_available_port(random.randint(4000, 5000))

# 获取路由器的 Worker Spec
spec = _compute_spec_router(args, model_idx=model_idx, model_cfg=model_cfg)
ports = dict(primary=router_port, prometheus=prometheus_port)
assert set(ports) == {info.name for info in spec.port_infos}

# 创建运行时上下文并生成最终的启动命令字符串
launch_command = spec.launch_command(LaunchCommandContext(host=router_ip, ports=
ports))

actor_handle = _launch_command_on_head(launch_command)
wait_tcp_ready(
    router_ip,
    router_port,
    is_alive=functools.partial(_actor_is_alive, actor_handle),
    timeout=_SERVER_READY_TIMEOUT_SECS,
)
logger.info(f"Router launched at {router_ip}:{router_port}")
return router_ip, router_port

def _launch_command_on_head(launch_cmd: str) -> ray.actor.ActorHandle:
    """在集群头节点上创建一个 CommandActor 并执行给定的命令字符串。"""
    actor_handle = create_head_worker_actor(worker_cls=CommandActor, env_vars={}, num_
cpus=0.2, ctor_kwargs={})
    # 现在直接接收并运行一个字符串命令
    actor_handle.run.remote(cmd=launch_cmd, envs={})
    return actor_handle

```

miles/utils/workers/worker_spec.py

扩展了 Worker Spec 的基础数据模型，引入了 `LaunchCommandContext` 并使 `CommandWorkerSpec` 支持动态命令生成。

miles/utils/workers/worker_spec.py (关键变更部分)

```

class PortInfo(FrozenStrictBaseModel):
    name: str
    static_port: int
    # 为 mode 和 allow_dynamic 添加了默认值，简化了创建。
    mode: Literal["per_worker", "master"] = "per_worker"
    allow_dynamic: bool = False
    num_consecutive: int = 1
    offset_by_cell: bool = False
    # ... ( 验证器保持不变 )

class SchedulingSpec(FrozenStrictBaseModel):

```

```

num_cells: int
num_workers_per_cell: int
num_gpus_per_worker: float

@classmethod
def single(cls, num_gpus_per_worker: float) -> "SchedulingSpec":
    """为单进程、单工作单元工作者创建调度规范的便捷方法。"""
    return SchedulingSpec(
        num_cells=1,
        num_workers_per_cell=1,
        num_gpus_per_worker=num_gpus_per_worker,
    )

```

```

class LaunchCommandContext(FrozenStrictBaseModel):
    """传递给命令生成函数的运行时上下文，包含主机和分配好的端口映射。"""
    host: str
    ports: dict[str, int] # 例如 {"primary": 8000, "prometheus": 9001}

```

```

class CommandWorkerSpec(BaseWorkerSpec):
    # 关键变更：从固定的字符串改为可调用对象，接受上下文并返回命令字符串。
    launch_command: Callable[[LaunchCommandContext], str]

```

评论区精华

唯一的 PR 评论来自 [guapisolo](#) 的审核批准，其中提到 "Reviewed in stack context: #2006 completes the worker-spec context contract...". 这表明此 PR 是在一个更大的重构堆栈中进行审核的，并且与其他相关 PR（如 #2006 为工作者规范定义了上下文契约）紧密衔接。没有发现其他公开的技术争议讨论。

- 暂无高价值评论线程

风险与影响

- 风险：
 1. 接口破坏性变更风险：start_router 函数签名发生根本性变化（从接受布尔标志到接受模型索引和配置对象），这是一个破坏性变更。所有直接调用 start_router 的代码都必须更新。目前只看到 rollout_server.py 一处调用被更新，需要确认是否有其他未显示的调用点（如测试、其他模块）。
 2. 动态命令生成风险：CommandWorkerSpec.launch_command 改为 Callable 引入了间接调用。如果闭包或上下文对象构造错误，可能在运行时（而非静态类型检查时）才暴露出问题，增加调试难度。
 3. 逻辑迁移正确性风险：核心的路由器命令生成逻辑从 router_manager.py 完整迁移到 specs/inference.py。需要确保迁移过程中没有遗漏参数或改变逻辑（例如 has_pd_disaggregation 的断言检查现在依赖于 model_cfg 对象的属性，而非函数参数）。测试覆盖是关键，现有的测试套件已进行更新。

4. 命名一致性风险：将 group 重命名为 controller 是跨多个文件的重构。虽然此次 PR 覆盖了 `ft_test_actions.py` 及其测试，但需确认 `miles/ray/train/group.py` 中对 `FTTestActionGroupExecutor` 的使用是否已正确更新（从代码片段看，更新了导入但使用处仍用 `group=self`，参数名不匹配，存在潜在的命名不一致）。- 影响：对用户：无直接影响，这是内部架构重构。对系统：这是 Rollout 模块规范化重构的关键一步。将路由器的描述和启动逻辑解耦并标准化，有助于提高系统的可维护性和可测试性。为未来支持更复杂的端口管理、多模型路由器部署以及更灵活的工作者生命周期管理奠定了基础。对团队：为工程师提供了一个更清晰、更一致的模式来定义和使用各种后台进程。新引入的 Worker Spec 模式可以被后续 PR（如会话服务器、推理引擎的规范实现）复用，加速重构进程。

- 风险标记：接口破坏性变更，逻辑迁移正确性，命名一致性风险

关联脉络

- PR #1837 Tracking issue for refactoring and enhancements: 此 PR 是 Issue #1837 中描述的重构和增强工作的一部分。
- PR #1999 Implement a worker spec for the session server: 同一重构链中的先行或并行 PR，为会话服务器实现了类似的工作者规范。
- PR #2006 Launch inference engines through the inference engine worker spec: 同一重构链中的后续 PR，完成了工作者规范的上下文契约，并将其应用于推理引擎。
- PR #1842 Split RolloutManager into InferenceController and RolloutExecutor: PR 中重命名了 `group` -> `controller`，与本次 PR 中测试代码的重命名相呼应，属于同一架构拆分方向。