PR #49515 完整报告

vllm-project/vllm

[ROCm][CI] Select CPU platform for native no-GPU jobs

合并时间：2026-08-15 14:30

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/49515

执行摘要

- 一句话：修复 ROCm CI 无 GPU Job 的 CPU 平台选择
- 推荐动作：值得快速阅读：如果你负责平台抽象或 ROCm/CPU CI，可以学习“显式环境变量目标优先于 wheel 元数据和宿主机探测”的设计模式；同时注意后续维护 cpu_platform_plugin() 时保持其不会在 VLLM_TARGET_DEVICE=cpu 下返回 None 的不变量。该 PR 不需要深度精读，但可作为平台选择逻辑的参考案例。

功能与动机

PR body 明确指出：Basic Models Test (Other CPU) 与 Async Engine, Inputs, Utils, Worker, Config (CPU) 两个 Buildkite 测试组在 ROCm CI 上失败。原生 no_gpu: true Job 已正确隐藏 MI300 设备，但复用的 ROCm wheel 后缀导致平台探测无法选中 CPU。修复后“native job 期望零 GPU 时导出 VLLM_TARGET_DEVICE=cpu，显式 CPU 选择优先于 wheel 元数据和宿主机加速器探测”，且保持现有 dind: false 调度模型不变。

实现拆解

实现分四步：

1. 平台插件入口改造（vllm/platforms/__init__.py）：cpu_platform_plugin() 在开始处先判断 envs.VLLM_TARGET_DEVICE == "cpu"，为真则直接判定 CPU 平台可用；原来的 vllm_version_matches_substr("cpu") wheel 后缀判断与 macOS 判断全部移入 else 分支。这是根因修复：native no-GPU Job 复用 ROCm wheel，版本后缀永远不匹配 cpu。
2. 平台解析短路（resolve_current_platform_cls_qualname()）：函数开头新增显式 CPU 分支，直接调用 cpu_platform_plugin() 并 assert 非 None 后返回，不再逐个探测 tpu/cuda/rocm/xpu/cpu 插件。注释解释了原因：native CPU-only CI Job 可能运行在加速器宿主机上，全面探测会同时激活 CPU 与宿主机加速器两个插件，触发多插件冲突。
3. CI 脚本注入环境变量（.buildkite/scripts/hardware_ci/run-amd-test.sh）：在 initialize_native_environment() 中，当 VLLM_CI_EXPECTED_GPU_COUNT 为 0 时导出 VLLM_TARGET_DEVICE=cpu 并 export。默认值 :-1 保证普通 GPU Job 不受影响，只影响明确声明零 GPU 的 Job。
4. 测试配套（tests/test_zen_cpu_platform_detection.py）：新增 test_cpu_target_selects_cpu_platform_from_non_cpu_wheel，设置环境变量后 mock 掉版本匹配、ZEN CPU 探测和 ROCm 插件，断言解析结果为

vllm.platforms.cpu.CpuPlatform，并验证 vllm_version_matches_substr 与 rocm_platform_plugin 均未被调用，覆盖“非 CPU wheel 但显式声明 CPU 目标”的关键场景。

关键文件：

- vllm/platforms/__init__.py（模块 平台选择；类别 source；类型 dependency-wiring；符号 cpu_platform_plugin, resolve_current_platform_cls_qualname）：平台选择核心入口：cpu_platform_plugin() 新增 VLLM_TARGET_DEVICE=cpu 优先判断，resolve_current_platform_cls_qualname() 新增显式 CPU 短路分支，是本次修复的根因所在。
- tests/test_zen_cpu_platform_detection.py（模块 平台测试；类别 test；类型 test-coverage；符号 test_cpu_target_selects_cpu_platform_from_non_cpu_wheel）：新增 test_cpu_target_selects_cpu_platform_from_non_cpu_wheel 单测，验证显式 CPU 目标在非 CPU wheel 下仍能正确解析并绕过版本探测与 ROCm 插件。
- .buildkite/scripts/hardware_ci/run-amd-test.sh（模块 CI 脚本；类别 infra；类型 infrastructure；符号 initialize_native_environment）：CI 脚本在 VLLM_CI_EXPECTED_GPU_COUNT == 0 时导出 VLLM_TARGET_DEVICE=cpu，是触发平台选择修复的上游注入点。

关键符号：cpu_platform_plugin, resolve_current_platform_cls_qualname, initialize_native_environment

关键源码片段

vllm/platforms/__init__.py

平台选择核心入口：cpu_platform_plugin() 新增 VLLM_TARGET_DEVICE=cpu 优先判断，resolve_current_platform_cls_qualname()新增显式CPU短路分支，是本次修复的根因所在。

```
# vllm/platforms/__init__.py
# cpu_platform_plugin() 的改造：显式 VLLM_TARGET_DEVICE=cpu 优先级最高。
def cpu_platform_plugin() -> str | None:
    logger.debug("Checking if CPU platform is available.")

    # native no-GPU CI Job 复用 ROCm wheel，不能用 wheel 后缀判断；
    # 先看环境变量，命中则直接跳过所有探测。
    is_cpu = envs.VLLM_TARGET_DEVICE == "cpu"
    if is_cpu:
        logger.debug(
            "Confirmed CPU platform is available because "
            "VLLM_TARGET_DEVICE is set to CPU."
        )
    else:
        try:
            is_cpu = vllm_version_matches_substr("cpu")
            if is_cpu:
                logger.debug(
                    "Confirmed CPU platform is available because vLLM is built "
```

```
                "with CPU."
            )
        if not is_cpu:
            import sys

            is_cpu = sys.platform.startswith("darwin")
            if is_cpu:
                logger.debug(
                    "Confirmed CPU platform is available because the machine "
                    "is MacOS."
                )
    except Exception as e:
        logger.debug("CPU platform is not available because: %s", str(e))

if not is_cpu:
    return None

# AMD Zen + zentorch 时走 ZenCpuPlatform，否则退回 CpuPlatform
if _is_amd_zen_cpu():
    try:
        import zentorch # noqa: F401

        logger.info(
            "AMD Zen CPU detected with zentorch installed, using ZenCpuPlatform."
        )
        return "vllm.platforms.zen_cpu.ZenCpuPlatform"
    except ImportError:
        logger.debug(
            "AMD Zen CPU detected but zentorch not installed, "
            "falling back to CpuPlatform."
        )

return "vllm.platforms.cpu.CpuPlatform"


def resolve_current_platform_cls_qualname() -> str:
    # 显式 CPU target 是权威的：native CPU-only CI Job 复用加速器 wheel，
    # 且可能运行在加速器宿主机上，逐个探测会同时激活 CPU 与宿主机加速器，
    # 因此一旦明确要求 CPU，就直接短路所有插件探测。
    if envs.VLLM_TARGET_DEVICE == "cpu":
        cpu_platform_cls_qualname = cpu_platform_plugin()
        assert cpu_platform_cls_qualname is not None
        logger.debug("Explicitly selected CPU platform.")
        return cpu_platform_cls_qualname

    # 以下为原有探测逻辑，未变更
    platform_plugins = load_plugins_by_group(PLATFORM_PLUGINS_GROUP)
    activated_plugins = []
```

```
    for name, func in chain(builtin_platform_plugins.items(), platform_plugins.items()):
        try:
            assert callable(func)
            platform_cls_qualname = func()
            if platform_cls_qualname is not None:
                activated_plugins.append(name)
        except Exception:
            pass

    # 多插件冲突检测与单插件选定逻辑保持不变
```

tests/test_zen_cpu_platform_detection.py

新增 test_cpu_target_selects_cpu_platform_from_non_cpu_wheel 单测，验证显式 CPU 目标在非 CPU wheel 下仍能正确解析并绕过版本探测与 ROCm 插件。

```
# tests/test_zen_cpu_platform_detection.py

def test_cpu_target_selects_cpu_platform_from_non_cpu_wheel(
    monkeypatch: pytest.MonkeyPatch,
):
    # 模拟 native no-GPU CI Job 注入的显式目标设备
    monkeypatch.setenv("VLLM_TARGET_DEVICE", "cpu")

    with (
        patch("vllm.platforms.vllm_version_matches_substr") as version_matches,
        patch("vllm.platforms._is_amd_zen_cpu", return_value=False),
        patch("vllm.platforms.rocm_platform_plugin") as rocm_plugin,
    ):
        # 复用 ROCm wheel 时也应解析到 CpuPlatform
        assert (
            resolve_current_platform_cls_qualname() == "vllm.platforms.cpu.CpuPlatform"
        )

    # 显式 target 不依赖 wheel 后缀，也不应触发宿主机加速器探测
    version_matches.assert_not_called()
    rocm_plugin.assert_not_called()
```

.buildkite/scripts/hardware_ci/run-amd-test.sh

CI 脚本在 VLLM_CI_EXPECTED_GPU_COUNT == 0 时导出 VLLM_TARGET_DEVICE=cpu，是触发平台选择修复的上游注入点。

```
# .buildkite/scripts/hardware_ci/run-amd-test.sh
# initialize_native_environment() 内，仅对零 GPU 的原生 Job 生效。
# 复用 ROCm wheel 的 CPU Job 需要显式声明目标设备，绕开 wheel 后缀探测。
if [[ "${VLLM_CI_EXPECTED_GPU_COUNT:-1}" == "0" ]]; then
    VLLM_TARGET_DEVICE=cpu
    export VLLM_TARGET_DEVICE
fi
```

评论区精华

PR 内没有实质性的代码设计讨论，主要评论为流程性内容：

- mergify[bot] 两次提醒 pre-commit 检查失败，要求作者先运行 uv pip install pre-commit>=4.5.1 并执行 pre-commit run --all-files 后重新提交，属于流程性提醒。
- claude[bot] 说明该 PR 来自 fork，自动 review 被禁用，需维护者评论 @claude review 触发一次性 review。
- 最终由 tjtanaa 给出空内容的 APPROVED 通过，说明方案经过人工确认。
- pre-commit 检查失败两次 (other): 作者后续提交通过 CI，两次提醒均未涉及业务逻辑问题。
- fork PR 自动 review 被禁用 (other): 未触发 claude review，最终由 tjtanaa 人工 approve 通过。

风险与影响

- 风险：主要风险点：
  - 环境变量全局生效：VLLM_TARGET_DEVICE=cpu 一旦被设置（无论 CI 还是用户环境），resolve_current_platform_cls_qualname() 会直接短路到 CPU 平台，跳过所有加速器探测与多插件冲突检查。普通 GPU 用户需要确认没有外部脚本意外导出该变量，否则会静默放弃 GPU 平台。
  - 断言依赖隐式不变量：短路分支中的 assert cpu_platform_cls_qualname is not None 在正常逻辑下恒成立，但若未来 cpu_platform_plugin() 被修改为在某些条件下返回 None，会直接触发崩溃，需要后续维护者留意。
  - 测试覆盖有限：单测只覆盖“显式 CPU + 非 CPU wheel”这一条路径；未覆盖“显式 CPU 但宿主机可探测到 CUDA/ROCm”的冲突场景，也未覆盖 run-amd-test.sh 的环境变量注入是否真正到达 Python 进程。建议后续补 shell 层面的集成检查。
  - 回归风险较低：GPU/CUDA 默认路径在 VLLM_TARGET_DEVICE 未设置时行为完全不变，改动集中在 ROCm CI 与平台选择入口。
  - 影响：对用户与系统：vllm/platforms/__init__.py 是平台选择的核心入口，改动在有显式环境变量时改变解析顺序，但默认行为不变；对普通 GPU 用户无感知。对团队：修复了 ROCm CI 上两个 CPU 测试组（Basic Models Test (Other CPU)、Async Engine, Inputs, Utils, Worker, Config (CPU)）的失败问题，AMD CI 维护者可恢复对 native no-GPU Job 的调度。影响范围限定在 ROCm/CI 场景，改动面小。
  - 风险标记：平台选择核心路径变更，环境变量全局生效，测试覆盖有限，仅 ROCm CI 验证

关联脉络

- PR #52400 [ROCm]: Drop pybind11 from Dockerfile.rocm to prevent version mismatch: 同属 ROCm CI 稳定性修复，都处理 wheel/ 环境元数据与运行时不匹配的 ROCm 特有问题。
- PR #52326 [CI] Shard Humming H100 eval: ROCm/AMD CI 关键路径优化，属于同一 ROCm CI Job 结构调整上下文。

- PR #50597 [ROCm]Remove special-case SiTU support model-specific gating: 同属 ROCm 平台路由 / 选择逻辑的简化与收敛，背景一致（减少平台特判、让配置优先）。