

PR #52264 完整报告

vllm-project/vllm

[CI][AMD] Improve Kubernetes failure diagnostics

合并时间: 2026-08-18 07:09

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/52264>

执行摘要

- 一句话: 重构 AMD CI 失败诊断, 采集 K8s 与 cgroup 证据
- 推荐动作: 值得精读, 尤其适合关注 CI 可观测性和 Bash 脚本工程化的读者。重点关注两点: 一是 cgroup v2 路径解析的实现方式 (结合 mountinfo 还原宿主机路径), 二是 artifact 路径逃逸校验与 symlink 拒绝策略, 这是 CI 安全设计的好范例。合并前请确认 ci-infra#475 已合并。

功能与动机

PR body 明确指出目标是“Collect Kubernetes-scoped AMD CI failure evidence while keeping the failing test log concise and immediately visible” (采集 Kubernetes 作用域内的 AMD CI 失败证据, 同时保持失败测试日志简洁、立即可见)。关联 Issue #475 说明这是两仓库迁移的第一阶段, 用来替代旧的将诊断内联写进终端日志的做法, 避免诊断信息丢失, 并消除自定义上传器与上传超时的重复逻辑。

实现拆解

1. 新增 Kubernetes 环境变量与清理机制: 在脚本头部引入 `amd_diagnostics_workspace`、`amd_diagnostics_pod_name`、`amd_diagnostics_k8s_namespace`、`amd_diagnostics_k8s_node_name`, 从 `VLLM_CI_*` 或 `POD_NAME`、`NODE_NAME`、`HOSTNAME` 等 Downward API 注入值获取; 同时在 `clear_ci_orchestration_env` 中清理 `VLLM_CI_K8S_POD_NAME` 与 `VLLM_CI_K8S_NAMESPACE`, 避免跨任务环境残留污染。
2. 重构诊断执行函数: 将原 `run_amd_diagnostic` 拆分为 `run_failure_diagnostic`, 并新增 `append_failure_diagnostic_section`、`append_failure_diagnostic_note`、`append_failure_diagnostic_file` 等辅助函数; 所有命令输出从 `tee` 改为直接重定向到日志文件, 并引入 25 秒总预算与 5 秒单命令超时, 防止诊断采集阻塞 CI 主流程。
3. 采集 cgroup v2 与进程资源证据: 实现 `resolve_current_cgroup_v2_dir`, 通过 `/proc/self/cgroup` 与 `/proc/self/mountinfo` 解析当前进程在宿主机视角下的 cgroup v2 目录; `collect_cgroup_diagnostics` 读取 `memory`、`cpu`、`cpuset`、`pids`、`io` 等关键文件; `collect_process_diagnostics` 抓取 shell 进程的 `limits`、`status` 关键字段及按 RSS 排序的 top 100 进程快照。
4. 采集挂载、设备与 GPU 证据并强化路径安全: 记录 `checkout`、`workspace`、`tmp`、`shared-memory` 等挂载信息, 检查设备节点权限、GPU 健康与网络时间戳; 对写入的 artifact 路径做逃逸校验, 拒绝通过 symlink 解析到 `checkout` 之外的路径, 避免 PR 控制

symlink 导致 CI 泄露或破坏。

5. 输出摘要与日志组织：最终打印无第三方依赖的两列 Bash 摘要，不携带 emoji 短代码或独立 Buildkite 诊断分组；保留 `--- Test log` 边界，使 pytest 失败与基础设施摘要保持在同一最后展开的 section 中；移除自定义上传超时，交由 Buildkite 在命令退出后的 post-command 阶段统一上传。

测试配套：没有新增独立测试用例，验证依赖真实 Buildkite 构建（PR 中提供 [amd-ci/builds/12112](#) 与 [ci/builds/84117](#) 的测试链接），并与 ci-infra#475 联合验证。

关键文件：

- `.buildkite/scripts/hardware_ci/run-amd-test.sh`（模块 CI 脚本；类别 infra；类型 infrastructure）：唯一变更文件，承载全部 Kubernetes 失败诊断逻辑：新增 cgroup 解析、进程 / 挂载 / GPU 证据采集、路径安全校验和两列摘要输出，是整个 PR 的核心。

关键符号：`run_failure_diagnostic`, `append_failure_diagnostic_section`, `append_failure_diagnostic_note`, `append_failure_diagnostic_file`, `decode_mountinfo_path`, `resolve_current_cgroup_v2_dir`, `collect_cgroup_files`, `collect_cgroup_diagnostics`, `collect_process_diagnostics`, `collect_mount_diagnostic`

关键源码片段

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

唯一变更文件，承载全部 Kubernetes 失败诊断逻辑：新增 cgroup 解析、进程 / 挂载 / GPU 证据采集、路径安全校验和两列摘要输出，是整个 PR 的核心。

```
# 通过 /proc/self/cgroup 与 /proc/self/mountinfo 解析当前进程所在的 cgroup v2 目录。
# 容器内看到的 cgroup 路径是相对于挂载根的，必须结合 mountinfo 还原宿主机可见的绝对路径。
resolve_current_cgroup_v2_dir() {
    local cgroup_path=""
    local mount_root=""
    local mount_point=""
    local relative_path=""
    local candidate=""

    # 从 /proc/self/cgroup 中取第 2 列为空（即 v2 层级）的路径，例如 /docker/abc123
    cgroup_path=$(awk -F: ' $2 == "" { print $3; exit }' /proc/self/cgroup 2>/dev/null)
    [[ -n "${cgroup_path}" ]] || return 1

    # 遍历 mountinfo 中 cgroup2 类型的挂载，找到能拼出有效目录的挂载点
    while IFS=$'\t' read -r mount_root mount_point; do
        mount_root=$(decode_mountinfo_path "${mount_root}")
        mount_point=$(decode_mountinfo_path "${mount_point}")

        # 根据挂载根与 cgroup 路径的关系计算相对路径
        if [[ "${cgroup_path}" == "/" ]]; then
            relative_path=""
        elif [[ "${mount_root}" == "/" ]]; then
            relative_path="${cgroup_path}"
        fi
    done
```

```

elif [[ "${cgroup_path}" == "${mount_root}" ]]; then
    relative_path=""
elif [[ "${cgroup_path}" == "${mount_root}/"* ]]; then
    relative_path="/${cgroup_path#${mount_root}/}"
else
    continue
fi

candidate="${mount_point%}/${relative_path}"
# 只有目录真实存在才接受，避免拼出错误路径
if [[ -d "${candidate}" ]]; then
    printf '%s\n' "${candidate}"
    return 0
fi
done < <(awk '
{
    for (i = 6; i <= NF; i++) {
        if ($i == "-" && $(i + 1) == "cgroup2") {
            print $4 "\t" $5
            break
        }
    }
}
' /proc/self/mountinfo 2>/dev/null)

return 1
}

```

评论区精华

该 PR 没有实质性的技术讨论线程，[khluu](#) 直接给予 APPROVED。唯一的 review 来自 [claude\[bot\]](#) 的自动提示，说明仓库配置了手动 review 流程，不构成技术意见。核心设计决策都写在 PR body 与关联 Issue #475 中：包括依赖合并顺序、artifact 路径 pattern 的精确匹配、以及避免重复上传的迁移策略。

- 暂无高价值评论线程

风险与影响

- 风险：

1. cgroup 路径解析依赖环境细节：resolve_current_cgroup_v2_dir 依赖 /proc/self/cgroup 的字段格式与 mountinfo 中 cgroup2 挂载的表示方式，不同容器运行时或内核版本下可能解析失败；脚本对此有 unavailable 回退，不会中断主流程，但会丢失部分资源证据。
2. 诊断采集超时风险：虽然设置了 25 秒总预算和 5 秒单命令超时，但如果系统负载极高导致 timeout 命令本身被拖慢，仍可能小幅拉长 CI 失败后的收尾时间。

3. 路径安全校验可能误拒绝合法路径：拒绝 symlink 解析到 checkout 外的设计偏向保守，某些合法的共享内存或 tmp 挂载路径可能被过滤，导致对应诊断文件缺失。
4. 跨仓库合并顺序依赖：必须等 ci-infra#475 先合并，否则新的 diagnostics.log 路径在 Buildkite artifact pattern 上匹配不到，导致诊断产物丢失；当前脚本已通过“匹配不到无副作用”的设计保证过渡安全，但顺序颠倒仍是明显风险。- 影响：对最终用户无影响，不涉及模型推理或服务逻辑。对 AMD/ROCm CI 的可观测性有显著正面影响：失败时能拿到 Kubernetes 作用域内的 cgroup、进程、挂载、GPU 健康等多维证据，大幅缩短故障根因定位时间。对团队而言，统一了诊断产物路径与上传方式，消除自定义上传器维护成本，并依赖 ci-infra 的标准 artifact 流程。影响范围仅限 AMD CI 脚本，其他平台（NVIDIA、CPU 等）不受影响。- 风险标记：跨仓库依赖顺序，Bash 脚本复杂度高，诊断采集超时敏感，路径安全校验严格

关联脉络

- PR #475 [CI][AMD] Upload failure diagnostics as Buildkite artifacts: 本 PR 明确依赖该 ci-infra PR 先合并，共同构成两仓库迁移的第一阶段：ci-infra 配置 artifact_paths 模式，vLLM 侧改写 diagnostics.log 并交由 Buildkite 上传。