PR #48746 完整报告

vllm-project/vllm

[CI][ROCm] Stabilize ci_base hash calculation and image handoff

合并时间：2026-07-15 23:56

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/48746

执行摘要

- 一句话：修复 ROCm CI base 镜像哈希计算与传递
- 推荐动作：该 PR 为 CI 稳定性修复，值得精读以了解 CI 镜像选型中的竞态问题及解决方案。关注其设计模式：通过多次计算并比对、显式失败代替静默默认值来提升确定性。

功能与动机

在 AMD CI 中，由于 ci_base content hash 的计算和传递存在竞态条件，不同 Buildkite job 可能为同一 commit 计算出不同的 hash tag，导致 test-image job 拉取不存在的镜像，阻止所有下游 AMD 测试运行。具体失败案例见于 https://buildkite.com/vllm/amd-ci/builds/10880/canvas?sid=019f6502-2333-428d-a9fb-d252d393038d&tab=output。

实现拆解

1. ci-bake-rocm.sh：重写 compute_ci_base_content_hash 函数 - 将原函数重命名为 compute_ci_base_content_hash_once，并增加错误返回（|| return 1），当 hash_dockerfile_arg_values 或摘要解析失败时立即退出。 - 新增包装函数 compute_ci_base_content_hash：循环调用 _once 三次，每次间隔 5 秒（可通过 CI_BASE_HASH_ATTEMPTS 和 CI_BASE_HASH_RETRY_DELAY 配置）；若任何一次失败，或三次结果不完全一致，则输出错误并返回 1；否则输出稳定后的 hash。 - 修改 hash_dockerfile_arg_values：当 BASE_IMAGE 的 resolve_image_digest 返回空字符串时，不再输出 unknown，而是直接报错并返回 1，防止静默失败。
2. build-test-image.sh：添加 use_ci_base_if_present 函数 - 在 main() 开头调用该函数，从 Buildkite 元数据 rocm-ci-base-image 读取前序步骤选定的 ci_base 镜像并导出为 CI_BASE_IMAGE 环境变量。 - 调整 use_refreshed_base_if_present 函数，移除其内部的 CI_BASE_IMAGE 设置逻辑，改为仅处理 BASE_IMAGE 和 IMAGE_TAG_LATEST，职责更清晰。 - 确保即使元数据不存在，函数也返回成功（|| true），不阻塞后续流程。

关键文件：

- .buildkite/scripts/ci-bake-rocm.sh（模块 CI 脚本；类别 other；类型 core-logic；符号 compute_ci_base_content_hash_once, compute_ci_base_content_hash, hash_dockerfile_arg_values）：核心变更文件，重写了哈希计算函数，加入重试、比对和错误传播逻辑。

- .buildkite/scripts/rocm/build-test-image.sh（模块 CI 脚本；类别 other；类型 core-logic；符号 use_ci_base_if_present, main）：新增 use_ci_base_if_present 函数，确保 test-image 步骤使用前序步骤确定的 ci_base 镜像。

关键符号：compute_ci_base_content_hash_once, compute_ci_base_content_hash, hash_dockerfile_arg_values, use_ci_base_if_present

关键源码片段

.buildkite/scripts/ci-bake-rocm.sh

核心变更文件，重写了哈希计算函数，加入重试、比对和错误传播逻辑。

```
# 新增的单次哈希计算函数，hash_dockerfile_arg_values 失败时返回非零
compute_ci_base_content_hash_once() {
    local -a content_paths=()
    local -a content_args=()
    local dockerfile="${CI_BASE_DOCKERFILE:-}"
    # … 收集依赖路径和参数
    if [[ -n "${dockerfile}" ]]; then
        printf 'dockerfile:%s\n' "${dockerfile}"
        printf 'resolved-build-args:\n'
        hash_dockerfile_arg_values "${dockerfile}" "${content_args[@]}" \
            || return 1 # 失败时立即返回，不静默
        # …
    fi
} | sha256sum | cut -d' ' -f1

# 包装函数：重试多次并要求结果一致
compute_ci_base_content_hash() {
    local attempts="${CI_BASE_HASH_ATTEMPTS:-3}"
    local delay_secs="${CI_BASE_HASH_RETRY_DELAY:-5}"
    local -a hashes=()

    for ((attempt = 1; attempt <= attempts; attempt++)); do
        if ! hash=$(compute_ci_base_content_hash_once); then
            echo "ci_base content hash calculation ${attempt}/${attempts} failed" >&2
            failed=1
        else
            hashes+=("${hash}")
            echo "ci_base content hash calculation ${attempt}/${attempts}: ${hash}" >&2
        fi
        if ((attempt < attempts)); then sleep "${delay_secs}"; fi
    done

    if ((failed)) || ((${#hashes[@]} != attempts)); then
        echo "Could not calculate a reliable ci_base content hash" >&2
        return 1
    fi
```

```
    for hash in "${hashes[@]:1}"; do
        if [[ "${hash}" != "${hashes[0]}" ]]; then
            echo "ci_base content hash changed between calculations" >&2
            return 1
        fi
    done
    printf '%s\n' "${hashes[0]}"
}

# hash_dockerfile_arg_values 中当 BASE_IMAGE digest 解析失败时直接报错
hash_dockerfile_arg_values() {
    # ...
    if [[ "${arg_name}" == "BASE_IMAGE" && -n "${arg_value}" ]]; then
        digest=$(resolve_image_digest "${arg_value}")
        if [[ -z "${digest}" ]]; then
            echo "Failed to resolve digest for BASE_IMAGE=${arg_value}" >&2
            return 1 # 替代原 printf 'unknown'
        fi
        printf 'arg:%s.digest=%s\n' "${arg_name}" "${digest}"
    fi
}
```

.buildkite/scripts/rocm/build-test-image.sh

新增 use_ci_base_if_present 函数，确保 test-image 步骤使用前序步骤确定的 ci_base 镜像。

```
# 新增函数：尝试从 Buildkite 元数据读取前序步骤选定的 ci_base 镜像
use_ci_base_if_present() {
    local ci_base_image=""
    ci_base_image="$(metadata_get rocm-ci-base-image)"
    if [[ -z "${ci_base_image}" ]]; then
        return 1 # 元数据不存在，静默失败
    fi
    export CI_BASE_IMAGE="${ci_base_image}"
    echo "Using ROCm ci_base image selected by the preceding build step: ${CI_BASE_IMAGE}"
}

main() {
    local base_refreshed=0
    use_ci_base_if_present || true # 无论是否成功，继续执行
    if use_refreshed_base_if_present; then
        base_refreshed=1
    fi
    # ...
}
```

评论区精华

无 review 评论，仅 claude[bot] 自动回复且未触发审查，以及 dllehr-amd 的批准。无技术讨论。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 低风险：变更集中在 CI 构建脚本，不涉及核心推理逻辑。
  - 新增的 hash 重试逻辑和摘要解析硬失败可能会在极端网络抖动下增加 CI 失败率，但本意正是要暴露此类不稳定，属于预期行为。
  - use_ci_base_if_present || true 确保即使元数据缺失也能继续，不会阻塞 CI。
- 影响：
  - 影响范围：仅影响 AMD ROCm CI 流水线。
  - 影响程度：中等。修复后可减少因 hash 不一致导致的镜像拉取失败，提高 CI 可靠性。
  - 对用户无直接影响。
  - 风险标记：CI 脚本变更，少量测试覆盖

关联脉络

- 暂无明显关联 PR