PR #23156 完整报告

sgl-project/sglang

[AMD] prepare for MI300x PR runner pool: registry mirror, runner routing, threshold tuning

合并时间：2026-04-21 15:58

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23156

执行摘要

- 一句话：为 AMD CI 准备 MI300x PR 运行器池，优化镜像拉取、动态路由并调整性能阈值。
- 推荐动作：该 PR 对于负责 CI/CD 基础设施和 AMD 平台支持的工程师非常值得精读。重点关注： 1) 如何设计镜像拉取的回退和重试策略以提升鲁棒性； 2) 如何利用 GitHub Actions 的表达式动态选择运行器环境； 3) 大规模性能测试阈值调整的策略和具体数值，可作为硬件平台适配的参考案例。

功能与动机

根据 PR 描述，目标是为 AMD CI 做好准备，以便在 linux-mi300-*gpu-sglang 运行器池上运行 PR 测试，同时保持计划任务在 linux-mi325-*gpu-sglang 池上运行。这涉及三个耦合变更：1）使用内网镜像注册表避免 PR 任务频繁拉取 Docker Hub 导致限流；2）动态路由 PR、计划任务和手动触发到不同的运行器池；3）针对 MI300x 硬件（相比 MI325x 内存带宽更低）调整测试性能预算（包括阈值、预估时间和超时），并跳过两个已知的 MI300x 内核级数值问题导致的非确定性测试。

实现拆解

1. 优化 Docker 镜像拉取策略以使用局域网注册表 - 修改 release-docker-amd-nightly.yml 和 release-docker-amd-rocm720-nightly.yml：在发布夜间镜像后，新增 push_local_registry 任务，将镜像重新标记为内网注册表地址（172.29.8.23:5000/）并推送。 - 修改 scripts/ci/amd/amd_ci_start_container.sh 和 amd_ci_start_container_disagg.sh：调整 find_latest_image 函数逻辑，不再通过 docker manifest inspect 探测内网注册表（因网络可达性问题）。在实际拉取时，优先尝试 docker pull "${LOCAL_DOCKER_REGISTRY}/${IMAGE}" 从内网注册表拉取，失败后回退到公共 Docker Hub 并辅以带指数退避的重试机制（retry_with_backoff）。拉取失败的错误信息会被捕获并添加 [local-pull] 前缀输出以便诊断。脚本还会在检测到 DOCKERHUB_AMD_USERNAME 和 DOCKERHUB_AMD_TOKEN 环境变量时执行 docker login 以提升公共拉取速率限制。 - 在相关 CI 工作流文件（如 pr-test-amd.yml）的 env 块中注入上述 Docker Hub 凭证，确保脚本可以获取到。
2. 根据 CI 触发类型动态选择 AMD 运行器池 - 修改 .github/workflows/pr-test-amd.yml：将 GPU 密集型作业的 runs-on 从硬编码的 linux-mi325-*-sglang 改为动态表达式：${{ format('linux-{0}-Ngpu-sglang', inputs.runner_arch || (github.event_name == 'pull_request' && 'mi300' || 'mi325')) }}。这意味着： pull_request 事件使用 MI300x 池

（更快反馈），schedule 事件使用 MI325x 池（全覆盖夜间测试），workflow_dispatch 事件则允许用户通过新的 runner_arch 输入进行选择（默认为 mi300）。 - 为作业名称添加格式化，例如 name: ${{ format('stage-a-test-1-gpu-small-amd (linux-{0}-1gpu-sglang)', ...) }}，使 Actions UI 中能清晰显示实际使用的运行器架构和 GPU 数量。 - 暂时禁用了 Docker 构建缓存（通过条件判断）。

3. 为 MI300x 硬件放宽性能测试阈值并处理已知问题 - 修改多个性能测试文件，在 is_in_amd_ci() 条件分支中放宽延迟、吞吐量等断言阈值。例如：
   - test/registered/perf/test_bench_serving_1gpu_part2.py：对 test_vlm_offline_throughput、test_score_api_batch_scaling、test_embeddings_api_latency_throughput 等测试，将 MI300x 的期望输出吞吐量从 2000 token/s 降至 900 token/s，延迟上限提高等。
   - test/registered/perf/test_bench_serving_1gpu_part1.py：移除原本在 AMD CI 中跳过 LoRA 测试的代码，并为 test_online_lora_latency 和 test_online_lora_latency_with_concurrent_adapter_updates 放宽 TTFT（首令牌时间）阈值。
   - test/registered/moe/test_torch_compile_moe.py：在非 CUDA 且为 AMD CI 时，将吞吐量阈值从≥270 tok/s 放宽至≥240 tok/s。
   - 修改 test/registered/tokenizer/test_multi_tokenizer.py 和 test/registered/amd/test_deepseek_v32_basic.py 等，放宽 TTFT 阈值或延长预估执行时间（est_time）。
   - 在 pr-test-amd.yml 中将 stage-c 步骤的超时从 90 分钟延长至 120 分钟。
   - 使用 @unittest.skipIf(is_in_amd_ci(), ...) 跳过两个在 MI300x 上因内核级数值非确定性而失败的测试：
   - test/registered/sampling/test_pytorch_sampling_backend.py::test_greedy：贪心解码在多次运行间无法保证比特一致。
   - sgl-kernel/tests/test_moe_topk_sigmoid.py::test_topk_sigmoid[2048-32-4]：特定参数下 sigmoid 分数在 fp32 ULP 内时，tie-breaking 逻辑与 torch.topk 不同（但结果仍有效）。

关键文件：

- .github/workflows/pr-test-amd.yml（模块 CI 流水线；类别 infra；类型 infrastructure）：这是 CI 动态路由的核心配置文件，定义了运行器选择逻辑（PR 用 MI300，计划任务用 MI325）、作业名称格式化、环境变量注入以及临时禁用缓存等关键基础设施变更。
- scripts/ci/amd/amd_ci_start_container.sh（模块 CI 脚本；类别 infra；类型 infrastructure）：负责在 AMD CI 中启动容器的关键脚本，其变更实现了优先从内网注册表拉取镜像、失败后回退到 Docker Hub 并重试的鲁棒策略，是解决 Docker Hub 限流和加速 CI 的关键。
- test/registered/perf/test_bench_serving_1gpu_part2.py（模块 性能测试；类别 test；类型 test-coverage；符号 test_vlm_offline_throughput, test_score_api_batch_scaling, test_embeddings_api_latency_throughput, test_embeddings_api_batch_scaling）：包含多个关键性能测试的断言阈值调整，是适配 MI300x 硬件性能（尤其是内存带宽较低）的集

中体现。其变更直接影响了 CI 在 MI300x 运行器上的通过 / 失败判定。

- test/registered/perf/test_bench_serving_1gpu_part1.py（模块 性能测试；类别 test；类型 test-coverage；符号 test_online_lora_latency, test_online_lora_latency_with_concurrent_adapter_updates）：修改了关键的 LoRA 延迟测试，移除了之前为 AMD CI 跳过的逻辑，并调整了 TTFT 阈值，反映了对 MI300x 硬件上 LoRA 性能的重新评估。
- test/registered/sampling/test_pytorch_sampling_backend.py（模块 采样测试；类别 test；类型 test-coverage；符号 test_greedy）：通过装饰器跳过了在 AMD CI 中因内核级数值非确定性而失败的贪婪解码测试，展示了处理硬件特定已知问题的一种策略。

关键符号：test_vlm_offline_throughput, test_score_api_batch_scaling, test_embeddings_api_latency_throughput, test_online_lora_latency, test_online_lora_latency_with_concurrent_adapter_updates, test_throughput, test_greedy, find_latest_image

关键源码片段

scripts/ci/amd/amd_ci_start_container.sh

负责在 AMD CI 中启动容器的关键脚本，其变更实现了优先从内网注册表拉取镜像、失败后回退到 Docker Hub 并重试的鲁棒策略，是解决 Docker Hub 限流和加速 CI 的关键。

```
# 定义本地注册表地址，支持通过环境变量覆盖（review 后采纳的建议）
LOCAL_DOCKER_REGISTRY="${LOCAL_DOCKER_REGISTRY:-172.29.8.23:5000}"

# 在拉取镜像的主逻辑中
IMAGE=$(find_latest_image "${GPU_ARCH}")
# 优先尝试从本地注册表拉取
if local_pull_output=$(docker pull "${LOCAL_DOCKER_REGISTRY}/${IMAGE}" 2>&1); then
    echo "Pulled from local docker registry: ${LOCAL_DOCKER_REGISTRY}/${IMAGE}"
    docker tag "${LOCAL_DOCKER_REGISTRY}/${IMAGE}" "${IMAGE}"
else
    # 本地拉取失败，回退到公共注册表并重试
    echo "Local docker registry pull failed; falling back to public registry: ${IMAGE}" >&2
    printf '%s\n' "${local_pull_output}" | sed 's/^/ [local-pull] /' >&2 #
    输出带前缀的错误信息便于调试
    retry_with_backoff 6 docker pull "${IMAGE}" # 指数退避重试函数
fi

# 如果提供了 Docker Hub 凭证，则登录（以提升拉取速率限制）
if [[ -n "${DOCKERHUB_AMD_USERNAME}" && -n "${DOCKERHUB_AMD_TOKEN}" ]]; then
    retry_with_backoff 3 docker login -u "${DOCKERHUB_AMD_USERNAME}" -p "${DOCKERHUB_
    AMD_TOKEN}"
fi
```

test/registered/perf/test_bench_serving_1gpu_part2.py

包含多个关键性能测试的断言阈值调整，是适配 MI300x 硬件性能（尤其是内存带宽较低）的集中体现。其变更直接影响了 CI 在 MI300x 运行器上的通过 / 失败判定。

```
def test_score_api_batch_scaling(self):
    """测试评分API在不同批次大小下的性能"""
    batch_sizes = [10, 25, 50]
    for batch_size in batch_sizes:
        res = run_score_benchmark(...)
        self.assertEqual(res["successful_requests"], res["total_requests"])
        # 根据是否在 AMD CI 中调整延迟边界
        if is_in_amd_ci():
            # 为 MI300x 放宽边界（HBM3 带宽较低，延迟更高）
            bounds = {10: (60, 65), 25: (70, 80), 50: (80, 90)}
            default_bounds = (90, 90)
        else:
            # 非 AMD CI（如 CUDA 或其他）保持原有严格边界
            bounds = {10: (45, 50), 25: (50, 60), 50: (60, 65)}
            default_bounds = (60, 65)
        avg_latency_bound, p95_latency_bound = bounds.get(batch_size, default_bounds)
        self.assertLess(res["avg_latency_ms"], avg_latency_bound)
        self.assertLess(res["p95_latency_ms"], p95_latency_bound)
```

评论区精华

reviewer gemini-code-assist[bot] 提出了两项主要改进建议：

1. 避免硬编码内网注册表地址：建议在 amd_ci_start_container.sh 和 amd_ci_start_container_disagg.sh 中使用环境变量 LOCAL_DOCKER_REGISTRY 并设置默认值（${LOCAL_DOCKER_REGISTRY:-172.29.8.23:5000}），以提高脚本的可配置性和可维护性。最终代码采纳了此建议，将硬编码改为环境变量。
2. 清理注释掉的代码：指出脚本中残留的注释行（如 CACHE_HOST=/home/runner/sglang-data）应被移除以保持代码整洁。此建议也被采纳，相关注释代码在最终提交中被删除。 这些讨论体现了对代码质量（可维护性、清晰度）的关注，并达成了共识。

- 使用环境变量提高脚本可配置性 (design): 建议被采纳，代码修改为 LOCAL_DOCKER_REGISTRY="${LOCAL_DOCKER_REGISTRY:-172.29.8.23:5000}"。
- 清理注释掉的代码以保持代码整洁 (style): 建议被采纳，相关注释代码在最终提交中被删除。

风险与影响

- 风险：
  - 测试阈值放宽可能掩盖真实性能回归：由于 MI300x 硬件性能较低，多项性能测试的断言阈值被显著放宽（例如输出吞吐量从 2000 降至 900）。虽然这是为了适应硬件差异，但可能使得在 MI300x 池上运行的 PR 测试对代码变更引入的性能退化不够敏感。
  - 硬编码配置降低可维护性：尽管 review 后改用了环境变量，但默认值仍指向固定的内网 IP 和端口（172.29.8.23:5000）。若内网注册表地址变更，需要更新多个脚本中的默认值或确保环境变量被正确覆盖。
  - 跳过测试可能隐藏潜在问题：test_greedy 和特定参数的 test_topk_sigmoid 在 AMD CI 中被跳过，理由是 MI300x 上存在已知的“内核级数值抖动”。虽然这些是已知问题且也存在于 MI325x 基线，但跳过测试意味着在这些运行器上失去了对这些功能正确性的持续验

证。

- 运行器池切换的副作用：PR 测试现在默认运行在 MI300x 而非 MI325x 上，性能基线不同。这要求工程师重新校准对 CI 结果中性能数字的期望，并注意比较基准的一致性。

- 影响：
  - 对 CI 系统的影响：显著提升了 AMD CI 的稳定性和效率。通过优先使用内网镜像注册表，减少了对外部 Docker Hub 的依赖和可能的限流影响，加速了镜像拉取。动态路由确保了 PR 测试能获得更快的反馈（使用更轻量 / 空闲的 MI300x 池），而计划任务仍能在性能更强的 MI325x 池上进行全覆盖测试。
  - 对开发团队的影响：提交 PR 的开发者将看到测试在 MI300x 硬件上运行，性能阈值已针对该硬件调整。团队需要了解新的性能基线（例如 LoRA TTFT 约慢 2 倍），并意识到一些非确定性测试在 AMD CI 中被跳过。
  - 对系统的影响：此变更纯粹是 CI/CD 基础设施和测试配置的调整，不影响 SGLang 核心运行时、模型推理逻辑或用户 API。
  - 风险标记：测试阈值放松，硬编码配置，跳过测试

关联脉络

- PR #23073 [AMD] mirror nightly images to local registry and prefer LAN pulls: 该 PR 同样涉及 AMD CI 镜像注册表的优化（将夜间镜像推送到内网注册表并优先拉取），与当前 PR 的“内网镜像注册表”变更目标部分重叠。从提交历史（提交 *a9a033b6*）可知，当前 PR 在开发过程中与 #23073 的变更产生了冲突并进行了合并解决，表明这两项工作是并行推进的 AMD CI 基础设施改进。