

PR #56601 完整报告

vllm-project/vllm

[CI] Extend Arm CPU kernel shard timeout

合并时间: 2026-09-15 16:00

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56601>

执行摘要

- 一句话: Arm CPU kernel 分片超时预算由 30 分钟放宽到 60 分钟
- 推荐动作: 值得快速浏览而非精读: 改动只有一行 timeout 与两行注释, 10 分钟内即可读完。阅读时建议关注三点设计判断: 其一, 作者用“不同套件、相同墙钟边界、无 pytest summary”作为区分“预算不足”与“测试失败”的判据, 这是处理 CI 超时类问题的可复用方法; 其二, 只放宽 kernel 分片而不是全局放宽, 体现了对影响面的克制; 其三, 验证路径被明确限定为“必须由真实 Arm runner 端到端确认”, 读者应跟进去看后续 nightly 是否真的在 60 分钟内跑完。对维护者的后续建议: 为该分片补充耗时基线记录 (例如各套件的实际耗时), 当运行时间开始逼近 60 分钟时优先考虑拆分 shard 而不是继续加 timeout; 同时此类 CI 脚本改动的合并最好留下 Arm runner 验证结论与人工审阅记录, 避免作者自审自并成为常态。

功能与动机

PR body 直言: “The Arm CPU kernel shard is being terminated by its own 30-minute wrapper while tests are still making progress.” 作者给出了完整证据链: main 构建 #88419、#88482 都在 job 启动约 32 分钟后结束, 测试命令在套件中途消失且没有 pytest 失败摘要; nightly main #88612 在两个不同的 Arm runner 上复现同一边界——首次在 31m46s 停在一个 CPU fused-MoE 用例 (此前已有一次 846 个测试的调用通过), 未改动重试也在 31m37s 停在一个 CPU attention 用例。作者据此判断: “Four runs stopping in different suites at the same wall-time boundary confirm the timeout 30m wrapper rather than a deterministic test failure.” 随后的两条 issue 评论又补齐了第 5 次 (nightly #88685, 约 31m47s) 和第 6 次 (nightly #88743, 第一个套件 240 个测试 19m00s 通过后, CPU attention 套件在 30 分钟边界被停) 复现, 并强调 “An unchanged retry would only repeat the known budget defect.”。

实现拆解

1. 变更入口定位: 改动集中在 .buildkite/scripts/hardware_ci/run-cpu-test-arm.sh 末尾按 SHARD_ID 分发的 case 语句。脚本前面通过 export -f kernel_tests model_tests serving_tests 把三个 shell 测试函数导出到容器内的 bash -c 中执行, case 分支只负责为每个分片套上 timeout 包装器, 属于典型的“预算配置点”而非测试逻辑。
2. 核心改动: 0) timeout 30m bash -c kernel_tests ;; 改为 0) timeout 60m bash -c kernel_tests ;;, 即只把 kernel 分片的测试预算翻倍; 新增两行注释解释原因 (kernel 分

片包含多个大规模 parametrized 套件，健康 Arm runner 上也可能超过 30 分钟），让后续读者不必回查 Buildkite 链接就能理解这个魔数的来历。

3. 影响面控制：1) timeout 30m bash -c model_tests 与 2) timeout 30m bash -c serving_tests 保持原样，与 PR body 中“Give only the kernel shard a 60-minute test budget. The model and serving shards keep their existing 30-minute limits.”的表述一致，避免在缺少同类复现证据的情况下全局放宽预算、掩盖其它分片的真实挂死。
4. 验证配套：变更未附带新测试，作者声明跑了 bash -n .buildkite/scripts/hardware_ci/run-cpu-test-arm.sh、对同一文件跑 pre-commit（含 shellcheck）以及 git diff --check，全部通过；同时明确“The Arm hardware suite is not available locally; the exact Buildkite shard is required for end-to-end validation.”，即端到端结论只能由真实 Arm runner 给出。commit 只有一个（a08f54a），消息中含 Co-authored-by: Sherlock 与 AI 辅助声明，说明本次改动有 AI 参与起草、仍需人工逐行确认。

关键文件：

- .buildkite/scripts/hardware_ci/run-cpu-test-arm.sh（模块 持续集成；类别 infra；类型 configuration；符号 kernel_tests, SHARD_ID）：唯一改动文件：Arm CPU 硬件 CI 的分片分发脚本，负责按 SHARD_ID 为 kernel/model/serving 三个套件设置 timeout 预算，本次把 kernel 分支由 30m 提升到 60m 并补充原因注释。

关键符号：kernel_tests

关键源码片段

.buildkite/scripts/hardware_ci/run-cpu-test-arm.sh

唯一改动文件：Arm CPU 硬件 CI 的分片分发脚本，负责按 SHARD_ID 为 kernel/model/serving 三个套件设置 timeout 预算，本次把 kernel 分支由 30m 提升到 60m 并补充原因注释。

```
# .buildkite/scripts/hardware_ci/run-cpu-test-arm.sh（分片分发逻辑，按语义重排后的片段）
# 三个测试套件以 shell 函数形式定义并导出到容器内的 bash -c 中执行，
# case 分支只负责给每个分片套一个 timeout 包装器，是纯粹的“时间预算配置点”。
export CONTAINER_NAME
export -f kernel_tests model_tests serving_tests
```

```
case "$SHARD_ID" in
  # kernel 分片包含多个大规模 parametrized 套件，在健康的 Arm runner 上也可能超过 30 分钟：
  # 此前 6 次 Buildkite 运行都在约 31 分钟的同一墙钟边界被 wrapper 杀掉，
  # 当时测试仍在持续报告 passed/skipped，没有任何断言失败或 traceback，
  # 因此判定为预算不足而非测试缺陷，只有这一分支放宽到 60m。
  0) timeout 60m bash -c kernel_tests ;;
  # model / serving 分片未观察到同类边界，继续保持 30m，
  # 以免在真实挂死场景下把 Arm runner 的占用时间放大一倍。
  1) timeout 30m bash -c model_tests ;;
  2) timeout 30m bash -c serving_tests ;;
esac
```

评论区精华

本 PR 的 review 通道几乎没有内容：唯一的 review 记录来自 `claude[bot]`，内容是 “This pull request is from a fork — automated review is disabled. A repository maintainer can comment `@claude review` to run a one-time review.”，即自动化审查因 fork 来源被禁用；`review_comments_count` 为 0，也没有任何人工 reviewer 的逐行评论，最终由作者 khluu 自己合并。真正有价值的讨论发生在 issue 评论区且全部由 khluu 自己追加：第 5 次复现 (nightly #88685) 强调 “Do not retry unchanged; the narrow 60-minute kernel-shard budget still needs exact Arm validation and human line review.”；第 6 次复现 (nightly #88743) 进一步给出量化细节——第一个套件 240 个测试 19m00s 通过，第二个 CPU attention 套件 “was still producing passes/skips when the wrapper stopped it at 30 minutes”，并再次定性为 “the existing `timeout 30m bash -c kernel_tests` boundary”。这些评论把“超时预算不足”从单点猜测升级为跨 runner、跨套件、共 6 次的可重复现象，是本次改动能被接受的主要依据。

- 以“同一墙钟边界反复复现”归因超时预算不足而非测试失败 (testing): 归因成立并据此把 kernel 分片预算提升到 60m；但归因针对的是“症状边界”，并未给出 kernel 套件在 60 分钟内必然跑完的耗时基线。
- 只放宽 kernel 分片，model/serving 维持 30 分钟 (design): 采用分片差异化预算，避免在缺少同类复现证据时全局放宽 timeout 而掩盖其它分片的真实挂死。
- fork PR 自动 review 被禁用且最终由作者自行合并 (other): 变更在无人逐行审阅的情况下合并，仅有静态检查 (bash -n、shellcheck、git diff --check) 作为把关；考虑到改动是单行数值 + 注释，直接风险有限，但流程上留下了缺口。

风险与影响

- 风险：
 1. 只缓解症状、未定位根因：改动把 wrapper 边界从 30 分钟推到 60 分钟，但没有回答 “kernel 套件为什么越来越慢”。证据显示 CPU attention 套件在 30 分钟时仍在持续产出 pass/skip，而 60 分钟是否够用完全没有基线支撑（已知的量化数据只有第一个套件 240 个测试 19m00s）；如果第二个套件本身超过 41 分钟，新预算同样会被击穿，届时表现为更昂贵的同款失败。
 2. CI 资源成本上升：kernel 分片最坏情况下会占用 Arm runner 60 分钟，是原先的两倍；一旦真的出现死锁（而非慢），排队与资源占用会被放大，而这类“真挂死”正是 timeout 包装器原本要防的场景。
 3. 验证不充分且流程把关薄弱：Arm 硬件套件本地不可用，作者只做了 bash -n、pre-commit/shellcheck、git diff --check 这类静态检查，端到端结论依赖后续 Buildkite 运行；同时 PR 来自 fork，自动 review 被禁用、无人工 review 评论，PR body 中 “A human maintainer must review every changed line” 的要求在实际流程中并未留下确认记录。
 4. 回归风险低：改动仅涉及一个 timeout 数值与两行注释，不触碰测试集合、并发度或容器参数，对非 Arm 平台和 model/serving 分片没有行为影响。- 影响：对最终用户无直接影响。对系统与团队：Arm CPU kernel 分片此前是“必然失败”的门禁，任何依赖该

job 的 Arm/CPU 相关 PR 都会被误判为不健康；本次调整让其有合理机会跑完，恢复这条流水线的可用性。影响幅度局部——同一脚本中 model 与 serving 分片仍是 30 分钟，其他硬件 CI（.buildkite/scripts/hardware_ci 下其它脚本）不受影响。成本方面，Arm runner 上该 job 的最长墙钟占用上限由约 31 分钟升至约 61 分钟，对 CI 集群容量属于轻微但持续的增量。对后续维护者的认知影响是正向的：注释把“30 分钟在 Arm 上不够”这个隐含经验固化在脚本里。 - 风险标记：CI 超时预算放宽，症状缓解而非根因修复，缺少 Arm runner 端到端验证，无人工 review 记录

关联脉络

- PR #56597 [CI/Build] Retry CPU image stream interruptions: 同样改动 . buildkite/scripts/hardware_ci/ 下的 CPU 硬件 CI 脚本，与本 PR 的 run-cpu-test-arm.sh 属同一套 CPU CI 基础设施，且都是通过调整脚本增强 CI 在异常情况下的鲁棒性。
- PR #56735 [CI] Extend DSv4 engine readiness timeout on main: 同类问题模式：Buildkite 配置中的超时预算不足导致 job 提前终止，通过拉长 timeout（600s→900s）修复，与本 PR 的 30m→60m 属于同一类 CI 超时代调。
- PR #56677 [CI] Select the supported DCP backend for PCP eval: 同样是修复 nightly main 上因配置未同步导致的 job 启动 / 运行失败，反映了近期 CI 配置类修复的集中脉络，可作为了解 CI 维护节奏的参照。