

PR #56610 完整报告

vllm-project/vllm

[ROCm][Bugfix] Fix elastic EP scaling deadlock

合并时间: 2026-09-13 07:09

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56610>

执行摘要

- 一句话: ROCm 弹性 EP 死锁修复: warm-up 延迟至 commit
- 推荐动作: 值得精读。核心设计决策是「把 collective warm-up 从通信器构建时推迟到全局静默点 (调度器暂停的 commit 阶段)」, 这背后是对 NCCL 并发 communicator 无 forward progress 保证的深刻理解 (引用了 NVIDIA/nccl#574 的官方维护者评论)。同时注意 ROCm/CUDA 行为分叉的取舍: CUDA 未复现故不引入 downtime 代价, 平台差异被严格门控。建议阅读 `defer_comm_warmup_on_rocm` 与 `_elastic_join_warmup_ctx` 的配契约, 理解「所有参与方必须同时 defer 否则 collective 无人配对」这一约束。

功能与动机

Elastic EP 在 ROCm 上扩缩容时偶发 hang, AMD CI 中可稳定复现 (如 Buildkite amd-ci build 12870)。PR body 说明根因定位过程: "This was root-caused via RCCL logs: during elastic-EP prepare the standby/joining groups are built while the engine is still serving, so a new communicator warm-up (which does an all-reduce) might run concurrently with forward-pass collectives on the existing communicators, causing a race and potentially a deadlock if different collectives are enqueued in different order on different ranks." 同时指出本 PR 取代 draft #53010, 因为后者把 deferral 限制在 ROCm + cudagraph, "does not cover all cases (including eager) in which the hangs occur"。

实现拆解

整个修复围绕「把通信器 warm-up 从构建时推迟到全局静默点 (commit)」展开, 共 4 个文件、5 步:

1. `pynccl.py` 引入线程局部 defer 原语: 新增 `_warmup_tls = threading.local()`、`_warmup_deferred()` 与 `defer_comm_warmup_on_rocm()` context manager。非 ROCm 平台直接透传; `_init_comm` 在 `ncclCommInitRank` 之后检查 `_warmup_deferred()`, 为真则跳过一元素 `all_reduce` 与 `stream.synchronize()`。这一步把「构建通信器」与「warm-up 校验」解耦, 是后续所有平台门控的基础。
2. `standby_state.py` 存量 rank 侧延迟: `create_standby_groups` 将 DP/EP/EPLB 三个 `_init_stateless_group` 调用统一包进 `defer_comm_warmup_on_rocm()`。此时 engine 仍在服务, standby 组只建通信器不做 warm-up。

3. `parallel_state.py` 新加入 rank 侧对称延迟：新增 `_elastic_join_warmup_ctx()`，在 `envs.VLLM_ELASTIC_EP_SCALE_UP_LAUNCH` 存在时返回 `defer_comm_warmup_on_rocm()`，否则 `nullcontext()`；`initialize_model_parallel` 中 `_DP/_EP/_EPLB` 的 `stateless` 初始化全部包进该 `ctx`。关键点在于集体通信必须所有参与方一致：存量 rank 已 `defer`，若 `scale-up` 新 worker 仍执行 `warm-up`，`all_reduce` 将无人配对而 `hang`。
4. `elastic_execute.py` 调整调用时机：`prepare_reconfiguration` 与 `prepare_new_worker` 在 `ROCM` 上跳过 `_warm_target_groups`（避免服务中死锁）；`commit_scale_up/commit_scale_down` 在调度器暂停后补回——非 `reuse` 路径由 `warm_and_capture()` 的 `dummy forward` 顺带触发 `deferred warm-up`，`_can_reuse_fused_moe_kernel()` (`nixl_ep`) 路径则显式调用 `_warm_target_groups(get_dp_group(), get_ep_group())`。
5. 测试与验证配套：未新增定向单测，回归依赖现有 `tests/distributed/test_elastic_ep.py`（Elastic EP Scaling 测试组）。PR body 给出 4x MI300X 实测：改动前 `enforce_eager_light/enforce_eager_heavy` 各失败 1 次，改动后 5 项全部通过（跑两遍），GSM8K accuracy 无变化。

关键文件：

- `vllm/distributed/device_communicators/pynccl.py`（模块 设备通信；类别 `source`；类型 `core-logic`；符号 `_warmup_deferred`, `defer_comm_warmup_on_rocm`, `_init_comm`）：机制核心：线程局部 `defer` 标志 + `context manager`，`_init_comm` 依据标志跳过 `warm-up` `all_reduce`，是平台门控与延迟预热的基础设施。
- `vllm/distributed/parallel_state.py`（模块 并行状态；类别 `source`；类型 `core-logic`；符号 `_elastic_join_warmup_ctx`, `initialize_model_parallel`）：以对称方式处理 `scale-up` 新加入 worker： `_elastic_join_warmup_ctx` 由环境变量门控，保证 `joining rank` 与 `存量 rank` 的 `defer` 行为一致，否则 `warm-up collective` 无人配对。
- `vllm/distributed/elastic_ep/standby_state.py`（模块 弹性 EP；类别 `source`；类型 `dependency-wiring`；符号 `create_standby_groups`）：存量 rank 侧在 `prepare` 阶段跳过 `standby DP/EP/EPLB` 组 `warm-up` 的落点，是死锁根因的直接对应改动。
- `vllm/distributed/elastic_ep/elastic_execute.py`（模块 弹性 EP；类别 `source`；类型 `dependency-wiring`；符号 `prepare_reconfiguration`, `commit_scale_up`, `commit_scale_down`, `prepare_new_worker`）：`prepare` 与 `commit` 各路径的平台门控：`prepare` 中跳过服务期 `warm-up`，`commit` 调度器暂停后按 `reuse` 路径显式补齐，是保证 `deferred warm-up` 不丢失的关键。

关键符号：`defer_comm_warmup_on_rocm`, `_warmup_deferred`, `_elastic_join_warmup_ctx`, `_init_comm`, `create_standby_groups`, `prepare_reconfiguration`, `commit_scale_up`, `commit_scale_down`, `prepare_new_worker`

关键源码片段

`vllm/distributed/device_communicators/pynccl.py`

机制核心：线程局部 defer 标志 + context manager, `_init_comm` 依据标志跳过 warm-up all_reduce，是平台门控与延迟预热的基础设施。

```
# 线程局部开关：defer 只影响当前线程的通信器构建
_warmup_tls = threading.local()
```

```
def _warmup_deferred() -> bool:
    return getattr(_warmup_tls, "deferred", False)
```

```
@contextlib.contextmanager
def defer_comm_warmup_on_rocm():
    """ROCM 上构建通信组时跳过 warm-up 集体通信。
```

背景：弹性 EP 在 prepare 阶段构建 standby 组时 engine 仍在服务，新通信器的 warm-up all_reduce 会与现有通信器上的 forward 集体通信并发执行；不同 rank 上 collectives 的入队顺序不同可能互相等待，形成死锁。因此 warm-up 推迟到 commit 阶段（调度器暂停）再执行。

```
"""
```

```
from vllm.platforms import current_platform
```

```
# CUDA 上从未复现该死锁，且延迟 warm-up 会增加重配置停机时间，
# 因此非 ROCm 平台直接透传，保持原有行为
if not current_platform.is_rocm():
    yield
    return
```

```
prev = _warmup_deferred()
_warmup_tls.deferred = True
try:
    yield
finally:
    # 恢复外层状态，支持 context manager 嵌套使用
    _warmup_tls.deferred = prev
```

```
def _init_comm(self, device: int | str | torch.device) -> None:
    """在 device 上根据已解析的 unique_id / rank / world_size 创建通信器。"""
    # device 归一化：统一转为 torch.device 对象
    if isinstance(device, int):
        device = torch.device(f"cuda:{device}")
    elif isinstance(device, str):
        device = torch.device(device)
    assert isinstance(device, torch.device)
    self.device = device

    # nccl 通信器和流都使用该 device
    with torch.accelerator.device_index(device.index):
```

```

self.comm = self.nccl.ncclCommInitRank(
    self.world_size, self.unique_id, self.rank
)

if _warmup_deferred():
    # 关键改动：defer 期间跳过 warm-up 一元素 all_reduce。
    # 此刻同步 all_reduce 会与服务中的 forward 集体通信竞争并死锁，
    # 真正的 warm-up 推迟到 commit 阶段由 dummy forward 或
    # _warm_target_groups 触发
    return

stream = current_stream()
# 一元素 all_reduce 作为通信器就绪校验
data = torch.zeros(1, device=device)
self.all_reduce(data)
stream.synchronize()
del data

```

vllm/distributed/elastic_ep/standby_state.py

存量 rank 侧在 prepare 阶段跳过 standby DP/EP/EPLB 组 warm-up 的落点，是死锁根因的直接对应改动。

```

# create_standby_groups 核心改动 (standby_state.py)
# 前置：按 new_dp_size / tp_size / pp_size 计算 standby 组 rank 布局
all_ranks = torch.arange(new_world_size_across_dp).reshape(
    -1, new_dp_size, pp_size, tp_size
)
# DP 组按 rank 横切、EP 组按 DP*TP 聚合，布局逻辑与改动前相同
standby_dp_ranks = all_ranks.transpose(1, 3).reshape(-1, new_dp_size).unbind(0)
standby_dp_ranks = [x.tolist() for x in standby_dp_ranks]
standby_ep_ranks = all_ranks.transpose(1, 2).reshape(
    -1, new_dp_size * tp_size
).unbind(0)
standby_ep_ranks = [x.tolist() for x in standby_ep_ranks]

# 关键改动：standby 组在 engine 仍服务时构建，把每个通信器的 warm-up
# 统一推迟到 commit 阶段（调度器暂停）执行，避免与服务中的 forward
# collectives 并发导致死锁；非 ROCm 平台该 context manager 为空操作
with defer_comm_warmup_on_rocm():
    _STANDBY_DP = _init_stateless_group(
        standby_dp_ranks, "dp", master_ip, backend, coord_store=coord_store
    )
    _STANDBY_EP = _init_stateless_group(
        standby_ep_ranks, "ep", master_ip, backend,
        coord_store, use_all2all=use_all2all,
    )
if enable_eplb:
    _STANDBY_EPLB = _init_stateless_group(
        standby_ep_ranks, "eplb", master_ip, backend,

```

```
        coord_store=coord_store,  
    )
```

vllm/distributed/elastic_ep/elastic_execute.py

prepare 与 commit 各路径的平台门控：prepare 中跳过服务期 warm-up，commit 调度器暂停后按 reuse 路径显式补齐，是保证 deferred warm-up 不丢失的关键。

```
# commit 阶段补做 warm-up (elastic_execute.py 改动点)  
def commit_scale_up(self, is_existing_worker: bool) -> None:  
    if is_existing_worker:  
        self.broadcast_expert_mapping()  
        retired_groups = self.switch_and_prepare()  
    else:  
        mapping = self.receive_expert_mapping()  
        self.worker.model_runner.setup_eplb_from_mapping(mapping)  
  
    if not self._can_reuse_fused_moe_kernel():  
        # 非 reuse 路径：warm_and_capture 的 dummy forward 会顺带触发  
        # 被推迟的 dp / ep 组 warm-up all_reduce  
        self.warm_and_capture()  
    elif current_platform.is_rocm():  
        # reuse 路径 (nixl_ep) 跳过 warm_and_capture，ROCM 上需显式补上  
        # 被推迟的 warm-up，此时调度器已暂停，不会与服务中的 collectives 竞争  
        self._warm_target_groups(get_dp_group(), get_ep_group())  
  
    self._perform_eplb_resuffle(async_op=True)  
    if is_existing_worker:  
        self._start_group_cleanup(retired_groups)  
  
# commit_scale_down 采用相同模式：非 reuse 走 warm_and_capture，  
# ROCm + reuse 走显式 _warm_target_groups 补齐延迟的 warm-up  
def commit_scale_down(self, new_dp_size: int, removing: bool) -> None:  
    self.perform_scale_down_eplb_resuffle(new_dp_size)  
    if removing:  
        self.switch_and_remove()  
    else:  
        retired_groups = self.switch_and_prepare()  
        if not self._can_reuse_fused_moe_kernel():  
            self.warm_and_capture()  
        elif current_platform.is_rocm():  
            self._warm_target_groups(get_dp_group(), get_ep_group())  
        self._start_group_cleanup(retired_groups)
```

评论区精华

核心交锋一：是否应为 CUDA 延迟 warm-up (design)

itayalroy: "The warmup hang is a AMD specific issue, we have not seen a single hang with CUDA, so we will not defer warmup for CUDA, this degrades reconfiguration downtime"

作者最初实现平台无关，理由是担心机制通用性，但在回应中认可差异并改为 ROCm 限定：CUDA 上并发 communicator 的 kernel 可能同时下发执行，而 ROCm 上可能被串行化。作者还引用了 NCCL maintainer 的观点佐证该问题在 NCCL 语义下具有普遍风险。

核心交锋二：PR 是否应拆分 (design)

itayalroy: "I think there should be 3 separate PRs here, the shared experts bugfix and the scale down bugfix should be reviewed separately, they are unrelated to the AMD warmup issue"

作者回应 shared experts 修复只是一行 typo、拆分论据较弱；最终合入版本已收敛为单一 ROCm warm-up 主题。

细节疑问：EPLB 分支语义 (question)

itayalroy 在 `standby_state.py` 102 行评论 "EPLB is always enabled in Elastic EP"；作者澄清这是既有代码（原第 93 行），本次只是缩进进 defer 上下文管理器，语义未变。

最终批准

AndreasKaratzas: "These changes pretty much touch only ROCm and also they are proven to render the test green again."

- warm-up 延迟是否应限定 ROCm (design): 改为 `current_platform.is_rocm()` 门控，CUDA 行为不变，满足维护者的停机时间诉求。
- 并发 communicator 死锁根因与 NCCL 语义 (correctness): 确认死锁机制在 NCCL 语义下具有普遍性；修复选择在调度器暂停窗口 (commit) 执行 warm-up，而非依赖 NCCL 行为。
- PR 拆分: shared experts / scale down 修复与 warm-up 分离 (design): 最终合入版本收敛为单一 ROCm warm-up 延迟主题，其他修复未出现在最终 diff 中。
- EPLB 分支是否应保留 `enable_eplb` 判断 (question): 确认为既有逻辑仅缩进调整，无行为变化。

风险与影响

• 风险:

1. 环境变量契约风险 (`parallel_state.py`) : `_elastic_join_warmup_ctx` 依赖 `VLLM_ELASTIC_EP_SCALE_UP_LAUNCH` 在 scale-up 启动时被正确注入。若该变量缺失，joining worker 不会 defer，而存量 rank 已 defer，warm-up all_reduce 无人配对，会从偶发 hang 变成确定性 hang。
2. defer 后漏 warm-up 风险 (`elastic_execute.py`) : warm-up 被推迟后，必须保证 commit 路径必然执行 `warm_and_capture()` 或 `_warm_target_groups()`。当前两个 commit 分支都显式覆盖，但未来若新增 `can_reuse_fused_moe_kernel()` 后端或修改 `warm_and_capture()` 调用条件，可能漏掉 deferred warm-up，导致首轮使用未初始化通信器。

3. 线程局部性限制 (pyncccl.py) : `_warmup_tls` 是 `threading.local()`, `defer` 只对当前线程生效。当前 `_init_stateless_group` 为同步调用, 风险低; 若未来异步线程创建通信器则会绕过 `defer`, 且与存量 `rank` 的 `defer` 状态不一致。
4. 回归面收敛性: 非 ROCm 平台与普通 (非 elastic EP) 初始化路径完全不进入 `defer` 分支, CUDA 行为零变化; 但改动位于分布式通信核心路径, 仍需依赖 Elastic EP Scaling 集成测试持续覆盖。- 影响: 用户侧: ROCm (MI 系列) 上 Elastic EP 扩缩容不再偶发 hang, 且覆盖 `eager` 模式 (相比 #53010 仅覆盖 `cudagraph` 是明显改进); CUDA 用户行为完全不变, 不增加重配置 `downtime`。系统侧: `warm-up` 从 `prepare` (服务中执行) 挪到 `commit` (调度器暂停窗口), 重配置路径不再与服务中的 `forward collectives` 竞争 GPU 队列; 4x MI300X 上 Elastic EP Scaling 测试组 5/5 稳定通过, GSM8K accuracy 无变化。团队侧: 确立了一个平台差异门控 + 延迟预热到静默点的模式, 为后续跨平台通信初始化策略提供了参照。- 风险标记: 分布式通信核心路径变更, 依赖 `VLLM_ELASTIC_EP_SCALE_UP_LAUNCH` 环境变量契约, 无新增定向单测, 线程局部开关有跨线程穿透限制

关联脉络

- PR #54985 [Elastic EP] Reuse CUDA graphs across reconfiguration: 同一功能线 (Elastic EP 重配置优化)。本 PR 的 `commit_scale_up/commit_scale_down` 修改必须兼容其引入的 `_can_reuse_fused_moe_kernel()` (`nixl_ep`) 路径: `reuse` 路径跳过 `warm_and_capture`, 因此 ROCm 上需显式补做 `warm-up`。
- PR #56323 [6/N][warmup][DSv4] Migrate sampling, and DFlash JIT kernels: 同一「把预热从运行时挪到安全窗口」的主题脉络: 该系列把 JIT 内核预热迁移到共享 `warmup` 框架, 本 PR 把通信器 `warm-up` 推迟到调度器暂停的 `commit` 阶段, 共同体现 vLLM 对启动 / 重配置阶段预热时机的一致设计。
- PR #50178 [9/N][warmup][DSv4] Migrate MHC TileLang kernels: `warmup` 迁移系列的另一环, 与上述 #56323 相同, 佐证 `warmup` 时机治理是近期跨模块的持续演进方向。