PR #47206 完整报告

vllm-project/vllm

[AMD][Bugfix][EPLB] Fix elastic EP scaling accuracy on ROCm

合并时间：2026-07-25 04:35

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47206

执行摘要

- 一句话：修复 ROCm 上弹性 EP 缩放准确率并消除冗余重排通信开销
- 推荐动作：值得精读，尤其是 rank_load_imbalance 的 replica-aware 不均衡度计算和 ROCm 平台专有条件的取舍。设计者对分布式数值一致性的强调和 review 中的集体决策讨论具有借鉴意义。

功能与动机

分布式测试 test_elastic_ep.py 在 ROCm 上 GSM8K 准确率从正常值崩溃至 0.19/0.016，冗余重排导致约 16s 的横向 all-to-all 颠簸使引擎饥饿，详见 PR body。

实现拆解

1. 防止 expert 拓扑缓冲区被清零：在 vllm/model_executor/layers/fused_moe/routed_experts.py 的 update_expert_map_info() 中，将五个 expert 拓扑缓冲区（_expert_map、expert_mask、expert_global_to_physical、expert_physical_to_global、expert_local_to_global）的 register_buffer 调用添加 persistent=False 参数。这些缓冲区是每 rank 派生的元数据，不应出现在 state_dict() 中，从而避免 ROCm dummy 权重初始化的 zero_() 和 batch_transfer_weights() 的覆盖。
2. 计算负载不均衡并跳过低增益重排：在 vllm/distributed/eplb/eplb_state.py 的 rearrange() 中增加跳过逻辑。定义 rank_load_imbalance() 函数，根据全局 expert 负载和物理到逻辑映射计算每个 rank 的负载不均衡度（考虑 replica 分摊）。比较当前映射和新映射的不均衡度相对改进，若改进 <5% 则设置 skip_rearrange = True。
3. 条件性跳过重排：skip_rearrange 为 True 时跳过 rearrange_expert_weights_inplace() 和 _commit_eplb_maps()，避免不必要的 all-to-all 通信。该逻辑仅在 ROCm 平台、非 profile 模式、非缩放场景（rank_mapping is None）且当前映射有效时生效。

关键文件：

- vllm/distributed/eplb/eplb_state.py（模块 负载均衡；类别 source；类型 core-logic；符号 rank_load_imbalance）：核心逻辑修改：定义 rank_load_imbalance 函数，在 rearrange() 中插入 ROCm 专用跳过低增益重排的逻辑，避免 16s 冗余通信。
- vllm/model_executor/layers/fused_moe/routed_experts.py（模块 MoE 层；类别 source；类型 data-contract）：数据契约修改：将 expert 拓扑缓冲区注册为非持久化，避免 ROCm dummy 权重初始化清零和 batch_transfer_weights 覆盖。

关键符号：rank_load_imbalance, update_expert_map_info

关键源码片段

vllm/distributed/eplb/eplb_state.py

核心逻辑修改：定义 rank_load_imbalance 函数，在 rearrange() 中插入 ROCm 专用跳过低增益重排的逻辑，避免 16s 冗余通信。

```
# 仅在 ROCm、非 profile、非缩放、且当前映射有效时启用跳过检查
skip_rearrange = False
if (
    current_platform.is_rocm()
    and not is_profile
    and rank_mapping is None
    and bool((eplb_model_state.physical_to_logical_map >= 0).all())
):
    logical_loads = global_expert_load_window.float()
    ep_size = ep_group.size()

    def rank_load_imbalance(
        mapping: torch.Tensor,
        logical_loads: torch.Tensor = logical_loads,
        ep_size: int = ep_size,
    ) -> float:
        # 将映射移动到负载张量所在设备并转为 long
        mapping = mapping.to(
            device=logical_loads.device,
            dtype=torch.long,
        )
        # 统计每个 physical expert 被映射到的 replica 数量
        replica_counts = torch.zeros_like(logical_loads)
        replica_counts.scatter_add_(
            dim=1,
            index=mapping,
            src=torch.ones_like(mapping, dtype=logical_loads.dtype),
        )
        # 将每个 logical expert 的负载均匀分摊到其所有 replica
        loads_per_replica = torch.gather(
            logical_loads / replica_counts.clamp_min(1),
            dim=1,
            index=mapping,
        )
        # 按 ep_size 重组并求和得到每个 rank 的总负载
        loads_per_rank = loads_per_replica.reshape(
            logical_loads.shape[0], ep_size, -1
        ).sum(dim=(0, 2))
        mean_load = loads_per_rank.mean()
        if mean_load == 0:
            return 1.0
```

```
            return (loads_per_rank.max() / mean_load).item()

        current_imbalance = rank_load_imbalance(
            eplb_model_state.physical_to_logical_map
        )
        proposed_imbalance = rank_load_imbalance(
            new_physical_to_logical_map
        )
        relative_improvement = (
            current_imbalance - proposed_imbalance
        ) / current_imbalance
        skip_rearrange = relative_improvement < 0.05
        if skip_rearrange and is_main_rank:
            logger.info(
                "[EPLB] Skip rearrange: imbalance %.4f -> %.4f "
                "(no material gain)",
                current_imbalance,
                proposed_imbalance,
            )

    if not skip_rearrange:
        rearrange_expert_weights_inplace(
            eplb_model_state.physical_to_logical_map,
            new_physical_to_logical_map,
            eplb_model_state.model.expert_weights,
            eplb_model_state.expert_buffer,
            ep_group,
            eplb_model_state.communicator,
            is_profile,
            rank_mapping,
        )
        if not is_profile:
            _commit_eplb_maps(
                eplb_model_state,
                new_physical_to_logical_map=new_physical_to_logical_map,
            )
```

评论区精华

线程 1：tlrmchlsmth 担心不同 rank 的跳过决策因数值差异导致死锁。okorzh-amd 解释所有输入（global_expert_load_window 已 all-reduce、physical_to_logical_map 为全局不变式、new_physical_to_logical_map 来自确定性策略）均 rank 一致，不会分歧。后续更新添加了 all_reduce 确保集体决策。

线程 2：tlrmchlsmth 建议简化代码并考虑在非 AMD 硬件也启用跳过重排（因增加了一次 all-reduce）。最终决策保留了 ROCm 专用，但代码块被大幅简化。

线程 3：itayalroy 指出 batch_transfer_weights() 排除了 expert_map 但未排除 expert_mask，导致新 rank 的 expert_mask 被发送者覆盖。okorzh-amd 确认并最初添加了

排除，但最终通过 persistent=False 从根本上避免此问题。

- 跳过重排的决策分歧与死锁风险 (correctness): 确认输入一致无分歧，且后续代码中通过注释说明；review 后保持当前实现。
- 是否在非 AMD 硬件也启用跳过 (design): 最终保留 ROCm 专用，但代码块被简化，避免引入全局行为变化。
- batch_transfer_weights 中 expert_mask 排除不完整 (correctness): 通过 persistent=False 使这些缓冲区完全不出现在 state_dict 中，从根本上避免覆盖。

风险与影响

- 风险：数值一致性与集体决策：skip_rearrange 的当前实现依赖本地浮点计算，虽作者声称输入一致不会分歧，但未使用 all_reduce 同步决策，在极端浮点误差下存在不同 rank 决策不一致的风险，可能导致集体通信调用（rearrange_expert_weights_inplace）在部分 rank 上被跳过而其他 rank 不跳过的死锁。 平台隔离：跳过逻辑限定于 ROCm，非 AMD 平台无变更，但引入平台分支增加了维护成本。 缓冲区持久性更改：将 expert 拓扑缓冲区设为 persistent=False 使其不参与 state dict，不影响模型正确的加载 / 保存（它们是派生的），但需确保所有使用这些缓冲区的路径均通过 update_expert_map_info() 重建。
- 影响：对用户：ROCm 用户进行弹性 EP 缩放时，GSM8K 准确率从 0.19/0.016 恢复至 ≥0.58，且 Sync-EPLB 的冗余重排被消除，吞吐明显提升。非 ROCm 用户无直接影响。对系统：减少每次 step_interval 可能触发的 16s all-to-all 通信，降低网络压力，提升整体调度效率。 对团队：明确了 ROCm dummy 权重初始化与 CUDA 的行为差异（整数 tensor 清零），以及 EPLB 重排频率对实际负载平衡的敏感性。
- 风险标记：数值决策未 all_reduce 同步，ROCm 专用分支增加维护成本，缓冲区持久性变更需确保重建路径

关联脉络

- 暂无明显关联 PR