PR #43637 完整报告

vllm-project/vllm

[Feature] Detect all2all peer fault with fault tolerance backend and prevent corrupted output

合并时间：2026-07-01 00:00

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43637

执行摘要

- 一句话：EP all2all 通信故障检测与异常终止
- 推荐动作：该 PR 是 fault tolerance 关键基础设施，值得所有 MoE 部署团队精读。设计决策（类变量使用、开关控制、异步安全性）体现了 real system 的权衡。建议关注后续 #44428 的 recovery 实现。

功能与动机

nixl_ep 默认启用 fault tolerance，当 rank 超时后内核会设置 mask 继续运行，但当前前向传播可能已产生损坏 token 并返回给用户（PR body："currently, under the nixl_ep backend, this corrupted token is returned to EngineCore and ultimately to the end user"）。此 PR 旨在拦截此场景，保证用户不会收到损坏输出。

实现拆解

1. 定义抽象接口：在 All2AllManagerBase（base_device_communicator.py）中添加 query_active_mask、query_fault 抽象方法和 support_fault_tolerance 属性，默认返回 False。
2. 实现DeepEPLL后端：在DeepEPLLAll2AllManager（all2all.py）中增加类变量_buffer、_mask、_last_mask，通过 low_latency_query_mask_buffer 获取 mask，比较当前与上次 mask 判定故障。support_fault_tolerance 设为 False，对应 enable_shrink 参数。
3. 实现 NixlEP 后端：在 NixlEPAll2AllManager（all2all.py）中增加类似类变量，通过 query_mask_buffer 获取 mask，支持弹性 EP 的形状自适应，support_fault_tolerance 设为 True。
4. 集成到 GPU 运行器：在 AsyncGPUModelRunnerOutput（gpu_model_runner.py）中新增 check_ep_fault 参数和 _has_fault 字段。在异步拷贝阶段调用 query_fault()，在 get_output() 检测到故障时查询当前 mask 并抛出 RuntimeError。在 GPUModelRunner 初始化时根据配置启用故障检查。
5. 异常传播处理：在 multiproc_executor.py 的 enqueue_output 中捕获 get_output() 抛出异常，将错误转换为 FAILURE 响应，避免工作进程崩溃。
6. 公共函数导出：将 all2all_utils.py 中的 _get_ep_all2all_manager 重命名为 get_ep_all2all_manager，供 GPU 运行器调用。

关键文件：

- vllm/distributed/device_communicators/all2all.py（模块 通信层；类别 source；类型 core-logic；符号 query_active_mask, query_fault）：核心实现文件，为 DeepEPLL 和 NixlEP 后端添加故障检测方法 query_active_mask、query_fault 及类变量。
- vllm/distributed/device_communicators/base_device_communicator.py（模块 通信层；类别 source；类型 core-logic；符号 query_active_mask, query_fault）：抽象基类增加故障检测抽象方法，定义接口契约。
- vllm/v1/worker/gpu_model_runner.py（模块 GPU 运行器；类别 source；类型 data-contract）：集成故障检测到模型运行器，在异步拷贝和 get_output 中检查故障并抛出异常。
- vllm/model_executor/layers/fused_moe/all2all_utils.py（模块 MoE 工具；类别 source；类型 data-contract；符号 _get_ep_all2all_manager, get_ep_all2all_manager）：将 _get_ep_all2all_manager 重命名为 get_ep_all2all_manager，供外部模块调用。
- vllm/v1/executor/multiproc_executor.py（模块 执行器；类别 source；类型 core-logic）：捕获 get_output 抛出的异常，避免工作进程崩溃并转为 FAILURE 响应。

关键符号：query_active_mask, query_fault, get_ep_all2all_manager

关键源码片段

vllm/distributed/device_communicators/all2all.py

核心实现文件，为 DeepEPLL 和 NixlEP 后端添加故障检测方法 query_active_mask、query_fault 及类变量。

```
# vllm/distributed/device_communicators/all2all.py

class DeepEPLLAll2AllManager(DeepEPAll2AllManagerBase):
    """DeepEP Low-Latency 后端 all2all 通信管理器，支持故障检测"""
    _buffer: Any = None # 类级缓存 buffer 句柄，per-GPU
    _mask: torch.Tensor | None = None # 可复用的 GPU mask 张量
    _last_mask: torch.Tensor | None = None # 上次查询的 mask 快照

    def __init__(self, cpu_group, tcp_store_group=None):
        super().__init__(cpu_group, tcp_store_group)
        self.support_fault_tolerance = False # 关闭 FT，待 recovery PR 启用

    def query_active_mask(self) -> torch.Tensor:
        """向 DeepEP 内核查询当前活跃 rank mask"""
        buf = DeepEPLLAll2AllManager._buffer
        assert buf is not None
        if DeepEPLLAll2AllManager._mask is None:
            DeepEPLLAll2AllManager._mask = torch.zeros(
                self.world_size, device="cuda", dtype=torch.int32
            )
        buf.low_latency_query_mask_buffer(DeepEPLLAll2AllManager._mask)
        return DeepEPLLAll2AllManager._mask

    def query_fault(self) -> torch.Tensor:
```

```
        """比较当前 mask 与 last_mask，返回是否有 rank 退出"""
        current = self.query_active_mask()
        if DeepEPLLAll2AllManager._last_mask is None:
            DeepEPLLAll2AllManager._last_mask = torch.zeros_like(current)
        has_fault = (current != DeepEPLLAll2AllManager._last_mask).any()
        return has_fault

class NixlEPAll2AllManager(All2AllManagerBase):
    """NIXL EP 后端 all2all 通信管理器，默认支持故障检测"""
    _buffer: _NixlEPBufferState | None = None
    _mask: torch.Tensor | None = None # 完整 mask 张量 (max_num_ep_ranks)
    _last_mask: torch.Tensor | None = None # 上次活跃 mask 快照

    def __init__(self, cpu_group, tcp_store_group=None):
        super().__init__(cpu_group, tcp_store_group)
        self.support_fault_tolerance = True # NIXL 默认启用 FT
        self.max_num_ep_ranks = envs.VLLM_NIXL_EP_MAX_NUM_RANKS

    def query_active_mask(self) -> torch.Tensor:
        state = NixlEPAll2AllManager._buffer
        assert state is not None
        if NixlEPAll2AllManager._mask is None:
            NixlEPAll2AllManager._mask = torch.zeros(
                self.max_num_ep_ranks, device="cuda", dtype=torch.int32
            )
        state.buffer.query_mask_buffer(NixlEPAll2AllManager._mask)
        return NixlEPAll2AllManager._mask[: state.active_ep_size]

    def query_fault(self) -> torch.Tensor:
        current = self.query_active_mask()
        last = NixlEPAll2AllManager._last_mask
        if last is None or last.shape != current.shape:
            NixlEPAll2AllManager._last_mask = torch.zeros_like(current)
            last = NixlEPAll2AllManager._last_mask
        has_fault = (current != last).any()
        return has_fault
```

vllm/distributed/device_communicators/base_device_communicator.py

抽象基类增加故障检测抽象方法，定义接口契约。

```
# vllm/distributed/device_communicators/base_device_communicator.py

class All2AllManagerBase:
    """all2all 通信管理器的抽象基类"""

    def __init__(self, cpu_group, tcp_store_group=None):
        # ... 原有初始化逻辑
        self.support_fault_tolerance = False # 子类可覆盖

    def query_active_mask(self) -> torch.Tensor:
```

```
        """返回当前活跃的 rank mask (GPU tensor)。"""
        raise NotImplementedError

    def query_fault(self) -> torch.Tensor:
        """返回 has_fault scalar (GPU tensor)。"""
        raise NotImplementedError
```

vllm/v1/worker/gpu_model_runner.py

集成故障检测到模型运行器，在异步拷贝和 get_output 中检查故障并抛出异常。

```
# vllm/v1/worker/gpu_model_runner.py

class AsyncGPUModelRunnerOutput(AsyncModelRunnerOutput):
    def __init__(self, ..., check_ep_fault: bool = False):
        # ... 原有初始化
        self._has_fault: torch.Tensor | None = None

        # 异步拷贝阶段（已在 async_output_copy_stream 上）
        if check_ep_fault:
            has_fault = get_ep_all2all_manager().query_fault()
            self._has_fault = has_fault.to("cpu", non_blocking=True)

    def get_output(self) -> ModelRunnerOutput:
        self.async_copy_ready_event.synchronize()
        # ...
        if self._has_fault is not None and self._has_fault.item():
            mask = get_ep_all2all_manager().query_active_mask()
            raise RuntimeError(
                "Fault detected in EP all2all communication: "
                "one or more ranks timed out during dispatch/combine. "
                f"Mask: {mask.cpu().tolist()}"
            )
        return output
```

评论区精华

- 类变量 vs 实例变量：gemini-code-assist[bot] 指出类变量在多模型 / 推测解码场景下不安全。作者 fangyuchu 解释这些变量反映 GPU 硬件状态（per-GPU），且当前不支持同一 GPU 多 EP 配置，评审者最终接受该设计。
- DeepEP LL 的 Fault Tolerance 开关：SageMoore 询问为何设置 support_fault_tolerance = False。fangyuchu 回应因为 recovery 机制尚未实现（见 PR #44428），所以故意禁用 enable_shrink。tlrmchlsmth 建议明确变量命名，最终保持现状。
- 异步竞争时序：tlrmchlsmth 担心 query_fault() 和 get_output() 在异步调度中可能重叠，导致读取不稳定。fangyuchu 最初认为不存在，后承认可能，并调整设计：query_fault() 不再返回 current_mask，改在异常分支单独查询，避免 GPU 张量的异步竞争。
- 类变量安全性问题 (design): fangyuchu 回应 class-level 变量是故意的，因为 mask 反映 GPU 硬件状态（per-GPU），且当前 vLLM 拓扑不支持同一 GPU 上多个 EP 实例；评审者

最终接受该解释。

- DeepEP LL Fault Tolerance 开关 (design): fangyuchu 解释 recovery 尚未实现，此 PR 只做检测，待 PR #44428 完成后再启用。tlrmchlsmth 确认该决策并建议变量更名表达用途（未改）。
- 异步调度中 fault 检测竞争 (correctness): 通过分离故障检测与 mask 获取，query_fault 只返回 has_fault scalar，异常时再 query_active_mask。评审者认可。

风险与影响

- 风险：
  - all2all.py：类变量共享若未来拓扑扩展可能引发误判；query_active_mask 执行 GPU 内核调用可能带来额外延迟。
  - gpu_model_runner.py：check_ep_fault 在每次模型执行的前向路径中添加 GPU→CPU 读取，对正常无故障场景引入少许开销（一次布尔拷贝）。但 PR 将其控制在异常路径，并使用 non_blocking 避免阻塞。
  - multiproc_executor.py：异常捕获将任何 get_output 抛出的错误转为 FAILURE，可能掩盖其他非故障异常，但通过日志保留细节。
  - 缺乏测试覆盖：目前没有自动化测试验证 fault 注入场景，仅靠手动测试，存在回归风险。
- 影响：
  - 用户：之前在 nixl_ep 后端发生 peer rank 超时时可能收到损坏 token，现在会抛出 RuntimeError，避免无声错误。用户体验变为显式错误，但需要用户配置合适的超时或容错策略。
  - 系统：为整个 fault tolerance 框架打下基础，后续可与 PR #44428 的 recovery 联动。正常路径性能影响极小，仅在异步拷贝流上多一次 GPU→CPU 拷贝（一个标量）。
  - 团队：需要关注后续 recovery PR 的协作，保证接口兼容。
  - 风险标记：缺少自动化测试，类变量共享风险，异步时序竞争

关联脉络

- 暂无明显关联 PR