PR #37047 完整报告

sgl-project/sglang

fix(vlm): contain multimodal feature transport failures

合并时间：2026-09-01 13:46

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37047

执行摘要

- 一句话：修复多模态特征传输失败导致的进程级崩溃，实现跨 rank 错误共识与资源安全释放。
- 推荐动作：本 PR 是一个 高价值的关键 bug 修复，强烈建议精读。它解决了一个可能导致生产环境进程崩溃的严重问题，并引入了严谨的分布式错误处理模式。核心设计决策——延迟错误、跨 rank 共识、防御性资源释放——对于理解 SGLang 如何处理多模态输入失败非常有启发性。建议重点审查以下方面： 1) scheduler.py 中 _materialize_cuda_vmm_inputs 和 _gather_vmm_materialization_errors 的错误收集与同步逻辑； 2) mm_utils.py 中 ShmPointerMMData 的 __setstate__ 和 close_and_unlink 方法如何安全处理各种失败场景； 3) schedule_batch.py 中 release_transport_proxies 的实现及其在 from_processor_output 和 set_finish_with_abort 中的调用点。新增的单元测试覆盖了核心失败路径，值得参考。

功能与动机

当前实现中，一个 rank-local 的 SHM open、clone、reshape 或 cleanup failure 会逃逸调度器的接收循环。这可能导致一个调度器 rank 崩溃，而其 peer rank 继续使用格式错误或资源受限的多模态输入处理该请求，最终转化为进程级故障。PR body 明确指出需要“contain multimodal feature transport failures”并“synchronize per-request failures across TP/CP ranks before model collectives”。

实现拆解

1. 错误封装与跨 rank 共识 (scheduler.py)：在 scheduler.py 中引入了 _MultimodalInputBroadcast 数据类和 _MultimodalInputProcessingError 异常。核心变更在于重构 _materialize_cuda_vmm_inputs 方法，使其能够捕获本地 MultimodalInputs.from_processor_output 调用产生的异常，并通过新增的 _gather_vmm_materialization_errors 方法（使用 torch.distributed.all_gather_object）与所有 TP/CP rank 同步错误状态。任何 rank 的失败都会被记录并上报。
2. 失败请求的安全处理与分发 (scheduler.py)：新增 _dispatch_tokenized_mm_requests 方法。当 _materialize_cuda_vmm_inputs 返回任何错误时，该方法被调用，绕过常规的 _request_dispatcher，直接调用 handle_generate_request 或 handle_embedding_request 并传递 mm_input_error 参数，从而将失败请求转换为内部错误并正确中止。

3. 共享内存（SHM）传输的健壮性增强 (mm_utils.py)：重构 ShmPointerMMData 类。在 __init__ 和 __setstate__ 中引入了 _materialization_error 属性，使得 SHM 打开失败能被延迟到 materialize() 调用时才抛出，而不是在反序列化阶段崩溃。新增 close_and_unlink 方法，确保在任何异常情况下（包括 unlink 失败）都能安全清理 SHM 资源。新增 discard_shm_features 函数，用于主动释放不再需要的 SHM 特征。
4. 多模态输入重构失败的回滚与代理释放 (schedule_batch.py)：在 MultimodalInputs.from_processor_output 方法中添加了 try-except 块，当 CUDA IPC 代理重构失败时，会遍历所有 mm_items 并调用新增的 release_transport_proxies 方法，以确保相关的 GPU 内存池切片被正确归还。Req.set_finish_with_abort 方法被增强，现在会主动释放请求所拥有的多模态特征（但会话共享的特征除外），防止内存泄漏。
5. 全面的测试覆盖（新增 / 修改测试文件）：新增 test_mm_shm_error_consensus.py 测试文件，验证了跨 rank 的 SHM 物化失败共识传播。修改 test_gpu_feature_transport.py，覆盖了重构失败回滚、代理释放和共享内存清理失败的场景。新增 test_multimodal_abort_cleanup.py 测试文件，验证了请求中止时的特征释放逻辑。修改 test_cuda_ipc_transport.py，测试了 CUDA IPC 重构失败和拒绝请求的资源回收。

关键文件：

- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 _MultimodalInputBroadcast, _MultimodalInputProcessingError, _materialize_cuda_vmm_inputs, _tokenized_requests）：核心变更：重构多模态特征 materialization 流程，引入跨 rank 错误共识机制，修改请求分发逻辑以处理失败请求。
- python/sglang/srt/managers/mm_utils.py（模块 多模态工具；类别 source；类型 core-logic；符号 close_and_unlink, _discard_tensor_or_list, discard_shm_features）：关键变更：增强 ShmPointerMMData 类以安全处理 SHM 打开 / 清理失败，新增 discard_shm_features 函数。
- python/sglang/srt/managers/schedule_batch.py（模块 批次管理；类别 source；类型 core-logic；符号 release_transport_proxies, set_finish_with_abort）：关键变更：增强请求中止逻辑以释放多模态特征，改进 MultimodalInputs.from_processor_output 在重构失败时的资源回滚。

关键符号：Scheduler._materialize_cuda_vmm_inputs, Scheduler._gather_vmm_materialization_errors, Scheduler._dispatch_tokenized_mm_requests, Scheduler._process_and_broadcast_mm_inputs, ShmPointerMMData.setstate, ShmPointerMMData.materialize, ShmPointerMMData.close_and_unlink, discard_shm_features, MultimodalDataItem.release_transport_proxies, MultimodalInputs.from_processor_output, Req.set_finish_with_abort

关键源码片段

python/sglang/srt/managers/scheduler.py

核心变更：重构多模态特征 materialization 流程，引入跨 rank 错误共识机制，修改请求分发逻辑以处理失败请求。

```
# 在 Scheduler 类中新增的数据类，用于在 TP ranks 间广播多模态输入或其处理错误。
@dataclasses.dataclass(frozen=True)
class _MultimodalInputBroadcast:
    inputs: Optional[MultimodalInputs] = None
    error: Optional[str] = None

# 自定义异常，用于表示多模态输入处理失败。
class _MultimodalInputProcessingError(RuntimeError):
    pass

# 重构后的方法：为每个请求收集并同步多模态特征重构错误。
def _materialize_cuda_vmm_inputs(self, recv_req) -> Optional[List[Optional[str]]]:
    """Materialize each request and agree on failures across TP ranks."""
    tokenized_reqs = self._tokenized_requests(recv_req)
    if not tokenized_reqs:
        return None

    request_errors = []
    for tokenized_req in tokenized_reqs:
        local_error = None
        try:
            # 尝试从处理器输出构建 MultimodalInputs，这可能涉及 SHM 打开和 CUDA IPC 重构
            if tokenized_req.mm_inputs is not None and not isinstance(
                tokenized_req.mm_inputs, MultimodalInputs
            ):
                tokenized_req.mm_inputs = MultimodalInputs.from_processor_output(
                    tokenized_req.mm_inputs
                )
        except Exception as error:
            # 捕获本地异常
            local_error = f"{type(error).__name__}: {error}"

        # 通过 all_gather_object 收集所有 TP rank 的错误状态
        rank_errors = self._gather_vmm_materialization_errors(local_error)
        failed_ranks = [
            rank for rank, error in enumerate(rank_errors) if error is not None
        ]
        if failed_ranks:
            # 任何 rank 失败，记录错误并将该请求的 mm_inputs 置为 None
            details = "; ".join(
                f"rank {rank}: {rank_errors[rank]}" for rank in failed_ranks
            )
            error_msg = f"Multimodal feature reconstruction failed ({details})"
            logger.error(error_msg)
            tokenized_req.mm_inputs = None
            request_errors.append(error_msg)
        else:
            request_errors.append(None)
    return request_errors
```

python/sglang/srt/managers/mm_utils.py

关键变更：增强 ShmPointerMMData 类以安全处理 SHM 打开 / 清理失败，新增 discard_shm_features 函数。

```
class ShmPointerMMData:
    def __init__(self, tensor: torch.Tensor, precomputed_hash: Optional[int] = None):
        # 初始化时即声明可能为 None 的属性，确保 setstate 和 materialize 可以安全访问
        self._shm_handle = None
        self.tensor = None
        self._materialization_error = None
        # ... [ 原有初始化逻辑 ] ...

    def __setstate__(self, state):
        # ... [ 原有状态恢复逻辑 ] ...
        self._shm_handle = None
        self.tensor = None
        self._materialization_error = None

        # 保持反序列化无失败，以便所有 TP ranks 能完成广播
        handle = None
        tensor = None
        try:
            handle = shared_memory.SharedMemory(name=self.shm_name)
            tensor = torch.frombuffer(handle.buf, dtype=self.dtype)
            self.tensor = tensor.reshape(self.shape)
            self._shm_handle = handle
        except Exception as error:
            tensor = None
            if handle is not None:
                try:
                    handle.close()
                except Exception:
                    logger.warning(
                        "Failed to close a malformed multimodal SHM handle",
                        exc_info=True,
                    )
            # 将错误信息保存起来，推迟到 materialize() 时抛出
            self._materialization_error = f"{type(error).__name__}: {error}"

    def materialize(self) -> torch.Tensor:
        """Clone tensor from shm to owned memory, then release shm handle."""
        try:
            if self._materialization_error is not None:
                # 如果之前反序列化时就出错了，在这里统一抛出
                raise RuntimeError(self._materialization_error)
            return self.tensor.clone()
        finally:
            # 无论成功还是失败，都尝试清理 SHM 资源
```

```
            self.close_and_unlink()

    def close_and_unlink(self) -> None:
        """Release this rank's view and unlink the shared feature segment."""
        handle = self._shm_handle
        self._shm_handle = None
        self.tensor = None
        if handle is None:
            # 如果没有本地 handle，尝试重新打开进行清理
            try:
                handle = shared_memory.SharedMemory(name=self.shm_name)
            except FileNotFoundError:
                return
            except OSError:
                logger.warning(
                    "Failed to reopen a multimodal SHM segment for cleanup",
                    exc_info=True,
                )
                return
        try:
            try:
                handle.unlink()
            except FileNotFoundError:
                pass
            except OSError:
                logger.warning(
                    "Failed to unlink a multimodal SHM segment",
                    exc_info=True,
                )
        finally:
            try:
                handle.close()
            except Exception:
                logger.warning(
                    "Failed to close a multimodal SHM handle",
                    exc_info=True,
                )
```

python/sglang/srt/managers/schedule_batch.py

关键变更：增强请求中止逻辑以释放多模态特征，改进 MultimodalInputs.from_processor_output 在重构失败时的资源回滚。

```
class MultimodalDataItem:
    def release_transport_proxies(self, consumer_count: int = 1) -> None:
        """Best-effort release of proxies left by an abandoned request."""
        # 收集 feature, precomputed_embeddings 和 model_specific_data 中的所有代理
        values = [self.feature, self.precomputed_embeddings]
        values.extend(self.model_specific_data.values())
        for value in values:
```

```
            if not isinstance(value, CudaIpcTensorTransportProxy):
                continue
            # 使用正确的 consumer_count 进行释放
            count = self._resolve_transport_consumer_count(value, consumer_count)
            try:
                value.release_without_reconstruction(count)
            except Exception:
                logger.warning(
                    "Failed to release an abandoned multimodal transport proxy",
                    exc_info=True,
                )

class MultimodalInputs:
    @staticmethod
    def from_processor_output(obj: MultimodalProcessorOutput):
        # ... [ 原有 mm_items 过滤逻辑 ] ...

        # 尝试从 cuda-ipc 重构特征
        reconstruct_device = None
        try:
            for mm_item in mm_items:
                if (
                    mm_item.has_cuda_ipc_proxy()
                    and not mm_item.can_defer_cuda_ipc_feature_reconstruction()
                ):
                    if reconstruct_device is None:
                        reconstruct_device = torch.cuda.current_device()
                    mm_item.reconstruct(reconstruct_device)
        except BaseException:
            # 重构失败时，释放所有项目中的传输代理，防止 GPU 内存泄漏
            for mm_item in mm_items:
                mm_item.release_transport_proxies()
            raise

        # ... [ 后续的 hash 计算和缓冲区逻辑 ] ...

class Req:
    def set_finish_with_abort(
        self,
        error_msg: str,
        status_code: int = HTTPStatus.BAD_REQUEST,
        err_type: str = "BadRequestError",
    ):
        if get_parallel().tp_rank == 0:
            logger.error(f"{error_msg}, {self.rid=}")
        # Session 请求共享历史多模态输入，由会话关闭时释放。普通请求在此释放。
        if self.multimodal_inputs is not None and self.session is None:
            self.multimodal_inputs.release_features()
        self.multimodal_inputs = None
```

```
    # ... [ 后续的重置逻辑 ] ...
```

评论区精华

由于 PR 评论区为空，本节主要基于代码变更和设计模式提炼关键的技术讨论点。核心设计决策在于错误传播的 时机和 范围：选择在 process_input_requests 循环内，对每个请求独立进行 materialization 并收集跨 rank 错误，而不是在更早的 broadcast 阶段或更晚的 forward 阶段。这确保了错误能被隔离到单个请求，避免了错误扩散导致整个 batch 或进程失败。另一个关键点是 资源释放的 防御性编程：在 release_features、release_transport_proxies 和 close_and_unlink 中普遍使用了 try-except 块，确保即使释放操作本身失败（如 unlink 被拒绝），也不会引发进一步异常，而是记录日志并继续清理。这体现了对分布式环境下不可靠操作的处理经验。

- 跨 TP/CP rank 的多模态特征物化错误共识机制 (correctness): 采用 all_gather_object 收集每个请求的本地错误字符串列表，形成跨 rank 共识。任何 rank 报告错误都会导致整个请求被中止。
- 共享内存（SHM）打开 / 重构失败的延迟处理与安全清理 (correctness): 错误延迟到物化时抛出，资源清理操作封装在 close_and_unlink 中，并用 try-except 包裹，防止清理失败引发二次异常。
- 请求中止时的多模态特征资源释放 (design): 在 set_finish_with_abort 中增加释放逻辑，仅对非会话请求调用 release_features。在 MultimodalInputs.from_processor_output 中增加 try-except 块，重构失败时遍历并释放所有代理。

风险与影响

- 风险：
  1. 回归风险：对核心请求处理路径（process_input_requests、handle_generate_request）和多模态特征生命周期管理（ShmPointerMMData、MultimodalInputs.from_processor_output）的修改可能引入细微的行为变化，特别是对于异常路径和边界条件（如会话请求）。需要确保现有 VLM 功能（如 Qwen-VL, Kimi）不受影响。
  2. 性能影响：_materialize_cuda_vmm_inputs 中新增的 all_gather_object 调用增加了每次带有多模态特征的请求处理时的 CPU 和通信开销。虽然该操作仅在 mm_feature_transport == "cuda_vmm" 时触发，但在高并发 VLM 场景下可能成为瓶颈。
  3. 内存泄漏 / 资源泄漏：虽然本 PR 的主要目标是修复资源泄漏，但新的释放逻辑（如 release_transport_proxies、close_and_unlink）本身也需要确保健壮。特别是 ShmPointerMMData.close_and_unlink 在 handle 为 None 时重新打开 SHM 进行清理的逻辑，如果 SHM 名称不存在或已被其他进程清理，需正确处理 FileNotFoundError。
  4. 兼容性：变更了 Req.set_finish_with_abort 的签名（新增 status_code 和 err_type 参数），并修改了 MultimodalInputs.from_processor_output 和 ShmPointerMMData.__setstate__ 的行为。任何依赖这些接口或旧的错误处理模式的下游代码或测试可能需要调整。 - 影响：影响范围：本 PR 直接影响 SGLang 的多模态请求处理流程，特别是使用共享内存（SHM）或 CUDA IPC 进行特征传输的路径。主要影响模块为 sglang.srt.managers.scheduler、sglang.srt.managers.mm_utils 和 sglang.srt.managers.schedule_batch。

影响程度：

- 对用户：提高了服务的 稳定性和可靠性。在处理损坏或不完整的多模态输入（如格式错误的图片、损坏的共享内存段）时，系统不再会因单个请求的失败而导致整个调度器进程崩溃。用户将收到明确的 InternalServerError（HTTP 500）错误响应，而不是连接中断或无响应。
- 对系统：增强了系统的 容错能力和 资源管理。通过隔离失败和确保资源释放，减少了因 VLM 请求处理失败导致的内存泄漏、GPU 内存池枯竭或进程挂起的风险。系统的整体健壮性得到提升，特别是在多租户或不信任输入的部署环境中。
- 对团队：引入了更清晰的多模态错误处理模式和更全面的测试用例，为后续开发相关的健壮性功能提供了参考。但同时也增加了代码的复杂性，需要维护跨 rank 错误同步和资源安全释放的逻辑。
- 风险标记：核心路径变更，分布式通信开销，资源清理逻辑复杂

关联脉络

- PR #37030 [Stacked PR base]: 本 PR 明确指出是 'Stacked on #37030'，表明它是基于 #37030 进行的开发。#37030 可能是进行了一些前置的重构或特性引入。