PR #27011 完整报告

sgl-project/sglang

[Bugfix] Clean up failed NIXL sender state

合并时间：2026-06-03 12:15

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27011

执行摘要

- 一句话：修复 NIXL 发送器失败后房间状态未清理
- 推荐动作：该 PR 值得合并，修复了明确的状态泄漏 bug。建议后续考虑对 staging_ctx 加锁或明确其线程安全模型。

功能与动机

NIXL 传输工作线程失败会被记录并暴露为 KVPoll.Failed，但 NixlKVSender.failure_exception() 在抛出异常前未清理 per-room 状态，导致请求终止后残留 request_status、transfer_infos、req_to_decode_prefix_len、failure_records 以及 staging prefetch 状态。这与 Mooncake 和 Mori 的 sender 失败路径不一致，需要修复。

实现拆解

变更集中在两个文件：

1. python/sglang/srt/disaggregation/nixl/conn.py：
   - 新增 clear() 方法：先调用父类 CommonKVSender.clear() 清理通用状态（request_status、transfer_infos 等），再针对 NIXL staging 场景，从 kv_mgr._staging_ctx.prefetched_rooms 中移除当前 room，并从 prefetch_requested 中删除所有以当前 room 为 key 的记录。
   - 重写 failure_exception() 方法：
   - 先弹出异常和失败原因（加锁保护），标记 sender 为失败状态（_send_failed = True，conclude_state = KVPoll.Failed）。
   - 调用 self.clear() 清理房间状态。
   - 然后按优先顺序抛出异常：_send_error → 弹出的 exception → 弹出的 failure_reason → 默认 RuntimeError。
2. test/registered/unit/disaggregation/test_nixl_sender_failure_cleanup.py：
   - 新增 CPU 单元测试 test_failure_exception_cleans_room_state_before_raising，注册到 base-a-test-cpu 测试套件。
   - 通过 SimpleNamespace 模拟 kv_mgr 和 staging 上下文，验证：
   - 异常被正确抛出（assertRaises）。
   - sender 的 _send_failed 和 conclude_state 被正确设置。

- 所有房间相关状态（request_status、req_to_decode_prefix_len、transfer_infos、exceptions、failure_records）被清除。
- staging prefetch 状态中当前 room 被清除，而不影响其他 room。

关键文件：

- python/sglang/srt/disaggregation/nixl/conn.py（模块 传输层；类别 source；类型 core-logic；符号 clear, failure_exception）：核心修复文件，新增 clear() 方法并重写 failure_exception()，确保失败时清理 per-room 状态。
- test/registered/unit/disaggregation/test_nixl_sender_failure_cleanup.py（模块 测试；类别 test；类型 test-coverage；符号 TestNixlSenderFailureCleanup, test_failure_exception_cleans_room_state_before_raising）：新增 CPU 单元测试，验证 failure_exception() 中清理行为的正确性，覆盖 staging 场景。

关键符号：clear, failure_exception

关键源码片段

python/sglang/srt/disaggregation/nixl/conn.py

核心修复文件，新增 clear() 方法并重写 failure_exception()，确保失败时清理 per-room 状态。

```
# python/sglang/srt/disaggregation/nixl/conn.py
# NixlKVSender 类新增 clear() 方法，重写 failure_exception() 方法

def clear(self) -> None:
    # 先调用父类清理通用的 per-room 状态 (request_status, transfer_infos 等 )
    super().clear()
    # 如果启用了 staging，还需要清理 staging prefetch 状态
    if (
        getattr(self.kv_mgr, "enable_staging", False)
        and getattr(self.kv_mgr, "_staging_ctx", None) is not None
    ):
        # 从 prefetched_rooms 集合中移除当前 room
        self.kv_mgr._staging_ctx.prefetched_rooms.discard(self.bootstrap_room)
        # 从 prefetch_requested 集合中过滤掉所有以当前 room 开头的请求
        self.kv_mgr._staging_ctx.prefetch_requested = {
            key
            for key in self.kv_mgr._staging_ctx.prefetch_requested
            if key[0] != self.bootstrap_room
        }

def failure_exception(self):
    # 先从 kv_mgr 中弹出异常和失败原因
    exc = self.kv_mgr.exceptions.pop(self.bootstrap_room, None)
    with self.kv_mgr.failure_lock:
        failure_reason = self.kv_mgr.failure_records.pop(self.bootstrap_room, None)

    # 标记 sender 为失败状态
    if self.conclude_state is None:
```

```
            self.conclude_state = KVPoll.Failed
        self._send_failed = True

        # 清理房间状态 —— 先于异常抛出，确保资源释放
        self.clear()

        # 按优先顺序抛出异常：自定义错误 > 传输异常 > 失败原因 > 默认 RuntimeError
        if self._send_error is not None:
            raise self._send_error
        if exc is not None:
            raise exc
        if failure_reason is not None:
            raise RuntimeError(failure_reason)
        raise RuntimeError("NIXL KVSender Exception")
```

test/registered/unit/disaggregation/test_nixl_sender_failure_cleanup.py

新增 CPU 单元测试，验证 failure_exception() 中清理行为的正确性，覆盖 staging 场景。

```
# test/registered/unit/disaggregation/test_nixl_sender_failure_cleanup.py
# CPU 单元测试，注册到 base-a-test-cpu 套件
import threading
import unittest
from types import SimpleNamespace

from sglang.srt.disaggregation.base.conn import KVPoll
from sglang.srt.disaggregation.nixl.conn import NixlKVSender
from sglang.test.ci.ci_register import register_cpu_ci

register_cpu_ci(est_time=2, suite="base-a-test-cpu")

class TestNixlSenderFailureCleanup(unittest.TestCase):
    def test_failure_exception_cleans_room_state_before_raising(self):
        room = 7
        expected_exc = RuntimeError("transfer failed")
        # 直接通过 __new__ 创建 sender 实例，避免初始化复杂依赖
        sender = NixlKVSender.__new__(NixlKVSender)
        sender.bootstrap_room = room
        sender.conclude_state = None
        sender._send_failed = False
        sender._send_error = None

        # 构造 staging 上下文，包含当前 room 和其他 room 的数据
        staging_ctx = SimpleNamespace(
            prefetched_rooms={room, 8},
            prefetch_requested={(room, 0, "session-a"), (8, 0, "session-b")},
        )
        # 构造 kv_mgr 模拟器，包含所有需要清理的状态
        sender.kv_mgr = SimpleNamespace(
            enable_staging=True,
```

```
            _staging_ctx=staging_ctx,
            request_status={room: object()},
            req_to_decode_prefix_len={room: 3},
            transfer_infos={room: object()},
            exceptions={room: expected_exc},
            failure_records={room: "transfer failed"},
            failure_lock=threading.Lock(),
        )

        with self.assertRaises(RuntimeError) as cm:
            sender.failure_exception()

        # 验证异常被正确传播
        self.assertIs(cm.exception, expected_exc)
        # 验证 sender 被标记为失败
        self.assertTrue(sender._send_failed)
        self.assertEqual(sender.conclude_state, KVPoll.Failed)
        # 验证房间状态被清理
        self.assertNotIn(room, sender.kv_mgr.request_status)
        self.assertNotIn(room, sender.kv_mgr.req_to_decode_prefix_len)
        self.assertNotIn(room, sender.kv_mgr.transfer_infos)
        self.assertNotIn(room, sender.kv_mgr.exceptions)
        self.assertNotIn(room, sender.kv_mgr.failure_records)
        # 验证 staging 状态中当前 room 被清理，但其他 room 不受影响
        self.assertNotIn(room, staging_ctx.prefetched_rooms)
        self.assertNotIn((room, 0, "session-a"), staging_ctx.prefetch_requested)
        self.assertIn(8, staging_ctx.prefetched_rooms)
        self.assertIn((8, 0, "session-b"), staging_ctx.prefetch_requested)

if __name__ == "__main__":
    unittest.main()
```

评论区精华

Reviewer ShangmingCai 批准了 PR，并提及 @YAMY1234 @ishandhanani，表明需要相关开发者关注或测试。无其他讨论。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 回归风险：clear() 方法重用了父类清理逻辑，且在 failure_exception() 中先标记失败再清理，不会影响正常成功路径。
  2. 并发安全：清理 failure_records 时使用了 failure_lock，但 clear() 中访问 staging_ctx 时未加锁，若 staging_ctx 被多个线程并发修改可能引入竞态。建议确认 staging_ctx 的线程安全设计。

3. 测试覆盖：单元测试模拟了典型场景，但未覆盖并发场景或 staging 关闭时的路径（*enable_staging=False*）。 - 影响：影响范围限于 NIXL 传输失败时的资源清理。修复后，失败请求的资源将被及时释放，不会干扰后续请求。对用户透明，系统资源利用率和稳定性提升。 - 风险标记：staging_ctx 并发安全需确认

关联脉络

- 暂无明显关联 PR