PR #36029 完整报告

sgl-project/sglang

fix(disagg): refresh stale prefill bootstrap metadata

合并时间：2026-08-26 03:29

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36029

执行摘要

- 一句话：修复 prefill 重启后解码端复用旧端口，缓存代际按对象身份失效
- 推荐动作：值得精读。该 PR 展示了一个典型的"缓存失效需精确到对象身份"的并发设计：通过记录谁引用了哪一代缓存、在失效时用 is 比较避免误删并发替换，是处理分布式 / 多接收者缓存失效的佳例。同时，将 _connect_to_bootstrap_server 移入 try 块并在发送失败时同步失效，体现了错误恢复与缓存一致性联动的思路。建议重点关注 _setup_bootstrap_infos 中 setdefault 与对象身份判断的组合用法。

功能与动机

PR body 指出：Decode workers 缓存的 prefill bootstrap 元数据中包含临时 per-rank 端口，当 prefill 进程以相同 host 和 bootstrap 地址重启后，rank 端口改变但缓存键不变，后续请求会无限期复用旧端口。前一个修复 PR #31144 虽然限制了失败的 ZMQ send，但缓存代际仍保留，导致每个后续请求继续失败。本 PR 的目标是精确失效"仅当前接收者使用过"的缓存条目，同时避免误删其他接收者并发写入的新代际。

实现拆解

1. 记录接收者使用的缓存条目：在 CommonKVReceiver.__init__ 中新增 _connection_pool_entries: Dict[str, List[Dict]]，用于保存该接收者从 connection_pool 选中的条目（key 为 bootstrap_key，value 为缓存列表对象）。
2. 重构缓存读写路径：_setup_bootstrap_infos 中把 if bootstrap_key not in connection_pool 改为在 connection_lock 保护下 get；写入时使用 setdefault 并比对对象身份，若已有并发写入的新代际则采用对方的值，同时更新 _connection_pool_entries。
3. 新增身份安全失效方法：invalidate_cached_bootstrap_infos() 在锁内遍历 _connection_pool_entries，仅当 connection_pool.get(key) is 目标对象时才删除，随后清空本地追踪；这样不会误删另一个接收者刚写入的替换代际。
4. 接入失败路径：_setup_bootstrap_infos 的拉取失败、_register_kv_args 失败以及 _check_waiting_timeout 超时分支均调用失效方法；Mooncake、MORI、NIXL 的 send_metadata 和 _register_kv_args 把 _connect_to_bootstrap_server 移入 try 块，并在 zmq.ZMQError 时调用失效方法。
5. 配套测试：新增 test/registered/unit/disaggregation/test_receiver_connection_pool.py（注册到 base-a-test-cpu），覆盖身份安全失效、并发替换保留、多 CP 条目、路由重拉、

等待超时失效；test_nixl_backend_basic.py 的 fixture 补上 _connection_pool_entries ={} 初始化。

关键文件：

- python/sglang/srt/disaggregation/common/conn.py（模块 分拆连接；类别 source；类型 core-logic；符号 invalidate_cached_bootstrap_infos）：核心修改文件：新增 _connection_pool_entries 追踪与 invalidate_cached_bootstrap_infos() 身份安全失效逻辑，并重构 _setup_bootstrap_infos 的缓存读写路径。
- test/registered/unit/disaggregation/test_receiver_connection_pool.py（模块 分拆测试；类别 test；类型 test-coverage；符号 _ConcreteReceiver, poll, failure_exception, _receiver）：新增的 CPU 单元测试，完整覆盖失效逻辑的关键分支，是验证修复正确性的核心证据。
- python/sglang/srt/disaggregation/mooncake/conn.py（模块 传输后端；类别 source；类型 core-logic）：将 _connect_to_bootstrap_server 移入 try 块，并在 ZMQ 发送失败时调用 invalidate_cached_bootstrap_infos，统一错误恢复行为。
- python/sglang/srt/disaggregation/mori/conn.py（模块 传输后端；类别 source；类型 core-logic）：与 Mooncake 相同的错误恢复修正，确保 MORI 后端在发送失败时也触发缓存失效。
- python/sglang/srt/disaggregation/nixl/conn.py（模块 传输后端；类别 source；类型 core-logic）：与 Mooncake、MORI 相同的错误恢复修正，NIXL 后端同样接入缓存失效。
- test/registered/unit/disaggregation/test_nixl_backend_basic.py（模块 NIXL 测试；类别 test；类型 test-coverage）：测试 fixture 需要补上 _connection_pool_entries 初始化，以兼容新增的失效逻辑。

关键符号：invalidate_cached_bootstrap_infos, _setup_bootstrap_infos, _check_waiting_timeout, send_metadata, _register_kv_args

关键源码片段

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

核心修改文件：新增 _connection_pool_entries 追踪与 invalidate_cached_bootstrap_infos() 身份安全失效逻辑，并重构 _setup_bootstrap_infos 的缓存读写路径。

```
def invalidate_cached_bootstrap_infos(self) -> None:
    # 只在锁内删除本接收者实际引用过的缓存代际。
    # 用对象身份（is）比较，避免误删其他接收者并发写入的替代代际。
    with self.kv_mgr.connection_lock:
        for bootstrap_key, bootstrap_infos in self._connection_pool_entries.items():
            if self.kv_mgr.connection_pool.get(bootstrap_key) is bootstrap_infos:
                del self.kv_mgr.connection_pool[bootstrap_key]
        # 清空本地追踪，避免同一接收者再次调用时重复删除。
        self._connection_pool_entries.clear()

def _setup_bootstrap_infos(self):
```

```
all_bootstrap_infos = []
# 缓存 key 由 bootstrap_addr、prefill_dp_rank、prefill_cp_rank、target_tp_rank 唯一确定
for target_cp_rank in self.target_cp_ranks:
    bootstrap_key = f"{self.bootstrap_addr}_{self.prefill_dp_rank}_{target_cp_rank}_{self.target_
    tp_rank}"

    with self.kv_mgr.connection_lock:
        cached_bootstrap_infos = self.kv_mgr.connection_pool.get(bootstrap_key)

    if cached_bootstrap_infos is None:
        # 未命中缓存 → 逐 rank 从 bootstrap server 拉取
        bootstrap_infos = []
        for target_tp_rank in self.target_tp_ranks:
            for target_pp_rank in reversed(self.target_pp_ranks):
                bootstrap_info = self._get_bootstrap_info_from_server(
                    self.prefill_dp_rank,
                    target_cp_rank,
                    target_tp_rank,
                    target_pp_rank,
                )
                if bootstrap_info is None:
                    # 拉取失败 → 标记失败并立即清除本接收者用过的缓存代际
                    self.kv_mgr.record_failure(
                        self.bootstrap_room,
                        f"Failed to fetch bootstrap info from {self.bootstrap_addr}",
                    )
                    self.kv_mgr.update_status(self.bootstrap_room, KVPoll.Failed)
                    self.bootstrap_infos = None
                    self.invalidate_cached_bootstrap_infos()
                    return
                # 省略 is_dummy 等字段处理，逻辑与旧版一致
                bootstrap_infos.append(bootstrap_info)

        self.bootstrap_infos = bootstrap_infos
        self._connection_pool_entries[bootstrap_key] = self.bootstrap_infos

        if not self._register_kv_args():
            # 注册失败 → 缓存不能残留失败代际，立即失效
            self.invalidate_cached_bootstrap_infos()
            return

        with self.kv_mgr.connection_lock:
            # setdefault 保证并发下只保留一份；若已有其他 receiver 写入新代际，则采用对方的
            cached_bootstrap_infos = self.kv_mgr.connection_pool.setdefault(
                bootstrap_key, self.bootstrap_infos
            )
            if cached_bootstrap_infos is not self.bootstrap_infos:
                self.bootstrap_infos = cached_bootstrap_infos
```

```
            self._connection_pool_entries[bootstrap_key] = self.bootstrap_infos
        else:
            self.bootstrap_infos = cached_bootstrap_infos
            self._connection_pool_entries[bootstrap_key] = self.bootstrap_infos

        assert len(self.bootstrap_infos) > 0
        all_bootstrap_infos.extend(self.bootstrap_infos)

    self.bootstrap_infos = all_bootstrap_infos
```

test/registered/unit/disaggregation/test_receiver_connection_pool.py

新增的 CPU 单元测试，完整覆盖失效逻辑的关键分支，是验证修复正确性的核心证据。

```
class TestReceiverConnectionPool(CustomTestCase):
    def test_invalidate_preserves_concurrent_replacement_generation(self):
        # 模拟场景：另一个 receiver 已用新端口（2001）替换了同一 key 的缓存，
        # 本 receiver 持有的 stale 代际（1001）应当被安全丢弃，但不能误删新代际。
        stale = [{"rank_ip": "10.0.0.1", "rank_port": 1001}]
        replacement = [{"rank_ip": "10.0.0.1", "rank_port": 2001}]
        receiver = _receiver(
            {"key": replacement}, # connection_pool 当前存的是新代际
            {"key": stale}, # 本 receiver 引用的旧代际
        )

        receiver.invalidate_cached_bootstrap_infos()

        # 只有旧代际被移除，新代际保留
        self.assertEqual(receiver.kv_mgr.connection_pool, {"key": replacement})

    def test_next_receiver_refetches_after_invalidation(self):
        # 失效后，下一个 receiver 应重新调用 _get_bootstrap_info_from_server 拉取新端口
        stale = [{"rank_ip": "10.0.0.1", "rank_port": 1001}]
        connection_pool = {"prefill:8998_0_0_0": stale}
        stale_receiver = _receiver(connection_pool, {"prefill:8998_0_0_0": stale})
        stale_receiver.invalidate_cached_bootstrap_infos()

        receiver = _fetching_receiver(connection_pool) # fetch_count 初始为 0
        receiver._setup_bootstrap_infos()

        self.assertEqual(receiver.fetch_count, 1) # 证明确实重新拉取
        self.assertEqual(receiver.bootstrap_infos[0]["rank_port"], 2001)
```

评论区精华

ShangmingCai 在 _check_waiting_timeout 新增失效调用处提问：“what if prefill is just slow, not dead?”（如果 prefill 只是慢，而不是死掉怎么办？），担心超时失效会误伤响应慢的 prefill。该问题未在 thread 内直接回复，但从实现看，超时意味着当前接收者已放弃该次请求，失效后下个请求会重新拉取；若 prefill 只是慢，重新拉取仍可成功并自愈，因此不会造成永久性错误。ShangmingCai 最终批准并留言：“Overall LGTM, let us check the CI.”

- 等待超时失效是否会误杀慢 prefill？ (correctness): 未在 thread 内直接回复，但实现上失效只影响缓存复用，下个请求会重新拉取；若 prefill 可用则自动恢复，不会造成永久性失败。

风险与影响

- 风险：并发安全：失效方法在 connection_lock 内遍历和删除，避免与 _setup_bootstrap_infos 的读写竞争；锁粒度小，条目数通常等于 CP 数，不会成为热点。对象身份依赖：失效依赖 Python is 比较，要求缓存中保存的是原始列表对象；当前所有写入路径都直接保存原对象，不存在序列化重建，行为正确，但未来若有克隆写入需特别注意。超时误杀：等待超时立即失效可能对慢 prefill 触发一次额外重拉，但重拉成本低且结果正确，属于可接受的权衡。影响范围：只影响 PD 分离的错误恢复路径，不改变正常数据面；三个传输后端行为对齐，统一处理 ZMQ 发送失败。
- 影响：对用户而言，prefill 重启后解码端请求将不再无限失败，而是自动刷新并恢复，显著提升 PD 分离场景的鲁棒性；对系统而言，happy path 仅增加短锁保护的缓存查找 / 插入操作，无额外数据面开销；对团队而言，统一了 Common/Mooncake/MORI/NIXL 四个后端的失效逻辑，降低了后续维护成本，并通过 CPU 测试锁定了关键行为。
- 风险标记：核心错误恢复路径，按对象身份失效，跨后端统一，锁内失效

关联脉络

- PR #36351 fix(disagg): snapshot affected rooms before iterating outside the lock: 同一模块（disaggregation/common/conn.py）的并发正确性修复，与本 PR 都在处理连接池跟踪与锁语义。