PR #33048 完整报告

sgl-project/sglang

[Bugfix] Hold references to fire-and-forget tasks in disaggregation

合并时间：2026-08-30 14:26

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33048

执行摘要

- 一句话：修复 disaggregation 中任务引用丢失，防止后台任务被 GC 回收
- 推荐动作：值得精读，但收益主要在“模式”而不是“逻辑”：这是一个将 asyncio 官方建议落地为统一代码惯用法的教科书式小补丁。推荐关注三点：set + add_done_callback(set.discard) 的组合如何同时解决强引用与自动释放；rebase 过程中如何识别出上游已用 asyncio.wait 兜住引用的变化并主动丢弃 hunk；以及审阅者关于 hardening 与 bugfix 的定性争论——它提醒我们在评估此类修复时要区分“理论隐患”与“实际触发路径”。

功能与动机

PR body 直接引用 asyncio 官方文档的警告：“Save a reference to the result of this function, to avoid a task disappearing mid-execution. The event loop only keeps weak references to tasks.” 作者通过 AST 扫描 create_task / ensure_future 返回结果被当作裸表达式丢弃的位置，发现 srt/disaggregation 下共 5 处。其中前 4 处是进程生命周期循环，属于最坏情况：一旦被 GC，_result_listener 停止消费 worker 结果、_worker_watchdog 不再检测死 rank、_cleanup_stale_mappings 停止清理过期映射、_cleanup_expired_entries 使 room_to_dp_rank 无界增长。第五处是 encode 分发任务，任务被间接等待，但 wait_for 超时展开栈帧时 encode 可能仍在飞行中，仅靠局部变量无法覆盖。

实现拆解

1. 病灶定位与模式对齐：先以 AST 扫描确认 5 个丢弃点的分布，再对照 MMEncoder.background_tasks（encode_server.py 中已有、被 3 处调用）作为标准实现模板，确定统一的 set + done-callback 方案。
2. DPDispatcher 修复（runtime.py）：在 __init__ 中新增 self.background_tasks: Set[asyncio.Task]，将 start() 中三条裸 create_task 改写为遍历协程元组，逐个 create_task、add 进集合，并以 task.add_done_callback(self.background_tasks.discard) 在完成时自动移除；set.discard 天然容忍重复回调，避免 double-callback 抛异常。覆盖 _result_listener、_worker_watchdog、_cleanup_stale_mappings 三个进程生命周期循环。
3. CommonKVBootstrapServer 修复（conn.py）：在 __init__ 中新增 self._background_tasks: Set[asyncio.Task]，在 _run_server() 中把 self._loop.create_task(self._cleanup_expired_entries()) 的结果保存并注册到集合，同样绑定 add_done_callback 移除。

4. rebase 中自动消化的第四处：原 encode_receiver.py:1668 的 encode 分发任务在文件迁移到 disaggregation/encoder/receiver.py 后，代码已变为 asyncio.wait({encode_task, recv_task}, ...)，wait 集合自身持有强引用，覆盖了任务整个生命周期，因此该 hunk 被作者主动丢弃，合入版本无需改动。
5. 测试与 CI 配套：未新增单元测试——作者在 PR body 说明 GC 竞态测试本质上是 flaky，必须人为触发 GC 并断言任务未消失，且 5 个调用点都需要活的 dispatcher / bootstrap server / receiver 才能到达；CI 上的两次红色（test_batch_result_processor_hidden_states.py 的 think_end_ids 字段更名、XPU test_xpu_graph.py 超时）均与本次改动无关，rebase 当前 main 后转绿。

关键文件：

- python/sglang/srt/disaggregation/encoder/runtime.py（模块 DP 分发器；类别 source；类型 core-logic；符号 DPDispatcher.start, DPDispatcher.background_tasks, DPDispatcher._result_listener, DPDispatcher._worker_watchdog）：合并版本的核心源码改动：DPDispatcher.start() 中三个进程生命周期协程（_result_listener、_worker_watchdog、_cleanup_stale_mappings）的 Task 此前全部只被 create_task() 表达式弱引用，是本次修复最重要的调用点；文件由原 encode_server.py 迁移而来，改动被完整保留在此。
- python/sglang/srt/disaggregation/common/conn.py（模块 引导服务；类别 source；类型 core-logic；符号 CommonKVBootstrapServer._run_server, CommonKVBootstrapServer._background_tasks, CommonKVBootstrapServer._cleanup_expired_entries）：CommonKVBootstrapServer._run_server() 此前裸创建 _cleanup_expired_entries 清理任务，若被 GC 回收会导致 room_to_dp_rank 无界增长；改动为此处的长生命周期任务补上强引用注册，是第二个真实调用点。

关键符号：DPDispatcher.start, DPDispatcher._result_listener, DPDispatcher._worker_watchdog, DPDispatcher._cleanup_stale_mappings, CommonKVBootstrapServer._run_server, CommonKVBootstrapServer._cleanup_expired_entries

关键源码片段

python/sglang/srt/disaggregation/encoder/runtime.py

合并版本的核心源码改动：DPDispatcher.start() 中三个进程生命周期协程（_result_listener、_worker_watchdog、_cleanup_stale_mappings）的 Task 此前全部只被 create_task() 表达式弱引用，是本次修复最重要的调用点；文件由原 encode_server.py 迁移而来，改动被完整保留在此。

```
class DPDispatcher:
    """Routes encode requests across DP ranks by least-pending count."""

    def __init__(self, ...):
        ...
        # 事件循环只对 Task 持有弱引用；若任务仅被 create_task() 表达式引用，
```

```
        # 可能在运行中被 GC 提前回收。这里为进程生命周期循环保存强引用。
        self.background_tasks: Set[asyncio.Task] = set()

    def start(self) -> None:
        # 三个协程均为长生命周期循环：结果监听、rank 看门狗、过期映射清理。
        # 任何一个被 GC 都会让 DP 分发器静默退化，因此统一注册并强引用。
        logger.info(f"DP dispatcher started: {self.dp_size} ranks (all remote)")
        for coro in (
            self._result_listener(),
            self._worker_watchdog(),
            self._cleanup_stale_mappings(),
        ):
            task = asyncio.create_task(coro)
            self.background_tasks.add(task)
            # 任务完成时自动从集合中移除；set.discard 天然容忍重复回调，
            # 不会因 done callback 被触发两次而抛异常。
            task.add_done_callback(self.background_tasks.discard)
```

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

CommonKVBootstrapServer._run_server() 此前裸创建 _cleanup_expired_entries 清理任务，若被 GC 回收会导致 room_to_dp_rank 无界增长；改动为此处的长生命周期任务补上强引用注册，是第二个真实调用点。

```
class CommonKVBootstrapServer(BaseKVBootstrapServer):
    def __init__(self, host: str, port: int):
        self.host = host
        self.port = port
        self.app = web.Application()
        self.store = dict()
        self.lock = asyncio.Lock()
        # 事件循环只对 Task 持有弱引用；长生命周期任务需要强引用防 GC，
        # 否则 _cleanup_expired_entries 可能提前终止，room_to_dp_rank 无界增长。
        self._background_tasks: Set[asyncio.Task] = set()
        ...

    def _run_server(self):
        try:
            # 事件循环
            self._loop = asyncio.new_event_loop()
            asyncio.set_event_loop(self._loop)

            # 清理任务是守护型循环：注册强引用并在完成时移除，防止被回收
            cleanup_task = self._loop.create_task(self._cleanup_expired_entries())
            self._background_tasks.add(cleanup_task)
            cleanup_task.add_done_callback(self._background_tasks.discard)

            # aiohttp app 在同一事件循环内启动，随后 run_forever 阻塞
            access_log = None
            if logging.getLogger(__name__).getEffectiveLevel() <= logging.DEBUG:
```

```
                access_log = self.app.logger
            self._runner = web.AppRunner(self.app, access_log=access_log)
            self._loop.run_until_complete(self._runner.setup())
            site = web.TCPSite(self._runner, host=self.host, port=self.port)
            self._loop.run_until_complete(site.start())
            ...
            self._loop.run_forever()
        except Exception as e:
            logger.error(f"Server error: {str(e)}", exc_info=True)
        finally:
            self._loop.run_until_complete(self._runner.cleanup())
            self._loop.close()
```

评论区精华

审阅讨论的核心交锋集中在三点：一是 dreamer-89 在旧 encode_receiver.py hunk 上提出的异常传播问题；二是 ShangmingCai 对“bugfix vs hardening”的定性；三是作者对不补测试和 rebase 丢弃 hunk 的详细说明。

- encode 任务失败缺少错误传播 (design): 认可为独立的跟进问题，dreamer-89 创建 issue #34434 用于审计剩余 fire-and-forget 任务，本次改动范围不扩大。
- hardening 还是 bugfix 的定性争论 (question): 双方认可代码质量，按原计划合入；PR 标题保留 Bugfix，合并信息中也保留了防御性语义。
- 为何不补单元测试 (testing): 审阅方接受跳过测试，以跟进 issue #34434 作为审计入口。
- CI 红色与 PR 无关的排除分析 (other): liusy58 多次 /rerun-failed-ci，rebase 当前 main 并解决冲突后 CI 转绿合并。

风险与影响

- 风险：技术风险整体较低，但需注意： 1) 无直接测试覆盖——GC 竞态难以构造确定性用例，合入后只能依赖后续 issue #34434 的审计与真机长稳暴露问题； 2) 任务完成回调依赖 set.discard，若任务异常终止，done callback 仍会执行并移除引用，但异常本身仍不被传播给调用方（这正是 dreamer-89 提出、留给 #34434 的问题）； 3) 改动位于编码分发与 bootstrap 注册表等关键路径，若未来有人在这两个类上重复调用 start() / _run_server()，集合会累积任务引用，但当前调用点单例化，实际风险可忽略； 4) rebase 后 encode_receiver.py 的 hunk 被丢弃，若未来 asyncio.wait 被重构回裸 create_task，需要重新套用同一模式，属于隐性回归面。
  - 影响：对用户与系统：提升 PD（prefill-decode disaggregation）编码链路的可靠性，消除一种极端时序下 dispatcher 静默失效、死 rank 检测失效、room_to_dp_rank 与过期映射长期累积的隐患，属于防御性加固而非行为变化。对团队：改动集中、语义清晰（2 个文件、+17/-4），并让 disaggregation 包与既有 MMEncoder.background_tasks 模式保持一致，降低了后续维护者对异步任务生命周期的认知负担；同时为“审计全仓库 fire-and-forget 任务”这一横切动作留下 issue #34434 作为入口。
  - 风险标记：缺少直接测试覆盖，涉及编码分发核心路径，GC 竞态难以复现验证，异常仍无传播路径

关联脉络

- PR #35281 [PD] Align defensive protocol behavior across Mooncake, NIXL, and Mori: 同为 disaggregation 模块的防御性行为修复，体现 PD 后端近期系统性加固的同一脉络。
- PR #37166 fix(staging): make empty staging rings reusable: 同为 PD 链路中隐蔽时序问题的修复，与本次任务生命周期加固共同指向 disaggregation 可靠性的持续投入。