PR #26298 完整报告

sgl-project/sglang

Fail-fast on PD subprocess exit and scheduler exception

合并时间：2026-05-26 07:42

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26298

执行摘要

- 一句话：添加 PD 测试与调度器异常快速终止机制
- 推荐动作：值得精读。PR 展示了如何在分布式系统中设计安全、可选的中断机制，其 opt-in 设计、测试夹具的 teardown 顺序、环境变量命名规范都可作为内部可靠性改进的参考模板。

功能与动机

PR body 指出：PD 分解测试中子进程崩溃后，评估客户端会卡在连接重试约 30 秒到 2 分钟才能报错；调度器异常会导致兄弟进程输出数千行 NCCL HeartbeatMonitor/TCPStore 回溯，干扰问题定位。快速终止可显著缩短测试反馈时间并降低日志噪音。

实现拆解

1. 测试工具函数：在 python/sglang/test/test_utils.py 新增 start_subprocess_fail_fast_watcher，启动守护线程监控命名子进程列表，任一非零退出时 kill 兄弟进程并 os._exit(rc)。
2. 夹具集成：在 python/sglang/test/server_fixtures/disaggregation_fixture.py 中，launch_all 启动所有子进程后调用 watcher，tearDownClass 先 stop watcher 再清理进程，避免误终止 pytest。
3. 调度器 killpg：在 python/sglang/srt/managers/scheduler.py 的异常处理块中，发送 SIGQUIT 给父进程后，若环境变量 SGLANG_KILLPG_ON_SCHEDULER_EXCEPTION 为真，则调用 os.killpg SIGKILL 整个进程组。
4. 环境变量：在 python/sglang/srt/environ.py 添加 SGLANG_KILLPG_ON_SCHEDULER_EXCEPTION = EnvBool(False)，默认关闭，用户可通过导出环境变量开启。

关键文件：

- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic）：核心调度器异常处理路径，添加可选的进程组 SIGKILL 以快速终止兄弟进程，避免 NCCL/TCPStore 大量错误日志。
- python/sglang/test/test_utils.py（模块 测试工具；类别 test；类型 test-coverage；符号 start_subprocess_fail_fast_watcher, watcher）：新增 start_subprocess_fail_fast_watcher 函数，为多进程测试提供可复用的快速终止监控器。

- python/sglang/test/server_fixtures/disaggregation_fixture.py（模块 分解夹具；类别 test；类型 test-coverage）：在 PD 分解测试夹具中集成 fail-fast watcher，监控 prefill/decode/lb 子进程。
- python/sglang/srt/environ.py（模块 环境配置；类别 source；类型 core-logic）：添加 SGLANG_KILLPG_ON_SCHEDULER_EXCEPTION 环境变量，作为 killpg 行为的开关。

关键符号：start_subprocess_fail_fast_watcher

关键源码片段

python/sglang/srt/managers/scheduler.py

核心调度器异常处理路径，添加可选的进程组 SIGKILL 以快速终止兄弟进程，避免 NCCL/TCPStore 大量错误日志。

```
# python/sglang/srt/managers/scheduler.py
# ... 前面的上下文
    except Exception:
        traceback = get_exception_traceback()
        logger.error(f"Scheduler hit an exception: {traceback}")
        parent_process.send_signal(signal.SIGQUIT)
        # Opt-in: 如果启用了 killpg 环境变量，则 SIGKILL 整个进程组，
        # 这样兄弟进程不会输出数千行 NCCL/TCPStore 回溯就立即退出。
        if envs.SGLANG_KILLPG_ON_SCHEDULER_EXCEPTION.get():
            try:
                os.killpg(os.getpgrp(), signal.SIGKILL)
            except Exception:
                pass
    finally:
        if scheduler is not None:
            # FPM 有后台 ZMQ 发布线程，需要显式关闭以刷新指标队列
            scheduler.metrics_reporter._shutdown_fpm()
```

python/sglang/test/test_utils.py

新增start_subprocess_fail_fast_watcher函数，为多进程测试提供可复用的快速终止监控器。

```
# python/sglang/test/test_utils.py
def start_subprocess_fail_fast_watcher(
    named_procs: list[tuple[str, subprocess.Popen]],
) -> threading.Event:
    """一旦任何被监控的子进程非零退出，立即终止整个测试进程。
    调用方必须在主动 teardown 前 `.set()` 返回的 Event，
    否则正常清理也会触发快速终止。"""
    stop = threading.Event()

    def watcher():
        while not stop.is_set():
            for name, proc in named_procs:
                rc = proc.poll() if proc else None
                if rc is None or rc == 0:
```

```
                continue
            if stop.is_set():
                return
            sys.stderr.write(
                f"[FIXTURE FAIL-FAST] {name} (pid={proc.pid}) exited "
                f"rc={rc}; aborting.\n"
            )
            sys.stderr.flush()
            # 杀死所有兄弟进程（但不包括已退出的这个）
            for _, sib in named_procs:
                if sib and sib is not proc:
                    try:
                        kill_process_tree(sib.pid, wait_timeout=10)
                    except Exception:
                        pass
            # POSIX: 信号退出码为 128+N（os._exit 通过 & 0xff 处理负数）
            os._exit(rc if rc >= 0 else 128 + (-rc))
        time.sleep(0.1)

    threading.Thread(target=watcher, daemon=True, name="SubprocFailFastWatcher").start()
    return stop
```

评论区精华

review 主要由 gemini-code-assist[bot] 提出三点建议：

- watcher 干扰 teardown：os._exit 会在 tearDownClass 中误杀 pytest。该问题已被修复，代码改为在 teardown 中先 .set() stop Event 再 kill 子进程，确保 watcher 正常退出不触发 os._exit。
- 跨平台 os.killpg：os.killpg 仅 Unix 存在，虽然被 try-except 保护，但 reviewer 建议显式 hasattr(os, 'killpg') 检查以避免依赖异常控制流。当前版本保持 try-except 方式。
- 动态进程引用：watcher 闭包捕获的是创建时的进程对象列表，若后续进程被重启，监控会指向过时对象。reviewer 建议在循环中从类属性动态获取进程对象。当前版本未采纳此建议，夹具中没有重启场景，风险可控。
  - Watcher 在 teardown 时错误触发 os._exit (correctness)：已在后续 commit 中修复：tearDownClass 先 stop watcher 再杀进程，且 watcher 逻辑中已在触发前检查 stop.is_set()。
  - os.killpg 跨平台兼容性 (design)：当前版本仍使用 try-except，维护者认为 Windows 非目标平台，因此保留现有方式。
  - 子进程引用动态性 (design)：当前测试夹具中不会重启子进程，风险可控，未修改实现。

风险与影响

- 风险：
  1. 测试中断风险：如果 watcher 在 teardown 前未被正确 stop，正常进程退出也可能触发 os._exit。当前通过在 tearDownClass 中先 stop 后清理规避。

2. 平台依赖：os.killpg 和 os.kill 在 Windows 上不可用，但通过 try-except 避免崩溃。如果未来需要 Windows 支持，应增加平台检测。
3. 误杀风险：调度器 killpg 默认关闭，用户需显式开启，降低意外影响。 - 影响：对默认用户无行为变更（killpg 默认关闭）。对测试开发者，PD 分解测试快速失败可节省约 30 秒至 2 分钟的重试等待时间。对运维人员，开启 killpg 后可大幅减少异常时的分布式错误日志量。影响范围限于 PD 分解测试和主动开启 killpg 的环境。 - 风险标记：平台依赖 killpg，测试中断顺序敏感，默认未启用

关联脉络

- PR #26281 [CI] Enable EPD CI for EPD architecture enhancements: 同为 PD 分解测试相关 CI 改进，本 PR 的 fail-fast watcher 可提升该 CI 场景的可靠性。