PR #30110 完整报告

sgl-project/sglang

[diffusion] fix: shut down diffusion workers on serve exit

合并时间：2026-07-04 20:42

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30110

执行摘要

- 一句话：为 diffusion 服务添加优雅关闭 worker 进程的机制
- 推荐动作：值得精读，尤其是 launch_server.py 中的层级式进程关闭模式（join → terminate → kill）设计和 kill_itself_when_parent_died 的强化实现。该 PR 为 diffusion 服务稳定性做出了重要改进。

功能与动机

在 SGLang Office Hour 7/2 讨论中，用户反馈 Ctrl-C 后后台 scheduler 进程仍然存活，需要手动 kill，有时会残留 GPU 内存占用导致 OOM（PR body 原文："Ctrl-C often leaves the background scheduler process alive, requiring manual kill and sometimes leaving enough GPU memory occupied to OOM a large model"）。本 PR 旨在通过优雅关闭机制解决该问题。

实现拆解

1. 在 launch_server.py 中添加关闭辅助函数：新增 _process_names、_join_processes_with_deadline、_terminate_alive_processes、_kill_alive_processes、_request_monolithic_scheduler_shutdown 和 shutdown_scheduler_processes。这些函数实现了带超时的层级式进程关闭：先尝试 join，超时后 terminate，再超时后 kill。
2. 在服务入口集成关闭逻辑：在 launch_server 函数的 finally 块以及 pool_disagg_launch_server 和 disagg_role_launch_server 中调用 shutdown_scheduler_processes，确保无论正常退出还是异常中断都能清理进程。
3. 重写 utils.py 中的 kill_itself_when_parent_died：移除平台分支，直接使用 Linux prctl(PR_SET_PDEATHSIG, SIGKILL) 并处理错误；如果父进程已死亡（pid 变为 1），则立即自杀，避免 worker 成为孤儿。
4. 在 gpu_worker.py 的 run_scheduler_process 入口处调用 kill_itself_when_parent_died：使得每个 worker 进程在启动时绑定父进程死亡信号。
5. 添加单元测试文件 test_launch_server_shutdown.py：使用 _FakeProcess 模拟进程行为，测试 monolithic 关闭流程（验证发送了 ShutdownReq，调用了进程的 terminate 和 kill），测试 scheduler 请求失败时仍强制执行关闭，以及非 monolithic 角色不发送关闭请求。

关键文件：

- python/sglang/multimodal_gen/runtime/launch_server.py（模块 启动器；类别 source；类型 core-logic；符号 _process_names, _join_processes_with_deadline, _terminate_alive_processes, _kill_alive_processes）：核心实现文件，添加了 shutdown_scheduler_processes 及辅助函数，并在多个服务入口集成关闭逻辑。
- python/sglang/multimodal_gen/test/unit/test_launch_server_shutdown.py（模块 测试；类别 test；类型 test-coverage；符号 _FakeProcess, init, join, is_alive）：新增单元测试，覆盖 monolithic 关闭流程、错误路径和 disagg role 跳过场景，使用 FakeProcess 模拟进程行为。
- python/sglang/multimodal_gen/utils.py（模块 工具；类别 source；类型 core-logic；符号 kill_itself_when_parent_died）：重写 kill_itself_when_parent_died 函数，使其更健壮并处理 parent 已死亡的情况。
- python/sglang/multimodal_gen/runtime/managers/gpu_worker.py（模块 GPU 工作器；类别 source；类型 core-logic；符号 run_scheduler_process）：在 worker 进程执行入口调用 kill_itself_when_parent_died，确保 worker 在父进程死亡时自动退出。

关键符号：_process_names, _join_processes_with_deadline, _terminate_alive_processes, _kill_alive_processes, _run_http_server_process, _request_monolithic_scheduler_shutdown, shutdown_scheduler_processes, kill_itself_when_parent_died, run_scheduler_process

关键源码片段

python/sglang/multimodal_gen/runtime/launch_server.py

核心实现文件，添加了 shutdown_scheduler_processes 及辅助函数，并在多个服务入口集成关闭逻辑。

```
# 带超时的进程连接，确保不会永久阻塞
def _join_processes_with_deadline(processes, timeout_s: float) -> None:
    deadline = time.monotonic() + timeout_s
    for process in processes:
        remaining_s = max(0.0, deadline - time.monotonic())
        process.join(timeout=remaining_s)

# 终止仍然存活的进程，返回依然存活的进程列表
def _terminate_alive_processes(processes, timeout_s: float) -> list:
    alive = [p for p in processes if p.is_alive()]
    if not alive:
        return []
    logger.warning("Worker process(es) did not exit in time; terminating: %s", _process_
    names(alive))
    for process in alive:
        process.terminate()
    _join_processes_with_deadline(alive, timeout_s)
    return [p for p in alive if p.is_alive()]

# 强制杀死仍然存活的进程
```

```
def _kill_alive_processes(processes, timeout_s: float) -> None:
    alive = [p for p in processes if p.is_alive()]
    if not alive:
        return
    logger.warning("Worker process(es) did not terminate in time; killing: %s", _process_
    names(alive))
    for process in alive:
        process.kill()
    _join_processes_with_deadline(alive, timeout_s)

# 向 monolithic scheduler 发送关闭请求
def _request_monolithic_scheduler_shutdown(server_args: ServerArgs) -> None:
    if server_args.disagg_role != RoleType.MONOLITHIC:
        return
    client = SchedulerClient()
    try:
        client.initialize(server_args)
        client.forward(ShutdownReq(), timeout_ms=_SCHEDULER_SHUTDOWN_TIMEOUT_MS)
    except Exception as e:
        logger.warning("Failed to request graceful scheduler shutdown: %s", e)
    finally:
        client.close()

# 主关闭入口：先尝试优雅关闭 scheduler，再逐步强制退出 workers
def shutdown_scheduler_processes(
    server_args: ServerArgs | None,
    processes: list,
    *,
    request_shutdown: bool = True,
) -> None:
    if not processes:
        return

    # 1. 尝试优雅关闭 monolithic scheduler
    if request_shutdown and server_args is not None:
        _request_monolithic_scheduler_shutdown(server_args)

    # 2. 等待 worker 进程正常退出（最长 _WORKER_JOIN_TIMEOUT_S 秒）
    _join_processes_with_deadline(processes, _WORKER_JOIN_TIMEOUT_S)

    # 3. 对未退出的 worker 发送 SIGTERM
    still_alive = _terminate_alive_processes(processes, _WORKER_TERMINATE_TIMEOUT_S)

    # 4. 对仍然未退出的 worker 发送 SIGKILL
    if still_alive:
        _kill_alive_processes(processes, _WORKER_KILL_TIMEOUT_S)
```

评论区精华

Review 建议：在请求 scheduler 关闭前检查 master 进程是否存活

gemini-code-assist[bot] 提出：如果 master 进程已经死亡（例如被直接 SIGKILL），则 scheduler 肯定未运行，此时 _request_monolithic_scheduler_shutdown 会超时等待 5 秒，造成不必要的延迟。建议增加 processes[0].is_alive() 检查。该建议未被采纳，作者未回复，PR 已合并。可能作者认为在正常关闭路径中 master 进程不会先于 scheduler 退出，或者该优化影响较小。

- 检查 master 进程存活以避免不必要的 scheduler shutdown 超时 (performance): 作者未回应，PR 已合并，未采纳该建议。

风险与影响

- 风险：
  1. 进程关闭超时风险：如果 worker 进程在 terminate 或 kill 后仍不退出（例如陷入死循环或内核态阻塞），_join_processes_with_deadline 会超时返回，可能导致关闭不彻底。已在代码中添加日志警告，但无法强制恢复。
  2. 依赖 SchedulerClient 网络调用：在 _request_monolithic_scheduler_shutdown 中会初始化 SchedulerClient 并发送请求，如果网络异常或 shcheduler 未监听，会触发异常并被 catch 后继续关闭流程，不会阻塞过程，但可能留下未处理的连接（finally 中已调用 close()）。
  3. kill_itself_when_parent_died 在非 Linux 平台直接返回：原代码已通过 if sys.platform != "linux": return 保护，不会误杀，但 worker 失去该保护。 - 影响：用户影响：使用 diffusion 服务的用户在执行 Ctrl-C 或正常关闭服务时，不再需要手动清理残留进程，避免 GPU 内存泄漏导致 OOM。系统影响：关闭流程增加最多约 (join 10s + terminate 1s + kill 1s) = 12s 的额外延迟，但通常在正常关闭时 worker 能快速退出，只有异常情况会触发更长等待。团队影响：新增的 shutdown_scheduler_processes 函数和测试为后续维护提供了清晰的关闭契约，有助于防止进程泄漏。
- 风险标记：进程关闭超时风险，依赖网络调用，非 Linux 平台保护缺失

关联脉络

- 暂无明显关联 PR