

PR #34712 完整报告

sgl-project/sglang

[Fix] Spawn, don't fork, the benchmark server process

合并时间: 2026-09-16 12:44

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/34712>

执行摘要

- 一句话: benchmark 启动改 spawn, 修复 XPU/CUDA fork 崩溃
- 推荐动作: 值得精读, 但重点不在修复本身 (核心只有一行), 而在配套测试的设计: 如何用 stub 掉设备探测、如何选择 patch 粒度 (BaseProcess 而非 get_context)、如何断言进程类型来固定 spawn 语义。这是为 " 进程启动方式 " 类缺陷补回归测试的范例, 对 benchmark 工具链及其他 spawn/fork 敏感代码的后续维护有参考价值。

功能与动机

Issue #34709 定位到根因: `launch_or_reuse_server()` 用平台默认 start method (Linux 即 fork) 启动服务器子进程, 而 CLI 父进程已在 `ServerArgs.__post_init__` 中初始化加速器——XPU 侧通过 `torch.xpu.mem_get_info()`、CUDA 侧通过 `torch.cuda.get_device_capability()` (SGLANG_OPT_FP8_WO_A_GEMM 默认开启即触发)。fork 子进程继承死掉的设备上下文, 首次设备操作即报 `RuntimeError: Cannot re-initialize XPU in forked subprocess`。PR body 强调 `launch_server()` 虽强制 spawn, 但那是在已 fork 的子进程内部才通过 `_set_envs_and_config()` 生效, 为时已晚; 该问题在纯文本 CUDA 场景被掩盖 (子进程在转交 scheduler worker 前不做设备操作), 但在 XPU 上必现, 多模态模型的 warmup 设备操作 (如 `image.to(device)`) 也会暴露。

实现拆解

1. 变更入口与根因确认: `python/sglang/benchmark/endpoint.py` 的 `launch_or_reuse_server()` 是所有 `sglang.benchmark.*` 工具共用的启动 / 复用决策点。原实现先调用 `server_args.resolve_once()` 解析参数 (会探测设备: 默认 attention 后端读 CUDA 算力、XPU 读显存信息), 随后用 `multiprocessing.Process` 启动子进程, 在 Linux 上即 fork。fork 子进程继承父进程已初始化的设备上下文, 而 CUDA/XPU 不允许在 fork 子进程中重新初始化, 首次设备操作即崩溃。`launch_server()` 内部的强制 spawn (`_set_envs_and_config()`) 发生在已 fork 的子进程内, 无法补救。
2. 核心修改: 将 `multiprocessing.Process(...)` 改为 `multiprocessing.get_context("spawn").Process(...)`, 让服务器子进程以 spawn 方式启动 (重新 import 模块、干净初始化设备)。同一提交还修正了 `BenchEndpoint._proc` 的类型注解, 从 `Optional[multiprocessing.Process]` 改为 `Optional[multiprocessing.process.BaseProcess]`——因为 `SpawnProcess` 与 `Process` 是兄弟类而非子类, 原注解在 spawn 语义下类型不成立。spawn 对 target/args 有可 pickle

要求，当前 `_launch_server_target` 是模块级函数、`server_args` 是 `msgspec` 记录，均满足。

3. 测试配套：新增 `test/registered/unit/bench/test_benchmark_endpoint.py`，注册到 CPU-only CI 套件（`register_cpu_ci`, suite `base-a-test-cpu`）。测试用 `_StubServerArgs` 把设备探测的 `resolve_once()` 替换为空操作、用模块级 `_noop_launch_server` 作为启动 `target`，`patch BaseProcess.start` 与 `server_is_up` 短路启动循环后调用 `launch_or_reuse_server()`，断言返回的进程对象是 `multiprocessing.context.SpawnProcess`。测试刻意 `patch BaseProcess` 而非 `get_context`，避免 `mock` 掉被测的 `start method`；断言对象类型而非 `mock` 调用，固定 `spawn` 语义。作者验证修复前（`fork`）红、修复后（`spawn`）绿。
4. 无其他配套：无配置、`schema`、文档或部署改动；`spawn` 增加的数秒一次性启动开销发生在 `server_is_up()` 就绪之前，不进入任何测量窗口。

关键文件：

- `python/sglang/benchmark/endpoint.py`（模块 基准工具；类别 `source`；类型 `core-logic`；符号 `launch_or_reuse_server`, `BenchEndpoint`, `acquire_endpoint`）：核心修复文件：`launch_or_reuse_server()` 是 `sglang.benchmark.*` 工具链共用的服务器启动决策点，此处把平台默认的 `fork` 改为显式 `spawn` 上下文，并同步修正 `BenchEndpoint._proc` 的类型注解，是本次变更的主路径。
- `test/registered/unit/bench/test_benchmark_endpoint.py`（模块 基准工具；类别 `test`；类型 `test-coverage`；符号 `_noop_launch_server`, `_StubServerArgs`, `TestLaunchOrReuseServer`, `test_server_child_is_a_spawn_process`）：新增的 CPU-only 回归测试，从已关闭的 #37000 移植并重构：断言服务器子进程必须是 `SpawnProcess`，修复前（`fork`）红色、修复后（`spawn`）绿色，挂载到 `base-a-test-cpu` 套件防止回归。

关键符号：`launch_or_reuse_server`, `acquire_endpoint`, `BenchEndpoint.close`, `test_server_child_is_a_spawn_process`

关键源码片段

`python/sglang/benchmark/endpoint.py`

核心修复文件：`launch_or_reuse_server()` 是 `sglang.benchmark.*` 工具链共用的服务器启动决策点，此处把平台默认的 `fork` 改为显式 `spawn` 上下文，并同步修正 `BenchEndpoint._proc` 的类型注解，是本次变更的主路径。

```
def launch_or_reuse_server(launch_server_func: Callable, server_args: ServerArgs):
    # 注意：resolve_once() 会探测设备——默认 attention 后端读取 CUDA 算力，
    # XPU 后端读取显存信息。调用后本进程就持有了活跃的设备上下文。
    server_args.resolve_once()

    base_url = resolve_base_url("", server_args.host, server_args.port)

    # 复用已运行的服务器，避免第二个实例占用端口后成为孤儿进程、
    # 并与现有实例争抢 GPU 导致 OOM。
```

```

if server_is_up(base_url, timeout=5):
    print(
        f"WARNING: reusing the server already running at {base_url} "
        f"(--model and server-launch args ignored). Pass --base-url to silence."
    )
    return None, base_url

```

核心修复：必须使用 spawn 而不是平台默认的 fork。
 # fork 会继承 resolve_once() 初始化后的设备上下文，
 # 而 CUDA/XPU 不允许在 fork 出来的子进程中重新初始化。

```

proc = multiprocessing.get_context("spawn").Process(
    target=_launch_server_target,
    args=(
        launch_server_func,
        server_args,
    ),
)
proc.start()

start_time = time.time()
while time.time() - start_time < DEFAULT_TIMEOUT:
    # 服务器在启动期间退出时快速失败（例如 OOM），
    # 而不是傻等满 600 秒超时。
    if not proc.is_alive():
        raise RuntimeError(
            f"Server process exited during startup (exit code "
            f"{proc.exitcode}); see the traceback above for the cause."
        )
    if server_is_up(base_url):
        return proc, base_url
    time.sleep(10)

```

超时：杀掉半启动的服务器，避免留下孤儿进程占着显存与端口。
 kill_process_tree(proc.pid)
 raise TimeoutError("Server failed to start within the timeout period.")

```
@dataclasses.dataclass
```

```
class BenchEndpoint:
```

```

    """一个 base URL，加上任何我们为支撑它而启动的服务器的生命周期。
    close() 会销毁已启动的服务器；对连接模式则是空操作。
    """

```

```
    base_url: str
```

SpawnProcess 是 multiprocessing.Process 的兄弟类而非子类，
 # 所以这里必须用 BaseProcess 才能同时容纳两种进程对象。

```
_proc: Optional[multiprocessing.process.BaseProcess] = None
```

```
def close(self) -> None:
```

```
    if self._proc is not None:
```

```
kill_process_tree(self._proc.pid)
self._proc = None
```

test/registered/unit/bench/test_benchmark_endpoint.py

新增的 CPU-only 回归测试，从已关闭的 #37000 移植并重构：断言服务器子进程必须是 SpawnProcess，修复前（fork）红色、修复后（spawn）绿色，挂载到 base-a-test-cpu 套件防止回归。

```
def _noop_launch_server(server_args):
    """必须是模块级函数而非 lambda：spawn 在 Process.start() 时要 pickle target。"""
```

```
class _StubServerArgs:
    """resolve_once() 是真实 ServerArgs 中探测设备的入口；stub 掉它才能让测试在纯 CPU 环境运行。创建子进程前只读取 host 和 port。"""
```

```
host = "127.0.0.1"
port = 30000
```

```
def resolve_once(self):
    pass
```

```
class TestLaunchOrReuseServer(CustomTestCase):
    def test_server_child_is_a_spawn_process(self):
        """回归测试 #34709：fork 子进程继承了父进程已初始化的加速器上下文，首次设备操作即崩溃；因此子进程必须由 spawn 创建。"""
        with (
            # 只 patch BaseProcess.start，不要 patch get_context：
            # 把 context mock 掉会连被测的 start method 一起抹除。
            mock.patch.object(multiprocessing.process.BaseProcess, "start"),
            mock.patch.object(
                multiprocessing.process.BaseProcess, "is_alive", return_value=True
            ),
            # server_is_up 第一次探测返回 False（触发启动分支），
            # 第二次返回 True（启动成功）；若启动循环再增加
            # 探测次数，需要同步扩展这个 side_effect 列表。
            mock.patch(
                "sglang.benchmark.endpoint.server_is_up", side_effect=[False, True]
            ),
        ):
            proc, _ = launch_or_reuse_server(_noop_launch_server, _StubServerArgs())

            # 断言进程对象类型，而不是断 mock 调用：类型断言直接固定 spawn 语义，
            # 防止未来有人悄悄改回 multiprocessing.Process。
            self.assertIsInstance(proc, multiprocessing.context.SpawnProcess)
```

评论区精华

最核心的讨论围绕回归测试的移植与设计展开。YuvrajSinghBhadoria2 主动提出其已关闭的 PR #37000 中有一个 CPU-only 回归测试可复用；dayanandav 采纳并做了两处关键调整：一是放到 `test/registered/unit/bench/` 并注册 `register_cpu_ci`（因为 `run_suite.py` 只发现 `test/registered/**`，原 `test/srt/` 路径不会被 CI 执行），二是断言创建出的进程对象类型 `assertIsInstance(proc, multiprocessing.context.SpawnProcess)`，而不是 mock 掉 `start` method 后断言调用——mock 掉 `get_context` 会连被测的 `start` method 一起抹除。两位 reviewer (siju-samuel、mingfeima) 均批准，mingfeima 明确认可 "explicit spawn context correctly avoids inheriting an initialized GPU context through fork"。作者还报告了三次 CI 失败（imageio 模块缺失、依赖安装超时、NPU 性能断言 $51.18 > 51.0$ ），均判定为环境问题、与本次变更无关。

- 移植 #37000 的 CPU-only 回归测试 (testing): 测试被成功移植并验证（修复前 fork 红色、修复后 spawn 绿色），原 PR 作者确认 "Glad the test was useful! Looks good to me."
- 断言 `SpawnProcess` 类型而非 mock 调用 (testing): 该写法被采纳并合并，成为后续 spawn/fork 回归测试的可复用模式。
- CI 失败归因 (other): 失败被识别为环境问题，未阻塞合并。

风险与影响

- 风险：
 1. 行为变更范围：Linux 上从 fork 切到 spawn 后，子进程不再继承父进程内存态，所有经 `launch_or_reuse_server()` 传入的 `launch_server_func` 与 `server_args` 必须可 pickle。当前均为模块级函数和 `msgspec` 记录，满足要求；但未来若有人传入闭包、lambda 或带文件句柄的对象，会在 `Process.start()` 时直接抛 `TypeError`，属于新增的隐性约束。
 2. 启动开销：spawn 需要重新 import `sglang` 及其依赖，增加数秒一次性成本；benchmark 场景下服务器启动本身耗时远超此值，且计时从 `server_is_up()` 之后开始，实际无影响。
 3. 影响面控制：改动仅触及 `python/sglang/benchmark/endpoint.py` 的工具链路径，不涉及 `scheduler`、模型执行等推理核心路径，回归风险集中在 benchmark 工具本身。
 4. 测试局限：新测试是 CPU-only 单测，验证的是进程类型而非真实设备行为；XPU/CUDA 上的实际验证依赖作者声明与后续硬件 CI，未纳入自动化。- 影响：用户侧：XPU 用户运行 `sglang.benchmark.*`（如 `one_batch_server`）不再因 fork 崩溃；CUDA 下多模态模型 warmup 设备操作的潜在崩溃被消除；Linux 与 Windows/macOS 的启动语义对齐。系统侧：无推理路径影响，输出逐位一致（作者声明已在 NVIDIA GPU 与 Intel XPU 验证）。团队侧：新增 CPU-only 回归测试挂载到 `base-a-test-cpu` 套件，后续任何把 spawn 改回 fork 的改动都会被 CI 拦截，形成可复用的 "进程启动方式" 回归模式。- 风险标记：进程启动方式变更，spawn 参数需可 pickle，影响仅限 benchmark 工具链，新增测试覆盖

关联脉络

- PR #37000（原题未提供，已关闭）CPU-only regression test for benchmark server spawn start method: YuvrajSinghBhadoria2 在 issue 评论中主动指出该已关闭 PR 提供

了完全针对此问题的 CPU-only 回归测试，本 PR 的测试文件由它移植并重构而来。