PR #27894 完整报告

sgl-project/sglang

[CI][PD] Add NIXL disaggregation functional tests

合并时间：2026-07-21 18:17

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27894

执行摘要

- 一句话：添加 NIXL 分解 E2E 功能测试
- 推荐动作：该 PR 值得精读，尤其对于需要为特定分解后端添加 E2E 测试的开发者。它展示了如何通过 fixture 类封装后端差异、如何注入故障并验证系统健壮性，以及如何在 CI 中注册此类多 GPU 测试。作者与 review 的讨论（默认值处理、异常包装）也体现了严谨的测试编写态度。

功能与动机

当前分解测试仅覆盖 Mooncake 后端；为 NIXL 提供对等的 E2E 测试，以验证 NIXL KV 传输的正确性和健壮性。

实现拆解

1. 在 python/sglang/test/server_fixtures/disaggregation_fixture.py 中添加 configure_nixl_pd_backend（配置 NIXL 传输后端）和 assert_process_healthy（携带清晰错误信息的进程健康检查）两个函数。
2. 创建 test/registered/disaggregation/test_disaggregation_nixl.py，定义 NIXL 后端配置辅助函数（_nixl_backend_config、_nixl_prefill_ucx_backend_env、_get_configured_nixl_backend_probe_error 等），用于探测和验证 NIXL 后端可用性。
3. 实现 NixlPDDisaggregationServerBase 基类（继承 PDDisaggregationServerBase），设定 TP4+TP4 拓扑，重写 start_prefill 以注入 UCX 后端参数环境变量。
4. 实现 TestDisaggregationNixlBasic 测试类，包含测试用例：启动 P/D/R 并通过 generate 请求完成 NIXL KV 传输、logprob 请求验证、请求后 worker 存活检查。
5. 实现 TestDisaggregationNixlFailure 测试类，通过 SGLANG_TEST_DISAGG_FAILURE_PROB=0.05 注入传输故障，运行 200 例 GSM8K 评估，容忍部分请求失败，并检查负载均衡器及 worker 健康。通过 register_cuda_ci 将测试注册到 base-c 阶段的 8-gpu-h20 runner。

关键文件：

- test/registered/disaggregation/test_disaggregation_nixl.py（模块 NIXL 测试；类别 test；类型 test-coverage；符号 _nixl_backend_config, _nixl_prefill_ucx_backend_env, _get_configured_nixl_backend_probe_error, _has_configured_nixl_backend）：新增文件，包含所有 NIXL 专用测试逻辑：后端探测、服务器启动、基本功能测试、故障注入测试

和 GSM8K 准确率验证。

- python/sglang/test/server_fixtures/disaggregation_fixture.py（模块 测试夹具；类别 test；类型 test-coverage；符号 configure_nixl_pd_backend, assert_process_healthy）：修改文件，添加了 configure_nixl_pd_backend 和 assert_process_healthy 两个辅助函数，供 NIXL 测试使用。

关键符号：_nixl_backend_config, _get_configured_nixl_backend_probe_error, _require_configured_nixl_backend, configure_nixl_pd_backend, assert_process_healthy, NixlPDDisaggregationServerBase.start_prefill

关键源码片段

test/registered/disaggregation/test_disaggregation_nixl.py

新增文件，包含所有 NIXL 专用测试逻辑：后端探测、服务器启动、基本功能测试、故障注入测试和 GSM8K 准确率验证。

```
import json
import os
import uuid
from sglang.srt.environ import envs

def _nixl_backend_config(
    backend,
    backend_params_json,
    ucx_num_threads=2, # 默认 2 个 UCX 线程，降低 mlx 设备负载
):
    # 解析后端参数 JSON 字符串
    backend_params = json.loads(backend_params_json)
    # 校验必须是 dict[str, str]
    if not isinstance(backend_params, dict) or not all(
        isinstance(key, str) and isinstance(value, str)
        for key, value in backend_params.items()
    ):
        raise ValueError(
            "SGLANG_DISAGGREGATION_NIXL_BACKEND_PARAMS must be a JSON object "
            "with string keys and string values"
        )
    # 根据不同后端设置默认线程数
    if backend == "UCX":
        backend_params["num_threads"] = str(ucx_num_threads)
    elif backend == "OBJ":
        backend_params.setdefault("num_threads", "8")
    elif backend == "GDS_MT":
        backend_params.setdefault("thread_count", "8")
    elif backend == "UCCL":
        backend_params.setdefault("num_cpus", "8")
    return backend, backend_params

def _get_configured_nixl_backend_probe_error():
```

```
    # 读取环境变量配置
    backend = envs.SGLANG_DISAGGREGATION_NIXL_BACKEND.get()
    backend_params_json = envs.SGLANG_DISAGGREGATION_NIXL_BACKEND_PARAMS.get()
    # 尝试导入 NIXL 库
    try:
        from nixl._api import nixl_agent, nixl_agent_config, nixl_thread_sync_t
    except ImportError as e:
        return f"NIXL import failed: {e}"
    # 解析并验证后端参数
    try:
        backend, backend_params = _nixl_backend_config(backend, backend_params_json)
    except (json.JSONDecodeError, ValueError) as e:
        return str(e)
    # 创建探针 agent 并尝试创建后端
    try:
        probe_num_threads = 2 if backend == "UCX" else 8
        agent_config = nixl_agent_config(
            backends=[],
            num_threads=probe_num_threads,
            sync_mode=nixl_thread_sync_t.NIXL_THREAD_SYNC_STRICT,
        )
        agent = nixl_agent(f"sglang_nixl_probe_{uuid.uuid4()}", agent_config)
        available_plugins = agent.get_plugin_list()
        if backend not in available_plugins:
            return (
                f"NIXL backend {backend!r} not found. "
                f"Available plugins: {available_plugins}."
            )
        agent.create_backend(backend, backend_params)
    except Exception as e:
        return f"NIXL backend probe failed: {e}"
    return None
```

python/sglang/test/server_fixtures/disaggregation_fixture.py

修改文件，添加了 configure_nixl_pd_backend 和 assert_process_healthy 两个辅助函数，供 NIXL 测试使用。

```
import requests

def configure_nixl_pd_backend(test_cls):
    # 设置分解传输后端为 NIXL
    test_cls.transfer_backend = ["--disaggregation-transfer-backend", "nixl"]
    # NIXL 的网络配置由环境变量驱动，不需要 Mooncake 的 --disaggregation-ib-device
    test_cls.rdma_devices = []

def assert_process_healthy(test_case, name, process, url, health_path="/health"):
    # 确保进程存在
    test_case.assertIsNotNone(process, f"{name} process was not started")
    # 确保进程仍在运行
```

```
    test_case.assertIsNone(
        process.poll(),
        f"{name} exited unexpectedly with code {process.returncode}",
    )
    # 通过健康端点检查服务是否可用
    try:
        response = requests.get(f"{url}{health_path}", timeout=10)
    except requests.RequestException as e:
        test_case.fail(f"Failed to connect to {name} health endpoint: {e}")
    test_case.assertEqual(response.status_code, 200, response.text)
```

评论区精华

- SGLANG_DISAGGREGATION_NIXL_BACKEND_PARAMS 未设置时的潜在崩溃：gemini-code-assist[bot] 指出若环境变量未设置，envs.get() 返回 None 导致 json.loads() 引发 TypeError。作者澄清，在 envs 实现中，未设置的环境变量默认返回 "{}"，而非 None，因此不存在该问题。
- 健康检查异常处理：bot 建议在 assert_process_healthy 中包装 requests.get 的异常，提供更清晰的失败信息。作者采纳并实现了 try-except-fail 模式。
- 文件组织：ShangmingCai 建议将 disaggregation_utils.py 中的函数直接合并到现有的 disaggregation_fixture.py，避免创建新文件。作者采纳并迁移。
- RDMA 支持和 UCX 内存错误调试：在 CI 运行中发现 UCX 初始化失败（Out of memory），作者通过减少 UCX 线程数（num_threads=2）来降低 mlx 设备负载，最终解决。
  - 环境变量未设置时的错误处理 (correctness): 作者解释在 envs 实现中未设置的环境变量默认返回 '{}'，因此不会出现 None。但在最终代码中未额外处理，仅通过测试通过验证。
  - assert_process_healthy 异常处理 (correctness): 作者采纳建议，在 assert_process_healthy 中添加了 try-except 块，捕获 RequestException 并调用 test_case.fail。
  - 文件组织：合并到现有 fixture (design): 作者采纳，将 configure_nixl_pd_backend 和 assert_process_healthy 迁移到 disaggregation_fixture.py，并删除了 utils 文件。

风险与影响

- 风险：
  - 环境依赖：测试依赖 NIXL 库、RDMA 设备和特定 GPU 拓扑（8 卡 H20），在非符合环境的机器上会通过 _has_configured_nixl_backend 跳过（CI 外）或报错。
  - 故障注入稳定性：故障注入测试依赖随机概率，可能因故障率波动导致测试断言不稳定。当前 GSM8K 阈值 0.62 与 Mooncake 一致，但 200 例样本下方差较大。
  - CI 资源：测试需要 8-GPU runner，且运行时间较长（est_time=700s），可能加重 CI 队列压力。
- 影响：
  - 用户：无直接影响，仅为测试增强。
  - 系统：增加 CI 中对 NIXL 分解的回归覆盖，有助于及早发现 NIXL 相关回归。

- 团队：提供可复用的 NIXL 端到端测试模式和辅助函数，降低后续为 NIXL 添加测试的门槛。
- 风险标记：测试环境依赖，随机故障注入波动，CI 资源占用

关联脉络

- PR #31035 Fix libnvshmem missing issue: 依赖修复，该 PR 为 NIXL 测试提供了 libnvshmem 库，否则测试会因 ImportError 失败。
- PR #30997 NIXL functional tests passed on 8-gpu-h20: 参考 PR，验证测试在 CI 上通过，为后续合并提供信心。