PR #30748 完整报告

sgl-project/sglang

Route PD server warmup to every DP rank

合并时间：2026-07-15 15:59

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30748

执行摘要

- 一句话：显式路由 PD warmup 到每个 DP rank，修复覆盖不全问题
- 推荐动作：值得精读。设计上使用显式路由到每个 DP rank、并通过 asyncio.gather 并发控制，是典型的批量初始化模式。建议关注 return_exceptions 讨论，根据实际容错需求决定是否添加。

功能与动机

根因是 bootstrap-room affinity 在 total_tokens 等显式负载均衡策略下不被使用，startup warmup 不能保证每个 rank 都执行 dry run（PR body 描述）。某个 rank 首次被真实流量接触时才触发延迟的 JIT 或初始化成本，引入 rank skew。

实现拆解

1. 新增异步预热函数：在 http_server.py 中新增 _send_disaggregation_warmup_requests，接收 server_args、url、headers、ssl_verify 和 timeout 参数，返回每个 rank 的状态码列表。
2. 构造每个 rank 的显式路由请求：内部 send_request 协程为单个 dp_rank 构建 JSON 体，其中 routed_dp_rank 和 bootstrap_room 均设置为该 rank 值，bootstrap_host 固定为 FAKE_BOOTSTRAP_HOST，输入为标量 input_ids。
3. 并发发送所有请求：函数内创建 aiohttp.ClientSession，使用 asyncio.gather 并发执行所有 rank 的 send_request，所有请求共享同一个 session。
4. 修改预热入口：在 _execute_server_warmup 中，将原来单次 requests.post 批次请求替换为调用 asyncio.run(_send_disaggregation_warmup_requests(...))，检查返回的状态码列表，仅当全为 200 时才标记预热成功。
5. 新增单元测试：新建 test_http_server_warmup.py，通过 mock aiohttp.ClientSession 验证函数并发发送 N 个请求、每个请求的 routed_dp_rank 覆盖 0~dp_size-1，且所有响应状态码 200。

关键文件：

- python/sglang/srt/entrypoints/http_server.py（模块 预热逻辑；类别 source；类型 core-logic；符号 _send_disaggregation_warmup_requests, send_request, _execute_server_warmup）：核心修改文件，新增异步并发 warmup 函数并修改预热入口。

- test/registered/unit/entrypoints/test_http_server_warmup.py（模块 预热测试；类别 test；类型 test-coverage；符号 TestDisaggregationServerWarmup, test_sends_concurrent_scalar_request_to_each_dp_rank, Response, Session）：新增测试验证并发发送到每个 rank 的正确性。

关键符号：_send_disaggregation_warmup_requests, send_request, _execute_server_warmup

关键源码片段

python/sglang/srt/entrypoints/http_server.py

核心修改文件，新增异步并发 warmup 函数并修改预热入口。

```
async def _send_disaggregation_warmup_requests(
    server_args: ServerArgs,
    url: str,
    headers: Dict[str, str],
    ssl_verify: Union[bool, str],
    timeout: int,
) -> List[int]:
    """发送异步并发 warmup 请求到每个 DP rank。"""
    # 根据 ssl_verify 创建 SSL 上下文
    ssl_context = (
        ssl_verify
        if isinstance(ssl_verify, bool)
        else ssl.create_default_context(cafile=ssl_verify)
    )

    # 内部函数：为单个 dp_rank 发送一个 /generate 请求
    async def send_request(session: aiohttp.ClientSession, dp_rank: int) -> int:
        # 构造请求体，显式指定 routed_dp_rank 和 bootstrap_room
        json_data = {
            "sampling_params": {"temperature": 0.0, "max_new_tokens": 8, "ignore_eos": True},
            "bootstrap_host": FAKE_BOOTSTRAP_HOST,
            "bootstrap_room": dp_rank, # 作为 rank 唯一标识
            "input_ids": [10, 11, 12, 13],
            "routed_dp_rank": dp_rank, # 显式路由到此 rank
        }
        async with session.post(url + "/generate", json=json_data, ssl=ssl_context) as response:
            await response.read()
            return response.status

    # 使用共享的 aiohttp ClientSession 并发发送所有请求
    async with aiohttp.ClientSession(
        timeout=aiohttp.ClientTimeout(total=timeout),
        headers=headers,
    ) as session:
        # 注意：此处未设置 return_exceptions=True，若某个请求失败会直接抛出异常
        return await asyncio.gather(
```

```
            *(send_request(session, dp_rank) for dp_rank in range(server_args.dp_size))
        )
```

test/registered/unit/entrypoints/test_http_server_warmup.py

新增测试验证并发发送到每个 rank 的正确性。

```
class TestDisaggregationServerWarmup(unittest.IsolatedAsyncioTestCase):
    async def test_sends_concurrent_scalar_request_to_each_dp_rank(self):
        server_args = SimpleNamespace(dp_size=4)
        all_started = asyncio.Event()
        calls = []
        sessions = []

        class Response:
            # mock 响应：状态码 200，__aenter__ 屏障确保并发
            status = 200

            async def __aenter__(self):
                # 当所有 dp_size 个调用都到达后释放屏障
                if len(calls) == server_args.dp_size:
                    all_started.set()
                await asyncio.wait_for(all_started.wait(), timeout=5)
                return self

            async def __aexit__(self, *args):
                pass

            async def read(self):
                return b""

        class Session:
            # mock ClientSession，记录每个调用及其 kwargs
            def __init__(self, **kwargs):
                self.kwargs = kwargs
                sessions.append(self)

            async def __aenter__(self):
                return self

            async def __aexit__(self, *args):
                pass

            def post(self, *args, **kwargs):
                calls.append((args, kwargs))
                return Response()

        with patch("sglang.srt.entrypoints.http_server.aiohttp.ClientSession", Session):
            status_codes = await _send_disaggregation_warmup_requests(
                server_args=server_args,
```

```
            url="http://localhost:30000",
            headers={"Authorization": "Bearer token"},
            ssl_verify=False,
            timeout=123,
        )

        # 验证所有 rank 都收到 200
        self.assertEqual(status_codes, [200] * server_args.dp_size)
        # 验证每个 rank 的请求体中 routed_dp_rank 唯一且覆盖 0..dp_size-1
        calls_by_rank = {kwargs["json"]["routed_dp_rank"]: (args, kwargs) for args, kwargs in calls}
        self.assertEqual(set(calls_by_rank), set(range(server_args.dp_size)))
```

评论区精华

审阅者 ShangmingCai 建议在 asyncio.gather 中添加 return_exceptions=True（line 2055），以避免单个 rank 请求失败时整个预热抛出异常而中断。该建议未被作者采纳，最终提交未包含该参数。讨论状态未关闭，仍属未解决事项。

- 建议在 asyncio.gather 中添加 return_exceptions=True (design): 未采纳，代码保持无 return_exceptions，单个 rank 失败将直接中断预热。

风险与影响

- 风险：
  1. 启动时间增加：现在发送 dp_size 个并发请求，但预热超时长达 1800 秒（默认），实际并发开销小，风险可控。
  2. 单个 rank 失败风险：未设置 return_exceptions=True，若某个 rank 的预热请求失败（如网络抖动），asyncio.gather 会直接抛出异常，导致服务器就绪失败。但这也是一种严格保障，确保所有 rank 都预热成功。
  3. 依赖aiohttp：新增导入aiohttp，但该库已是项目常用依赖，不需额外安装，风险极低。 - 影响：影响范围：仅改动 disaggregated 预热路径，不涉及正常请求路由。对用户：DP-Attention 部署的 PD 服务器在启动时可靠地预热所有 rank，消除首次流量时的 rank skew。对系统：启动时多出 dp_size 个并发 HTTP 请求，但总耗时相近，因为原有批次请求本身需要等待所有 rank 处理。对团队：代码更清晰，可维护性改善。 - 风险标记：单个 DP rank 失败会阻止服务器就绪，启动时间线性增长（dp_size 个并发请求）

关联脉络

- 暂无明显关联 PR