PR #23755 完整报告

sgl-project/sglang

[SGLang Tracing] Add pd disaggregation mooncake backend tracing

合并时间：2026-06-03 16:43

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23755

执行摘要

- 一句话：为 Mooncake 后端添加 PD 分解追踪功能
- 推荐动作：值得精读。该 PR 展示了如何在现有 OpenTelemetry 追踪框架中安全添加模块化追踪，其 copy_for_thread 跨线程上下文传播设计可复用，trace_modules 过滤模式也值得参考。

功能与动机

提升 PD 分离系统中 Mooncake 后端的数据传输可观测性，帮助开发者调试和性能分析。PR 描述提出为 Mooncake 后端实现追踪，并作为多模块追踪能力的首次落地。

实现拆解

1. 新增多模块追踪控制：在 server_args.py 中添加 --trace-modules 参数（默认 "request"），trace.py 中的 TraceReqContext.__init__ 根据 server_args.trace_modules 判断当前模块是否启用，若不启用则设置 tracing_enable=False，实现模块粒度的开关。
2. 定义 Mooncake 追踪阶段：新建 mooncake_trace.py，定义 MooncakeRequestStage 类包含 5 个阶段（如 MOONCAKE_SEND、MOONCAKE_WORKER_SEND），并提供两个工具函数 mooncake_trace_slice（直接追踪时间段）和 mooncake_trace_func（装饰器自动追踪函数执行）。
3. 扩展核心追踪上下文：在 trace.py 中重命名 __get_host_id 为 _get_host_id，新增 copy_for_thread 方法用于跨线程复制追踪上下文（保留 root_span_context，重建 thread_context），并增加 TraceNullContext.is_copy 标记。
4. 在 Mooncake 连接器中集成：在 conn.py 的 MooncakeKVSender 中添加 _init_trace_ctx 方法初始化 trace_ctx，在 transfer_worker 线程启动时设置线程信息，处理每个 TransferKVChunk 时重建上下文并记录阶段开始 / 结束；在 poll 和 abort 方法中调用 trace_req_finish 完成请求追踪。
5. 数据传输单元携带上下文：在 TransferKVChunk 中新增 trace_ctx 字段（默认为 TraceNullContext），实现数据与追踪上下文一起传递。
6. 测试与文档配套：更新单元测试 test_trace.py 以 mock get_global_server_args；更新集成测试 test_tracing_disaggregation.py 增加 --trace-modules request,mooncake；更新文档 server_arguments.mdx 说明新参数。

关键文件：

- python/sglang/srt/observability/mooncake_trace.py（模块 追踪框架；类别 source；类型 core-logic；符号 MooncakeRequestStage, mooncake_trace_slice, mooncake_trace_func）：核心新增文件，定义 Mooncake 追踪阶段和工具函数，是 PR 的主体。
- python/sglang/srt/observability/trace.py（模块 追踪框架；类别 source；类型 core-logic；符号 _get_host_id, copy_for_thread）：核心追踪框架修改，新增 copy_for_thread 方法，重命名 _get_host_id，并增加模块过滤逻辑。
- python/sglang/srt/disaggregation/mooncake/conn.py（模块 Mooncake 连接器；类别 source；类型 core-logic；符号 _init_trace_ctx, abort）：Mooncake 连接器核心修改，在发送 / 接收逻辑中集成追踪调用。
- python/sglang/srt/disaggregation/common/utils.py（模块 通用工具；类别 source；类型 dependency-wiring）：在 TransferKVChunk 中添加 trace_ctx 字段，实现上下文随数据传递。
- python/sglang/srt/server_args.py（模块 配置；类别 source；类型 core-logic）：新增 --trace-modules 参数，支持模块级追踪控制。
- test/registered/unit/observability/test_trace.py（模块 测试；类别 test；类型 test-coverage；符号 _mock_get_global_server_args）：单元测试更新，mock get_global_server_args 以适配新依赖。
- test/registered/observability/test_tracing_disaggregation.py（模块 测试；类别 test；类型 test-coverage）：集成测试配置更新，添加 --trace-modules 参数以验证 Mooncake 追踪。
- scripts/convert_otel_2_perfetto.py（模块 脚本；类别 source；类型 bugfix）：对 rid 属性进行防御性获取，避免因缺失 KeyError。
- docs_new/docs/advanced_features/server_arguments.mdx（模块 文档；类别 other；类型 documentation）：文档更新，添加 --trace-modules 参数说明。

关键符号：mooncake_trace_slice, mooncake_trace_func, copy_for_thread, _init_trace_ctx, _get_host_id

关键源码片段

python/sglang/srt/observability/mooncake_trace.py

核心新增文件，定义 Mooncake 追踪阶段和工具函数，是 PR 的主体。

```
import time
from typing import Union

from sglang.srt.observability.req_time_stats import (
    RequestStageConfig,
    convert_time_to_realtime_ns,
)
from sglang.srt.observability.trace import TraceNullContext, TraceReqContext

class MooncakeRequestStage:
    # 定义 Mooncake 数据传输的各个阶段，每个阶段有一个名称和追踪级别
```

```
    MOONCAKE_SEND = RequestStageConfig("mooncake_send", level=1)
    MOONCAKE_RECV = RequestStageConfig("mooncake_recv", level=1)
    MOONCAKE_WORKER_SEND = RequestStageConfig("mooncake_worker_send", level=1)
    MOONCAKE_WORKER_SEND_SESSION = RequestStageConfig(
        "mooncake_worker_send_session", level=2
    )
    MOONCAKE_WORKER_RECV = RequestStageConfig("mooncake_worker_recv", level=1)


def mooncake_trace_slice(
    trace_ctx: Union[TraceReqContext, TraceNullContext],
    stage: RequestStageConfig,
    start_ts: float,
    thread_finish_flag=False,
):
    """
    记录一个完整的时间段（start -> end）。
    如果 trace_ctx 为 None 或未启用追踪，则直接返回。
    """
    if trace_ctx is None:
        return
    if not trace_ctx.tracing_enable:
        return
    # 将相对时间转换为纳秒级实时时间戳
    start_ts = convert_time_to_realtime_ns(start_ts)
    trace_ctx.trace_slice_start(stage.stage_name, stage.level, start_ts)
    trace_ctx.trace_slice_end(
        stage.stage_name,
        stage.level,
        thread_finish_flag=thread_finish_flag,
    )


def mooncake_trace_func(stage: RequestStageConfig):
    """
    装饰器：自动在函数执行前后添加追踪事件。
    用于需要追踪整个函数执行时间的场景。
    """
    def decorator(func):
        def wrapper(self, *args, **kwargs):
            if self.trace_ctx is None:
                return func(self, *args, **kwargs)
            start_ts = convert_time_to_realtime_ns(time.perf_counter())
            self.trace_ctx.trace_slice_start(stage.stage_name, stage.level, start_ts)
            ret = func(self, *args, **kwargs)
            self.trace_ctx.trace_slice_end(stage.stage_name, stage.level)
            return ret
        return wrapper
    return decorator
```

python/sglang/srt/observability/trace.py

核心追踪框架修改，新增 copy_for_thread 方法，重命名 _get_host_id，并增加模块过滤逻辑。

```
def copy_for_thread(self) -> "TraceReqContext":
    """
    创建一个当前追踪上下文的副本，用于跨线程传播。
    副本共享 root_span_context，但拥有独立的 thread_context。
    接收方需要调用 rebuild_thread_context() 来初始化线程状态。
    """
    # 快速路径：未启用追踪或没有根上下文时返回空上下文
    if not self.tracing_enable or not self.root_span_context:
        return TraceNullContext()

    # 从当前线程提取上一个 span 上下文（如果有子 span 则取子 span）
    prev_span_context = self.last_span_context
    if self.thread_context and self.thread_context.cur_slice_stack:
        cur_slice = self.thread_context.cur_slice_stack[0]
        if cur_slice.span:
            prev_span_context = cur_slice.span.get_span_context()

    # 创建新实例，共享 root_span_context（不可变，无需深拷贝）
    copied = TraceReqContext.__new__(TraceReqContext)
    copied.tracing_enable = self.tracing_enable
    copied.rid = self.rid
    copied.bootstrap_room = self.bootstrap_room
    copied.start_time_ns = self.start_time_ns
    copied.role = self.role
    copied.trace_level = self.trace_level
    copied.module_name = self.module_name
    copied.is_copy = True # 标记为副本，便于调试
    copied.pid = self.pid
    # thread_context 设为 None，接收方线程需调用 rebuild_thread_context()
    copied.thread_context = None
    copied.root_span = None
    copied.root_span_context = self.root_span_context
    copied.last_span_context = prev_span_context
    return copied
```

python/sglang/srt/disaggregation/mooncake/conn.py

Mooncake 连接器核心修改，在发送 / 接收逻辑中集成追踪调用。

```
def _init_trace_ctx(self):
    # 如果全局启用了追踪，则创建 TraceReqContext
    # 否则直接使用 TraceNullContext
    if get_global_server_args().enable_trace:
        self.trace_ctx = TraceReqContext(
            rid=str(hex(self.bootstrap_room)), # 使用房间号作为请求 ID
            bootstrap_room=self.bootstrap_room,
            role="Sender",
```

```
            module_name="mooncake", # 模块名，用于 trace_modules 过滤
        )
        # 如果当前模块未在 trace_modules 中，tracing_enable 将被设为 False
        if not self.trace_ctx.tracing_enable:
            self.trace_ctx = TraceNullContext() # 回退为空上下文
    else:
        self.trace_ctx = TraceNullContext()

# 在 transfer_worker 线程循环中使用（关键片段）:
# 从队列获取 chunk 后：
if self.enable_trace:
    # 重建工作线程的追踪上下文（因为 chunk 是从主线程传递过来的副本）
    kv_chunk.trace_ctx.rebuild_thread_context()
    # 记录 worker_send 阶段开始
    kv_chunk.trace_ctx.trace_slice_start(
        MooncakeRequestStage.MOONCAKE_WORKER_SEND.stage_name,
        MooncakeRequestStage.MOONCAKE_WORKER_SEND.level,
    )
# ... 处理传输 ...
if self.enable_trace:
    # 记录 worker_send 阶段结束
    kv_chunk.trace_ctx.trace_slice_end(
        MooncakeRequestStage.MOONCAKE_WORKER_SEND.stage_name,
        MooncakeRequestStage.MOONCAKE_WORKER_SEND.level,
        thread_finish_flag=True,
    )
```

评论区精华

- 函数命名风格：ShangmingCai 指出 __init_trace_ctx 双下划线前缀奇怪，sufeng-buaa 改为 _init_trace_ctx。
- 门控逻辑设计：ShangmingCai 询问 enable_trace 与 trace_ctx.tracing_enable 的区别，sufeng-buaa 解释前者是全局开关，后者进一步受 trace_modules 过滤，不匹配时设为 TraceNullContext 避免后续检查；ShangmingCai 建议在 transfer_worker 中保留 get_global_server_args().enable_trace 门控以提高可读性，最终保留。
- 循环导入预防：ShangmingCai 指出 trace.py 直接导入 ServerArgs 可能导致循环导入，sufeng-buaa 改为 TYPE_CHECKING 条件导入。
- abort 时缺失追踪结束：ShangmingCai 指出 abort 方法需要调用 trace_req_finish 以正确结束追踪周期，sufeng-buaa 修复。
- 测试配置位置：ShangmingCai 指出追踪日志注入在 transfer_worker（prefill 侧），测试中的 --trace-modules 应加在 prefill 而非 decode 侧，sufeng-buaa 确认并修正。
- 函数命名双下划线 (style): 改为单下划线 _init_trace_ctx。
- trace_ctx 门控逻辑设计 (design): 保留 enable_trace 门控作为第一道检查，避免在未启用时执行额外代码。
- 循环导入风险 (design): 改为 TYPE_CHECKING 条件导入，避免运行时循环依赖。

- abort 时缺失 trace_req_finish (correctness): 在 abort 中调用 self.trace_ctx.trace_req_finish()。
- 测试参数位置 (testing): 将 --trace-modules request,mooncake 移至 prefill 侧参数。

风险与影响

- 风险:
  - 性能开销：在 transfer_worker 等热路径注入追踪事件会增加延迟，但通过 enable_trace 门控和 TraceNullContext 短路可避免不必要开销。
  - 跨线程上下文正确性：copy_for_thread 方法需要正确处理 root_span_context 共享和 thread_context 重建，若实现有误可能导致 span 错乱或内存泄漏。
  - 模块控制兼容性：新增 --trace-modules 默认 "request"，已有仅使用 --enable-trace 的脚本不会自动启用 mooncake 追踪，符合预期但需用户知晓。
  - 分布式追踪标识：使用 rid（房间号）作为关联键，在主键冲突时可能导致追踪关联错误，但 Mooncake 内部使用 bootstrap_room 保证唯一性。
- 影响:
  - 对用户：启用 --trace-modules mooncake 后，可在 Jaeger/Perfetto 中观察到 Mooncake 数据传输的详细阶段，提升 PD 分离调试效率。
  - 对系统：新增一个模块级追踪开关，不影响现有请求追踪行为；Mooncake 连接器代码增加约 70 行追踪逻辑。
  - 对团队：为后续扩展其他模块追踪（如 HiCache）提供了可复用的模式：阶段定义 + 工具函数 + 上下文传播。
  - 风险标记：跨线程上下文传播，性能开销，模块控制兼容性

关联脉络

- 暂无明显关联 PR