PR #23169 完整报告

sgl-project/sglang

feat(observability): add OpenTelemetry tracing for pipeline parallelism

合并时间：2026-04-28 17:05

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23169

执行摘要

- 一句话：为 PP pipeline 添加 OpenTelemetry 追踪
- 推荐动作：值得精读。PR 展示了如何在调度循环的轻量热路径中注入可观测性代码，设计上注重最小侵入和性能开销控制。命名讨论和属性传递方式的简化体现了对可维护性的关注，可作为同类追踪功能的参考模式。

功能与动机

根据可观测性路线图 issue #13511，实现 PP（Pipeline Parallelism）场景的 OpenTelemetry 追踪，以分析 PP 各阶段的耗时分布。

实现拆解

1. 新增阶段定义和时间记录方法：在 RequestStage 枚举中新增 RUN_BATCH_CPU 阶段（level=4），在 SchedulerReqTimeStats 中新增 run_batch_cpu_start_time 字段以及 set_run_batch_cpu_start_time / set_run_batch_cpu_end_time 方法，后者调用 trace_slice 记录起止时间跨度。
2. 在 PP 调度循环中插桩：修改 python/sglang/srt/managers/scheduler_pp_mixin.py 的 _pp_launch_batch 方法，调用 set_time_batch 在 run_batch 前后分别记录 set_run_batch_cpu_start_time 和 set_run_batch_cpu_end_time（仅追踪启用时），并将 pp_mb_id 作为属性传入。
3. 扩展线程上下文与属性传播：在 python/sglang/srt/observability/trace.py 的 TraceThreadInfo 增加 pp_rank 字段，trace_set_thread_info 接收 pp_rank 参数；在 __create_thread_context 中将 PP rank 追加到线程名称和 span 属性中；调整 python/sglang/srt/managers/scheduler.py 中调用处传入 pp_rank。
4. 更新 Perfetto 转换脚本：在 scripts/convert_otel_2_perfetto.py 的 generate_perfetto_span 中，若存在 pp_rank 属性，将其附加到线程名称后，例如 -PP0。
5. 测试适配：修改 test/registered/unit/observability/test_trace.py 中 TraceThreadInfo 的构造调用，补上 pp_rank 参数以通过类型检查。

关键文件：

- python/sglang/srt/observability/req_time_stats.py（模块 可观测性；类别 source；类型 core-logic；符号 set_run_batch_cpu_start_time, set_run_batch_cpu_end_time, set_time_batch）：核心变更：新增 RUN_BATCH_CPU 阶段定义、时间字段起止记录方法

，修改 set_time_batch 以支持 attrs 参数。

- python/sglang/srt/managers/scheduler_pp_mixin.py（模块 调度器；类别 source；类型 dependency-wiring）：在 PP 调度循环中插桩，调用 set_time_batch 记录 run_batch 起止时间，是追踪数据的埋入点。
- python/sglang/srt/observability/trace.py（模块 可观测性；类别 source；类型 core-logic）：扩展 TraceThreadInfo 和 trace_set_thread_info 以包含 pp_rank，在 span 名称和属性中添加 PP rank。
- scripts/convert_otel_2_perfetto.py（模块 转换工具；类别 source；类型 core-logic）：在 Perfetto 转换脚本中解析 pp_rank 和 dp_rank，以便在可视化线程名称中显示。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic）：在调度器启动时传入 pp_rank 参数，使线程上下文正确关联 PP rank。
- test/registered/unit/observability/test_trace.py（模块 测试；类别 test；类型 test-coverage）：适配 TraceThreadInfo 构造函数的新 pp_rank 参数，修复测试失败。

关键符号：set_run_batch_cpu_start_time, set_run_batch_cpu_end_time, set_time_batch, trace_set_thread_info

关键源码片段

python/sglang/srt/observability/req_time_stats.py

核心变更：新增 RUN_BATCH_CPU 阶段定义、时间字段起止记录方法，修改 set_time_batch 以支持 attrs 参数。

```
# 在 RequestStage 枚举中新增 CPU 侧 run_batch 阶段
class RequestStage:
    # ... 其他阶段 ...
    RUN_BATCH_CPU = RequestStageConfig(
        'run_batch_cpu',
        level=4, # level=4 表示在 prefill_forward / decode_forward 内部更深一层
    )

class SchedulerReqTimeStats(ReqTimeStatsBase):
    # ... 其他字段 ...
    run_batch_cpu_start_time: float = 0.0 # 记录 CPU run_batch 起始时间

    def set_run_batch_cpu_end_time(self, ts=None, attrs=None):
        '''结束 CPU run_batch 时间片，记录 trace span'''
        ts = ts or time.perf_counter()
        if self.run_batch_cpu_start_time > 0.0:
            # 使用 trace_slice 记录 span
            self.trace_slice(
                RequestStage.RUN_BATCH_CPU,
                self.run_batch_cpu_start_time,
                ts,
                attrs
            )
```

```
            self.run_batch_cpu_start_time = 0.0 # 重置，避免重复记录

def set_time_batch(
    reqs: List[Any],
    set_func: str,
    trace_only: bool = False,
    attrs: Optional[Dict[str, Any]] = None, # 新增 attrs 参数支持传递自定义属性
):
    if reqs is None or len(reqs) == 0:
        return
    if trace_only and not get_global_tracing_enabled():
        return

    ts = time.perf_counter()
    for req in reqs:
        method = getattr(req.time_stats, set_func)
        if attrs is None:
            method(ts)
        else:
            method(ts, attrs) # 传递属性给设置方法
```

python/sglang/srt/managers/scheduler_pp_mixin.py

在 PP 调度循环中插桩，调用 set_time_batch 记录 run_batch 起止时间，是追踪数据的埋入点。

```
from sglang.srt.observability.req_time_stats import set_time_batch

class SchedulerPPMixin:
    def _pp_launch_batch(self: Scheduler, mb_id, pp_proxy_tensors, mb_metadata, last_rank_
    comm_queue):
        with torch.profiler.record_function('run_batch'):
            with self.forward_stream_ctx:
                self.forward_stream.wait_stream(self.schedule_stream)
                # 记录 CPU run_batch 开始时间，仅追踪启用时生效
                set_time_batch(
                    self.cur_batch.reqs,
                    'set_run_batch_cpu_start_time',
                    trace_only=True,
                )
                result = self.run_batch(self.cur_batch, pp_proxy_tensors)
                # 记录 CPU run_batch 结束时间，并附加 PP micro-batch ID 作为属性
                set_time_batch(
                    self.cur_batch.reqs,
                    'set_run_batch_cpu_end_time',
                    trace_only=True,
                    attrs={'pp_mb_id': mb_id},
                )
                # 其余处理 ...
```

评论区精华

阶段命名争议

Reviewer sufeng-buaa 指出原始名称 pp_forward 不够精确，因为追踪的是 CPU 侧的 run_batch 而非整个 forward，建议改为 run_batch_cpu。作者接受并修改。

属性传递方式简化

sufeng-buaa 建议避免构建复杂的 _pp_trace_attrs 方法，直接将简单属性如 pp_mb_id 通过 set_time_batch 的 attrs 参数传递。作者采纳，最终实现移除多余方法，直接传递 {'pp_mb_id': mb_id}。

trace_enabled 缓存讨论

ShangmingCai 认为 get_global_tracing_enabled() 应缓存为调度器属性，减少调用开销。作者解释该函数仅返回全局变量，性能开销极小，无需缓存。

测试修复

ShangmingCai 报告 CI 中 test_trace_thread_context 因 TraceThreadInfo 缺少 pp_rank 参数而失败。作者修改测试用例构造，补上参数。

- 阶段命名调整：PP_FORWARD 改为 RUN_BATCH_CPU (design): 作者接受建议并修改。
- 属性传递方式简化：取消 _pp_trace_attrs (design): 作者采纳，最终实现仅传递 {'pp_mb_id': mb_id}。
- trace_enabled 检查是否应缓存 (performance): 未缓存，保持当前实现。
- 测试修复：TraceThreadInfo 构造缺少 pp_rank (testing): 作者修改测试用例中的构造参数。

风险与影响

- 风险：风险较低。改动集中在可观测性模块，且所有追踪调用均受 trace_only=True 和全局开关控制，仅在启用 OpenTelemetry 时才会执行额外函数调用。在未启用追踪的正常推理路径中，仅增加了一次函数调用（set_time_batch 内部立即返回），性能影响可忽略。set_time_batch 的 attrs 参数通过条件分支兼容旧方法，不会引发 TypeError。
- 影响：影响范围：仅影响启用了 OpenTelemetry 追踪且使用 PP（Pipeline Parallelism）的部署场景。 影响程度：用户无需更改配置或代码即可获得 PP 追踪数据（若已安装 OpenTelemetry）。新增的 span 数据可帮助分析 PP 各 micro-batch 的 CPU 执行耗时。团队影响：为后续其他并行模式（如 TP、DP）的追踪提供了可复用的插桩模式。
- 风险标记：核心调度路径插桩，微小开销仅追踪启用时，无兼容性问题

关联脉络

- 暂无明显关联 PR