PR #34335 完整报告

sgl-project/sglang

metrics: don't clock-rebase unset time sentinels in ReqTimeStats deserialization

合并时间：2026-08-12 14:27

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34335

执行摘要

- 一句话：修复 ReqTimeStats 0.0 哨兵误 rebase
- 推荐动作：值得精读：PR 很小但问题层次深，一行条件判断暴露了“哨兵值穿越 IPC 被时钟 rebase 污染”的观测链路反模式。重点关注：① __setstate__ 中 falsy 跳过策略与 convert_time_cross_thread 的配合；② 两 hop 测试如何用 mock 锚点精确复现跨进程时钟差，可作为 metrics 类 bug 回归测试的模板。

功能与动机

PR body 指出：ReqTimeStatsBase.__setstate__ 将每个 *time 字段 rebase 到接收进程时钟锚点，包括从未打戳的 0.0 哨兵，使其变成 0.0 + sender_diff − receiver_diff 的微小非零值，破坏下游 == 0.0 / > 0.0 的“是否已打戳”判断。实测在 PD 分离 decode 服务器上，prefill_finished_time 从未被本地打戳，epsilon 经 has_timing_data 到达 tokenizer manager 后通过首 token 检查，导致每请求记录一条约 −node-uptime 的 TTFT 观测和约 +node-uptime 的 ITL 观测；线上采集到 134,931 个 ITL 样本中 4,373 个落入 +Inf 桶，first_token_time=1.430511474609375e-06。

实现拆解

1. 修改核心反序列化逻辑：在 python/sglang/srt/observability/req_time_stats.py 中，ReqTimeStatsBase.__setstate__ 的时间字段循环判断由 if key.endswith("time") 改为 if key.endswith("time") and state[key]，跳过 falsy（即 0.0）哨兵，使 0.0 在 pickle 往返后保持精确 0.0；已打戳字段继续通过 convert_time_cross_thread 按接收进程时钟锚点 rebase。
2. 新增回归测试：新建 test/registered/unit/observability/test_req_time_stats.py，TestSetstatePreservesUnsetTimeSentinels.test_two_hop_round_trip 构造同时含已打戳（wait_queue_entry_time = 123.456）与未打戳（prefill_finished_time = 0.0）的 SchedulerReqTimeStats，通过 mock 三次不同的 global_diff_realtime_monotonic 模拟两跳 IPC，断言未打戳字段保持精确 0.0（修复前会变成 -9.0），已打戳字段按锚点差 rebase 为 123.456 - 9.0。
3. CI 配套：测试通过 register_cpu_ci(est_time=5, suite="base-a-test-cpu") 注册进 CPU CI，并触发了 /rerun-test，ubuntu-latest 1 个测试通过。
4. 演进过程：提交经历 main 分支合并与注释迁移（将 inline 注释移到 PR review comments），最终由 ispobock 合入 main。

关键文件：

- python/sglang/srt/observability/req_time_stats.py（模块 观测统计；类别 source；类型 core-logic；符号 setstate）：核心修复点：ReqTimeStatsBase.__setstate__ 时间字段循环增加真值判断，避免 0.0 哨兵被 rebase 成 epsilon。
- test/registered/unit/observability/test_req_time_stats.py（模块 指标测试；类别 test；类型 test-coverage；符号 TestSetstatePreservesUnsetTimeSentinels, test_two_hop_round_trip）：新增回归测试，两 hop pickle 往返验证哨兵保留与已打戳字段 rebase，并覆盖 has_timing_data 路径。

关键符号：ReqTimeStatsBase.setstate, convert_time_cross_thread, TestSetstatePreservesUnsetTimeSentinels.test_two_hop_round_trip

关键源码片段

python/sglang/srt/observability/req_time_stats.py

核心修复点：ReqTimeStatsBase.__setstate__ 时间字段循环增加真值判断，避免 0.0 哨兵被 rebase 成 epsilon。

```
def __setstate__(self, state: object):
    # 从字符串重建 disagg_mode：pickle 跨进程时枚举会被序列化为字符串
    disagg_mode_val = state.get("disagg_mode")
    if isinstance(disagg_mode_val, str):
        state["disagg_mode"] = DisaggregationMode(disagg_mode_val)

    # 从序列化字典重建 trace_ctx，区分异步与同步追踪上下文
    trace_ctx_state = state.get("trace_ctx")
    if isinstance(trace_ctx_state, dict):
        if trace_ctx_state.get("tracing_enable"):
            if trace_ctx_state.get("is_async"):
                trace_ctx = object.__new__(TraceReqContextAsync)
                trace_ctx.__setstate__(trace_ctx_state)
            else:
                trace_ctx = object.__new__(TraceReqContext)
                trace_ctx.__setstate__(trace_ctx_state)
            state["trace_ctx"] = trace_ctx
        else:
            state["trace_ctx"] = TraceNullContext()

    # 关键修复：仅对非 falsy 的时间戳字段执行跨进程时钟 rebase。
    # 0.0 表示“从未打戳”，若也参与 rebase 会变成 sender_diff − receiver_diff 的微小 epsilon，
    # 导致下游 == 0.0 / > 0.0 的“是否已打戳”判断失效，
    # TTFT / inter-token-latency 直方图被约节点运行时长的脏样本污染。
    for key in state.keys():
        if key.endswith("time") and state[key]:
            state[key] = convert_time_cross_thread(
                state[key],
                state["diff_realtime_monotonic"],
                global_diff_realtime_monotonic,
```

```
            )
        self.__dict__.update(state)
```

test/registered/unit/observability/test_req_time_stats.py

新增回归测试，两 hop pickle 往返验证哨兵保留与已打戳字段 rebase，并覆盖 has_timing_data 路径。

```
class TestSetstatePreservesUnsetTimeSentinels(CustomTestCase):
    def test_two_hop_round_trip(self):
        # 构造"已打戳 + 未打戳"混合的 SchedulerReqTimeStats 实例
        src = rts.SchedulerReqTimeStats()
        src.enable_metrics = True
        src.wait_queue_entry_time = 123.456
        src.prefill_finished_time = 0.0 # 未打戳哨兵，必须保持精确 0.0

        # 模拟两次 IPC 跳转，各自持有不同的时钟锚点
        # hop 1：scheduler → detokenizer；hop 2：detokenizer → tokenizer
        # 第二跳依赖 has_timing_data 让字段继续流动
        with mock.patch.object(rts, "global_diff_realtime_monotonic", 1_000_000.0):
            blob = pickle.dumps(src)
        with mock.patch.object(rts, "global_diff_realtime_monotonic", 1_000_005.0):
            hop1 = pickle.loads(blob)
            blob2 = pickle.dumps(hop1)
        with mock.patch.object(rts, "global_diff_realtime_monotonic", 1_000_009.0):
            hop2 = pickle.loads(blob2)

        # 未打戳字段必须原样保留 0.0（修复前会变成 -9.0）
        self.assertEqual(hop2.prefill_finished_time, 0.0)
        # 已打戳字段仍按锚点差值 rebase：123.456 - 9.0
        self.assertAlmostEqual(hop2.wait_queue_entry_time, 123.456 - 9.0)
```

评论区精华

两条 review 评论均由作者 sshleifer 自述关键设计：

- 在 req_time_stats.py 第 366 行说明：0.0 表示"从未打戳"，rebase 会伪造 sender_diff – receiver_diff 的 epsilon，并给出 PD decode server 上 TTFT/ITL 直方图被 ~ 节点运行时长的脏样本污染的完整链条。
- 在测试第 35 行补充 hop 语义：hop 1 是 scheduler → detokenizer，hop 2 是 detokenizer → tokenizer；base __getstate__ 会把 enable_metrics: False 写入序列化状态，因此第二跳专门验证 has_timing_data 让字段继续流动的路径。
- 审核者 sufeng-buaa 批准（APPROVED），无未解决疑虑。
- 0.0 哨兵被 rebase 成 epsilon 的失真链条 (correctness): 通过 `if key.endswith("time") and state[key]` 跳过 falsy 哨兵，已在 PR 中修复。
- 两跳测试的 hop 语义与 has_timing_data 路径 (testing): 测试设计获认可，无修改要求；sufeng-buaa 批准合并。

风险与影响

- 风险：风险点：
  - if state[key] 依赖 Python 真值语义：除 0.0 外，任何 falsy 值（如 None）也会被跳过。当前所有 *time 字段默认 0.0，未见 None 用法，但未来若新增以 None 表示合法状态的字段，会被静默跳过 rebase 导致时间基准错误。
  - __setstate__ 是通用反序列化路径，影响不限于 PD 分离场景，还包括 DP controller、跨进程 JSON 编解码等所有 ReqTimeStats 输送路径；跳过 rebase 后若某进程在反序列化后基于本地时钟补充打戳，时序基准需重新核对。
  - 修复只覆盖时间戳哨兵，diff_realtime_monotonic 本身的跨机器时钟漂移假设未变，极端时钟偏差场景仍可能产生偏差。
  - 新增测试仅覆盖 CPU 单测，未包含真实 PD 多进程端到端验证。
  - 影响：影响范围：仅限 ReqTimeStatsBase 系列统计对象（SchedulerReqTimeStats、APIServerReqTimeStats 等）的 IPC 反序列化行为，不触及业务推理路径。对 PD 分离部署、DP 多进程下依赖“是否已打戳”判断的指标（TTFT、ITL、首 token 记录逻辑）是直接修复：消除每请求一条约节点运行时长的 TTFT/ITL 脏样本，Prometheus 聚合回归真实数值。同进程场景行为不变。对团队而言是一次低风险、可快速回滚的观测链路修正，并补齐了此前缺失的指标序列化回归保护。
  - 风险标记：IPC 反序列化公共路径变更，依赖 Python falsy 语义，缺少 PD 端到端回归验证

关联脉络

- PR #34450 Raise PD zmq per-context socket cap via SGLANG_DISAGGREGATION_ZMQ_MAX_SOCKETS: 同为 PD 分离部署下的基础设施修复，一个解决连接数上限，一个解决跨进程时间统计失真，共同支撑 PD 场景稳定性。