PR #22789 完整报告

sgl-project/sglang

feat: emit per-iteration forward pass metrics via ZMQ PUB

合并时间：2026-05-13 01:28

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22789

执行摘要

- 一句话：新增基于 ZMQ PUB 的调度器前向传递指标实时推送
- 推荐动作：值得精读。关键设计决策：1）使用 IPC 而非 TCP 避免端口冲突和监听安全问题；2）复用 DeviceTimer 而非自行实现 GPU 计时，降低维护成本；3）通过 dp_rank 后缀解决多副本 IPC 冲突；4）waiting_queue 中所有请求视为 prefill 确保下游正确进行 TTFT 估计。建议同步关注下游 Dynamo 对 FPM 的消费逻辑。

功能与动机

下游应用需要每轮迭代的调度遥测来基于负载做伸缩决策（TTFT/ITL 在线预测）。当前 Prometheus 拉取模式无法满足毫秒级实时性。PR body 明确说明 'Downstream applications needs per-iteration scheduling telemetry to make load-based scaling decisions (TTFT/ITL prediction via online regression).'

实现拆解

实现过程分为以下几个步骤：

1. 定义 FPM 数据模型：在 forward_pass_metrics.py 中定义了 WelfordAccumulator（用于在线计算方差）、ScheduledRequestMetrics、QueuedRequestMetrics 和 ForwardPassMetrics，均使用 msgspec.Struct 实现零拷贝序列化。全局编解码器 encode/decode 用于与下游 Dynamo 兼容。
2. 实现 ZMQ 发布线程：_FpmPublisherThread 在后台通过 ZMQ PUB socket 发送序列化后的 ForwardPassMetrics 字节流。支持心跳机制（空闲时每秒发送空指标）。队列大小为 10,000，防止下游消费不及时阻塞调度器。
3. 集成调度指标收集：在 SchedulerMetricsMixin 中新增类属性 enable_fpm 和方法 _init_fpm、_emit_forward_pass_metrics 等。_init_fpm 在 tp_rank=0 且 pp_rank 为最后一个时启动 publisher，并根据 dp_rank 区分 IPC 端点避免冲突。
4. 埋入调度器事件循环：在 scheduler.py 的 get_next_batch_to_run 中记录批次开始时间（fpm_start_time），在 process_batch_result 末尾调用 _emit_forward_pass_metrics 计算指标并投递给 publisher 线程。
5. 复用 DeviceTimer 提升精度：利用已有 DeviceTimer（PR #24197 引入）替代手动 CUDA Event，FPM 注册为第二个 reporter 实现 GPU 精确的 wall_time。当环境变量 SGLANG_ENABLE_METRICS_DEVICE_TIMER 未定义时，FPM 自动创建独立的

DeviceTimer。

6. 支持 PD 分离部署：在 _build_queued_request_metrics 中根据 disaggregation_mode 读取正确队列（prefill 引擎读 bootstrap_queue，decode 引擎读 waiting_queue），并在 _build_scheduled_request_metrics 中通过 req.prefix_indices 准确计算前缀命中 token。
7. 添加 CLI 配置：在 server_args.py 新增 enable_forward_pass_metrics、forward_pass_metrics_worker_id、forward_pass_metrics_ipc_name 三个参数，其中 IPC 名称由框架自动生成并注入。
8. 编写测试套件：注册 CPU CI 的 test/registered/unit/observability/test_forward_pass_metrics.py 测试了 mixed batch 分离、disagg 分支、GPU timer 等核心逻辑；手动测试 test/manual/test_forward_pass_metrics.py 验证了 schema roundtrip、ZMQ PUB/SUB 端到端通信和心跳机制。

关键文件：

- python/sglang/srt/observability/forward_pass_metrics.py（模块 遥测模块；类别 source；类型 core-logic；符号 WelfordAccumulator, ScheduledRequestMetrics, QueuedRequestMetrics, ForwardPassMetrics）：新增 FPM 核心模块，定义数据模型、Welford 累加器、ZMQ 发布线程和编解码器，是 PR 的核心内容。
- python/sglang/srt/observability/scheduler_metrics_mixin.py（模块 调度器集成；类别 source；类型 core-logic；符号 _init_fpm, _build_scheduled_request_metrics, _build_queued_request_metrics, _emit_forward_pass_metrics）：集成 FPM 到调度器指标收集体系，新增 _init_fpm、_emit_forward_pass_metrics 等方法，是 PR 的集成枢纽。
- test/registered/unit/observability/test_forward_pass_metrics.py（模块 单元测试；类别 test；类型 test-coverage；符号 _FakeReq, _FakeForwardMode, _CollectingPublisher, TestForwardPassMetrics）：注册到 CI 的单元测试，覆盖 mixed batch、disagg 分支、GPU timer 等关键逻辑，确保质量。
- test/manual/test_forward_pass_metrics.py（模块 手动测试；类别 test；类型 test-coverage；符号 test_schema_roundtrip, test_zmq_pub_sub, test_heartbeat）：手动端到端测试，验证 schema roundtrip、ZMQ PUB/SUB 通信和心跳，确保与外部兼容。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 get_next_batch_to_run, process_batch_result, run_scheduler_process）：调度器主循环中埋入 FPM 计时和发射点，是功能生效的入口。
- python/sglang/srt/server_args.py（模块 服务配置；类别 source；类型 configuration）：新增三个 FPM 相关配置参数，控制开关和 IPC 命名。
- python/sglang/srt/utils/device_timer.py（模块 设备计时；类别 source；类型 core-logic；符号 add_reporter）：FPM 通过 add_reporter 方法复用 DeviceTimer 实现 GPU 计时，减少重复代码。
- python/sglang/srt/managers/schedule_batch.py（模块 调度批次；类别 source；类型 core-logic）：在 ScheduleBatch 中新增 fpm_start_time 字段，存储迭代起始时间戳。
- python/sglang/srt/managers/utils.py（模块 工具；类别 source；类型 refactor）：配合 schedule_batch 调整导入路径，修复循环引用。

关键符号：WelfordAccumulator.add, WelfordAccumulator.variance, ScheduledRequestMetrics, QueuedRequestMetrics, ForwardPassMetrics, encode, decode, _FpmPublisherThread.init, _FpmPublisherThread.publish, _FpmPublisherThread.shutdown, SchedulerMetricsMixin._init_fpm, SchedulerMetricsMixin._build_scheduled_request_metrics, SchedulerMetricsMixin._build_queued_request_metrics, SchedulerMetricsMixin._emit_forward_pass_metrics, SchedulerMetricsMixin._shutdown_fpm, DeviceTimer.add_reporter, Scheduler.get_next_batch_to_run, Scheduler.process_batch_result, Scheduler.run_scheduler_process

关键源码片段

python/sglang/srt/observability/forward_pass_metrics.py

新增 FPM 核心模块，定义数据模型、Welford 累加器、ZMQ 发布线程和编解码器，是 PR 的核心内容。

```
# Welford 在线方差累加器
class WelfordAccumulator:
    __slots__ = ('count', 'total', '_mean', '_m2')
    def __init__(self):
        self.count = 0
        self.total = 0
        self._mean = 0.0
        self._m2 = 0.0

    def add(self, v: int) -> None:
        self.count += 1
        self.total += v
        delta = v - self._mean
        self._mean += delta / self.count
        delta2 = v - self._mean
        self._m2 += delta * delta2

    def variance(self) -> float:
        if self.count == 0:
            return 0.0
        return self._m2 / self.count

# msgspec Struct，位置编码以便与下游 Dynamo 兼容
class ScheduledRequestMetrics(msgspec.Struct, frozen=True, gc=False):
    num_prefill_requests: int = 0
    sum_prefill_tokens: int = 0
    var_prefill_length: float = 0.0
    sum_prefill_kv_tokens: int = 0
    num_decode_requests: int = 0
    sum_decode_kv_tokens: int = 0
    var_decode_kv_tokens: float = 0.0
```

```
class QueuedRequestMetrics(msgspec.Struct, frozen=True, gc=False):
    num_prefill_requests: int = 0
    sum_prefill_tokens: int = 0
    var_prefill_length: float = 0.0
    num_decode_requests: int = 0
    sum_decode_kv_tokens: int = 0
    var_decode_kv_tokens: float = 0.0

class ForwardPassMetrics(msgspec.Struct, frozen=True, gc=False):
    version: int = 1
    worker_id: str = ''
    dp_rank: int = 0
    counter_id: int = 0
    wall_time: float = 0.0
    scheduled_requests: ScheduledRequestMetrics = ScheduledRequestMetrics()
    queued_requests: QueuedRequestMetrics = QueuedRequestMetrics()

# 模块级编码器，线程安全
_encoder = msgspec.msgpack.Encoder()
_decoder = msgspec.msgpack.Decoder(ForwardPassMetrics)
def encode(metrics: ForwardPassMetrics) -> bytes:
    return _encoder.encode(metrics)
def decode(data: bytes) -> ForwardPassMetrics:
    return _decoder.decode(data)
```

评论区精华

Review 中主要讨论了：wall_time 在事件循环重叠模式下的测量精度问题（tedzhouhk 指出 overlap 模式下 wall_time 会包含下一轮调度的开销）；variance 应使用完整 prompt 长度还是 chunk 长度的争论（最终保留完整长度，因 attention 时间取决于全上下文）；disaggregation 模式下 queued_requests 始终为零的 bug（需根据模式读取不同队列）；IPC 端点冲突导致 dp_size>1 时所有进程绑定同一 IPC 路径（通过追加 dp_rank 后缀修复）；是否重用 DeviceTimer 代替手动 CUDA 事件（最终实现 FPM 作为 DeviceTimer 的第二 reporter）；代码组织建议抽离 _init_fpm 方法等。

- wall_time 在 overlap 模式下的测量精度 (design): 采纳文档化说明的方式，保留现有测量点，因 ~1ms 误差对下游可接受。
- variance 使用完整 prompt 长度 vs chunk 长度 (correctness): 保留 len(req.origin_input_ids)，与 sum_prefill_tokens 语义区分。
- disaggregation 模式下 queued_requests 始终为零 (correctness): 已修复：PREFILL 模式读 bootstrap_queue，DECODE 读 waiting_queue。
- IPC 传输 vs TCP (design): 改为 IPC 传输，通过 server_args.forward_pass_metrics_ipc_name 暴露路径。
- 复用 DeviceTimer 替代手写 CUDA Event (design): FPM 不再自建 CUDA Event，通过 DeviceTimer.add_reporter 集成 GPU 计时。
- IPC 端点 dp_rank 冲突 (design): 已修复，endpoint 格式为 '{base}.{dp_rank}'。

风险与影响

- 风险：
  1. 墙时间测量精度：在 *event_loop_overlap* 模式下，wall_time 可能包含 ~1ms 的下一轮调度开销，可能引入噪声，需下游消费者理解此语义。
  2. schema 兼容性：ForwardPassMetrics 字段顺序必须与下游 Dynamo 严格一致，msgspec 使用位置编码，任何顺序错位都会静默数据损坏。已通过版本号和跨仓库协作文档约束，但部署时需确保两端同步。
  3. disagg 队列指标：已修复但生产环境中不同 disagg 实现的队列行为可能仍有差异，需持续验证。
  4. IPC 端点清理：finally 块确保 publisher 关闭，但若 scheduler 初始化中途异常退出，ZMQ 上下文可能未正确终结，遗留 IPC 文件。
  5. 默认关闭不会造成性能影响，但开启后 ZMQ 后台线程占用少量 CPU 和内存。 - 影响：用户：默认无影响；开启后可在外部消费毫秒级调度指标，用于自动伸缩、TTFT/ITL 预测等。系统：每个调度器进程增加一个 ZMQ PUB 线程（空闲时 CPU 占用 ~0），队列最大 10k 条消息。团队：新增 forward_pass_metrics.py 模块需与下游 Dynamo 团队共同维护 schema；测试覆盖了 CI 注册测试和手动 PUB/SUB 测试。 - 风险标记：核心路径变更，跨团队接口依赖，默认关闭回归风险，DP 多副本兼容性，IPC 清理安全

关联脉络

- PR #24197 Refactor device timer into model runner: 引入 DeviceTimer 基础设施，本 PR 利用其 *add_reporter* 实现 GPU 精确计时，体现了模块复用模式。
- PR #24932 [PD] Refactor hybrid state transfer: 重构 PD 状态传输，本 PR 的 disagg 队列读取依赖于同一 *disaggregation_mode* 枚举，两个 PR 共同演进 disagg 可观测性。