

PR #37636 完整报告

sgl-project/sglang

[Metrics] Export scheduler stage wall time

合并时间: 2026-09-05 01:45

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/37636>

执行摘要

- 一句话: 新增调度器阶段墙壁时间互斥统计, 导出为 Prometheus 指标。
- 推荐动作: 该 PR 值得精读。它展示了一个清晰、低侵入性的可观测性增强模式, 通过装饰器和上下文管理器将新的度量收集逻辑无缝集成到复杂的核心路径中, 同时保持了与现有分析器和 NVTX 插桩的兼容性。对于负责系统性能分析、监控或调度器开发的工程师, 理解此 PR 的设计 (特别是 SchedulerStageMetricsRecorder 的互斥状态管理和 scheduler_stage_method 装饰器) 非常有价值。

功能与动机

动机源于对调度器性能瓶颈的可观测性需求。正如 PR body 所述: ‘Scheduler process CPU time and GPU forward occupancy show whether an engine is busy, but not which scheduler phase owns gaps between forwards.’ 因此, 需要导出按阶段划分的互斥墙壁时间, 以帮助工程师定位性能问题。

实现拆解

1. 核心记录器与装饰器实现: 新增 python/sglang/srt/observability/scheduler_stage_metrics.py。其中定义了 SchedulerStageMetricsRecorder 数据类, 负责累加互斥的阶段时间; 以及 scheduler_stage_method 装饰器, 用于包装调度器方法。两者通过 enter/exit 或上下文管理器 record 来确保阶段切换的原子性和互斥性。时间采样基于 time.monotonic_ns。
2. 指标收集器集成: 修改 python/sglang/srt/observability/metrics_collector.py, 在 SchedulerMetricsCollector 中新增 scheduler_stage_seconds_total Prometheus 计数器 and 对应的 increment_scheduler_stage_seconds 方法。计数器按 category 标签 (即阶段名) 区分, 并在初始化时预设所有阶段标签以避免计数器缺失。
3. 记录器实例化与注入: 修改 python/sglang/srt/managers/scheduler_components/metrics_reporter.py, 在 MetricsReporter 初始化时创建 SchedulerStageMetricsRecorder 实例。该记录器在 start_scheduler_time_accounting 和 _record_scheduler_time 中被启动和清空 (drain), 并将清空的数据推送给指标收集器。
4. 调度器核心路径插桩: 修改 python/sglang/srt/managers/scheduler.py, 将 @scheduler_nvtx_method 装饰器替换为 @scheduler_stage_method, 应用于 process_input_requests、get_next_batch_to_run、run_batch、process_batch_result、on_idle 等关键方法。同时, 将 scheduler_stage_metrics 实例从 MetricsReporter 传递给 Scheduler 并注入到 SchedulerRequestReceiver、PrefillBootstrapQueue、

SchedulerInvariantChecker 等组件。

5. 分布式与异构路径覆盖：修改 `python/sglang/srt/disaggregation/prefill.py` 和 `decode.py`，在预填充 / 解码分离（disaggregation）相关方法（如 `pop_bootstrapped`、`process_decode_queue`、`get_next_disagg_decode_batch_to_run`）上同样应用 `@scheduler_stage_method` 装饰器，并将 `scheduler_stage_metrics` 作为依赖传递。这确保了 `normal`、`pipeline-parallel` 和 `MLX` 调用路径都被覆盖。
6. 全面测试覆盖：新增 `test/registered/unit/observability/test_scheduler_stage_metrics.py`，单元测试验证互斥性、异常恢复、嵌套、装饰器行为等。同时修改了 `test/registered/unit/observability/test_forward_pass_metrics.py` 等多个测试文件，以适配新的指标收集接口。

关键文件：

- `python/sglang/srt/observability/scheduler_stage_metrics.py`（模块 可观测性；类别 `source`；类型 `core-logic`；符号 `SchedulerStageMetricsRecorder`, `start`, `enter`, `exit`）：核心变更文件。新增了负责调度器阶段墙壁时间累加与互斥管理的 `SchedulerStageMetricsRecorder` 类和 `scheduler_stage_method` 装饰器。
- `python/sglang/srt/observability/metrics_collector.py`（模块 可观测性；类别 `source`；类型 `core-logic`；符号 `increment_scheduler_stage_seconds`）：在 Prometheus 指标收集器中新增了 `scheduler_stage_seconds_total` 计数器和递增方法，是导出数据的出口。
- `python/sglang/srt/managers/scheduler_components/metrics_reporter.py`（模块 调度器组件；类别 `source`；类型 `dependency-wiring`）：负责创建并管理 `SchedulerStageMetricsRecorder` 实例，将其生命周期与调度器的空闲 / 活跃周期对齐，并定期将累计数据清空并推送到指标收集器。
- `python/sglang/srt/managers/scheduler.py`（模块 调度器；类别 `source`；类型 `dependency-wiring`）：调度器主循环所在，是应用阶段装饰器的核心位置，将多个关键方法标记为特定阶段。
- `test/registered/unit/observability/test_scheduler_stage_metrics.py`（模块 可观测性测试；类别 `test`；类型 `test-coverage`；符号 `TestSchedulerStageMetricsRecorder`, `test_category_names`, `test_nested_stages_are_exclusive`, `test_decorator_restores_stage_after_exception`）：针对核心记录器和装饰器的单元测试，验证了互斥性、异常恢复等关键行为。

关键符号： `SchedulerStageMetricsRecorder.start`,
`SchedulerStageMetricsRecorder.enter`, `SchedulerStageMetricsRecorder.exit`,
`SchedulerStageMetricsRecorder.record`, `SchedulerStageMetricsRecorder.drain`,
`SchedulerStageMetricsRecorder._sample`, `scheduler_stage_method`,
`SchedulerMetricsCollector.increment_scheduler_stage_seconds`

关键源码片段

`python/sglang/srt/observability/scheduler_stage_metrics.py`

核心变更文件。新增了负责调度器阶段墙壁时间累加与互斥管理的

`SchedulerStageMetricsRecorder` 类和 `scheduler_stage_method` 装饰器。

```
# python/sglang/srt/observability/scheduler_stage_metrics.py
# 本文件实现了调度器阶段墙壁时间的互斥统计。
```

```
def scheduler_stage_method(stage: str) -> Callable[[_F], _F]:
    """
    装饰器：将调度器方法标记为属于特定阶段，并自动管理时间统计和 NVTX 追踪。
    该装饰器通过访问实例的 `scheduler_stage_metrics` 属性来获取记录器。
    """

    if stage not in _SCHEDULER_STAGE_CATEGORY_SET:
        raise ValueError(f"Unknown scheduler stage: {stage}")
    trace_name = f"scheduler.{stage}"

    def decorator(func: _F) -> _F:
        # 包装原始函数以添加 NVTX 追踪能力
        profiled_func = scheduler_nvtx_method(trace_name)(func)

        @wraps(func)
        def wrapper(self, *args, **kwargs):
            # 从调度器实例获取记录器，若为 None（未启用）则直接执行
            recorder = self.scheduler_stage_metrics
            if recorder is None:
                return profiled_func(self, *args, **kwargs)

            # 进入新阶段，记录之前的阶段以便退出时恢复
            previous_stage = recorder.enter(stage)
            # 管理 torch.profiler 的追踪阶段名，避免嵌套时重复或冲突
            previous_trace_stage = recorder._trace_stage
            trace_stage_changed = previous_trace_stage != stage
            if trace_stage_changed:
                recorder._trace_stage = stage
            try:
                return profiled_func(self, *args, **kwargs)
            finally:
                # 无论是否异常，都恢复之前的追踪阶段和调度器阶段
                if trace_stage_changed:
                    recorder._trace_stage = previous_trace_stage
                recorder.exit(previous_stage)

        return cast(_F, wrapper)

    return decorator
```

python/sglang/srt/observability/metrics_collector.py

在 Prometheus 指标收集器中新增了 `scheduler_stage_seconds_total` 计数器和递增方法，是导出数据的出口。

```
# python/sglang/srt/observability/metrics_collector.py ( 相关片段 )
# 在 SchedulerMetricsCollector.__init__ 中新增计数器定义：
self.scheduler_stage_seconds_total = Counter(
```

```

    name="sglang:scheduler_stage_seconds_total",
    documentation=(
        "Total scheduler-loop wall time exclusively attributed to each stage."
    ),
    labelnames=list(labels.keys()) + ["category"],
)
# 预初始化所有阶段标签，确保指标存在：
for category in SCHEDULER_STAGE_CATEGORIES:
    self.scheduler_stage_seconds_total.labels(**labels, category=category)

# 新增递增方法：
def increment_scheduler_stage_seconds(self, stage: str, seconds: float) -> None:
    """为指定的调度器阶段增加累计的墙壁时间（秒）。"""
    self.scheduler_stage_seconds_total.labels(**self.labels, category=stage).inc(
        seconds
    )

```

评论区精华

Review 讨论极少，仅有一条 **APPROVED** 评论。这表明实现方案在技术层面未引发争议，可能因为这是一个纯粹的可观测性增强，设计模式清晰，且对现有逻辑的侵入性通过装饰器和上下文管理器得到了良好控制。提交历史中的第二次提交仅是为了统一代码风格（从 black 到 ruff-format），不涉及功能变更。

- 暂无高价值评论线程

风险与影响

- 风险：性能风险：在调度器热路径的每个阶段切换和采样点引入了 `time.monotonic_ns()` 调用。虽然开销通常很小，但在高吞吐、低延迟场景下可能需要关注。不过，实现中通过 `enabled` 标志和条件判断（如 `if not self.enabled: return`）将开销最小化。正确性风险：阶段互斥性依赖于正确的 `enter/exit` 配对和状态管理。如果调用顺序错误或异常未正确处理，可能导致时间统计不准确。但 `record` 上下文管理器和 `scheduler_stage_method` 装饰器中的 `try...finally` 块旨在确保健壮性。测试覆盖了主要场景。兼容性风险：修改了多个组件的接口（如 `PrefillBootstrapQueue`、`SchedulerRequestReceiver` 新增了 `scheduler_stage_metrics` 参数）。需要确保所有实例化点都正确传递了该参数。从代码变更看，修改是全面的，但依赖于调用方正确更新。
- 影响：用户影响：直接提升了 SGLang 引擎的可观测性。运维和开发者现在可以通过 `sglang:scheduler_stage_seconds_total` 指标（按 `category` 标签区分）精确了解调度器循环中各阶段（如请求接收、批处理选择、执行等）的时间消耗占比，从而更有效地诊断性能瓶颈和优化调度策略。系统影响：此变更仅影响可观测性层，不改变模型计算或核心调度逻辑。新增的指标收集逻辑在未启用指标时开销为零（`enabled=False`），在启用时开销极低。对现有基于 CPU 时间和 GPU 占用率的监控是补充而非替代。团队影响：为性能工程团队提供了一个强大的新工具。未来可以基于此指标构建更精细的仪表盘、告警规则或自动优化逻辑。该实现模式（记录器 + 装饰器 / 上下文管理器 + 指标收集器）为后续添加更多调度器级度量树立了良好范例。

- 风险标记：热路径性能开销，新指标正确性验证，组件接口变更

关联脉络

- PR #38067 [Profiler] Add SGLANG_PROFILE_BY_STAGE_DECODE_MIN_BS to defer the decode-stage capture: 同属可观测性 / 性能分析领域，涉及调度器阶段（decode-stage）的性能采集优化。
- PR #37843 [Router] Add load-aware prefill admission and bounded policy proposals: 涉及调度策略（scheduling），与本 PR 新增的调度器阶段可观测性有潜在关联，可用于评估调度策略效果。
- PR #34565 [Unified Tree] Support Branching-Point Caching for the SWA Component: 涉及统一缓存（unified-radix-cache），本 PR 中的 SCHEDULER_STAGE_SANITY_CHECK_CACHE 阶段包含缓存检查，可能使用类似机制。