

PR #37461 完整报告

sgl-project/sglang

[Metrics] Add rolling scheduler utilization counters

合并时间: 2026-09-04 23:06

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/37461>

执行摘要

- 一句话: 新增调度器利用率计数器, 区分空闲挂钟时间与进程 CPU 时间。
- 推荐动作: 这是一个有价值且实现良好的可观测性增强 PR。它填补了现有指标在调度器利用率监控方面的空白。实现考虑了所有调度器变体和状态, 测试覆盖充分。建议关注以下设计决策:
 1. 采用 `dataclass(slots=True)` 和 1 秒聚合窗口来高效跟踪时间状态。
 2. 将记录点嵌入事件循环的关键转换点, 确保状态跟踪的完整性。
 3. 处理引擎暂停状态时, 通过 `is_fully_idle()` 动态判断当前状态。该 PR 可精读, 特别是 `_SchedulerTimeAccountingSnapshot` 的实现和各事件循环中的集成方式, 对设计类似的可观测性功能有参考价值。

功能与动机

现有的前向执行指标 (如 GPU 计算时间) 无法直接反映调度器的利用率, 特别是当调度器处于空闲状态时。此变更旨在暴露累积信号, 以区分调度器空闲挂墙时间 (idle wall time) 与调度器进程 CPU 时间, 从而帮助用户和运维更准确地理解系统负载和性能瓶颈。PR body 中明确指出动机为“Expose cumulative signals that distinguish scheduler idle wall time from scheduler process CPU time”。

实现拆解

1. 引入时间核算快照类: 在 `python/sglang/srt/managers/scheduler_components/metrics_reporter.py` 中新增 `_SchedulerTimeAccountingSnapshot` 数据类, 用于跟踪时间核算周期。它维护起始时间、上次采样时间、累积空闲时间, 并提供 `init`、`sample`、`should_record`、`reset` 方法, 以 1 秒为周期进行记录。
2. 在调度器中集成状态记录: 在 `python/sglang/srt/managers/scheduler.py` 的多个事件循环方法 (`event_loop_normal`、`event_loop_overlap`、`run_batch`、`on_idle`) 中插入状态跟踪调用。关键是在每次接收新请求时调用 `record_scheduler_active()`, 在无工作可执行或引擎暂停时调用 `record_scheduler_idle()`。还新增了 `_record_scheduler_state_for_paused_engine` 方法以处理暂停状态。
3. 扩展指标收集器: 在 `python/sglang/srt/observability/metrics_collector.py` 中新增两个 Prometheus Counter: `sglang:scheduler_idle_seconds_total` 和 `sglang:scheduler_process_cpu_seconds_total`, 并提供

increment_scheduler_idle_seconds 和 increment_scheduler_process_cpu_seconds 方法。

4. 适配其他调度器循环：在 python/sglang/srt/hardware_backend/mlx/scheduler_mixin.py、python/sglang/srt/disaggregation/decode.py、python/sglang/srt/disaggregation/prefill.py 的相应事件循环中也插入了 _record_scheduler_state_for_paused_engine 调用，确保所有执行路径的状态都能被正确记录。
5. 增强测试覆盖：在 test/registered/unit/observability/test_forward_pass_metrics.py 中新增 TestSchedulerTimeAccounting 测试类，验证时间核算逻辑（如空闲时间累计、周期更新、CPU 时间采样）。在 test/registered/unit/managers/test_scheduler_pause_generation.py 中新增测试验证暂停引擎状态记录。同时更新了 test/registered/observability/test_metrics.py 以验证新指标端点，并在多个测试文件中初始化 metrics_reporter 以修复测试。

关键文件：

- python/sglang/srt/managers/scheduler_components/metrics_reporter.py（模块 指标报告器；类别 source；类型 core-logic；符号 _SchedulerTimeAccountingSnapshot, init, sample, should_record）：核心变更文件，新增了时间核算快照类和状态记录方法。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 _record_scheduler_state_for_paused_engine）：在所有调度器事件循环中集成状态记录调用，是实现的关键接入点。
- python/sglang/srt/observability/metrics_collector.py（模块 指标收集器；类别 source；类型 core-logic；符号 increment_scheduler_idle_seconds, increment_scheduler_process_cpu_seconds）：定义新的 Prometheus 计数器并添加对应的增量方法。
- test/registered/unit/observability/test_forward_pass_metrics.py（模块 指标单元测试；类别 test；类型 test-coverage；符号 TestSchedulerTimeAccounting, setUp, test_counts_idle_wall_time_and_process_cpu_time, test_state_transitions_accumulate_until_periodic_update）：为新增的时间核算功能添加了专门的单元测试，验证核心逻辑。

关键符号：_SchedulerTimeAccountingSnapshot.init,
_SchedulerTimeAccountingSnapshot.sample, _SchedulerTimeAccountingSnapshot.should_record, _SchedulerTimeAccountingSnapshot.reset,
SchedulerMetricsReporter.start_scheduler_time_accounting,
SchedulerMetricsReporter.record_scheduler_active,
SchedulerMetricsReporter.record_scheduler_idle,
SchedulerMetricsReporter._record_scheduler_time,
Scheduler._record_scheduler_state_for_paused_engine,
SchedulerMetricsCollector.increment_scheduler_idle_seconds,
SchedulerMetricsCollector.increment_scheduler_process_cpu_seconds

关键源码片段

python/sglang/srt/managers/scheduler_components/metrics_reporter.py

核心变更文件，新增了时间核算快照类和状态记录方法。

新增的时间核算快照类，用于跟踪调度器空闲 / 活跃状态的累积时间

@dataclass(slots=True)

class _SchedulerTimeAccountingSnapshot:

start_ns: int

last_sample_ns: int

start_process_cpu_ns: int

accumulate_idle_ns: int

Whether the last sample was idle (for example, no work on the engine).

is_idle: bool

@classmethod

def init(

cls, now_ns: int, now_process_cpu_ns: int, is_idle: bool

) -> _SchedulerTimeAccountingSnapshot:

return cls(

start_ns=now_ns,

last_sample_ns=now_ns,

start_process_cpu_ns=now_process_cpu_ns,

accumulate_idle_ns=0,

is_idle=is_idle,

)

def sample(self, now_ns: int, is_idle: bool) -> None:

如果之前处于空闲状态，将自上次采样以来的空闲时间累加

if self.is_idle:

self.accumulate_idle_ns += now_ns - self.last_sample_ns

self.last_sample_ns = now_ns

self.is_idle = is_idle

def should_record(self, now_ns: int) -> bool:

判断是否达到记录周期 (1 秒)

return now_ns - self.start_ns >= _SCHEDULER_TIME_ACCOUNTING_INTERVAL_NS

def reset(self, now_ns: int, now_process_cpu_ns: int, is_idle: bool) -> None:

重置核算周期

self.start_ns = now_ns

self.last_sample_ns = now_ns

self.start_process_cpu_ns = now_process_cpu_ns

self.accumulate_idle_ns = 0

self.is_idle = is_idle

在 SchedulerMetricsReporter 中添加的核算方法

def start_scheduler_time_accounting(self) -> None:

if not self.enable_metrics:

return

self._scheduler_time_accounting = _SchedulerTimeAccountingSnapshot.init(

```

        time.monotonic_ns(), time.process_time_ns(), True
    )

def record_scheduler_active(self) -> None:
    self._record_scheduler_time(is_idle=False)

def record_scheduler_idle(self) -> None:
    self._record_scheduler_time(is_idle=True)

def _record_scheduler_time(self, is_idle: bool) -> None:
    if not self.enable_metrics:
        return
    now_wall_ns = time.monotonic_ns()
    accounting = self._scheduler_time_accounting
    if accounting is None:
        # 首次调用时初始化快照
        self._scheduler_time_accounting = _SchedulerTimeAccountingSnapshot.init(
            now_wall_ns, time.process_time_ns(), is_idle
        )
    return
    accounting.sample(now_wall_ns, is_idle)
    if not accounting.should_record(now_wall_ns):
        return
    # 达到记录周期，将累积的时间增量发布到 Prometheus 计数器
    now_process_cpu_ns = time.process_time_ns()
    elapsed_process_cpu_ns = now_process_cpu_ns - accounting.start_process_cpu_ns
    if accounting.accumulate_idle_ns > 0:
        self.metrics_collector.increment_scheduler_idle_seconds(
            accounting.accumulate_idle_ns / 1e9
        )
    self.metrics_collector.increment_scheduler_process_cpu_seconds(
        elapsed_process_cpu_ns / 1e9
    )
    accounting.reset(now_wall_ns, now_process_cpu_ns, is_idle)

```

python/sglang/srt/observability/metrics_collector.py

定义新的 Prometheus 计数器并添加对应的增量方法。

```

# 在 SchedulerMetricsCollector.__init__ 中新增的两个计数器
self.scheduler_idle_seconds_total = Counter(
    name="sglang:scheduler_idle_seconds_total",
    documentation=(
        "Total wall time while the scheduler has no runnable or queued work."
    ),
    labelnames=labels.keys(),
)

self.scheduler_process_cpu_seconds_total = Counter(
    name="sglang:scheduler_process_cpu_seconds_total",
    documentation=(

```

```

        "Total CPU time consumed by the scheduler process across all threads."
    ),
    labelnames=labels.keys(),
)
# 预先初始化标签以避免后续 inc 时的 KeyError
self.scheduler_idle_seconds_total.labels(**labels)
self.scheduler_process_cpu_seconds_total.labels(**labels)

# 新增的增量方法
def increment_scheduler_idle_seconds(self, t: float) -> None:
    self.scheduler_idle_seconds_total.labels(**self.labels).inc(t)

def increment_scheduler_process_cpu_seconds(self, t: float) -> None:
    self.scheduler_process_cpu_seconds_total.labels(**self.labels).inc(t)

```

评论区精华

此 PR 主要为功能添加，代码审查讨论较少。唯一的审核者 [merrymercy](#) 给予了批准。讨论内容主要集中在 CI 测试的多次重跑上，而非技术设计争议。

- 调度器利用率指标的设计与实现 (design): 设计方案得到实施并合并。

风险与影响

- 风险:

1. 性能影响: 时间核算逻辑在调度器循环中频繁调用 `time.monotonic_ns()` 和 `time.process_time_ns()`，但采样点以 1 秒为周期进行聚合，开销可控。在空闲时和状态转换时记录，避免了高频调用。引入的 `dataclass(slots=True)` 和轻量级计算对性能影响极小。
2. 正确性: 状态跟踪逻辑需确保在所有调度器状态转换点（接收请求、启动批次、进入空闲、引擎暂停）都被正确调用。PR 通过覆盖所有事件循环（正常、重叠、解聚、MLX）来保证这一点。测试用例模拟了不同的时间序列和状态转换，验证了累计逻辑的正确性。
3. 兼容性: 新增的指标为增量式计数器，对现有指标无破坏性变更。仅当启用 `metrics` 收集时才生效，不会影响未启用 `metrics` 的部署。
4. 回归风险: 变更主要集中在可观测性代码，不改变模型计算或核心调度逻辑。但需确保在所有代码路径中正确调用记录方法，避免状态跟踪遗漏或错误。

- 影响:

1. 用户影响: 用户将获得两个新的 Prometheus 指标 (`scheduler_idle_seconds_total` 和 `scheduler_process_cpu_seconds_total`)，可用于构建调度器利用率面板（例如：`idle_time / (idle_time + process_cpu_time)`）。这有助于诊断调度器瓶颈，特别是在高负载或低利用率场景下。
2. 系统影响: 系统增加了少量状态跟踪代码，但整体开销极低。指标仅通过 Prometheus 端点暴露，不增加额外存储或通信负担。
3. 团队影响: 为运维和开发团队提供了更细粒度的系统监控能力，有助于性能调优和故障排查。为后续基于利用率的自动扩展或资源调度奠定了基础。

- 风险标记: 核心路径变更

, 监控指标新增

关联脉络

- PR #37898 sm120 32GB mem-tier: raise decode cuda-graph max_bs 24->48 + chunked_prefill 2k->4k: 同属性能与调度优化范畴, 虽然具体目标不同 (调整 GPU 资源配置), 但与本次新增调度器利用率指标有潜在关联, 指标可用于评估此类调整的效果。
- PR #37546 [sp] Make attention-TP sequence sharding a per-forward batch property: 涉及调度器和前向批处理的属性变更, 与本次在调度器循环中集成状态记录有相似的代码路径修改。
- PR #37836 Fix mamba radix cache ssm state indexing: 作为 bugfix 修复了调度和缓存相关问题, 而本次 PR 新增的指标可能有助于未来诊断类似问题。