PR #47444 完整报告

vllm-project/vllm

[Rust Frontend] Cache metric handles for scheduler & request stats

合并时间：2026-07-06 21:03

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47444

执行摘要

本 PR 在 Rust frontend 的 metrics 路径中引入 handles 缓存机制，避免每次记录指标时重复 get_or_create。通过预解析 scheduler 和 request-level 的 metrics handles 并缓存在 SchedulerStatsRecorder 和 RequestMetricHandles 中，显著降低开销。改动涉及 9 个文件，净增 343 行删除 237 行，核心逻辑在 rust/src/llm/src/request_metrics.rs 和 rust/src/engine-core-client/src/metrics.rs。

功能与动机

根据 PR 描述："This PR reduces repeated get_or_create lookups on scheduler & request-level metrics paths. Instead of calling get_or_create every time when we want to access the global metrics handle for a request, we pre-resolve per-engine metrics handles once ClientInner and GenerateOutputStream are created, and cache them for future use." 动机是消除每次指标记录时的哈希查找和潜在锁竞争，提升性能。

实现拆解

1. 引入 SchedulerStatsRecorder：在 rust/src/engine-core-client/src/metrics.rs 中新增结构体，包含 BTreeMap<u32, SchedulerStatsHandles>。构造时遍历所有 engine，调用 resolve_scheduler_stats_handles 预解析 handles，提供 record 方法缓存查找。原有 record_scheduler_stats 函数被替代。
2. 重构 RequestMetricsTracker：在 rust/src/llm/src/request_metrics.rs 中新增 RequestMetricHandles，缓存请求所有固定标签 handles。new 方法增加 engine_index 参数，初始化时预解析 handles。observe_output 等方法直接使用缓存 handles，不再传递 engine_index。
3. 传递 engine_index：在 rust/src/engine-core-client/src/client/stream.rs 中为 EngineCoreOutputStream 添加 engine_index 字段；在 rust/src/llm/src/lib.rs 中调整创建顺序：先调用 client.call 获取 stream，再基于 stream.engine_index() 创建 RequestMetricsTracker。
4. 简化输出流：rust/src/llm/src/output.rs 中 GenerateOutputStream 的 poll_next 调用 observe_output 不再传入 engine_index。
5. 测试适配：rust/src/llm/tests/generate.rs 相应调整。

关键源码片段

rust/src/llm/src/request_metrics.rs

核心文件：引入 RequestMetricHandles 结构体缓存所有请求级 metrics handles，重构 RequestMetricsTracker 使其在构造时预解析 handles，所有方法直接使用缓存 handles 而非重复 get_or_create。

```
/// 预解析并缓存请求级 metrics handles，避免每次记录时重复 get_or_create。
#[derive(Clone)]
struct RequestMetricHandles {
    labels: EngineLabels,

    // 请求级别计数器
    num_preemptions: U64Counter,
    prompt_tokens: U64Counter,
    prompt_tokens_local_compute: U64Counter,
    prompt_tokens_local_cache_hit: U64Counter,
    prompt_tokens_external_kv_transfer: U64Counter,
    prompt_tokens_cached: U64Counter,
    generation_tokens: U64Counter,

    // 请求生命周期指标（直方图等）
    request_success: Family<FinishedReasonLabels, U64Counter>,
    request_prompt_tokens: HistogramMetric,
    request_generation_tokens: HistogramMetric,
    request_max_num_generation_tokens: HistogramMetric,
    request_params_max_tokens: HistogramMetric,
    request_params_n: HistogramMetric,
    request_prefill_kv_computed_tokens: HistogramMetric,
    time_to_first_token_seconds: HistogramMetric,
    inter_token_latency_seconds: HistogramMetric,
    e2e_request_latency_seconds: HistogramMetric,
    request_queue_time_seconds: HistogramMetric,
    request_prefill_time_seconds: HistogramMetric,
    request_decode_time_seconds: HistogramMetric,
    request_inference_time_seconds: HistogramMetric,
    request_time_per_output_token_seconds: HistogramMetric,
}

impl RequestMetricsTracker {
    pub(crate) fn new(
        model_name: String,
        engine_index: u32, // 新增 engine_index 参数
        arrival_time: f64,
        prompt_len: u32,
        max_tokens_param: Option<u32>,
        n_param: u32,
    ) -> Self {
        Self {
            // 在构造时预解析所有 handles
```

```
            handles: resolve_request_metric_handles(&model_name, engine_index),
            arrival_time,
            prompt_len,
            max_tokens_param,
            n_param,
            is_prefilling: true,
            queued_ts: 0.0,
            scheduled_ts: 0.0,
            first_token_ts: 0.0,
            last_token_ts: 0.0,
            first_token_latency: 0.0,
            num_generation_tokens: 0,
            latest_num_cached_tokens: 0,
        }
    }

    pub(crate) fn observe_output(
        &mut self,
        batch_timestamp: f64,
        received_at: f64,
        output: &EngineCoreOutput,
    ) {
        // 直接使用缓存的 handles，无需每次 get_or_create
        self.handles.generation_tokens.inc_by(output.new_token_ids.len() as u64);
        // ... 简化，不再传递 engine_index
    }
}
```

rust/src/engine-core-client/src/metrics.rs

核心文件：引入 SchedulerStatsRecorder 结构体，预解析所有 scheduler stats 的 handles，提供 record 方法替代原有函数。

```
/// 缓存 scheduler stats 的 metric handles，按 engine 索引存储。
pub(crate) struct SchedulerStatsRecorder {
    engines: BTreeMap<u32, SchedulerStatsHandles>,
}

/// 单个 engine 的缓存的 metric handles。
struct SchedulerStatsHandles {
    labels: EngineLabels,
    scheduler_running: U64Gauge,
    scheduler_waiting: U64Gauge,
    scheduler_waiting_capacity: U64Gauge,
    scheduler_waiting_deferred: U64Gauge,
    kv_cache_usage: F64Gauge,
    prefix_cache_queries: U64Counter,
    prefix_cache_hits: U64Counter,
    external_prefix_cache_queries: U64Counter,
    external_prefix_cache_hits: U64Counter,
```

```
    spec_decode_num_drafts: U64Counter,
    spec_decode_num_draft_tokens: U64Counter,
    spec_decode_num_accepted_tokens: U64Counter,
    spec_decode_num_accepted_tokens_per_pos: Family<EnginePositionLabels, U64Counter>,
    estimated_flops_per_gpu: U64Counter,
    estimated_read_bytes_per_gpu: U64Counter,
    estimated_write_bytes_per_gpu: U64Counter,
    kv_block_lifetime_seconds: HistogramMetric,
    kv_block_idle_before_evict_seconds: HistogramMetric,
    kv_block_reuse_gap_seconds: HistogramMetric,
    log_stats: SchedulerLogStatsAccumulator,
}

impl SchedulerStatsRecorder {
    /// 在构造时遍历所有 engine，预解析 handles。
    pub(crate) fn new(
        metrics: &SchedulerMetrics,
        model_name: &str,
        engines: &[ConnectedEngine],
    ) -> Self {
        let engines = engines
            .iter()
            .filter_map(|engine| {
                let engine = engine.engine_id.engine_index()?;
                Some((
                    engine,
                    resolve_scheduler_stats_handles(metrics, model_name, engine),
                ))
            })
            .collect();
        Self { engines }
    }

    /// 记录 scheduler stats，直接使用缓存 handles。
    pub(crate) fn record(&self, engine_index: u32, stats: &SchedulerStats) {
        if let Some(handles) = self.engines.get(&engine_index) {
            record_scheduler_stats_with_handles(handles, stats);
        }
    }
}
```

评论区精华

没有实质性的技术讨论。PR 仅获得 njhill 的 approve 以及机器人自动评论（合并冲突和 pre-commit 提醒）。

风险与影响

- 风险：handles 缓存与底层 metrics 注册表一致；若 engine 在运行时动态增减（如 DP 扩容），SchedulerStatsRecorder 无法自动感知，需重建；动态标签指标（如 spec decoding per position）仍存在 child lookup 开销。
- 影响：对用户透明，提升 Rust frontend metrics 记录性能。改动涉及多个文件但逻辑单一，回归风险较低。

关联脉络

该 PR 属于 git-spice 管理的 stack 的一部分（关联 #47435）。未发现与同仓库近期其他 PR 直接关联。