

PR #57033 完整报告

vllm-project/vllm

[Rust Frontend][Metrics] Add per-request preemption histogram

合并时间: 2026-09-16 10:51

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/57033>

执行摘要

本 PR 为 Rust 前端补齐 per-request 抢占直方图 `vllm:request_num_preemptions`: 每个请求终止 (含 abort) 时记录一次抢占次数样本, 0 次抢占也记录, 桶边界与 Python 前端完全一致。这是 issue #44772 在 Rust 侧的落地, 改动仅 33 行、纯增量、无行为变更, 已通过 CI 与自动 / 人工审核, 是双前端指标对齐的一次小而稳妥的补全。

功能与动机

Rust 前端此前只有引擎级抢占计数器 `vllm:num_preemptions`, 只能看到累计总量。PR body 明确说明对齐目标:

Align the Rust frontend with the Python frontend by tracking preemptions for each request and observing `vllm:request_num_preemptions` when terminal request metrics are recorded, including requests with zero preemptions.

关联 issue #44772 进一步指出运维盲区: `Request.num_preemptions` 在调度器递增但从未暴露到 per-request 指标, "giving no visibility into how frequently individual requests are preempted". 直方图 (而非 counter) 的理由也已写明: 值是 per-request 的、有界且随请求重置, 支持 p50/p99 分位数查询与按严重程度分桶。

实现拆解

- 指标定义与注册 (`rust/src/metrics/src/request.rs`) - 新增常量 `REQUEST_NUM_PREEMPTIONS_BUCKETS`: 7 个桶 [1, 2, 3, 4, 5, 10, 20], 与 Python 前端 #49984 完全一致; - 新增构造函数 `request_num_preemptions_histogram()`; - `RequestMetrics` 结构体新增 `request_num_preemptions: HistogramFamily` 字段, 在 `RequestMetrics::register` 中注册 `vllm:request_num_preemptions`, 并加入 `Self` 返回体;
- 请求级状态跟踪 (`rust/src/llm/src/request_metrics.rs`) - `RequestMetricsTracker` 新增 `num_preemptions: u64` 字段, 初始为 0; - `RequestMetricHandles` 新增 `request_num_preemptions: HistogramMetric` 句柄; - `observe_events` 在 `EngineCoreEventType::Preempted` 分支中, 除原有引擎级 counter 递增外, 再对请求级计数 `self.num_preemptions += 1`; - `record_finished` 在请求终止时 `observe(self.num_preemptions as f64)`, 零抢占请求也会记录一次; - `resolve_request_metric_handles` 通过 `get_or_create_owned(&labels)` 解析句柄;

3. 测试配套 (rust/src/llm/tests/generate.rs) -

`generate_records_request_metrics_in_prometheus_output` (engine=4, 1 次抢占) 断言 `_sum` 为 1.0、`_count` 为 1; - `dropping_stream_records_abort_terminal_request_metrics` (abort, 0 次抢占) 断言 `_sum` 为 0.0、`_count` 为 1, 显式覆盖零值请求也被记录的行为。

无配置、schema 或部署配套改动, 属于纯增量观测能力。

rust/src/metrics/src/request.rs

指标面核心改动: 新增直方图桶常量、构造函数、结构体字段与注册逻辑, 是 `vllm:request_num_preemptions` 指标的定义入口。

```
// 每请求抢占次数的直方图分桶, 与 Python 前端 vllm:request_num_preemptions 保持一致。
//
// 与 token 数等采用 1-2-5 递增的桶不同, 这里使用接近均匀的边界:
// 抢占次数是小整数, 运维可以直观区分被抢占 1 次、2 次、3 次的请求分布,
// 便于按严重程度分级告警。
const REQUEST_NUM_PREEMPTIONS_BUCKETS: [f64; 7] = [1.0, 2.0, 3.0, 4.0, 5.0, 10.0, 20.0];

fn request_num_preemptions_histogram() -> Histogram {
    // 由 prometheus_client 的 Histogram::new 依据桶边界构造直方图。
    Histogram::new(REQUEST_NUM_PREEMPTIONS_BUCKETS.iter().copied())
}

// 在 RequestMetrics::register 中注册该指标族, 与其它 request 生命周期指标并列。
// 使用 new_with_constructor 延迟构造, 避免为每个标签组合预先创建直方图。
let request_num_preemptions =
    Family::new_with_constructor(request_num_preemptions_histogram as fn() -> Histogram);
registry.register(
    "vllm:request_num_preemptions",
    "Histogram of the number of times a request was preempted.",
    request_num_preemptions.clone(),
);
```

rust/src/llm/src/request_metrics.rs

请求生命周期跟踪核心: 新增 `num_preemptions` 状态、指标句柄、事件累计与终止时观测逻辑, 是数据流的主链路。

```
/// 从 engine-core 事件流中维护请求私有的抢占计数。
///
/// 引擎级 counter 展示的是累计总量, 这里额外保存 per-request 计数,
/// 供请求结束时观测进直方图, 从而支撑 p50/p99 级别的分布查询。
fn observe_events(&mut self, events: &[EngineCoreEvent]) {
    for event in events {
        match event.r#type {
            EngineCoreEventType::Queued => {
                self.queued_ts = event.timestamp;
            }
            EngineCoreEventType::Scheduled => {
```

```

        // 只记录首次调度时间，避免重复调度覆盖已修正的时间戳。
        if self.scheduled_ts == 0.0 {
            self.scheduled_ts = event.timestamp;
        }
    }
    EngineCoreEventType::Preempted => {
        // per-request 计数与引擎级 counter 同步累加，
        // 保证 vllm:num_preemptions 与 vllm:request_num_preemptions 口径一致。
        self.num_preemptions += 1;
        self.handles.num_preemptions.inc();
    }
}
}
}

/// 请求终止（含 abort）时收口所有生命周期指标。
pub(crate) fn record_finished(&self, received_at: f64, finish_reason: FinishReason) {
    let e2e_latency_seconds = received_at - self.arrival_time;

    // 每个请求恰好观测一次，包括从未被抢占的请求（num_preemptions == 0），
    // 保证直方图 _count 等于已结束请求总数，分位数计算不缺分母。
    self.handles.request_num_preemptions.observe(self.num_preemptions as f64);

    // 其余过程指标（队列时间、prefill/decode 时间、e2e 延迟等）统一在此观测，
    // 与 Python 前端 FinishedRequestStats 的收口逻辑一一对应。
    self.handles.e2e_request_latency_seconds.observe(e2e_latency_seconds);
    self.record_request_success(finish_reason);
}

```

评论区精华

该 PR 的 review 评论区没有实质技术交锋（review_comments_count = 0），主要结论来自两处：

ChatGPT Codex 自动审核： "Didn't find any major issues. More of your lovely PRs please." (Reviewed commit [21caed384](#))

维护者 BugenZhao 批准并致谢： "Thanks @reidliu41"

真正的设计讨论沉淀在 issue #44772：作者明确解释了 histogram 优于 counter 的理由——per-request 的值有界且随请求重置，可支持 p50/p99 分位数查询与按严重程度分桶。本 PR 的 0 次抢占记录与之呼应，保证 `_count` 分母完整。

风险与影响

技术风险

1. 事件流时序依赖：num_preemptions 依赖 Preempted 事件在终止前到达。若事件乱序会漏计，但与引擎级 counter 同源同路径，不会出现指标间不一致，属既有约束。

2. 双前端口径同步：Rust 与 Python 各自维护桶常量，Python 侧未来调整时 Rust 侧需手动跟进。
3. 零值样本语义：abort 与 0 次抢占请求计入 `_count`，用 `_sum / _count` 算均值会被稀释，应基于 `_bucket` 计算分位数。

影响范围

- 运维可在 Prometheus 查询每请求抢占分布，指标名与 Python 一致，现有 dashboard 可直接复用；
- 每请求多一次直方图 observe，开销可忽略；
- 纯新增指标，无 breaking change，不影响现有指标与告警。

关联脉络

本 PR 是 issue #44772 → Python 前端 #49984 → Rust 前端 #57033 这条功能链的最后一环。vLLM 正在将 v1 前端逐步迁移到 Rust 实现（同仓库近期 PR：51026 修复 Rust 前端 logprobs、56931 为 Rust 前端新增 `--hf-overrides`、57046 退役 Buf schema 发布），指标面的一致性是该迁移的重要验收标准之一。后续新增 per-request 指标时，可复用本 PR 确立的 "事件流累计 + `record_finished` 收口" 模板。