PR #45030 完整报告

vllm-project/vllm

[Rust Frontend][Metrics] Export `vllm:lora_requests_info` from frontend

合并时间：2026-06-11 21:45

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/45030

执行摘要

本 PR 在 Rust 前端新增 vllm:lora_requests_info 指标导出，用于支持外部路由器（如 llm-d）的 LoRA 感知路由。实现通过请求注册表追踪每个 LoRA 请求的调度阶段（等待 / 运行），从引擎事件中推断状态，与 Python 前端行为一致。同时移除了从未填充的 SchedulerStats LoRA 适配器字段。整体变更风险较低，但需注意未来与 Python 侧的一致性维护。

功能与动机

> "Add support for reporting LoRA adapter metrics to the rust frontend, this is required for llm-d compatibility with the lora-aware-routing affinity scorer."

Python 前端已经通过 LoRARequestStates 暴露了该指标，Rust 前端此前缺失。此 PR 填补该空白，使 Rust 前端部署也能被 llm-d 等系统用于路由决策。

实现拆解

1. 请求注册表扩展：在 state.rs 的 TrackedRequest 中新增 lora: Option<LoraRequestState> 字段，并定义 LoraPhase 枚举和 LoraRequestState 结构体。register() 方法新增 lora_name 参数。
2. 事件驱动阶段更新：apply_lora_events() 根据引擎输出中的 Queued / Preempted / Scheduled 事件推进阶段，在每次输出处理开始时调用。
3. 适配器状态聚合：lora_adapter_states() 遍历所有活跃请求，返回运行中和等待中的适配器名称集合。
4. 指标导出器：metrics.rs 的 LoraInfoExporter 负责将集合更新到 Prometheus gauge，当标签集变化时自动清理旧系列。
5. 输出循环集成：在 imp.rs 的 run_output_dispatcher_loop() 中创建 LoraInfoExporter 实例，每次处理完引擎输出后调用导出。
6. 字段清理：从 stats.rs 的 SchedulerStats 删除不再使用的 waiting_lora_adapters 和 running_lora_adapters。

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

实现 LoRA 阶段跟踪核心逻辑：添加 LoraPhase 枚举、LoraRequestState 结构体，修改 register 方法接受 lora_name，实现 apply_lora_events 和 lora_adapter_states。

```
/// LoRA 请求的调度阶段，与 Python 前端 `LoRARequestStates` 对应
#[derive(Debug, Clone, Copy, PartialEq, Eq)]
```

```
enum LoraPhase {
    Waiting,
    Running,
}

/// 前端侧每个 LoRA 请求的状态
#[derive(Debug)]
struct LoraRequestState {
    adapter_name: String,
    phase: LoraPhase,
}

/// 请求注册表中的条目扩展 LoRA 状态
#[derive(Debug)]
struct TrackedRequest {
    sender: OutputSender,
    engine_id: EngineId,
    lora: Option<LoraRequestState>, // 新增：LoRA 请求状态
}

impl RequestRegistry {
    /// 注册请求时接受可选的 lora_name
    pub fn register(
        &mut self,
        request_id: String,
        lora_name: Option<String>, // 新增参数
        data_parallel_rank: Option<u32>,
    ) -> Result<(EngineId, OutputReceiver)> {
        // ... 原有逻辑不变 ...
        self.requests.insert(
            request_id,
            TrackedRequest {
                sender: tx,
                engine_id: engine_id.clone(),
                lora: lora_name.map(|adapter_name| LoraRequestState {
                    adapter_name,
                    phase: LoraPhase::Waiting, // 初始化为 Waiting
                }),
            },
        );
        // ...
    }

    /// 根据引擎输出的事件推进 LoRA 阶段
    fn apply_lora_events(&mut self, output: &EngineCoreOutput) {
        let Some(events) = output.events.as_ref() else { return };
        let Some(lora) = self
            .requests
            .get_mut(output.request_id.as_str())
```

```
            .and_then(|t| t.lora.as_mut()) else { return };
        for event in events {
            lora.phase = match event.r#type {
                EngineCoreEventType::Queued | EngineCoreEventType::Preempted => LoraPhase::
                Waiting,
                EngineCoreEventType::Scheduled => LoraPhase::Running,
            };
        }
    }

    /// 收集所有活跃 LoRA 请求的适配器名称，分为运行中和等待中两组
    pub fn lora_adapter_states(&self) -> (BTreeSet<String>, BTreeSet<String>) {
        let mut running = BTreeSet::new();
        let mut waiting = BTreeSet::new();
        for req in self.requests.values() {
            if let Some(lora) = &req.lora {
                match lora.phase {
                    LoraPhase::Running => { running.insert(lora.adapter_name.clone()); },
                    LoraPhase::Waiting => { waiting.insert(lora.adapter_name.clone()); },
                }
            }
        }
        (running, waiting)
    }
}
```

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

实现 LoraInfoExporter 聚合指标并导出到 Prometheus gauge，提供单元测试验证 emit/replace/drain 语义。

```
/// 导出 `vllm:lora_requests_info` 系列，覆盖所有跨引擎的 LoRA 请求
#[derive(Default)]
pub(crate) struct LoraInfoExporter {
    current: Option<LoraInfoLabels>,
}

impl LoraInfoExporter {
    pub(crate) fn update(
        &mut self,
        metrics: &SchedulerMetrics,
        running: BTreeSet<String>,
        waiting: BTreeSet<String>,
    ) {
        // 当没有活跃 LoRA 时返回 None
        let next = (!running.is_empty() || !waiting.is_empty()).then_some(LoraInfoLabels {
            running_lora_adapters: LoraAdapterNames(running),
            waiting_lora_adapters: LoraAdapterNames(waiting),
        });
```

```
        // 如果标签集发生变化，移除旧的 Prometheus 系列
        if self.current != next
            && let Some(prev) = &self.current
        {
            metrics.lora_info.remove(prev);
        }

        // 设置当前值：与 Python 前端一致，取 Unix 时间戳
        if let Some(labels) = &next {
            metrics.lora_info.get_or_create(labels).set(now_unix_secs());
        }

        self.current = next;
    }
}

/// 获取当前 Unix 时间戳（秒，浮点数）
fn now_unix_secs() -> f64 {
    SystemTime::now()
        .duration_since(UNIX_EPOCH)
        .map(|d| d.as_secs_f64())
        .unwrap_or(0.0)
}

#[cfg(test)]
mod tests {
    #[test]
    fn lora_info_emits_clears_stale_and_drains() {
        // 验证：初始无适配器时不发射；有适配器时正确发射；更新时替换旧系列；全部完成后清空。
    }
}
```

评论区精华

> BugenZhao：“Thanks for the work! … this may not be a good metric for control-plane routing purposes. … or it's now a good time to reconsider about / redesign this, e.g., using the scheduler/engine-side information as the source of truth.”
>
> wseaton：“To be crystal clear, I am implementing this mainly to get API compatibility … I think I'd rather defer a redesign to another PR?”
>
> BugenZhao：“I agree that we may achieve functional parity first and defer refactoring the entire codebase. I've pushed a small commit for refactoring.”

风险与影响

- 风险：锁竞争（低）、与 Python 前端行为一致性维护（中）、缺失适配器负载状态（低）。
- 影响：用户获得 LoRA 指标；系统无性能退化；团队需注意后续同步。

关联脉络

该 PR 是 Rust 前端指标能力的补充，与 Python 前端已有的 LoRARequestStates 功能对等。未来可能通过 *engine/proto* 变更暴露更准确的适配器加载状态，实现更完善的路由支持。