

PR #45137 完整报告

vllm-project/vllm

[Rust Frontend] Add external→internal request-id map for abort()

合并时间: 2026-06-15 14:51

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/45137>

PR 分析报告: Rust 前端外部→内部请求 ID 映射中止功能

执行摘要

此 PR 为 Rust 前端的 Llm、TextLlm 和 ChatLlm 添加 abort() 功能, 通过引入 InflightRequests 实现外部用户请求 ID 到内部引擎 ID 的映射。同时修改 engine-core-client 以在本地合成中止输出流, 确保流不会挂起。这是 Rust 前端请求生命周期管理的重要改进, 为后续用户级 /abort_requests 端点奠定基础。

功能与动机

Python 前端 AsyncLLM 已经支持通过外部请求 ID 中止请求 (abort()), 但 Rust 前端缺乏此能力。用户只能通过内部随机化后的 ID 中止, 这对调试和外部负载均衡等场景不友好。此 PR 填补了这一空白, 同时为即将实现的用户级 /abort_requests 端点 (#44382) 准备了底层基础设施。

实现拆解

1. 新建 InflightRequests 模块 (rust/src/llm/src/inflight.rs) : 定义 InflightMap (HashMap<String, HashMap<String, usize>>) 受 Arc<Mutex<>> 保护。提供 track() 记录映射并返回 RAII RequestGuard, 在请求流 drop 时自动递减引用计数并清理映射。resolve() 将外部 ID 列表解析为当前活跃的内部 ID 列表。
2. 修改 Llm 结构 (rust/src/llm/src/lib.rs) : 新增 inflight 字段。在 generate() 中调用 client.call 成功后通过 inflight.track() 记录映射, 并将 guard 存入 GenerateOutputStream。新增 abort() 方法: 调用 inflight.resolve() 获取内部 ID, 然后委托 client.abort() 发送中止请求。
3. 暴露高层 API (rust/src/text/src/lib.rs, rust/src/chat/src/lib.rs) : TextLlm 和 ChatLlm 分别添加 abort() 方法, 直接委托给内部 Llm::abort()。
4. 本地合成中止输出 (rust/src/engine-core-client/src/client/state.rs, rust/src/engine-core-client/src/client/imp.rs) : RequestRegistry 新增 abort_many() 方法, 从注册表移除请求并立即发送 FinishReason::Abort 终端输出。ClientInner 通过 abort_requests_locally() 包装, 确保调用者不会因流无输出而挂起。
5. 修复指标记录 (rust/src/llm/src/request_metrics.rs) : 在 observe_output 中, 当输出没有新 token 时 (如合成中止输出), 跳过 time-to-first-token 和 inter-token 延迟的观测, 避免错误统计。

6. 测试覆盖: inflight.rs 新增 5 个单元测试, 验证映射解析、重复外部 ID、引用计数、guard 清理等。generate.rs 新增 2 个集成测试, 验证外部 ID 到内部 ID 的解析以及单个外部 ID 中止多个内部请求的行为。

rust/src/llm/src/inflight.rs

核心新模块, 定义了 InflightRequests 映射和 RAII guard

/// 追踪外部到内部请求 ID 映射, 用于支持通过外部 ID 中止请求。

/// 当启用请求 ID 随机化时, 引擎核心只知道内部 ID,

/// 因此需要通过此映射将用户提供的外部 ID 解析为内部 ID。

```
use std::collections::HashMap;
```

```
use std::sync::{Arc, Weak};
```

```
use parking_lot::Mutex;
```

/// 外部 ID → 内部 ID → 活跃 guard 计数。

```
type InflightMap = HashMap<String, HashMap<String, usize>>;
```

/// 管理所有正在飞行中的外部→内部 ID 映射。

/// 同一个外部 ID 可能对应多个内部 ID (重复提交相同外部 ID 时每个有自己的随机 ID)。

/// 同一 (external, internal) 对可能被多个 guard 持有引用计数。

```
#[derive(Default)]
```

```
pub(crate) struct InflightRequests {
```

```
    map: Arc<Mutex<InflightMap>>,
```

```
}
```

```
impl InflightRequests {
```

```
    pub(crate) fn new() -> Self {
```

```
        Self::default()
```

```
    }
```

/// 记录 `external` → `internal` 关系, 返回一个 guard,

/// 当请求输出流被 drop 时 (正常完成或取消) 自动移除该边。

```
pub(crate) fn track(&self, external: String, internal: String) -> RequestGuard {
```

```
    *self
```

```
        .map
```

```
        .lock()
```

```
        .entry(external.clone())
```

```
        .or_default()
```

```
        .entry(internal.clone())
```

```
        .or_insert(0) += 1;
```

```
    RequestGuard {
```

```
        map: Arc::downgrade(&self.map),
```

```
        external,
```

```
        internal,
```

```
    }
```

```
}
```

/// 将一组外部 ID 解析为它们对应的、当前仍在飞行中的内部 ID。

```

/// 未知或已完成的 ID 贡献空结果。
pub(crate) fn resolve(&self, external_ids: &[String]) -> Vec<String> {
    let map = self.map.lock();
    external_ids
        .iter()
        .filter_map(|external| map.get(external))
        .flat_map(|internal_ids| internal_ids.keys())
        .cloned()
        .collect()
}

#[cfg(test)]
fn is_empty(&self) -> bool {
    self.map.lock().is_empty()
}
}

/// RAI guard: 当 drop 时释放一个外部→内部边的引用计数。
/// 如果引用计数降为 0, 则移除该内部 ID; 如果外部 ID 下所有内部 ID 都被移除, 则清理该外部 ID
/// 条目。
/// 使用 `Weak` 引用, 所以即使 guard 存活超过 `InflightRequests` 实例不会阻止其释放。
pub(crate) struct RequestGuard {
    map: Weak<Mutex<InflightMap>>,
    external: String,
    internal: String,
}

impl Drop for RequestGuard {
    fn drop(&mut self) {
        let Some(map) = self.map.upgrade() else {
            return; // InflightRequests 已被释放, 无需清理
        };
        let mut map = map.lock();
        if let Some(internal_ids) = map.get_mut(&self.external) {
            if let Some(count) = internal_ids.get_mut(&self.internal) {
                *count -= 1;
                if *count == 0 {
                    internal_ids.remove(&self.internal);
                }
            }
        }
        if internal_ids.is_empty() {
            map.remove(&self.external);
        }
    }
}
}

```

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

新增 `abort_many` 方法，在注册表层合成终端中止输出，避免流挂起

```
/// 完成客户端发起的中止：从注册表中移除每个请求，并向其流发送一个终端输出（finish_reason =
Abort），
/// 然后 drop 掉发送者。返回仍然活跃的请求 ID。
pub fn abort_many<'a>(
    &mut self,
    request_ids: impl IntoIterator<Item = &'a String>,
    timestamp: f64,
) -> Vec<String> {
    let mut aborted = Vec::new();
    for request_id in request_ids {
        let Some((sender, engine_id)) = self.remove(request_id) else {
            continue; // 请求已经完成或不存在，跳过
        };
        // 合成一个终端输出，通知消费者该请求已被中止
        let output = EngineCoreStreamOutput {
            engine_index: engine_id.engine_index().unwrap_or(0),
            timestamp,
            output: EngineCoreOutput {
                request_id: request_id.clone(),
                finish_reason: Some(EngineCoreFinishReason::Abort),
                ..EngineCoreOutput::default()
            },
        };
        let _ = sender.send(Ok(output));
        aborted.push(request_id.clone());
    }
    aborted
}
```

评论区精华

tahsintunan：在无内部负载均衡时引擎不会为客户端中止返回 `finished_requests`，流可能永远不 `yield`。

BugenZhao：可以通过 `ClientInner` 在本地合成终端输出。

作者：已通过新增 `abort_many` 方法修复。

tahsintunan：建议 `PreparedGenerateRequest` 直接暴露 `external_id` 字段以避免 `expect`。

作者：`expect` 是故意使用的不可达断言，明确语义且不冗余。

tahsintunan：未来 `n>1` 时 `resolve` 需要展开父 ID 为子 ID。

作者：已添加 `TODO` 注释。

风险与影响

- 风险：本地合成中止输出可能影响正常输出流顺序，但仅在中止路径触发，且 `request_metrics` 已跳过无 `token` 输出；多引擎场景下引擎仍可能通过 `finished_requests`

发送重复输出，但本地注册表已移除请求，不会重复处理。整体风险可控。

- 影响：用户可用外部 ID 中止请求，提升调试和负载均衡体验；团队获取了 Rust 前端请求生命周期管理的基础设施。

关联脉络

本 PR 是 Rust 前端请求生命周期管理系列的一部分。之前 PR #44382 添加了 `/abort_requests` HTTP 路由，本 PR 提供底层外部→内部 ID 解析和本地中止逻辑。后续 PR 将在此基础上实现完整的用户级 abort 端点。此外，本 PR 与 `request_metrics` 的改进（防止合成输出错误指标）相关，确保指标精度。