PR #46833 完整报告

vllm-project/vllm

[Rust Frontend] Start current wave for a stale DP FirstRequest

合并时间：2026-06-30 13:13

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46833

执行摘要

本 PR 修复了 Rust 进程内 DP 协调器中的一个竞态条件：当 FirstRequest 入队后引擎提前完成 wave 导致 current_wave 前进时，旧的 FirstRequest 错误地回退 wave。通过引入 start_wave_for_first_request 方法，确保广播当前 wave 并在请求过时时唤醒所有引擎（exclude = None），行为与 Python 协调器一致。

功能与动机

> "In the in-process DP coordinator, notify_first_request enqueues FirstRequest{wave: W} while engines are paused, but the runner broadcasts START_DP_WAVE later. If engines report WaveComplete(W) in that gap, current_wave advances to W+1. The stale FirstRequest{wave: W} was then applied unconditionally, rewinding current_wave back to W and broadcasting a superseded wave."

需要确保 Rust 实现与 Python 协调器的行为一致，即：从不回退 wave，广播当前 wave，并在请求 wave 过时时排除 None（唤醒所有引擎）。

实现拆解

1. 修改 StartDpWaveMessage 和 broadcast_start_wave（inproc.rs）：将 exclude_engine_index 字段类型从 u32 改为 Option<u32>，以支持 None 值（唤醒所有引擎）。更新 broadcast_start_wave 签名和实现。
2. 新增 start_wave_for_first_request 方法（handle.rs）：在 CoordinatorStateSnapshot 上实现新方法，接收 request_wave 和 target_engine_index。将 engines_running 置为 true，然后根据 request_wave >= current_wave 决定 exclude：成立时排除目标引擎，否则为 None。始终返回 self.current_wave，确保不回退。
3. 重构 handle_command 的 FirstRequest 分支（inproc.rs）：不再直接设置 state.lock().current_wave = wave，而是调用 state.start_wave_for_first_request(wave, target_engine_index) 获取当前 wave 和 exclude，然后以此广播。
4. 更新另一处 broadcast_start_wave 调用：引擎通知过时 wave 时，改为传递 Some(engine_index)。
5. 添加单元测试：覆盖正常路径（请求 wave 等于当前 wave，排除目标引擎）和过时路径（请求 wave 小于当前 wave，exclude 为 None）。

rust/src/engine-core-client/src/coordinator/inproc.rs

核心变更文件：修改 broadcast_start_wave 参数类型，重构 handle_command 中 FirstRequest 分支逻辑，并新增单元测试。

```
// rust/src/engine-core-client/src/coordinator/inproc.rs ( 关键变更部分 )

/// 广播 START_DP_WAVE 消息给所有引擎。
/// exclude_engine_index 为 None 时唤醒所有引擎（用于过时请求 wave）。
async fn broadcast_start_wave(
    &mut self,
    wave: u32,
    exclude_engine_index: Option<u32>, // 从 u32 改为 Option<u32>
) -> Result<()> {
    let payload = encode_msgpack(&StartDpWaveMessage {
        wave,
        exclude_engine_index,
    })?;
    self.coordinator_input
        .send(
            ZmqMessage::try_from(vec![
                EngineCoreRequestType::StartDpWave.to_frame(),
                payload.into(),
            ])
            .expect("coordinator START_DP_WAVE message must contain two frames"),
        )
        .await?;
    Ok(())
}

// handle_command 中 FirstRequest 分支，不再直接设置 current_wave，
// 而是委托给 start_wave_for_first_request 处理过时判断。
CoordinatorCommand::FirstRequest {
    target_engine_id,
    wave,
} => {
    let target_engine_index = target_engine_id.engine_index().ok_or_else(|| {
        Error::UnsupportedCoordinatorEngineId {
            engine_id: target_engine_id.to_vec(),
        }
    })?;
    // 获取当前 wave 和 exclude 值，注意这里不改变 current_wave
    let (current_wave, exclude) = {
        let mut state = self.state.lock();
        state.start_wave_for_first_request(wave, target_engine_index)
    };
    debug!(
        current_wave,
        request_wave = wave,
        ?exclude,
        "starting DP wave after first request while engines were paused"
```

```
    );
    // 广播当前 wave，exclude 为 None 时唤醒所有引擎
    self.broadcast_start_wave(current_wave, exclude).await?;
}

// 单元测试：过时请求 broadcast 全部引擎
#[test]
fn stale_first_request_starts_current_wave_for_all_engines() {
    let mut state = CoordinatorStateSnapshot {
        current_wave: 4,
        engines_running: false,
    };
    // 请求 wave 3 小于 current_wave 4，说明 wave 已过时
    let (wave, exclude) = state.start_wave_for_first_request(3, 2);
    assert_eq!(wave, 4); // 广播当前 wave，不回退
    assert_eq!(exclude, None); // 唤醒所有引擎
    assert!(state.engines_running);
    assert_eq!(state.current_wave, 4); // current_wave 保持不变
}
```

rust/src/engine-core-client/src/coordinator/handle.rs

新增 start_wave_for_first_request 方法，封装过时 wave 判断逻辑；修改 exclude 字段类型为 Option。

```
// rust/src/engine-core-client/src/coordinator/handle.rs ( 新增方法 )

impl CoordinatorStateSnapshot {
    /// 为 FirstRequest 启动 wave，返回要广播的 wave 和要排除的引擎索引。
    /// 如果 request_wave 小于 current_wave（过时），则 exclude 为 None（唤醒所有引擎），
    /// 且绝不回退 current_wave。
    pub(crate) fn start_wave_for_first_request(
        &mut self,
        request_wave: u32,
        target_engine_index: u32,
    ) -> (u32, Option<u32>) {
        self.engines_running = true;
        // 当 request_wave >= current_wave 时，说明请求 wave 是当前的，
        // 排除目标引擎（它已经接收过请求）；否则（过时）None 唤醒所有引擎。
        let exclude = (request_wave >= self.current_wave).then_some(target_engine_index);
        (self.current_wave, exclude) // 始终返回 current_wave，不回退
    }
}

// StartDpWaveMessage 结构体中的 exclude 字段类型改变
struct StartDpWaveMessage {
    wave: u32,
    /// Engine index that already received the triggering request and so does not
    /// need an extra wakeup. `None` wakes every engine (used when the triggering
    /// request was for a stale wave).
    exclude_engine_index: Option<u32>, // 从 u32 改为 Option<u32>
```

}

评论区精华

Review 由 BugenZhao 批准，评论为 "LGTM. Thanks!"，无其他讨论或争议。

风险与影响

- 风险：较低。变更仅影响 Rust 进程内 DP 协调器，修复了一个明确的竞态条件 bug。测试覆盖了正常和过时路径。
- 影响：修复了 DP 数据并行场景下 wave 管理的一个潜在正确性问题，使 Rust 实现与 Python 协调器行为一致。不影响 Python 协调器或其他系统模块。

关联脉络

本 PR 是 Rust 前端系列工作的一部分（如 PR #45890），旨在逐步用 Rust 重写 vLLM 的前端组件。本次修复对齐了 Rust 协调器与 Python 协调器的行为，为后续功能演进打下基础。