PR #44591 完整报告

vllm-project/vllm

[Rust Frontend] Batch auto-abort requests by engine

合并时间：2026-06-05 17:59

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44591

执行摘要

本 PR 针对 Rust 前端自动中止功能进行性能优化，将原本逐个发送 Abort 消息改为按引擎批量发送，显著减少高并发断连场景下的 IPC 往返次数。改动简洁，并附有完整回归测试。

功能与动机

原实现中，run_abort_loop 在每次收到 AbortRequest 后立即调用 do_abort_requests 发送一条 Abort 消息。当大量流同时丢弃（如客户端批量断连），引擎端需要处理大量单请求 Abort 消息，造成不必要的 IPC 开销。本 PR 将多个 Abort 请求按引擎聚合为一条消息，降低负载。

实现拆解

1. 核心循环改造：在 run_abort_loop 中，使用 tokio::sync::mpsc::UnboundedReceiver::recv_many 替代原来的 recv，一次性取出最多 1024 个请求。
2. 分组与过滤：遍历收集到的请求，跳过不活跃的请求，并将活跃请求按 engine_id 归入 BTreeMap。
3. 批量发送：遍历分组结果，对每个引擎调用一次 do_abort_requests，传入该引擎的所有 request_id。
4. 清理依赖：移除不再需要的 std::slice 导入。
5. 测试覆盖：新增 dropping_multiple_live_streams_aborts_all_in_a_burst 集成测试，模拟三个流同时丢弃，验证引擎端只收到一条 Abort 消息且携带三个 ID。

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

实现了批量自动中止的核心逻辑

```
/// Background loop that listens for request IDs to abort and sends abort
/// messages to the engine. This is used to implement the auto-abort behavior
/// when a request stream is dropped without being properly terminated.
pub(crate) async fn run_abort_loop(
    inner: Arc<ClientInner>,
    mut abort_rx: mpsc::UnboundedReceiver<AbortRequest>,
) {
    // Coalesce bursts of auto-aborts into a single Abort message per engine.
    // A dropped-stream storm (e.g. many clients disconnecting at once under
    // high concurrency) would otherwise issue one engine round-trip per
    // request. `recv_many` returns as soon as at least one item is ready, so a
```

```
    // lone abort is still forwarded promptly.
    // 中文注释：批量收集自动中止请求，按引擎合并，减少 IPC 往返
    const MAX_DRAIN: usize = 1024;
    let mut batch: Vec<AbortRequest> = Vec::new();

    while abort_rx.recv_many(&mut batch, MAX_DRAIN).await > 0 {
        // 按引擎 ID 分组，每个引擎累积一个 request ID 列表
        let mut by_engine: BTreeMap<EngineId, Vec<String>> = BTreeMap::new();

        for AbortRequest { request_id, cause } in batch.drain(..) {
            let Some(engine_id) = inner.take_auto_abort_target(&request_id) else {
                debug!(request_id, "skip auto-abort for inactive request");
                continue;
            };

            match cause {
                AbortCause::DroppedStream => {
                    info!(request_id, "auto-aborting request due to dropped stream")
                }
                AbortCause::StopStringMatched => {
                    debug!(
                        request_id,
                        "auto-aborting request due to stop string matched"
                    )
                }
            }

            by_engine.entry(engine_id).or_default().push(request_id);
        }

        // 每个引擎发送一次批量 Abort 消息
        for (engine_id, request_ids) in by_engine {
            if let Err(error) = inner.do_abort_requests(&engine_id, &request_ids).await {
                warn!(
                    ?engine_id,
                    ?request_ids,
                    error = %error.as_report(),
                    "failed to auto-abort request streams"
                );
            }
        }
    }
}
```

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

新增集成测试验证批量自动中止

```
#[tokio::test]
async fn dropping_multiple_live_streams_aborts_all_in_a_burst() {
```

```
init_tracing();
let ipc = IpcNamespace::new().unwrap();
let handshake_address = ipc.handshake_endpoint();
let engine_id = b"engine-burst".to_vec();
let request_ids = ["req-1", "req-2", "req-3"];

// 启动 mock engine，先接收三个 Add 请求，然后发送三个输出，
// 最后验证只收到一条 Abort 消息且包含三个 ID
let (shutdown_tx, engine_task) = spawn_mock_engine_task(
    handshake_address.clone(),
    engine_id.clone(),
    |dealer, push| {
        Box::pin(async move {
            // 接收三个添加请求
            for _ in 0..3 {
                let add = recv_engine_message(dealer).await;
                assert_eq!(add[0].as_ref(), &[0x00]);
            }
            send_outputs(
                push,
                EngineCoreOutputs {
                    outputs: vec![
                        request_output("req-1", vec![99], None),
                        request_output("req-2", vec![99], None),
                        request_output("req-3", vec![99], None),
                    ],
                    ..Default::default()
                },
            )
            .await;

            // 期望只收到一条批量 Abort 消息
            let abort =
                timeout(Duration::from_secs(1), recv_engine_message(dealer)).await.unwrap();
            assert_eq!(abort[0].as_ref(), &[0x01]);
            let ids: Vec<String> = rmp_serde::from_slice(&abort[1]).unwrap();
            assert_eq!(
                ids,
                vec![
                    "req-1".to_string(),
                    "req-2".to_string(),
                    "req-3".to_string()
                ]
            );
            // 确认后续没有额外 Abort 消息
            assert!(
                timeout(Duration::from_millis(100), recv_engine_message(dealer)).await.is_err()
            );
        })
```

```
        },
    );

    let client = connect_client_with_ipc(
        handshake_test_config(
            handshake_address,
            1,
            "test-model",
            Duration::from_secs(2),
            0,
            None,
        ),
        &ipc,
    )
    .await;

    // 先打开所有请求流，让 engine 收到所有 Add
    let mut streams = Vec::new();
    for id in request_ids {
        streams.push(client.call(sample_request_with_id(id)).await.unwrap());
    }
    // 消费第一个 token
    for stream in streams.iter_mut() {
        let first = timeout(Duration::from_secs(1), stream.next()).await.unwrap().unwrap().
        unwrap();
        assert_eq!(first.new_token_ids, vec![99]);
    }
    // 同时丢弃所有流，触发批量自动中止
    drop(streams);

    let _ = shutdown_tx.send(());
    engine_task.await.unwrap();
    client.shutdown().await.unwrap();
}
```

评论区精华

无实质性讨论。BugenZhao 直接批准，LGTM. Thanks!。

风险与影响

- 风险：变更范围小，MAX_DRAIN 设为 1024，若超过会分多批，但功能正确；失败日志稍变，无安全风险。
- 影响：显著减少高并发断连时 IPC 往返，对单个请求无影响。仅影响 Rust 前端自动中止流程。

关联脉络

无直接关联的 Issue 或 PR。本 PR 是 Rust 前端持续优化的一部分，与近期其他 Rust 前端 PR（如 #44500 #44391）共同构建更高效的前端层。