

PR #1945 完整报告

radixark/miles

Retry a submit only when it provably never reached the server

合并时间: 2026-09-04 13:24

原文链接: <http://prhub.com.cn/radixark/miles/pull/1945>

执行摘要

- 一句话: 重构 RPC 客户端重试策略, 确保仅重试证明未到达服务器的请求。
- 推荐动作: 这个 PR 值得精读, 它展示了一个关键且经过深思熟虑的分布式系统设计决策。关注点在于:
 1. 设计哲学: 如何通过错误分类 (NEVER_REACHED_SERVER_ERRORS vs RETRYABLE_ERRORS) 来平衡可用性与幂等性。
 2. 实现模式: `retry_until_deadline` 工具如何将重试策略 (超时、退避、日志) 与业务逻辑 (`_submit_attempt`) 解耦。
 3. 测试方法: `FlakyProxy` 和 `_HookTransport` 如何模拟复杂网络故障场景, 是测试网络容错代码的优秀范例。

功能与动机

根据 commit message, 核心动机是: "A submit retries solely on connect-level failures, because anything that may have reached the worker could already be running and a second submit would run it twice." (提交仅在连接层失败时重试, 因为任何可能已到达工作器的请求可能已在运行, 第二次提交会导致其重复执行)。这解决了 RPC 客户端在网络不稳定或服务器部分响应时, 可能错误重试并引发副作用的潜在风险。

实现拆解

变更从通用工具到具体实现, 再到测试验证, 层层递进。

1. 新增通用重试工具: 在 `miles/utils/retry_utils.py` 中新增了 `retry_until_deadline` 异步函数。该函数提供了一个基于截止时间的重试循环, 支持指数退避、抖动和结构化日志。它接收一个回调函数 `fn(remaining_budget)`, 每次尝试时告知调用方剩余时间, 这是后续精细化重试策略的基础设施。
2. 重构 RPC 提交逻辑: 核心变更在 `miles/utils/workers/rpc/client/call.py`。`RpcCall._submit` 方法不再进行简单的重试, 而是调用 `retry_until_deadline`。关键决策在于 `retry_on` 参数被设置为 `NEVER_REACHED_SERVER_ERRORS` (仅包含 `httpx.ConnectError`, `httpx.ConnectTimeout`, `httpx.PoolTimeout`)。这意味着只有在证明请求从未到达服务器的连接错误时才会重试。原有的固定超时 `SUBMIT_TIMEOUT_SECONDS` 被拆分为 `SUBMIT_RETRY_WINDOW_SECONDS` (整体重试窗口) 和 `SUBMIT_ATTEMPT_TIMEOUT_SECONDS` (单次尝试超时)。新增了

`_submit_attempt` 方法封装单次请求，并引入了 `_ABORT_SLACK_SECONDS` 来提前中止超时，避免边界情况。

3. 复用重试工具改进健康检查： `miles/utils/workers/rpc/client/handle.py` 中的 `RpcWorkerHandle.wait_ready` 方法被重构，也复用了 `retry_until_deadline`。这统一了 `wait_ready` 和 `submit` 的重试模式（例如使用相同的退避策略），并移除了内联的重试循环。
4. 扩展异常分类与传输层：在 `miles/utils/workers/rpc/client/misc.py` 中，新增了 `RetryableResponseError` 异常类，并定义了 `RETRYABLE_ERRORS`（可重试的错误，如 5xx 响应）和 `NEVER_REACHED_SERVER_ERRORS` 两个元组。`RpcTransport.request` 方法被修改：显式禁用重定向 (`follow_redirects=False`) 以防止重复提交；同时增加了 `timeout` 计算中的 `+ _ABORT_SLACK_SECONDS`，确保连接超时早于 HTTP 读取超时；最后，对 5xx 响应状态码，抛出 `RetryableResponseError` 而不是 `RpcProtocolError`，将其归类为可重试（由上层决定是否重试）。
5. 全面增强测试覆盖：新增 `tests/fast/utils/workers/e2e/test_client_retry.py` 文件，使用 `FlakyProxy` 进行端到端测试，覆盖了各类错误（5xx, 4xx, 未知方法、连接断开）的重试行为。在 `tests/fast/utils/workers/rpc/client/test_handle.py` 中新增 `TestSubmitRetry` 测试类，用模拟传输层（`_HookTransport`）精细测试了重试窗口、超时、重定向等边界情况。同时，为 `retry_until_deadline` 添加了完整的单元测试。

关键文件：

- `miles/utils/retry_utils.py`（模块 重试工具；类别 `source`；类型 `dependency-wiring`；符号 `retry_until_deadline`）：新增核心重试工具函数 `retry_until_deadline`，为整个 RPC 重试策略重构奠定基础。
- `miles/utils/workers/rpc/client/call.py`（模块 RPC 客户端；类别 `source`；类型 `core-logic`；符号 `_submit_attempt`）：RPC 提交逻辑的核心重构点，应用了新的重试策略，将错误分类具体化。
- `tests/fast/utils/workers/e2e/test_client_retry.py`（模块 重试端测；类别 `test`；类型 `test-coverage`；符号 `short_retry_window`, `_StaleWorker`, `demo_removed_method`, `TestSubmitNoRetry`）：新增的端到端测试文件，全面验证新的重试策略在真实网络故障注入下的行为，是验证设计意图的关键。
- `miles/utils/workers/rpc/client/misc.py`（模块 RPC 客户端公共模块；类别 `source`；类型 `core-logic`；符号 `RetryableResponseError`）：定义了错误分类元组 (`NEVER_REACHED_SERVER_ERRORS`, `RETRYABLE_ERRORS`) 和新增异常类，是实现精细化重试策略的基石。

关键符号：`retry_until_deadline`, `RpcCall._submit`, `RpcCall._submit_attempt`, `RpcWorkerHandle.wait_ready`, `RpcTransport.request`

关键源码片段

`miles/utils/retry_utils.py`

新增核心重试工具函数 `retry_until_deadline`，为整个 RPC 重试策略重构奠定基础。

```
import asyncio
import logging
```

```

import random
import time
from collections.abc import Awaitable, Callable
from typing import Any, TypeVar

from miles.utils.tracking_utils.structured_log import log_structured

logger = logging.getLogger(__name__)

_DEFAULT_INITIAL_DELAY = 1.0
_DEFAULT_MAX_DELAY = 60.0
_DEFAULT_BACKOFF_FACTOR = 2.0
_DEFAULT_JITTER_RATIO = 0.1

_T = TypeVar("_T")

async def retry_until_deadline(
    fn: Callable[[float], Awaitable[_T]],
    *,
    total_seconds: float,
    retry_on: type[Exception] | tuple[type[Exception], ...],
    initial_delay: float = _DEFAULT_INITIAL_DELAY,
    max_delay: float = _DEFAULT_MAX_DELAY,
    backoff_factor: float = _DEFAULT_BACKOFF_FACTOR,
    jitter_ratio: float = _DEFAULT_JITTER_RATIO,
    log_fields: dict[str, Any] | None = None,
) -> _T:
    """在截止时间前重试，仅捕获 `retry_on` 指定的异常。

    fn 接收一个 `remaining` 参数（剩余秒数），允许调用方动态调整其超时。
    """
    expires_at = time.monotonic() + total_seconds
    delay = initial_delay
    attempt = 0

    while True:
        try:
            # 传递剩余时间给回调函数，供其设置请求超时
            return await fn(max(0.0, expires_at - time.monotonic()))
        except retry_on as e:
            attempt += 1
            # 计算带抖动的下一次延迟
            sleep_seconds = delay + random.uniform(0.0, delay * jitter_ratio)
            if expires_at - time.monotonic() <= sleep_seconds:
                logger.warning(f"retry_until_deadline: giving up after {total_seconds:.1f}s", exc_info=True)
                raise
            log_structured(

```

```

        logger.info,
        **{"op": "retry_until_deadline", **(log_fields or {})},
        phase="attempt_failed",
        attempt=attempt,
        sleep_s=round(sleep_seconds, 3),
        remaining_s=round(expires_at - time.monotonic(), 3),
        error=repr(e),
    )
    await asyncio.sleep(sleep_seconds)
    delay = min(delay * backoff_factor, max_delay)

```

miles/utils/workers/rpc/client/call.py

RPC 提交逻辑的核心重构点，应用了新的重试策略，将错误分类具体化。

```

from miles.utils.retry_utils import retry_until_deadline
from miles.utils.workers.rpc.client.misc import (
    NEVER_REACHED_SERVER_ERRORS,
    RETRY_INITIAL_DELAY_SECONDS,
    RETRY_MAX_DELAY_SECONDS,
    RETRYABLE_ERRORS,
    RpcTransport,
    RpcWorkerCallError,
)

```

重试窗口和单次尝试超时

```

SUBMIT_RETRY_WINDOW_SECONDS = 60.0
SUBMIT_ATTEMPT_TIMEOUT_SECONDS = 10.0

```

```

class RpcCall:

```

```

    # ... __init__ 不变 ...

```

```

    async def _submit(self) -> None:

```

```

        request = SubmitRequest(call_id=self._call_id, query=self._query)

```

```

        try:

```

```

            # 使用重试工具，仅在证明未到达服务器的错误上重试

```

```

            await retry_until_deadline(
                lambda remaining: self._submit_attempt(request=request, remaining=remaining),
                total_seconds=SUBMIT_RETRY_WINDOW_SECONDS,
                retry_on=NEVER_REACHED_SERVER_ERRORS, # 关键：只有连接错误才重试
                initial_delay=RETRY_INITIAL_DELAY_SECONDS,
                max_delay=RETRY_MAX_DELAY_SECONDS,
                log_fields={**self._log_fields, "op": "submit"},
            )

```

```

        except RETRYABLE_ERRORS as e:

```

```

            # 所有可重试错误（包括重试窗口耗尽后的最后一次错误）都转化为不可达错误

```

```

            log_structured(logger.warning, op="submit", phase="gave_up", **self._log_fields, error=
                repr(e))

```

```

            raise WorkerUnreachableError(f"{self._method_label} submit failed: {e!r}") from e

```

```

    async def _submit_attempt(self, *, request: SubmitRequest, remaining: float) -> None:

```

```
"""执行单次提交尝试，超时取`remaining`预算和默认超时的较小值。"""
await self._transport.request(
    "POST",
    SUBMIT_PATH.format(method_name=self._spec.name),
    seconds=min(SUBMIT_ATTEMPT_TIMEOUT_SECONDS, remaining),
    response_model=SubmitResponse,
    json=request.model_dump(exclude_none=True),
)
log_structured(logger.debug, op="submit", phase="accepted", **self._log_fields)
```

评论区精华

虽然只有一个批准评论，但 commit message 本身阐述了核心的设计权衡。

- 争议 / 决策点：是否应对服务器端错误（如 5xx）进行重试？结论是“否”。因为 5xx 响应意味着请求可能已被服务器处理，重试可能导致重复执行，破坏幂等性。因此，只有连接层失败（证明请求未发送 / 未被接收）才被重试。
- 设计结论：实现了一个“尽最大努力证明未到达”的重试策略。RETRYABLE_ERRORS 包含 5xx (RetryableResponseError)，但 NEVER_REACHED_SERVER_ERRORS 不包含。调用方（如 _submit）选择只在 NEVER_REACHED_SERVER_ERRORS 上重试，而 wait_ready 则可能在 RETRYABLE_ERRORS 上重试（因其幂等性）。
- 未解决疑虑：从上下文看，此设计决策在 PR 被批准前已达成共识。
- 仅在证明未到达服务器的连接错误上重试，避免重复执行 (design): 采用 NEVER_REACHED_SERVER_ERRORS 元组作为重试条件，5xx 响应被分类为 RetryableResponseError（属于 RETRYABLE_ERRORS），但不在 submit 的重试范围内。

风险与影响

- 风险：
 1. 行为变更风险：这是最主要的风险。之前的提交可能在遇到某些网络错误或服务器 5xx 时进行重试。新策略会立即放弃这些重试。这可能导致在某些网络抖动或服务器临时过载场景下，调用方更快地收到 WorkerUnreachableError，而不是等待或重试。这可能会改变现有系统在瞬态故障下的行为模式。
 2. 参数调整影响：SUBMIT_RETRY_WINDOW_SECONDS (60s) 和 SUBMIT_ATTEMPT_TIMEOUT_SECONDS (10s) 等常量值的引入和调整，会直接影响客户端重试的时机和超时行为，需要确保这些默认值在生产环境中是合理的。
 3. 超时计算复杂性：在 _submit_attempt 和 RpcTransport.request 中，超时计算现在涉及 remaining 预算和 _ABORT_SLACK_SECONDS，增加了复杂性。如果逻辑有误，可能导致请求过早或过晚被取消。
 4. 依赖变更：新增了对 retry_until_deadline 的依赖，这是对内部库的变更，需要确保其行为正确且稳定。
 - 影响：用户影响：最终用户（如使用 RPC 框架的工程师）将体验到更可预测的错误处理。对于证明未到达服务器的失败，系统会提供重试窗口；对于可能已到达服务器的失败，系统会快速失败。这提升了操作的安全性，但可能减少某些边缘情况下的自动恢复能力。系统影响：此变更仅影响 RPC 客户端的重试逻辑（

miles/utils/workers/rpc/client/)，对服务器端和其他模块无直接影响。但它是整个分布式工作器通信栈的一部分，其行为的改变可能影响依赖 RPC 的上层服务（如 Rollout 管理器）的稳定性和错误处理逻辑。团队影响：引入了更清晰、更安全的重试模式，为 RPC 客户端提供了更可靠的基础设施。相关的测试工具（如 FlakyProxy）也得到了增强，有助于团队未来编写更健壮的网络容错测试。

- 风险标记：重试行为变更，超时参数调整，依赖内部库变更

关联脉络

- PR #1947 Refuse a duplicate call id with 409 instead of running it twice: 该 PR 在服务器端防止了重复 call_id 的执行，而本 PR 则在客户端防止了因重试可能导致的重复提交。两者从不同层面保障了操作的幂等性和避免重复执行。
- PR #1948 Pin a client to one server process with a boot uuid: 该 PR 解决了客户端因服务器重启而丢失状态的问题，而本 PR 优化了客户端在网络故障下的重试策略。两者都是增强 RPC 客户端健壮性和可靠性的改进。
- PR #1950 Support non-finite floats across the wire: 该 PR 修复了 RPC 序列化 / 反序列化问题，而本 PR 优化了 RPC 的网络容错逻辑。两者都属于 RPC 通信栈的加固工作，可能被同一套上层服务使用。