Request Many

Metadata	Value
Date	2024-09-26
Author	@aricart, @scottf, @Jarema
Status	Partially Implemented
Tags	client, spec, orbit

Revision	Date	Author	Info
1	2024-09-26	@scottf	Document Initial Design

Problem Statement

Have the client support receiving multiple replies from a single request, instead of limiting the client to the first reply, and support patterns like scatter-gather and sentinel.

Basic Design

The user can provide some configuration controlling how and how long to wait for messages. The client handles the requests and subscriptions and provides the messages to the user.

- The client doesn't assume success or failure - only that it might receive messages.
- The various configuration options are there to manage and short circuit the length of the wait, and provide the user the ability to directly stop the processing.
- Request Many is not a recoverable operation, but it could be wrapped in a retry pattern.
- The client should communicate status whenever possible, for instance if it gets a 503 No Responders

Config

Total timeout

The maximum amount of time to wait for responses. When the time is expired, the process is complete. The wait for the first message is always made with the total timeout since at least one message must come in within the total time.

- Always used
- Defaults to the connection or system request timeout.

Stall timer

The amount time to wait for messages other than the first (subsequent waits). Considered "stalled" if this timeout is reached, indicating the request is complete.

- Optional
- Less than 1 or greater than or equal to the total timeout behaves the same as if not supplied.

- Defaults to not supplied.
- When supplied, subsequent waits are the lesser of the stall time or the calculated remaining time. This allows the total timeout to be honored and for the stall to not extend the loop past the total timeout.

Max messages

The maximum number of messages to wait for.

- Optional
- If this number of messages is received, the request is complete.
- If this number is supplied and total timeout is not set, total timeout defaults to the connection or system timeout.

Sentinel

While processing the messages, the user should have the ability to indicate that it no longer wants to receive any more messages.

- Optional
- Language specific implementation
- If sentinel is supplied and total timeout is not set, total timeout defaults to the connection or system timeout.

Notes

Message Handling

Each client must determine how to give messages to the user.

- They could all be collected and given at once.
- They could be put in an iterator, queue, channel, etc.
- A callback could be made.

End of Data

The developer should notify the user when the request has stopped processing and the receiving mechanism is not fixed like a list or iterator that termination is obvious. A queue or a callback for instance, should get a termination message. Implementation is language specific based on control flow.

Status Messages / Server Errors

If a status (like a 503) or an error comes in place of a user message, this is terminal. This is probably useful information for the user and can be conveyed as part of the end of data.

Callback timing

If callbacks are made in a blocking fashion, the client must account for the time it takes for the user to process the message and not consider that time against the timeouts.

Sentinel

If the client supports a sentinel with a callback/predicate that accepts the message and returns a boolean, a return of true would mean continue to process and false would mean stop processing.

If possible, the client should support the "standard sentinel", which is a message with a null/nil or empty payload.

Cancelling

A client can offer other ways for the user to be able to cancel the request. This is another pathway besides sentinel allowing that the dev can cancel the entire request-many arbitrarily.

Disconnection

It's possible that there is a connectivity issue that prevents messages from reaching the requester, It might be difficult to differentiate that timeout from a total or stall timeout. If possible to know the difference, this could be conveyed as part of the end of data.

Strategies

It's acceptable to make "strategies" via enum / api / helpers / builders / whatever. Strategies are just pre-canned configurations, for example:

Timeout or Wait - this is the default strategy where only the total timeout is used.

Stall - the stall defaults to the lessor of 1/10th of the total wait time (if provided) or the default connection timeout.

Max Responses - accepts a max response number and uses the default timeout.

Subscription Management

Since the client is in charge of the subscription, it should always unsubscribe upon completion of the request handling instead of leaving it up to the server to time it out.

Max Responses Optimization

On requests that specify max responses, and when not using mux inboxes, the client can unsubscribe with a count immediately after subscribing. Theoretically this unsub could be processed after a reply has come in and out of the server, so you still must check the count manually.

Mux Inbox

If possible, the implementation can offer the use of the mux inbox. Consider that the implementation of managing the subscription will differ from a non-mux inbox, for instance not closing the subscription and not implementing a max response optimization.