Pivotal

Timeout Pattern

Problem

- *Liveness*: The system executes the correct behavior, in a timely fashion.
- When system components have *Slow Responses*, they typically tie up resources:
  - Threads
  - Queued resources
- Slow components in synchronous block upstream dependent components,
  Potentially result in cascading failures
  - *Blocked Threads*
  - *Dog Piles*

Solution - Timeout

In preceding examples from *Fault Tolerance Patterns*, we might have used *Timeout* to *Fast-Fail* the downstream request before escalating to *Blocked Threads*:

- Abort the request if it does not complete in acceptable time.
- Can be considered a form of *Fail-Fast*.

Timeout Types - Socket Timeouts

- RESTful applications using HTTP protocol to integrate with down stream resource can generally set *Socket Timeouts*:
  - **Connect Timeout** - Time a client waits to establish a socket connection.
  - **Read Timeout** - Time a client waits for a request to complete.
  - Do not assume defaults, explicitly set them.

Timeout Types - Walkaway Timeouts

- A mechanism to monitor the time of a thread's execution
- After a specified timeout interval, if not complete, abandon the thread.
  - Cannot explicitly terminate threads in Java
  - Can send an `InterruptedException`, but not guaranteed the culprit thread will honor it
  - Cannot guarantee clean up of long running or stuck threads
- Typically used in combination of *Load Shedding*

Timeouts and 3rd Party Clients

- Beware of 3rd party clients - they may not support Socket timeouts.
- Apps using EJB or RMI integration generally *cannot* use client socket timeouts.
- Use Walk-Away Timeouts

Benefits

- Simple pattern
- For socket-based communication, most languages have a facility to set at least at network library.
- Can correlate to Service Level Indicators (SLIs).

Trade-offs

- Does not address root cause of *Slow Response*.
- Too short of timeout and too liberal retry policy can result in *Dog Piles* on slow producers.
- Does not address subsequent consequence of failures to the consumer (if retries fail).
- Higher operational tuning overheads, requires combined use of timeout, retry and/or backoff for resilience strategy based from SLOs.
- Walk-Away timeouts require use of independent thread-pools, at higher resource cost