

L U M B R E



Fallos, timeouts y patrones de resiliencia

Grado: 6

Materia: Sistemas Distribuidos · Ingeniería de Datos

11/10/2026

Antes de empezar

Por qué una dependencia lenta puede tumbar todo un sistema y cómo los patrones de resiliencia (timeout, retry con jitter, idempotencia, circuit breaker, bulkhead) acotan el fallo sin amplificarlo.

QUÉ APRENDERÁS

- Explicar el fallo parcial en sistemas distribuidos y aplicar timeouts, reintentos con backoff, idempotencia, circuit breaker y bulkhead para contener el daño.

Fallos, timeouts y patrones de resiliencia

VIDEO

Resiliencia frente a fallos distribuidos

Video explicativo (inglés): patrones de circuit breaker, retries y timeouts para manejar fallos en sistemas distribuidos.



Escanea para verlo en Lumbre

<https://www.youtube.com/watch?v=TEah5YpBm-A>

La pregunta no es si fallará, sino cuándo y de qué forma

En un solo proceso un error es una excepción; en un sistema distribuido es una posibilidad constante y parcial: la llamada puede tardar, perderse o ejecutarse dos veces. La resiliencia no es evitar fallar, sino fallar sin derribar todo.

Cuando un servicio llama por red a otro, entran tres enemigos que no existen dentro de un proceso: latencia incierta, particiones de red y fallos parciales. No sabes si tu petición se perdió al ir o si la respuesta se perdió al volver. Todos los patrones de resiliencia nacen de aceptar esa incertidumbre.

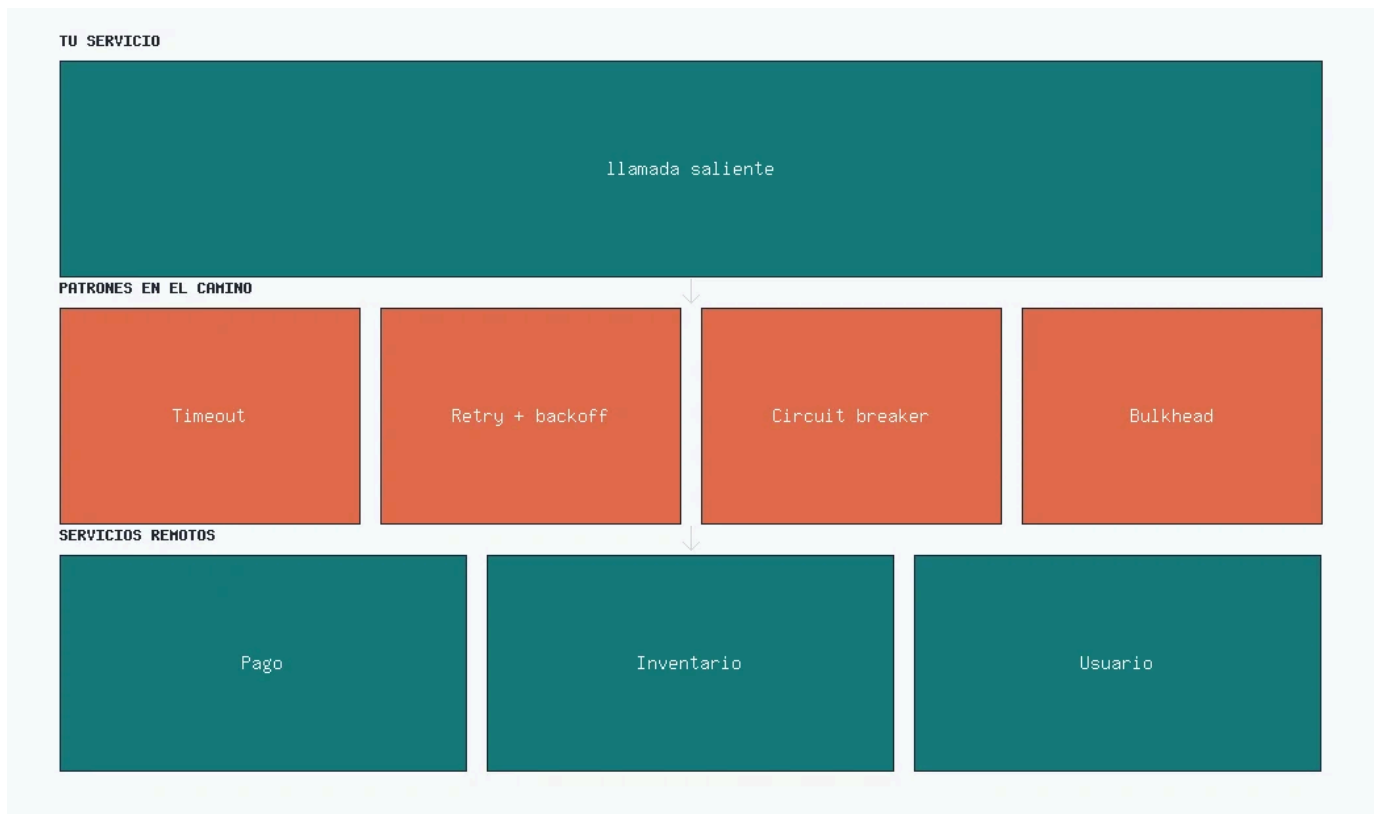
El fallo parcial

Un nodo puede estar "vivo" para unos y "muerto" para otros. Ante una llamada sin respuesta, no hay forma de distinguir en el instante si la petición llegó, se procesó o se perdió; por eso la corrección exige **idempotencia**, no solo reintentos.

- Caídas: el servicio remoto no responde.
- Latencia variable: responde, pero tarde, y bloquea a quien espera.
- Partición de red: ambos sanos pero incomunicados.

Timeouts: ponle reloj a todo

Toda llamada remota necesita un tiempo máximo. Sin timeout, un hilo espera para siempre, se agotan las conexiones y una dependencia lenta tumba a tu servicio. Un timeout demasiado corto, en cambio, cancela trabajo que iba a terminar y dispara reintentos inútiles.



Entre tu servicio y una dependencia lenta o caída se interponen los cuatro patrones que acotan el daño.

Reintentos: útiles y peligrosos

Reintentar salva fallos transitorios, pero reintento sin control amplifica la carga: si todos los clientes reintentan a la vez sobre un servicio caído, se arma una **tormenta de reintentos** que impide que se recupere.

- Backoff exponencial: espera cada vez más entre intentos.

- Jitter: aleatoriza la espera para que los clientes no coincidan.
- Presupuesto de reintentos: límite global para no amplificar el fallo.

Idempotencia: repetir sin dañar

Una operación es **idempotente** si ejecutarla varias veces tiene el mismo efecto que ejecutarla una. Es lo que hace seguros a los reintentos. Un PUT que fija un valor es idempotente; un POST que crea un cargo, no. Cuando la operación no es naturalmente idempotente, se añade una **clave de idempotencia** que el servidor usa para descartar duplicados.

Cada patrón cubre un modo distinto de fallo; se complementan.

Patrón	Qué protege	Riesgo si se omite
Timeout	Que una dependencia lenta congele recursos	Hilos y conexiones agotados
Retry + backoff	Fallos transitorios breves	Tormenta de reintentos
Circuit breaker	No martillar un servicio caído	Colapso en cascada
Bulkhead	Aislar recursos por dependencia	Un fallo consume todo el servicio
Idempotencia	Repetir sin duplicar efectos	Cargos o escrituras duplicadas

Circuit breaker: descansar al servicio caído

Un interruptor cuenta fallos; si superan un umbral, se **abre** y deja de llamar durante una ventana, fallando rápido sin sobrecargar al dependiente. Tras el descanso pasa a **medio abierto**: deja pasar algunas llamadas para probar; si van bien, se **cierra** y se reanuda el tráfico normal.

Bulkhead: compartimentos estancos

Como en un barco, se divide recursos (hilos, conexiones) por dependencia, para que una llamada lenta a un servicio llene solo su compartimento y no hunda el buque completo. Un fallo aislado no consume la capacidad de todo el servicio.

Errores frecuentes

- Reintentar de inmediato y sin tope sobre un error que no era transitorio.
- Poner timeouts mayores que el SLA del propio servicio, propagando esperas.
- Diseñar operaciones de escritura no idempotentes y exponerlas a reintentos.

Ejemplo resuelto: llamada a una pasarela de pago

Un servicio cobra llamando a una pasarela externa que a veces se tarda o responde mal. Secuencia de defensa:

1. Fijar un timeout por debajo del tiempo que el usuario está dispuesto a esperar.
2. Enviar una clave de idempotencia para que un reintento no cobre dos veces.
3. Reintentar con backoff exponencial y jitter, con presupuesto máximo.
4. Abrir un circuit breaker si la pasarela falla de forma sostenida.
5. Aislar con bulkhead las conexiones de pago para no afectar otras funciones.

¿Para qué sirve en la realidad?

Los grandes apagones de servicios en línea rara vez nacen de un solo fallo: nacen de un fallo pequeño que se amplificó por reintentos y esperas sin límites. La resiliencia es lo que convierte una molestia local en algo que el usuario casi no nota.

1. ¿Qué problema intenta resolver principalmente un circuit breaker?

- ☐ Acelerar el disco del servidor
- ☐ Evita seguir llamando a un servicio caído y amplificar su colapso
- ☐ Cifrar las respuestas
- ☐ Compresar el tráfico de red

2. Reintentar una operación no idempotente puede provocar efectos duplicados, como un doble cargo.

- ☐ Verdadero
- ☐ Falso

3. La espera creciente entre reintentos, con variación aleatoria para que los clientes no coincidan, se llama backoff exponencial con

_____.

4. Clasifica cada patrón según el problema que resuelve.

Elementos:

- bulkhead por dependencia
- clave de idempotencia
- circuit breaker abierto
- operación idempotente por diseño
- timeout que acota la espera

Aislar daño	. . .
Repetir seguro	. . .

5. Une cada patrón con su idea central.

5.	Timeout	-----
5.	Backoff + jitter	-----
5.	Circuit breaker	-----
5.	Bulkhead	-----

Opciones: tiempo máximo antes de rendirse · esperar más y dispersar reintentos · dejar de llamar al que falla · aislar recursos por dependencia

ACTIVIDAD INTERACTIVA DISPONIBLE EN
LUMBRE

6. Resiliencia distribuida



Fallos, timeouts y patrones de resiliencia

Resiliencia

Fallo parcial

latencia incierta

particion de red

Timeouts

acotar espera

Reintentos

backoff exponencial

jitter y presupuesto

Idempotencia

repetir sin danar

clave de idempotencia

Aislar

circuit breaker

bulkhead

El mapa mental interactivo es interactivo en la versión online.



Timeouts, reintentos y backoff con jitter (AWS Builders' Library) <https://aws.amazon.com/builders-library/timeouts-retries-and-backoff-with-jitter/> Explicación práctica y rigurosa de por qué y cómo reintentar sin amplificar fallos.

CircuitBreaker (Martin Fowler) <https://martinfowler.com/bliki/CircuitBreaker.html>
Descripción clásica del patrón con sus estados cerrado, abierto y medio abierto.

7. Tu servicio de checkout depende de una pasarela de pago que empieza a responder en 8 segundos aunque suele tardar 200 ms. Sin tocar la pasarela, ¿qué combinación de timeout, reintentos, breaker y bulkhead aplicarías para que ese deterioro no tumbe toda la tienda?

Referencia curricular

Grado:	6
Materia:	Sistemas Distribuidos · Ingeniería de Datos

Versión interactiva completa (con actividades, videos y autoevaluación):



<https://univ.lumbre.lat/object/cdia-sdis-004-fallos-timeouts-y-patrones-de-resiliencia>

Material educativo generado desde Lumbre · lumbre.lat