Engineer

Espera exponencial

⏳ espera para reintentar que se multiplica tras cada fallo

Qué es

La espera exponencial, o backoff exponencial, es una regla que determina cuánto esperar antes de volver a intentarlo. Tras cada intento fallido, la espera se multiplica por un factor fijo, normalmente dos, hasta un límite: un segundo, después dos, cuatro, ocho y así sucesivamente. La regla reside en el cliente, pero existe para proteger al sistema al que llama. Un servidor, una ruta de red o una base de datos que falla por exceso de carga no se recupera antes porque se le envíe otra solicitud de inmediato. Se recupera cuando quienes llaman reducen el ritmo, y una espera que crece con cada fallo hace que lo reduzcan más cuanto más dure el problema.

El ejemplo más antiguo y extendido está en TCP. RFC 6298, el documento de junio de 2011 en la vía de estandarización que especifica cómo calcula TCP su tiempo de espera de retransmisión, lo fija en un segundo antes de medir el primer tiempo de ida y vuelta y establece que, si el valor calculado es inferior a un segundo, debería redondearse hacia arriba hasta un segundo. Cada vez que vence el temporizador y hay que reenviar un segmento, el host debe duplicar el tiempo de espera. Se permite un límite superior siempre que sea de al menos 60 segundos. Cuando llega una nueva medida del tiempo de ida y vuelta, se recalcula a partir de las mediciones, lo que puede reducirlo de inmediato tras una serie de duplicaciones. El algoritmo se remonta al artículo de Van Jacobson de 1988, *Congestion Avoidance and Control*; la sección de seguridad del RFC explica qué está en juego: internet depende de un temporizador de retransmisión correcto para mantener estable la red y evitar el colapso por congestión.

En el código de una aplicación, la misma idea incluye unos pocos ajustes: una espera inicial, un multiplicador, una espera máxima que detiene el crecimiento y un límite del número de intentos o del tiempo total dedicado a reintentarlo. Expresada como fórmula, la espera tras el enésimo fallo es el menor valor entre el límite y la base × el multiplicador elevado a n, contando desde cero. El análisis

de AWS que se comenta más adelante lo llama espera exponencial limitada. El límite importa más de lo que parece: duplicar diez veces una espera de un segundo da 1.024 segundos, más de 17 minutos, más de lo que la mayoría de los clientes esperaría por una respuesta.

El último ajuste es el jitter, un componente aleatorio de cada espera. El protocolo de espera de reconexión de gRPC describe cómo se reconectan sus clientes tras fallar una conexión: empieza en un segundo, multiplica por 1,6, deja de crecer a los 120 segundos y aplica un jitter de 0,2, así que cada espera se desplaza al azar hacia arriba o abajo hasta un 20 %. También da al menos 20 segundos para completar cada intento de conexión y restablece la espera a su valor inicial cuando el servidor acepta una conexión; el protocolo considera que ese momento es la llegada de la trama SETTINGS. Permite otras implementaciones, por ejemplo, para un teléfono móvil que deba despertarse lo menos posible, pero exige separar en el tiempo los reintentos simultáneos y evitar un ritmo de intentos sustancialmente mayor.

Utilice el concepto en una decisión

Aplique «¿Qué es la espera exponencial?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Monitorización y alertas que avisan cuando un fallo aún es pequeño, y cifras que muestran con qué frecuencia se mantiene así.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Primer panel de observabilidad de la empresa — la primera visión de la empresa sobre sus propios sistemas

Primer panel de observabilidad de la empresa Póngase en contacto →

Por qué importa

Un reintento añade carga y llega en el peor momento: justo cuando falla lo que se está llamando. Si un servicio se ralentiza por sobrecarga y cada cliente responde reintentando de inmediato, la carga sube mientras cae la capacidad. Por eso la documentación de Google Cloud Storage considera un antipatrón reintentar sin espera progresiva: los reintentos inmediatos o muy próximos pueden producir fallos en cascada, donde uno desencadena otros. La espera progresiva convierte el reintento de amplificador en freno: cuanto más dure el problema, menos añade cada cliente.

La espera progresiva por sí sola no basta, porque los clientes que fallan juntos también esperan juntos. En *Exponential Backoff And Jitter*, publicado por AWS en marzo de 2015, Marc Brooker simuló numerosos clientes que usaban control

de concurrencia optimista para actualizar la misma fila en una base de datos remota. Si todos competían en cada ronda, el trabajo total crecía con el cuadrado del número de clientes. La espera exponencial limitada ayudaba poco: las llamadas seguían llegando en grupos, con intervalos tranquilos entre ellos, así que apenas bajaba la contención. Añadir jitter, eligiendo cada espera al azar entre cero y el valor exponencial limitado, eliminó los intervalos sin llamadas y dejó un ritmo aproximadamente constante de llamadas; en el caso con 100 clientes en contención, redujo las llamadas a menos de la mitad. Entre las variantes comparadas, la espera sin jitter era, en palabras de Brooker, «la clara perdedora». También expuso el límite: ninguno de los métodos cambia que el trabajo crezca con el cuadrado de la contención. El jitter reduce el desperdicio, no elimina el conflicto. Una actualización de mayo de 2023 señala que la mayoría de los SDK de AWS ya incluye espera exponencial y jitter en sus modos de reintento estándar y adaptativo.

La espera progresiva decide cuándo reintentar, no si es seguro hacerlo. La documentación de Cloud Storage antepone dos preguntas. Una es si la respuesta señala un problema transitorio: códigos de estado HTTP 408, 429 y 5xx, tiempos de espera agotados de sockets y desconexiones TCP. La otra es si la solicitud es idempotente. Una operación de listado se puede repetir libremente. Crear una notificación de Pub/Sub no, porque cada llamada satisfactoria crea otra. En medio están las operaciones condicionalmente idempotentes, que se pueden reintentar con seguridad si llevan una condición previa, como una subida que solo tiene éxito si el número de generación del objeto sigue coincidiendo con el que espera el cliente.

A veces el servidor sabe más que la fórmula del cliente. La cabecera Retry-After, definida en RFC 9110, indica cuánto debería esperar el cliente antes de otra solicitud, como fecha o número de segundos; con una respuesta 503 Service Unavailable indica cuánto tiempo espera el servicio estar indisponible. El estado 429 Too Many Requests de RFC 6585, que señala un límite de frecuencia, puede llevar la misma cabecera. Un cliente con un bucle de reintentos puede usar ese valor en lugar de su espera calculada, porque procede de quien ve su propia carga.

Cómo se aplica en la práctica

Un bucle de reintentos con espera exponencial limitada y jitter completo cabe en unas líneas de Python:

```
import random
import time

class TransientError(Exception):
    """A timeout, a 408, 429 or 5xx response, or a dropped connection."""

def call_with_backoff(op, attempts=5, base=1.0, cap=30.0):
    for attempt in range(attempts):
        try:
            return op()
        except TransientError:
            if attempt == attempts - 1:
                raise
            ceiling = min(cap, base * 2**attempt)
            time.sleep(random.uniform(0, ceiling))
```

Con estos valores, cinco intentos están separados por cuatro esperas cuyos límites son 1, 2, 4 y 8 segundos. El bucle duerme, por tanto, un máximo de 15 segundos antes de rendirse y unos 7,5 segundos de media, porque cada espera se elige con distribución uniforme entre cero y su límite. Solo se reintenta `TransientError`: un fallo de autorización o una solicitud mal formada genera una excepción de inmediato. El bucle repite todo lo que hace `op`, así que `op` debe poder repetirse con seguridad: una lectura o una escritura cuya repetición pueda reconocer el servidor.

Las bibliotecas cliente incluyen sus propias versiones del bucle y sus valores predeterminados difieren incluso para el mismo servicio. En Cloud Storage, el cliente de Python empieza en un segundo, duplica cada espera hasta 60 segundos y abandona al alcanzar un plazo total predeterminado de 120 segundos. El de C++ también empieza en un segundo y duplica, pero espera hasta cinco minutos entre intentos y sigue reintentando hasta 15 minutos, más de siete veces el plazo del cliente de Python. También reintenta operaciones que nunca son idempotentes, a menos que el programa active expresamente una política estricta de idempotencia. La línea de comandos gcloud usa lo que Google llama espera exponencial binaria truncada, con hasta 32 reintentos de forma predeterminada. Dos programas que llaman a la misma API y reciben el mismo fallo pueden, por tanto, comunicarlo tras dos minutos o tras un cuarto de hora, según el lenguaje en que estén escritos.

La espera no tiene por qué bloquear un hilo. En el trabajo en segundo plano, una tarea fallida puede volver a una cola con una fecha de ejecución posterior, y la espera progresiva pasa a ser una propiedad de la tarea en lugar de serlo de un proceso dormido. Ese es su lugar natural cuando nadie espera la respuesta.

En el correo con conmutación por error y registro de entregas, trabajo realizado en un puesto desempeñado desde 2025, la espera progresiva está en la parte asíncrona del sistema. El correo sale por tres proveedores: SendGrid como principal, seguido de SMTP2GO y Azure Communication Services, con conmutación automática entre ellos. La ruta de envío pasa por un cliente con un tiempo de espera de 30 segundos, y la de entrega asíncrona tiene reintentos y espera progresiva para que una interrupción breve no cueste un mensaje. La página resume el resultado: cuando un proveedor tiene una mala tarde, el correo sigue una ruta degradada en vez de sufrir una caída. También reconoce el coste: mantener tres integraciones funcionando en lugar de una.

Dónde surgen los problemas

Todos los clientes esperan lo mismo. Sin jitter, los clientes que fallaron juntos reintentan juntos y el servidor vuelve a ver el mismo pico tras uno, dos, cuatro y ocho segundos. La simulación de Brooker observó que la espera exponencial sin más apenas reducía la contención por ese motivo. Añada jitter: en su comparación, tanto Full Jitter como Decorrelated Jitter reducían mucho el trabajo de los clientes y la carga del servidor; cualquiera de los dos es una opción predeterminada razonable.

La espera no tiene límite o los reintentos no tienen fin. Duplicar sin límite supera los 17 minutos tras diez fallos, y un bucle sin límite mantiene viva una solicitud mucho después de que ya no quede nadie esperando su respuesta. Google considera un antipatrón ignorar los límites de reintento: reintentarlo indefinidamente puede agotar los recursos de la aplicación o seguir enviando solicitudes a un servicio que no se recuperará sin intervención. Fije una espera máxima y un plazo total adecuados para quien llama, cortos donde importa la latencia y más largos para el trabajo por lotes.

Se reintentan errores permanentes. Si hay un fallo de autorización o la solicitud es incorrecta, cada intento fracasa del mismo modo. Reintentar solo retrasa el error y, sin límite, el cliente puede acabar en un bucle infinito, como advierte la documentación de Cloud Storage. Clasifique los errores como transitorios o permanentes antes del bucle, reintente solo los transitorios, como 408, 429, 5xx y conexiones interrumpidas, y registre el resto con su causa.

Se reintenta a ciegas una operación no idempotente. Si una solicitud agotó su tiempo de espera después de que el servidor ya actuara, reintentar repite la acción, que la lista de antipatrones de Google describe como sobrescrituras o

eliminaciones involuntarias de datos. Reintente solo operaciones idempotentes, condicione las escrituras cuando lo permita la API y, en los demás casos, asegúrese de que el receptor pueda detectar duplicados.

Se acumulan reintentos en varias capas. Si la aplicación hace tres intentos y la biblioteca cliente situada debajo hace tres por cada uno, un fallo se convierte en nueve intentos. Es el propio ejemplo de Google, y cada capa adicional que reintenta por su cuenta vuelve a multiplicar la cifra. La documentación añade que tantos reintentos pueden provocar una limitación de la frecuencia de solicitudes y mayor latencia sin mejorar la tasa de éxito. Reintente en una sola capa, normalmente la biblioteca, y desactive o reduzca los reintentos en las demás.

La espera nunca se reinicia o nadie la ve. Un cliente que conserva su última espera larga después de la recuperación espera todo el intervalo máximo en el siguiente fallo pequeño; gRPC restablece la espera inicial al aceptarse una conexión, y TCP recalcula su temporizador con mediciones nuevas. Restablezca la espera tras un éxito. Además, como los reintentos ocultan problemas a los usuarios, también se los ocultan al equipo. Cuéntelos en la monitorización y genere una alerta cuando suba su frecuencia: una dependencia que necesita tres intentos por llamada ya está fallando.

Lecturas adicionales

- RFC 6298: calcular el temporizador de retransmisión de TCP — Las reglas de la vía de estandarización para el tiempo de espera de retransmisión de TCP, incluida su duplicación cada vez que vence el temporizador.
- Protocolo de espera de reconexión de gRPC — El algoritmo de reconexión, sus cinco parámetros con sus valores y cuándo se reinicia la espera.
- Espera exponencial y jitter — La simulación de clientes en contención de Marc Brooker de 2015, que compara la ausencia de jitter con Full, Equal y Decorrelated Jitter.
- Estrategia de reintentos — Qué considera Cloud Storage seguro de reintentar, los valores predeterminados de cada biblioteca cliente y cinco antipatrones de reintento.
- RFC 9110: semántica de HTTP — La cabecera Retry-After, con la que el servidor indica al cliente cuánto esperar antes de su siguiente solicitud.



Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es la espera exponencial?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/exponential-backoff/