Capítulo 11: Escalabilidad

Víctor Iranzo

2 de junio de 2018

1. Introducción

Podemos hacer todo lo posible para evitar que un sistema falle, pero llego un punto en la escala, la probabilidad de fallo es inevitable. Muchas organizaciones invierten mucho esfuerzo en evitar que un fallo se produzca, pero muy poco en mecanismos para recuperar el sistema una vez se ha producido.

Los mecanismos de escalabilidad de nuestro sistema han de ser conformes a sus necesidades: tener un sistema capaz de reaccionar automáticamente al incremento de carga o a un fallo es fantástico pero exagerado para ciertos servicios. También es necesario cuantos fallos pueden tolerar o cuánto tiempo puede estar el sistema caído para nuestros usuarios. Estos requisitos se pueden definir así:

- Latencia o tiempo de respuesta: cuánto tiempo puede una operación tardar. La carga que sufre un sistema puede influir en su tiempo de respuesta.
- Disponibilidad: cuánto tiempo está disponible un servicio respecto del tiempo que se deseaba que estuviera en funcionamiento. ¿Puede estar el servicio caído o tiene que estar disponible 24/7?
- Durabilidad de los datos: cuánto tiempo se deben almacenar ciertos datos y cuánta pérdida de datos se considera aceptable.
- Resilencia: es la capacidad de un sistema para tolerar fallos y continuar trabajando. Un sistema que por culpa de un servicio caído deja de funcionar es menos resilente que un sistema que puede continuar ofreciendo el resto de sus funcionalidades.

Existen varios modos de fallo que se pueden dar cuando un sistema está caído. Responder de forma lenta es uno de los peores porque aumenta la latencia de todas las peticiones para responder que el servicio está caído. Esto puede producir fallos en cascada dentro de la cadena de invocaciones.

2. Gestión de fallos en el sistema

En el libro "Antifragile"se explica como una organización como Netflix basada en la infraestructura de AWS prueba la tolerancia a fallos de sus servicios incitando al fallo de estos. Para ello, emplea un conjunto de programas que componen el "ejército de simios".

Chaos Monkey se encarga durante ciertas horas al día de apagar máquinas de forma aleatoria. Chaos Gorilla hace una función similar pero con los centros de datos. Latency Monkey simula redes de bajo rendimiento. Todos estos fallos pueden darse en producción y esta librería es una buena manera para comprobar si se está preparado para tolerarlos. Sin embargo, ¿cómo podemos gestionarlos una vez se producen?

2.1. Timeouts

Una manera de detectar los fallos es con una cuenta atrás. Esperar demasiado tiempo puede ralentizar al sistema entero. Esperar poco tiempo puede suponer considerar como fallidas tareas exitosas que han tardado más de lo normal. Una buena táctica para establecerlos es poner el mismo valor por defecto a todas las llamadas del sistema y ajustar los timeouts en función de los registros que obtengamos.

2.2. Cortocircuitos

Esta esta técnica se aplica cuando un servicio está caído o pasa un límite fijado por devolver timeouts o errores. Cuando esto ocurre, se dejan de mandar peticiones al servicio para que este devuelva un fallo rápido y no ralentice al resto del sistema. De esta forma, después de un período de tiempo podemos comprobar automáticamente si el servicio se ha recuperado y volverle a redirigir tráfico.

Como en el caso anterior, podemos establecer un cortocircuito con una condición por defecto y ajustarla según los casos específicos. Mientras la conexión está cerrada, las peticiones recibidas se pueden encolar para aten-

darlas más tarde. Existen diferentes librerías que implementan esto, como son la libería de Polly para .NET y la librería de Netflix Hystrix para Java.

Los cortocircuitos también se pueden emplear para tareas de mantenimiento programadas: cuando se tiene que modificar un servicio, se puede cortar la conexión de este y restablecerla una vez realizado el mantenimiento.

2.3. Bulkhead

Son una aproximación para aislar al sistema de un fallo en uno de sus componentes. Si tenemos una conexión distinta para cada servicio con el que interaccionemos, el que una de las conexiones vaya despacio no afectará al resto.

La separación en microservicios facilita el uso de bulkheads. A diferencia de los timeouts y los cortocircuitos que se emplean para actuar cuando un fallo se detecta, los bulkheads se pueden emplear para evitar que el fallo ocurra. Hystrix, por ejemplo, permite rechazar peticiones bajo ciertas condiciones para evitar que un servicio se sature, lo que se llama load schedding. Esto puede ser una buena medida para no convertir un servicio en un cuello de botella y evitar su fallo.

2.4. Aislamiento

Cuanto más dependa un servicio de otro para funcionar, más impacta la "‘salud"de uno sobre el otro. Esto se percibe sobretodo en las llamadas síncronas: un servicio puede tardar en responder no porque él no esté funcionando correctamente sino porque otro a quién ha invocado no contesta, produciendo un fallo en cascada.

3. Idempotencia

Si una operación se idempotente, podemos repetirla más de una vez sin un impacto adverso. Este tipo de operaciones es útil cuando no estamos seguros de si una operación se ha realizado y reenviamos la misma petición. En situaciones en las que tenemos varias instancias de un mismo servicio suscritas a los mismos eventos, aunque dos instancias procesen la misma petición no se producirá ningún error.

El sistema entero no tiene porque ser idempotente, solo las funcionalidades de negocio que se deseen. Algunos verbos HTTP como el GET o el PUT están definidos en su especificación como idempotentes. Si estos verbos no son idempotentes y los clientes piensan que sí, pueden surgir problemas a la hora de consumir el servicio.

4. Técnicas de escalabilidad

4.1. Escalabilidad vertical

4.2. Dividir la carga de trabajo de un servicio

4.3. Balanceadores de carga

4.4. Sistema basado en trabajadores

5. Escalabilidad en bases de datos

5.1. Escalabilidad en la lectura

5.2. Escalabilidad en la escritura

5.3. CQRS

6. Caching

6.1. Tipos de caching

7. Autoescalado

- Escalado reactivo:
- Escalado predictivo:

8. El teorema de CAP

8.1. Sistemas AP

8.2. Sistemas CP

8.3. Balance entre sistemas AP y sistemas CP

9. Descubrimiento de servicios

9.1. DNS

9.2. Herramientas para el descubrimiento de servicios

- Zookeeper
- Consul
- Eureka

10. Documentación de servicios

- Swagger
- HAL

11. Descripción de un servicio