Sistemas Distribuidos

Práctica 4 : Servicio de gestión de vistas

Autor: Unai Arronategui

Resumen

En esta práctica se introduce la tolerancia a fallos para nodos distribuidos con estado. El objetivo de la práctica es construir un servicio de almacenamiento clave/valor tolerante a fallos utilizando replicación Primario/Copia en memoria RAM. Esta pŕactica es la primera parte de una aplicación global que se completará con la práctica 5 (que dependerá de esta).

En esta 4ª práctica, se diseñará e implementará el servicio de gestión de vistas; y, en la práctica 5, el servicio clave/valor utilizando el servicio anterior. Es interesante señalar que el esquema planteado sigue presentando posibles fallos, como la posible caída del gestor de vistas.

Los protocolos a diseñar son los correspondientes al esquema de funcionamiento Primario/Copia planteado como ejemplo en la clase de teoría de replicación con estado.

Estas prácticas incluyen redactar una memoria, escribir código fuente y elaborar un juego de pruebas. El texto de la memoria y el código deben ser originales. Copiar supone un cero en la nota de prácticas.

Notas sobre esta práctica

- Aplicar "go fmt.ªl código fuente. Además : fijar en el editor máxima longitud de línea de 80 columnas, como mucho 15 instrucciones en una función (salvo situaciones especiales con bloques switch con multiples casos). Existen diferentes posibilidades de editores con coloración sintáctica: vscode, gedit, geany, sublimetext, gvim, vim, ...
- La solución aportada deberá funcionar para diferentes tests.

1. Objetivo de la práctica

Los objetivos de la práctica son los siguientes:

- Presentar una solución de tolerancia a fallos con estado como es el sistema de replicación Primario/Copia.
- Implementar el servicio de gestión de vistas de Primario/Copia para que sea capaz de recuperarse ante el fallo de una máquina.

2. El servicio de gestión de vistas

Se desea plantear un sistema de tolerancia frente a fallos basado en el servicio de vistas planteado en clase como ejemplo de gestión de réplicas Primario/Copia. El gestor de vistas no estará replicado para mayor sencillez, lo que implica que no se plantea una solución completa de tolerancia a fallos como la que obtiene mediante algoritmos de consenso como Raft, Paxos o Zab.

2.1. Protocolo del gestor de vistas

En el esquema general presentado en clase, cuando el primario falla el gestor de vistas promocionará la copia como primario. Si un nodo copia falla o es promocionado, y hay disponible un nodo en espera, el servicio de gestión de vistas promocionará este último como nodo copia.

El gestor de vistas gestionará una secuencia de "vistas" numeradas, cada una con una estructura de 3 valores, {nº de vista, direccion completa nodo primario, dirección completa nodo copia}, válido para cada vista.

El primario en una vista debe ser siempre el primario o una copia de la vista previa, para asegurar la preservación del estado del servicio clave/valor. *Excepcionalmente*, al inicio del servicio de vistas, el gestor debe admitir cualquier servidor como primario. Un nodo copia puede ser cualquier servidor, diferente al primario, o puede no existir (string vacio "") en ciertos momentos (problema de disponibilidad de servicio).

Cada servidor clave/valor (primario, copia y servidores en espera) debe enviar un latido como mensaje periódico, *como mucho*, cada *@intervalo_latido* (50 ms, por ejemplo), al gestor de vistas para notificar que está vivo. Puede haber latidos más frecuentes, con menos tiempo entre ellas (como ocurre en algunas situaciones del código de pruebas).

En dicho latido también se incluye el *nº de vista más reciente* conocida por el servidor primario y el servidor copia (*en esta práctica serán clientes del servidor de gestión de vistas*). El servidor de vistas les responderá con la descripción de la vista tentativa más reciente del gestor de vistas. Si el gestor de vistas no recibe ningún latido de corazón de alguno de los servidores clave/valor (o varios) durante un nº *@latidos_fallidos* de *@intervalo_latidos*, el gestor de vistas los considera caídos. Cuando uno de estos servidores rearranca después de una caida, debe enviar uno o más latidos con argumento *cero* para informarle al gestor de vistas que ha caido. A partir de aquí, se convertirá en un servidor en espera. Enviar latido(0, mi direccion de nodo) dos veces consecutivas se consideran como caídas consecutivas.

El gestor de vistas crea una nueva vista si :

- En la inicialización, cuando se incorporan los primeros servidores como primario y copia.
- No ha recibido un latido del primario o de la copia durante un nº *@latidos_fallidos* de *@intervalo_latidos*.
- Ha caido el primario o la copia, y se han perdido sus datos. Quizás, ha rearrancado a continuación, sin que el gestor de vistas haya detectado su fallo.
- Solo está vivo el primario y aparece un nuevo nodo.

El gestor de vistas manejará dos vistas diferentes, **la vista válida y la vista tentativa**. La vista válida es la única que proveerá a los clientes del servicio de almacenamiento (no a los clientes del servicio de gestión de vistas, que son los que utilizamos en esta práctica) para indicarles que ya hay un servidor Primario. Si el gestor de vistas detecta caídas de Primario y/o Copia, cuando reciba los latidos del resto de servidores les responderá con la vista tentativa, que no pasará a ser la válida hasta que sea confirmada por el primario en uno de sus latidos. Además, también se evita que los clientes del sistema de almacenamiento accedan al nuevo Primario, antes de que éste haya efectuado la copia completa de sus datos clave/valor a la nueva Copia (pero esto se tratará en la segunda parte). O de la copia correcta al nuevo primario. Durante este periodo, en el que la vista tentativa no coincide con la vista válida, los clientes del *sistema de almacenamiento* NO podrán acceder al servicio clave/valor.

Un ejemplo de diagrama de secuencia de cambios de vista se puede observar en la Figura 1. Por simplicidad, se ha omitido el servidor S3 que actúa como servidor en espera.

Como se puede observar en la figura, existen dos tipos de mensajes:

- Tipo mensaje de latido : {numVistaValidaConocida, direccion completa nodoEmisor}.
- Tipo mensaje de comunicación de vista. El mensaje consiste en una structura de la forma {NumVista, direccion completa Primario, direccion completa Copia}, donde

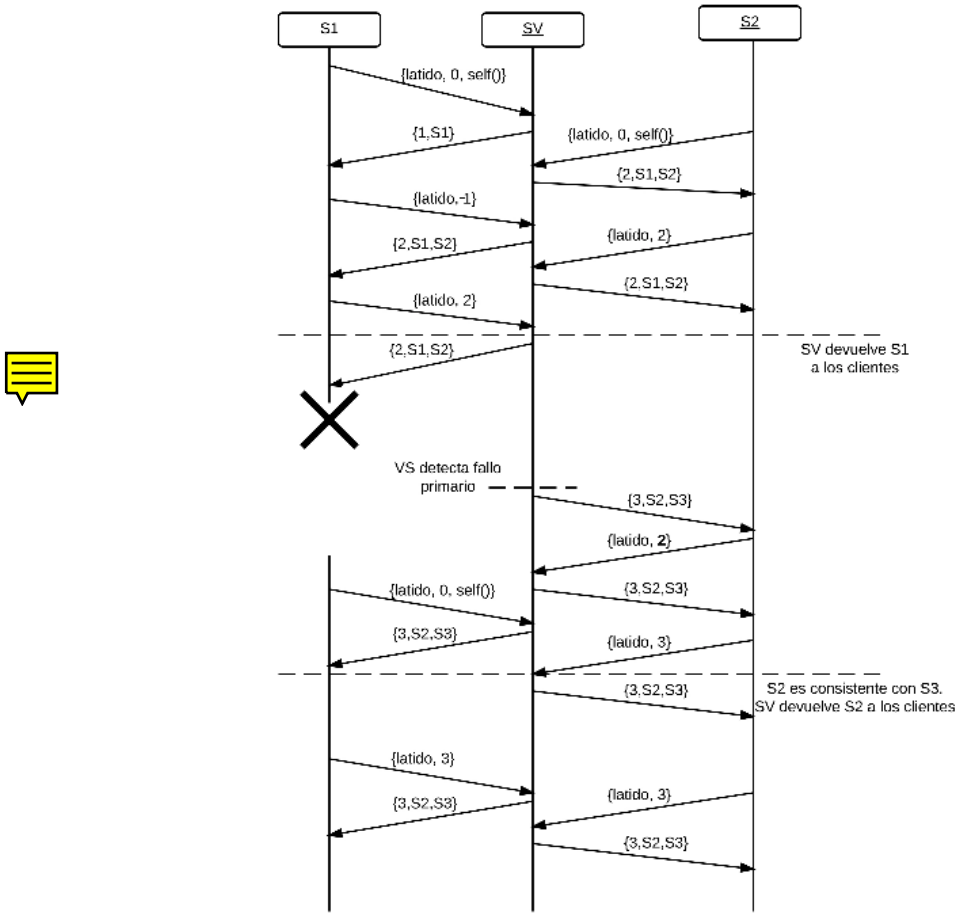

Figura 1: Diagrama de secuencia de inicialización y de recuperación de caída de nodos en el protocolo Primario/Copia

> NumVista es el nº de vista tentativa para el servidor de vistas, Primario es el servidor designado como primario en el sistema, y Copia es un servidor designado como Copia. En momentos específicos, la copia puede no estar definida (string vacio “”), como ocurre con el primario al comienzo de las operaciones del servicio.

Observando el diagrama de secuencia se puede ver que el servidor de vistas construye una nueva vista antes de que se le haya confirmado la anterior (sucede entre las vistas 2 y 3). Esto se debe a que el servidor que ha sido designado primario está realizando las tareas de copia necesarias para ser consistente con un nuevo servidor de Copia. Durante este proceso, el servidor responderá a los clientes con el servidor primario antiguo hasta que reciba el latido correspondiente a la vista 3 del primario (S2).

Además, el gestor de vistas puede recibir, también, el tipo de *mensaje de petición de primario {direccion completa del emisor}*, de un cliente del servicio de almacenamiento

para conocer quien es el primario válido del sistema de réplicas.

Algunas consideraciones :

- Al inicio, trás la recepción de la primera vista tentativa, el siguiente latido del primario es con *nº de vista = -1*, como situación especial, para no tener que reenviar un 0 (no queremos notificar una recaida) ni 1 (no queremos validar aún la vista).

- Es recomendable hacer un seguimiento del reconocimiento del primario de la vista tentativa para convertirla en vista válida.

- El servicio de vistas necesita tomar decisiones periódicas, por ejemplo, para promocionar un servidor copia a primario si el servicio de vistas no ha recibido un nº *@latidos_fallidos* de látidos no recibidos. El código que lo gestiona debería estar ubicado en una función que es llamada una vez cada *@intervalo_latido*.

- Puede haber más de 2 servidores enviando latidos. Los servidores extra (trás asignar primario y copia) son servidores en espera que están a la expectativa de convertirse en servidor copia si fuera necesario, y que el servidor de vistas debe gestionar.

- Tal como se ha comentado más arriba, tener en cuenta que el primario y/o copia pueden haber caido y rearrancar de forma rápida sin perder latidos, *pero perdiendo los datos almacenados en RAM !* El servicio de vistas tiene que darse cuenta de esa caida rápida, es decir, no os olvideis de manejar el latido(0, mi direccion completa de nodo) para notificar caida con pérdida de datos.

- También puede ocurrir que, por problemas de red (particiones de red), los latidos continuos no se reciban en el servidor de vistas; y este estime caído al cliente. Pero más tarde, la red funciona bien otra vez y los latidos llegan al servidor de vistas, no con latido(0), sino con el valor de vista normal en que habia quedado el cliente del gestor de vistas.

- Si cae el primario mientras la vista tentativa es diferente a la vista válida, entonces se han perdido los datos de todas las réplicas y *el sistema debe parar con un fallo crítico* !!!

2.2. Ejercicio

Se pide implementar el servicio de gestión de vistas presentado en la sección anterior, de tal manera que el sistema sea capaz de recuperarse ante el fallo de una réplica. Cuando el primario o la copia fallen, será el gestor de vistas el encargado de reemplazar la configuración con otro nodo operativo.

3. Notas sobre diseño e implementación en Golang

Se ponen a disposición de los alumnos un esqueleto de modulo con los directorios y paquetes que deberían completarse tanto para código cómun, código de servidor de gestor de vistas, código de cliente se servidor de vistas y coódigo de test de integración.

Tener en consideración que el paquete cliente implementa la funcionalidad de cliente del servidor de gestión de vistas, es decir, **lo que en la práctica 4 incorporareis a los servidores clave/valor como *una parte de su funcionalidad local interna***.

La puesta en marcha procesos servidor y cliente a través de *ssh* implica la utilización de la herramienta ssh SIN contraseña *con clave pública*. Verificar, previamente, que podeis conectaros con ssh sin contraseña sin interrupción, antes de lanzar una ejecución con el código que desarrolleis.

Los nombres de directorios, en el camino de acceso a vuestro código Golang, no deben contener el carácter espacio ni otros caracteres que dificulten el acceso a los ficheros fuente para la ejecución de procesos remotos.

3.1. Validación

Para el desarrollo inicial, se puede trabajar en la máquina local, pero para la validación final debe ejecutarse cada servidor en una máquina física diferente.

Se plantean las siguientes pruebas a superar y desarrollar test especificos para cada una de ellas :

1. No deberia haber primario (antes de tiempo).
2. Hay un primer primario correcto.
3. Hay un nodo copia.
4. Copia toma relevo si primario falla.
5. Servidor rearrancado se convierte en copia.
6. Servidor en espera se convierte en copia si primario falla.
7. Primario rearrancado es tratado como caido y convertido en nodo en espera.
8. Servidor de vistas espera a que primario confirme vista, pero este no lo hace.

9. Si anteriores servidores caen, un nuevo servidor sin inicializar no puede convertirse en primario.

La superación de las pruebas 1 a 5 supone la obtención de una B en la parte correspondiente a test. Para obtener una calificación de A, se deberá superar la prueba 6 y 7. La superación de los test 8 y 9 supone tener una calificación de A+. Para llevar a cabo esta implementación, se recomienda basarse en el código disponible.

Se deberán implementar las pruebas, cuyos comentarios ampliados estan disponibles en ese mismo fichero.

Los tests 8 y 9 necesitan de configuraciones de vistas especificas para su funcionamiento. Utilizar los látidos de forma adecuada para definir el contexto necesario para dichas configuraciones.

Podeis utilizar GoVector y Shiviz para para depurar la ejecución distribuida.

4. Evaluación

La realización de las prácticas es por parejas, pero los dos componentes de la pareja *deberán entregarla de forma individual*. En general, estos son los criterios de evaluación:

- Deben entregarse todos los programas, se valorará de forma negativa que falte algún programa / alguna funcionalidad.
- Los programas no tendrán problemas de compilación, se valorará de forma muy negativa que no compile algún programa.
- Todos los programas deben funcionar correctamente como se especifica en el problema través de la ejecución de la bateria de pruebas.
- Todos los programas tienen que seguir la guía de estilo de codificación de go fmt.
- Se valorará negativamente una inadecuada estructuración de la memoria, así como la inclusión de errores gramáticales u ortográficos.
- La memoria debería incluir diagramas de máquinas de estado y/o diagramas de secuencia para explicar los protocolos de intercambio de mensajes y los eventos de fallo.
- Cada uno de los servidores debe ejecutarse en una máquina física diferente en la prueba de evaluación.

4.1. Rúbrica

Con el objetivo de que, tanto los profesores como los estudiantes de esta asignatura por igual, puedan tener unos criterios de evaluación objetivos y justos, se propone la siguiente rúbrica en el Cuadro 1. Los valores de las celdas son los valores mínimos que hay que alcanzar para conseguir la calificación correspondiente y tienen el siguiente significado:

- A+ (excelente). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. Plantea *correctamente* el problema a partir del enunciado propuesto e identifica las opciones para su resolución. Aplica el método de resolución adecuado e identifica la corrección de la solución, *sin errores*. En el caso de la memoria, se valorará una estructura y una presentación adecuadas, la corrección del lenguaje así como el contenido explica de forma precisa los conceptos involucrados en la práctica. En el caso del código, este se ajusta exactamente a las guías de estilo propuestas.
- A (bueno). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. Plantea *correctamente* el problema a partir del enunciado propuesto e identifica las opciones para su resolución. Aplica el método de resolución adecuado e identifica la corrección de la solución, *con ciertos errores* no graves. Por ejemplo, algunos pequeños casos (marginales) no se contemplan o no funionan correctamente. En el caso del código, este se ajusta *casi* exactamente a las guías de estilo propuestas.
- B (suficiente). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. No plantea correctamente el problema a partir del enunciado propuesto y/o no identifica las opciones para su resolución. No aplica el método de resolución adecuado y / o identifica la corrección de la solución, pero *con errores*. En el caso de la memoria, bien la estructura y / o la presentación son mejorables, el lenguaje presenta deficiencias y / o el contenido no explica de forma precisa los conceptos importantes involucrados en la práctica. En el caso del código, este se ajusta a las guías de estilo propuestas, pero es mejorable.
- B- (suficiente, con deficiencias). En el caso de software, conoce y utiliza de forma autónoma y correcta las herramientas, instrumentos y aplicativos software necesarios para el desarrollo de la práctica. No plantea correctamente el problema a partir del enunciado propuesto y/o no identifica las opciones para su resolución. No se aplica el método de resolución adecuado y/o se identifica la corrección de la solución, pero *con errores* de cierta gravedad y/o sin proporcionar una solución completa. En el caso de la memoria, bien la estructura y / o la presentación son *manifiestamente* mejorables, el lenguaje presenta *serias* deficiencias y / o el contenido no explica de forma precisa los conceptos importantes involucrados en la práctica. En el caso del código, hay que mejorarlo para que se ajuste a las guías de estilo propuestas.

- C (deficiente). El software no compila o presenta errores graves. La memoria no presenta una estructura coherente y/o el lenguaje utilizado es pobre y/o contiene errores gramaticales y/o ortográficos. En el caso del código, este no se ajusta exactamente a las guías de estilo propuestas.

Calificación	Sistema	Tests	Código	Memoria
10	A+	A+ (test 1-9)	A+	A+
9	A+	A+ (test 1-9)	A	A
8	A	A (test 1-7)	A	A
7	A	A (test 1-7)	B	B
6	B	B (test 1-5)	B	B
5	B-	B-(test 1-4)	B-	B-
suspenso	1 C			

Cuadro 1: Detalle de la rúbrica: los valores denotan valores mínimos que al menos se deben alcanzar para obtener la calificación correspondiente

5. Entrega y evaluación

Cada alumno debe entregar un solo fichero en formato tar.gz o zip, a través de moodle en la actividad habilitada a tal efecto, no más tarde del día anterior a la siguiente sesión de prácticas (b5)

La entrega DEBE contener los diferentes ficheros de código Elixir y la memoria (con un máximo de 6 páginas la memoria principal y 10 más para anexos), en formato pdf. El nombre del fichero tar.gz debe indicar apellidos del alumno y nº de práctica. Aquellos alumnos que no entreguen la práctica no serán calificados. La evaluación "in situ" de la práctica se realizará durante la 5ª sesión de prácticas correspondiente.