Sistemas Distribuidos

Práctica 5: Servicio de almacenamiento replicado basado en primario/copia

Autor: Unai Arronategui

Resumen

En esta práctica se plantea desarrollar un servicio de almacenamiento distribuido clave/valor en memoria RAM, tolerante a fallos, utilizando el servicio de vistas implementado en la Práctica 4. Las operaciones que se pondrán a disponsición de los clientes serán: lectura de valor y escritura de valor. Los protocolos a diseñar son los correspondientes al esquema de funcionamiento Primario/Copia planteado como ejemplo en la clase de teoría. Se plantearan situaciones de fallo adicionales, como por ejemplo, el envío de escrituras duplicadas o la pérdida de mensajes.

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 los diferentes tests. Tener cuidado con los errores ligados a las temporizaciones (retardos ejecución y comunicación, y tiempos de expiración), son complicados de resolver.

1. Objetivo de la práctica

El objetivo de esta práctica es diseñar e implementar un servicio de almacenamiento distribuido clave/valor, en memoria RAM, que sea tolerante a fallos mediante el esquema Primario/Copia planteado en clases de teoría, y apoyandose en la implementación del servicio de vistas de la práctica 4.

2. El servicio de almacenamiento distribuido clave/valor basado en aproximación primario/copia

La Práctica 5 consiste en utilizar las indicaciones definidas en clase para el ejemplo de servicio de replicación primario/copia. En términos generales, el objetivo es diseñar e implementar tanto el protocolo cliente del servicio clave/valor como el protocolo de replicación entre primario y copia.

Se definen tres operaciones clave/valor que los clientes del servicio pueden utilizar :

- *lee(clave)* : devuelve el valor asociado a la clave, o cadena vacia (““) si no existe la clave.
- *escribe(clave, nuevo_valor)* : Actualizar valor asociado a *clave* con *nuevo_valor*. Si no existe clave, añade pareja (clave, nuevo_valor). Devuelve *nuevo_valor*.

  Tanto las claves como los valores son de tipo *string*.

Se aconseja la utilización de diagramas de máquina de estados, diagramas de interacción y diagramas de secuencia para modelar las interacciones entre clientes y servidores de almacenamiento, y para la interacción entre los diferentes servidores de almacenamiento con el protocolo de replicación.

Se debe gestionar la semántica “solo una vez” de la peticiones con reintentos y detección de duplicados (mediante secuenciación de peticiones). Los mensajes de propagación de peticiones de clientes entre primario y copia deben ser específicas para distinguirlas de las peticiones cliente al primario (por si se equivoca y lo envía al servidor copia). La transferencia completa de la base de datos clave/valor a un nuevo nodo copia se puede realizar en un solo envío. El estado relacionado con detección de duplicaciones debe ser también replicado correctamente entre el primario y copia para tolerar correctamente los fallos.

Para el desarrollo de la solución, se indican las siguientes recomendaciones :

- En primer lugar, recordar que cada servidor de almacenamiento debería enviar latidos al servicio de vistas y recibir su respuesta, para dar señales de vida y conocer la vista tentativa. Una vez que un servidor de almacenamiento conoce la vista tentativa, puede saber si es primario, copia o espera.

- Después, implementación del procesado ligado a la recepción de las operaciones *lee()* y *escribe()*. Para ello, se pueden almacenar las parejas clave/valor en el primario y la copia con estructura de datos tipo *map* de Golang. Suponemos que el tamaño del almacen es pequeño (pocos datos) y por lo tanto que supone un retardo muy pequeño en las operaciones de copia completa de almacen entre nodos primario y copia.

- Posteriormente, codificación de la operación escribe para que el primario solicite confirmación de actualización al servidor copia antes de responder al cliente.

- Implementación de la copia de la base de datos clave/valor, por parte del primario, cuando un servidor se convierte en copia en una nueva vista.

Se debe tener en consideración que, aunque el servicio de vistas de la Práctica 4 haya pasado las pruebas, pueden quedar errores, en dicho programa, que provoquen fallos en esta práctica.

2.1. Notas sobre diseño e implementación en Golang

Para la base de datos clave/valor en RAM se puede utilizar la estructura de datos *map* que teneis disponible en Golang. Para poder introducir aleatoriedad en los fallos, se puede utilizar las funciones *rand.Seed y rand.Intn* del paquete "math/rand" de la librería estándar de Golang.

El código cliente de servbicio de vistas, de la práctica 4, se puede integrar directamente en el servidor de almacenamiento de la práctica 5. Esto es debido a que este código será utilizado directamente, por cada servidor de almacenamiento (para latidos) y cada cliente del servicio de almacenamiento (para petición de primario), para interaccionar con el servidor de vistas.

Se aconseja aprovechar al máximo el código del servidor/gestor de vistas y del cliente de la práctica 4 para optimizar el tiempo dedicado a los aspectos específicos de esta práctica.

2.2. Validación

Podeis utilizar partes de código de la versión V2 del esqueleto de la práctica 4, en la prática 5. Por ejemplo, la puesta en marcha y la parada de los diferentes servidores con ssh (gestor de vistas, servidores de almacenamiento y clientes de almacenamiento), en máquina local y máquinas distribuidas, para la preparación de escenario en inicio y limpieza al final de cada test.

Se debe tener en cuenta que ejecuciones inacabadas de test (por errores) de integración pueden dejar procesos "main"sin morir. Esto puede interferir en validaciones posteriores. Podeis utilizar comandos "pkill main"hasta eliminar todos estos procesos de sistema antes de volver a ejecutar una validación. Podeis aplicarlo, también, en las máquinas remotas para validaciones distribuidas (mediante programa shell y ssh).

En la fase final de desarrollo y validación de vuestra solución, cada servidor de almacenamiento debe estar en una máquina física diferente al resto y ejecutarse de forma distribuida. En cambio el nodo test , el servidor gestor de vistas y los clientes del servicio de almacenamiento pueden ser ubicados en la misma máquina física desde donde se inicia la aplicación (máquina si fallos). En principio, como máximo necesitareis 4 máquinas físicas : 1 máquina para nodo test, gestor de vistas y varios clientes de almacenamiento (1 o varios clientes según necesidades de la prueba) y 3 máquinas más, cada una para un servidor almacenamiento diferente.

Tener cuidado con la variaciones de latencia que pueden ocurrir en las diferentes configuraciones que se prueben. Pueden dar errores complicados a detectar y resolver. Los **tiempos de expiración** (timeouts) y **retardos** en la ejecución de algunas operaciones deben adecuarse correctamente mediante *Process.sleep(milisegundos)*, comunicaciones con timeout o en timers.

Se pide implementar los siguientes pruebas como tests:

1. Arranque correcto de máquinas.
2. La operación de escritura se realiza correctamente
3. Escrituras concurrentes y comprobación de consistencia tras caída de primario.
4. Escrituras concurrentes y comprobación de consistencia tras caída de primario y copia. Se puede gestionar con cuatro nodos o con el primario rearrancado.
5. Petición de escritura inmediatamente después de la caída de nodo copia (con uno en espera que le reemplace).
6. Petición de escritura duplicada por perdida de respuesta (modificación realizada en BD), con primario y copia.

7. Comprobación de que un antiguo primario no debería servir operaciones de lectura.

La superación de las pruebas 1 a 3 supone la obtención de una nota B en la parte correspondiente a test. Para obtener una calificación de A, se debera superar las pruebas 4 y 5. La superación de los test 6 y 7 supone tener una calificación de A+. Para llevar a cabo esta implementación, se recomienda basarse en el código disponible.

3. Criterios de 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.
- Todos los programas deben funcionar correctamente como se especifica en el problema a través de la ejecución de la bateria de pruebas.
- Todos los programas tienen que seguir la guía de estilo de codificación "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.
- Cada uno de los servidores de almacenamiento debe ejecutarse en una máquina física diferente.

3.1. 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, en el Cuadro 1 se propone una rúbrica. Los valores de las celdas indican el mínimo que hay que alcanzar para conseguir la calificación correspondiente. Su significado es el siguiente:

- 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

Calificación	Sistema	Tests	Código	Memoria
10	A+	A+ (1-7)	A+	A+
9	A+	A+ (1-7)	A	A
8	A	A (1-5)	A	A
7	A	A (1-5)	B	B
6	B	B (1-3)	B	B
5	B-	B-	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

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.

4. Entrega y Defensa

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).

Se debe entregar, tanto la memoria (longitud máxima de 7 páginas para memoria y 10 para anexos), en formato pdf, como los ficheros de código fuente, en un solo fichero en formato tar.gz a través de moodle2 en la actividad habilitada a tal efecto. El nombre del fichero tar.gz debe indicar apellidos del alumno y nº de práctica. La entrega debe efectuarse, no más tarde del día anterior a la siguiente sesión de prácticas (6). 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.