Sistemas Distribuidos

Práctica 5, 1ª parte: Servicio de gestión de vistas

Autor: Unai Arronategui y Víctor Medel

Resumen

En esta práctica se introduce la tolerancia a fallos para nodos distribuidos con estado. El objetivo de la practica es construir un servicio clave/valor tolerante a fallos utilizando replicación Primario/Copia en memoria RAM. La práctica se divide en 2 partes; la primera, se desarrollará durane la sesión 5 y la segunda (que depende de la primera) en la sesión 6. Cada parte será entregada por separado tal como se espcifica al final del guión.

En la primera parte, se diseñará e implementará el servicio de gestión de vistas; y, en la segunda, el servicio clave/valor utilizando el servicio anterior. En interesante señalar que el esquema presentado 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.

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

- Seguir, en lo posible, la guía de estilo de codificación Erlang, en especial : fijar en el editor máxima longitud de línea de 80 columnas, como mucho 12 expresiones en una función (alvo situaciones especiales con bloques case y receive largos). Además, se deberá seguir completamente la sección 7 de la guía de estilo (`http://www.erlang.se/doc/programming_rules.shtml`). Existen diferentes posiblidades de editores con coloración sintáctica: geany, gedit, vim, emacs, sublimetext.

- La solución aportada deberá funcionar para los diferentes tests suministrados y los que podáis proponer.

1. Objetivo de la práctica

El objetivo de esta práctica es diseñar e implementar un servicio de gestión de vistas.

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/Bakcup. El gestor de vistas que lo gestiona no estará replicado para mayor sencillez (No planteamos solución completa de tolerancia a fallos como mediante algoritmos de consenso como Raft, Paxos o Zab).

El gestor de vistas gestionará una secuencia de "vistas" numeradas, cada una con el trio {nº de vista, identificador nodo primario, identificador nodo copia}; válido para cada vista.

El primario en una vista debe ser siempre el primario o el 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. El nodo copia puede ser cualquier servidor, diferente al primario, o puede no existir en ciertos momentos.

Cada servidor clave/valor (primario, copia y servidores en espera) debe enviar un latido como mensaje periódico, cada 100 ms, al gestor de vistas para notificar que está vivo (como latido de corazón). En dicho latido también se incluye el nº de vista más reciente conocida por el servidor primario y copia. El servidor de vistas les responderá con la descripción de la vista válida más reciente del gestor de vistas. Si el gestor de vistas no recibe ningún latido de corazón de alguno de los dos servidores clave/valor (o los dos), el gestor de vistas los considera caídos. Cuando uno de estos servidores rearranca después de una caida, debe enviar uno o mas pings con argumento cero para informarle al gestor de vistas que ha caido. (Enviar ping(0) 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 el periodo de fijado

El gestor de vistas manejará dos vistas diferentes, la vista válida y la vista tentativa. La vista válida es la única que provee a los clientes del servicio de almacenamiento (no del servicio de gestión de vistas) para indicarles que ya hay un primario y una copia completamente operativos. Si el gestor de vistas detecta una caída para el Primario y/o el 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 accedan al nuevo Primario, antes de que éste haya efectuado la copia completa de sus datos clave/valor al nuevo Copia.

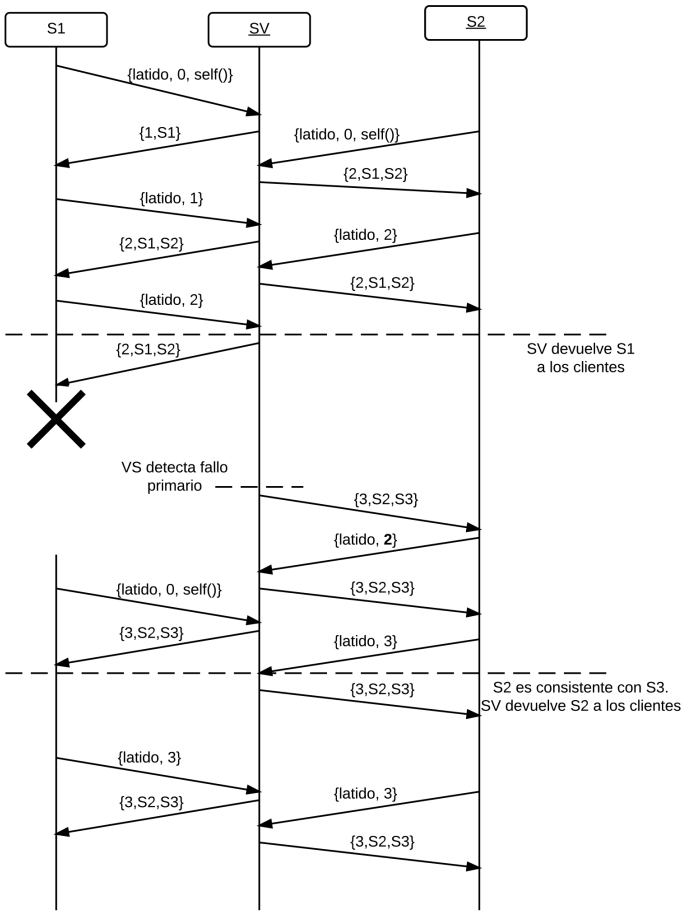

Figura 1: Diagrama de secuencia de inicialización y de recuperación de caída de nodos en el protocolo Primario/Copia

Durante este periodo, en el que la vista tentativa no coincide con la vista válida, los clientes 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:

- Mensaje de latido (ping).
- Mensaje de comunicación de vista. El mensaje consiste en una tupla de la forma {NumVista, Primario, Copia}, donde NumVista es la vista tentativa por el servidor de vistas, Primario es el servidor designado como primario en el sistema, y Copia es el servidor designado como Copia. Éste último puede no existir, si se trata de una situación de inicialización.

Observando el diagrama de secuencia se puede ver que el servidor de vista 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 el 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 tres del primario (S2).

Puede haber más de dos servidores clave/valor enviando latidos al gestor de vistas (es decir, no sólo el primario y la copia). Los servidores de más serán gestionados como servidores en espera por el gestor de vistas y serán utilizados como nodo copia si éste falta.

Algunas consideraciones :

- Es conveniente definir la forma de guardar el latido más reciente para cada servidor.
- Se pueden añadir campos al registro para conocer la vista válida.
- Podría tener interés hacerse 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 *PINGS_FALLIDOS* de látidos (pings) no recibidos. El código que lo gestiona debería estar ubicado en la función *procesar_situacion_servidores()* del fichero *servidor.erl*, que es llamada una vez cada *INTERVALO_PING*.
- 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 necesrio.
- Tal como se ha comentado más arriba, tener en cuenta que el primario o la copia pueden haber caido y rearrancar de forma rápida sin perder latidos. Pero el servicio de vistas tiene que darse cuenta de esa caida rápida, es decir, no os olvideis de manejar el ping(0).
- También puede ocurrir que, por problemas de red, los pings continuos no se reciban en el servidor de vistas; y este estime caido al cliente. Pero más tarde, la red funciona bien otra vez y los pings llegan al servidor de vistas, no con ping(0), sino con el valor de vista normal en que habia quedado el cliente.
- Estudiar los casos de prueba en el código que se os ha entregado. Si una de las pruebas falla, quizás os venga bien echar un vistazo al código fuente de prueba que teneis en el fichero *servicio_vistas_tests.erl*, para descubrir cómo es el escenario de error de ejecución.

2.1. Notas sobre diseño e implementación en Erlang

Se ponen a disposición de los alumnos 2 módulos completamente implementados: cliente, comun y tipo abstracto vista; junto a 2 módulos parcialmente implementados : servidor y los tests del servicio vistas. Se deberán estudiar y completar.

Además, se ha puesto a disposición un fichero de macros comunes (sv.hrl) cuya inclusión se invoca en los modulos previamente comentados.

La puesta en marcha de nodos Erlang a través de la librería *slave* 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 se os ha puesto a disposición.

2.2. Validación

Se provee el fichero de validación erlang *servicio_vistas_tests.erl*, basado en la infrasestructura *Eunit* de Erlang, y un par de ficheros shell, *validar_servicio_vistas.sh y compilar_clave_valor.sh*, para lanzar la compilación y ejecución de las pruebas.

En el fichero *servicio_vistas_tests.erl* se incluyen varias pruebas ya implementadas, a modo de ejemplo. Se debe implementar una amplicación del juego de pruebas disponibles, dentro este fichero, con algunas pruebas adicionales que se ejecutan en secuencia para aprovechar el estado de los nodos del test previo. Estas pruebas incluyen:

1. Copia toma el relevo si primario falla.
2. Servidor rearrancado se convierte en copia.
3. 3er servidor en espera se convierte en copia si primario falla.
4. Primario rearrancado es tratado como caido.
5. Servidor de vistas espera a que primario confirme vista, pero este no lo hace.
6. Si anteriores servidores caen, un nuevo servidor sin inicializar no puede convertirse en primario.

Para llevar a cabo esta implementación, se recomienda basarse en el código disponible.

Para ayudaros en la depuración, si ejecutais vuestras pruebas mediante el entrono Eunit, tener en cuenta que en lugar de *io:format()* para las trazas, es más aconsejable utilizar la macro *debugFmt()* disponible en fichero *eunit.hrl*. Esta macro maneja el mismo estilo de parámetros que io:format.

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 compilar correctamente, se valorará de forma muy negativa que no compile algún programa.

- Todos los programas deben funcionar correctamente como se especifica en el problemaa través de la ejecución de la bateria de pruebas.
- Todos los programas tienen que seguir la guía de estilo de codificación Erlang disponible en `http://www.erlang.se/doc/programming_rules.shtml`
- Se valorará negativamente una inadecuada estructuración de la memoria, así como la inclusión de errores gramáticales u ortográficos.

```
%% AUTOR: nombre y apellidos
%% NIA: n'umero de identificaci'on del alumno
%% FICHERO: nombre del fichero
%% TIEMPO: tiempo en horas de codificaci'on
%% DESCRIPCI'ON: breve descripci'on del contenido del fichero
```

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, 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+	A+	A+
9	A+	A+	A	A
8	A	A	A	A
7	A	A	B	B
6	B	B	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

4. Entrega y Defensa

Se debe entregar un solo fichero en formato tar.gz a través de moodle2 en la actividad habilitada a tal efecto, no más tarde del jueves anterior al comienzo de la sexta sesión de prácticas. EL *17 de diciembre de 2015* para los grupos A y el *7 de enero de 2016* para los grupos B.

La entrega debe contener los ficheros de código Erlang y la memoria, en formato pdf.