MiniProyecto Sistemas Distribuidos : 2ª parte
Servicio de almacenamiento Clave/Valor

Autor: Unai Arronategui

Límite entrega memoria en moodle : **28 de noviembre de 2016**

Resumen

Ahora que, en la primera parte, ya tenemos construido el servicio Paxos, lo vamos a utilizar para construir el sistema de almacenamiento distribuido clave/valor. Paxos funciona, localmente en cada servidor clave/valor, a modo de librería (aunque opere con varios procesos Erlang).

El servicio clave/valor utiliza a Paxos para implementar un registro de operaciones lee/escribe. Cada instancia de Paxos se puede ver como un elemento de dicho registro secuencial, y el orden de operaciones en ese registro es el orden en el que todos los servidores clave/valor van aplicar las operaciones a su base de datos local. Paxos se encargará que los servidores (réplicas) se pongan d acuerdo en este orden.

Vuestro sistema de almacenamiento distribuido tendrá algunas limitaciones, que sería necesario abordar si quisierais construir un sistema serio. No abordamos el problema de caidas de máquinas, ya que no guardamos en disco ni la base de datos clave/valor ni el estado de los servidores Paxos. Además, el número de servidores es estático. Finalmente, el algoritmo básico de Paxos utilizado es lento. Se necesitan muchos mensajes para cada operación lee() o escribe().

Más adelante veremos, de forma más conscisa, mejores soluciones que se ha ido proponiendo y que se están utilizando en un buen número de sistemas industriales (Spanner, Zookeeper,etc).

*Este miniproyecto incluye **redactar** una **memoria y escribir todo el código fuente (salvo el propocionado en la asignatura)**. El texto de la memoria y código debe ser original y bien explicado. Se puede comentarsobre el miniproyecto con otros alumnos, pero no está permitido mirar o copiar el código de otros.*

Notas sobre esta práctica

Seguir, en lo posible, la guía de estilo de codificación Erlang, **en especial la sección 7 por completo** : http://www.erlang.se/doc/programming_rules.shtml

1. Objetivos

Los objetivo de esta 2ª parte del mini proyecto de sistemas distribuidos es el diseño e implementación de un servicio de almacenamiento, distribuido y tolerante a fallos, basado en Paxos.

2. Servicio de almacenamiento clave/valor

El sistema de almacenamiento distribuido a construir está constituido por un conjunto de servidores clave/valor que funcionarán a modo de réplicas. Cada servidor funcionará en un nodo erlang separado. Cada servidor Paxos se ejecutará en cada nodo erlang para dar servicio al servidor clave/valor local a dicho nodo Erlang.

Por lo que necesitareis modificar ligeramente vuestro código Paxos para que la función “start(...)” del modulo “paxos.erl”, en lugar de poner en marcha un nodo Erlang (*slave:start*), ponga en marcha un proceso Erlang **en el mismo nodo Erlang** del servidor clave/valor que lo ha invocado. Es decir, el nodo erlang será el mismo para el servidor clave/valor y su servidor Paxos. El resto de vuestro código de Paxos, se mantiene igual.

Los servidores/réplicas clave/valor deberían ser idénticos; con la única excepción de que algunas réplicas pueden retrasarse con rspecto a otras si no son alcanzables (problemas de red). Si una réplica no puede comunicarse durante un rato, más adelante, debería poder alcanzar al estado que tienen las réplicas operacionales (obteniendo las operaciones que le faltan a partir de estas réplicas).Un servidor clave/valor se pone en marcha con la ejecución de sus prpio nodo Erlang dedicado.

Los clientes del servicio clave/valor deberían intentar, cada operación, en diferentes réplicas que conocen hasta encontrar una que responda; y no terminar su ejecución hasta que uno de ellos responda . Una servidor/réplica, que forme parte de una mayoría de replicas que se puedan comunicar entre ellas, debería ser capaz de dar servicio a la petición del cliente. Cada cliente se ejecuta también en su propio nodo Erlang (como si fuera un mauina remota que solicita servicio de almacenamiento distribuido), por lo que inicialmente habría que poner en marcha un nodo Erlang nuevo para cada cliente.

Vuestro sistema de almacenamiento debe proveer consistencia secuencial a las aplicaciones que utilicen su interfaz de cliente. Es decir, las llamadas completadas a las funciones **cliente:lee(clave), cliente:escribe(clave, valor) y cliente:escribe_hash(clave,valor)** en el fichero **cliente.erl** deben afectar a todas las réplicas en el mismo orden y con semántica de llamada remota **como_máximo_una_sola_vez**. Un operación **cliente:lee()** debería ver el valor escrito por la operación **cliente:escribe()** más reciente en la secuencia con la misma clave, o la cadena de caracteres vacía (“”) si no ha habido ninguna operación de escritura previamente para esa clave (la clave no existe aún en la base de datos). Una consecuencia de esto es que se debe asegurar que cada llamada de la aplicación a **cliente:escribe()** debe aparecer en ese orden solo una vez (es decir, escribir en la base de datos clave/valor sólo una vez), incluso si, internamente, el **cliente.erl** haya enviado múltiples veces mensajes escribe y escribe_hash hasta que encuentra un servidor/réplica clave/valor que le responda.

La operación **escribe_hash(clave, nuevo_valor)** permite encadenar los valores sucesivos para una determinada clave, aspecto de interés los tests y funciona de la siguiente forma : almacena el resultado de aplicar hash(anterior_valor_de misma_clave_en_BD, nuevo_valor) en la misma base de datos clave/valor, y devuelve el anterior_valor.

Para la implementación de la librería Paxos teneis disponible un par de ficheros shell ejecutables (*erlang.sh, compilar_clave_valor.sh y validar.sh*), especificado de la segunda parte, para lanzar un shell de Erlang con las opciones adecuadas o para validar mediante tests vuestro código. Para utilizarlos en vuestra implementación, deberiais configurar la conexión *ssh* con clave pública(*authorized_keys*). También teneis disponible un esqueleto básico de modulo de servidor (***servidor.erl***), modulo cliente (***cliente.erl***), modulo común (**comun.erl**) y código de pruebas de validación (***servicio_clave_valor_tests.erl***) que utiliza funcionalidades del modulo EUnit. Cuando os funcione vuestra implementación, la ejecución de *validar.sh* debería mostrar que todos los tests han pasado con éxito.

Vuestra implementación debe proporcionar, al exterior y al modulo de text, el interfaz de programación definido en el esqueleto de los modulos servidor y cliente:

Una posible forma de abordar la solución podría ser la siguiente :

1. Añadir información de estado que necesiteis a la estructura de datos básica del modulo servidor, con la información del “valor” que se requerirá acuerdo en Paxos y que haya sido suministrada por cada petición/operación cliente. Es decir, la estructura de datos de operación cliente es el “valor” sobre el que se debe llegar a un acuerdo en una instancia Paxos entre réplicas. Por ejemplo, se debería pasar esas estrcuturas de datos de operaciones a **paxos:start_instancia()**. El paso de mensajes de estructuras de dtos se hace de forma transparente en Erlang, en cuento que las serialización/deserialización que realiza internamente de forma implicita en la VM de Erlang.
2. Implementar el tratamiento a la recepción del **mensaje *escribe*** en la función **servidor:escribe()** del fichero **servidor.erl**. Debería introducir un operación escribe en el registro de instancias de Paxos. Es decir, utilizando Paxos para asignar un instancia de acuerdo cuyo valor incluye la clave, el valor y la operación escribe. De este modo, el resto de servidores Paxos sabrán de esta operación escribe.
3. Implementar el tratamiento a la recepción del mensaje **escribe_hash(clave, nuevo_valor)** procesando el nuevo_valor con el anterior_valor utilizando la función **hash** que teneis disponible en el fichero **comun.erl**. El argumento para la función **hash** debería ser la concatenación de las cadenas de caracteres correspondientes al anterior_valor y al nuevo_valor provisto para la misma clave. Se debería convertir el resultado de tipo entero, devuelto por la función hash, a tipo string utilizando *integer_to_list(Valor_hash);* como podeis observar en los tests. Si la clave no existe, **escribe_hash** puede no hacer nada, aunque otra opción es tratar el anterior valor como la cadena vacía (“”). Salvo en el tipo de dato que actualizan, la operación escribe y escribe_hash deberían ser tratados de forma idéntica (como se sugiere en el esqueleto de código).
4. Implementar el tratamiento ligado a la recepción del mensaje lee. Debería introducir la estructura de datos de la operación lee en el registro de instancias Paxos; y posteriormente interpretar el registro de instancias antes de este punto para asegurarnos que la base de datos clave/valor refleja la ejecución de las operaciones escribe más recientes.

5. Añadir código para tratar el caso del envio y recepción de mensajes escribe ***duplicados*** desde cliente. Es decir, situaciones en las que mensajes de operación tipo escribe sea enviadas, a partir del cliente, en múltiples peticiones destinadas a múltiples réplicas calve/valor a causa de la detección de fallos en la respuesta de ellas. Cada operación escribe() y escribe_hash() debe ser ejecutada en una sola ocasión !

Algunas consideraciones :

- Cada servidor debería intentar asignar la próxima instancia Paxos disponible (número de secuencia) a cada mensaje/petición entrante de cliente. Sin embargo, otras réplicas clave/valor pueden estar también intentando utilizar esa misma instancia para otra petición de operación desde otros clientes. Es decir, que cada servidor debe estar preparado para intentar diferentes instancias.
- Los servidores calve/valor de deberían comunicarse directamente, Sólo deberían interaccionar entre ellos a través del registro de instancias de Paxos y su protocolo de consenso.
- Se deberían identificar, de forma única, cada una de las operaciones cliente para asegurarnos que solo se ejecutan una vez. Se puede asumir que cada cliente solo tiene una operación escribe o lee sin acabar al mismo tiempo.
- Un servidor clave/valor no debería completar una operación lee() que no es parte de una mayoría (para que no lea un dato antiguo no consistente). esto significa que cada operación lee(), al igual que cada operación escribe(), debe implicar un acuerdo de Paxos.
- No os olvideis de llamar a la función paxos:hecho() cuando un servidor/réplica clave/valor haya procesado la operación asociada a una instancia decidida de Paxos, y ya no la necesite ni ella ni ninguna de las instancias anteriores.
- Vuestro código necesitará esperar a que las instancias Paxos completen sus acuerdosl. La única manera de hacerlo es llamar, periódicamente a la función paxos:estado(), durmiendo (timer:sleep()) entre lllamadas. ¿Cuánto tiempo dormir ? Un buen plan puede ser de comprobar con rapidez al principio, y después mas lentamente. Podeis inspiraros del código utilizado en la función **paxos_tests:esperar_aun_mas_tiempo()** para este cometido. Ajustar el máximo tiempo (a través del máximo número de iteraciones) a un tiempo razonable de espera suficientemente largo para las pruebas (30 segundos ?).

- Si uno de los servidores clave/valor queda retrasado, al no particpar en el acuerdo de alguna que otra instanciia, necesitará encontrar el valor acordado (si es que existe) más adelante. Una forma razonable de hacer esto es llamar a paxos:start_instancia() para descubrir el valor acordado previamente o provocar que el acuerdo ocurra finalmente. Pensar en que valor sería razonable pasar a paxos:start_instancia en esta situación.

Aconsejable utilizar un editor de código que ayude en la edición de Erlang (Geany, Sublime Text,...)

Documentación Erlang :

- http://www.erlang.org/doc.html
- http://learnyousomeerlang.com/content
- “Erlang Programming A Concurrent Approach to Software Development”. Francesco Cesarini, Simon Thompson. 2009. O'Reilly.

1. Evaluación

Para la evaluación de la primera parte del miniproyecto, se debe entregar una memoria de su diseño e implementación y todo el código fuente necesario para ejecutar vuestro programa, el cual debera pasar los tests incluidos en el fichero ***servicio_clave_valor_tests.erl*** que se os ha puesto a disposición junto con el guión. Estos tests deberan ejecutarse con éxito en máquinas del laboratorio 1.02.

La entrega se realizará en un solo fichero comprimido en una sección de entrega para la 2ª parte que se habilitará en la página moodle de la asignatura.