Sistemas Distribuidos - 22/23

Práctica 3: Raft 1ª parte

Autor: Unai Arronategui

Resumen

En las prácticas siguientes, se plantea construir un servicio de almacenamiento clave/-valor, en memoria RAM, tolerante a fallos utilizando replicación distribuida basada en Raft, una solución de máquina de estados replicada mediante un algoritmo de consenso. En el caso de esta práctica 3, se darán los primeros pasos para la operativa del algoritmo de elección de líder requerido, con algún que otro tipo de fallo, y el tratamiento, sin fallos, de añadir entradas (llamada rpc appendEntry) nuevas en las réplicas. Las referencias son el tema 8 de teoría y el documento adjunto al guión. La especificación de estado de las réplicas y las llamadas RPC de referencia se encuentran disponibles en el anexo de este mismo guión.

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 20 instrucciones en una función** (salvo situaciones especiales justificadas). Existen diferentes posibilidades de editores con coloración sintáctica, acceso a definición de tipos variables y funciones, etc: vscode, gedit, geany, sublimetext, gvim, vim, ...
- La solución aportada deberá funcionar para diferentes pruebas.

1. Objetivo de la práctica

Los objetivos de la práctica son los siguientes:

- Presentar una implementación del algoritmo de elección de líder de Raft que funcione en escenarios iniciales de fallos básicos.
- Implementar el tratamiento de la llamada de RPC *AppendEntry*, suponiendo un funcionamiento sin fallos.

2. El servicio de elección de líder

Como primeros pasos en la solución completa de replicación distribuida, es necesario construir el algoritmo de elección de líder por votación explicado en clase de teoría y en el documento que se entrega asociado a este guión. En esta práctica se implementará, **únicamente**, la elección por votación mayoritaria (al menos la mitad más uno de las réplicas). **La restricción, adicional, de nº de mandato y nº de índice para seleccionar al mejor líder (sección 5.4.1 del documento asociado) será implementado en las prácticas siguientes**.

2.1. Funcionamiento

Para la implementación, como comportamiento de alto nivel, seguir la especificación de la máquina de estados de la página 29 del tema 8 que especifica el comportamiento básico de cada nodo con respecto al algoritmo de elección de líder que deberá estar ejecutándose de forma continua. Por otra parte, la especificación más concreta de las llamadas RPC básicas y el estado de cada réplica se encuentra en la figura 2 disponible al final de este guión. En particular, el funcionamiento global de la elección del líder está diseminada en diferentes partes de esa figura.

Añadir el estado, definido en la figura 2, en el struct NodoRaft definido en el fichero *raft.go*. También se necesitará otro struct que represente la información de cada entrada del registro de operaciones (logs) de Raft. Completar los structs *RequestVoteArgs*, para los argumentos de la llamada RPC RequestVote y *RequestVoteReply* para la respuesta de esa llamada.

Crear una gorutina concurrente que se responsabilice de la gestión del líder, y ponga en marcha un proceso de elección si no recibe mensajes de nadie durante un tiempo. De esta forma podrá saber quien es el líder, si ya lo hay, o convertirse el mismo en líder. Implementar el método asociado a la llamada RPC *RequestVote()* para solicitar votos en una elección.

Define los structs asociados a argumento y respuesta de la llamada RPC *AppendEn-*

tries(). Implementar el método asociado a la llamada RPC *AppendEntries()* para que el líder envíe latidos de corazón, de forma periódica, al resto de réplicas, notificando su buen estado operativo.

Aseguraros que los **tiempos de expiración**, asociados a la **recepción de latidos** y utilizados para la detección de fallo del líder, no expiren a la vez en diferentes réplicas (introducir variaciones aleatorias en rango limitado). Si no, los nodos votarán por si mismos y ninguno podrá convertirse en líder. Tenéis disponible el paquete *rand* de la librería estándar para generar valores aleatorios en intervalos.

La **frecuencia de latidos** no debe ser superior a 20 veces por segundo. Y el periodo en el se decide un nuevo líder, tras detectar el fallo del anterior, no debe ser superior a 2,5 segundos. Se recuerda que puede requerirse varias elecciones para llegar a una decisión, por no conseguir mayorías. Luego hay que añadir un **tiempo de expiración** para una **elección**, en caso de no conseguir mayoría. Y debe ser suficientemente corto, **relativo** al **periodo de latido**, para conseguir varias elecciones en el periodo de decisión de 2,5 segundos. Si hace falta, utilizar intervalos aleatorios.

Apoyaros en el paquete *timer* de la librería estándar, tal como comentamos en la dispositiva 6 de Çomplementos Go,..."para gestionar acciones periódicas o tras retrasos.

EL RPC de Go solo envía structs con todos los campos exportados, es decir que empiezan en mayúsculas. Los campos de las substructuras internas (campos de la estructura de entrada del registro de operaciones de Raft) deben comenzar también con mayúsculas. Si no, no funcionan las llamadas RPC.

3. Llamada RPC *AppendEntry* sin fallos

Implementar la llamada *AppendEntry* completa siguiendo las indicaciones de la figura 2 para conseguir enviarse y comprometer, con mayoría de réplicas, operaciones cliente en entradas de registro, **sin aplicar las entradas comprometidas a la máquina de estados** (eso se hará en práctica 4). Y probar dicha operativa, **únicamente, durante funcionamiento sin fallos en el conjunto del sistema, incluido el líder**.

4. Organización de código

Se aconseja el desarrollo del código y las pruebas en un ordenador Unix (Linux, BSDs, Mac, subsistema Linux de Windows). No se da soporte a desarrollo en Windows.

Implementar la funcionalidad de Raft, como algoritmo de consenso, en el fichero -aft/internal/raft/raft.go", donde ya disponéis de un esqueleto.

El conjunto del código reside en el *modulo raft*, bajo el cual encontráis diferentes paquetes y funcionalidades, en subdirectorios. Subdirectorio "*raft/cmd*" es utilizado para ubicar código ejecutable (func main). El subdirectorio "*raft/internal*" se utiliza para paquetes de uso interno al modulo. En el subdirectorio "*raft/pkg*" se ubican paquetes a exportar, que pueden ser utilizados por código externo al modulo. Y el subdirectorio "*raft/vendor*" contiene los paquetes importados por el código de este modulo, y es obtenido ejecutando "**go mod vendor**" en el directorio raíz del modulo.

La implementación del servicio Raft de consenso debe ofrecer el siguiente interfaz de llamadas y tipo dato:

```
// Crear nuevo nodo Raft
nr:= NuevoNodo(nodos, yo, canalAplicar)

// Someter operación para acuerdo por consenso en entrada de registro
nr.SometerOperacion(operacion interface{}) (indice, mandato, esLider, idLider)

// Obtención de estado de nodo Raft: quien es, mandato en curso
//y si cree ser el lider
nr.ObtenerEstado() (yo, mandato, esLider, idLider)

// Metodo Para() utilizado cuando no se necesita mas al nodo
func (nr *NodoRaft) Para()

// cada vez que una nueva operación es comprometida en una entrada
// de registro, cada nodo Raft debe enviar un mensaje AplicaOperacion
// a la máquina de estados
type AplicaOperacion
```

Tenéis disponible la función "*CallTimeout*", de llamada a método remoto rpc con tiempo de expiración, en el fichero "*raft/internal/comun/rpctimeout*".

Tenéis un código básico de servidor rpc genérico (con tcp, no http) en fichero çmd/srvraft/main.go"para que lo adaptéis al funcionamiento de servidor Raft con registro de llamadas rpc *AppendEntries* y *RequestVote* desarrolladas en el fichero "*internal/raft.raft.go*".

Si os interesa, tenéis disponible código de despliegue de programas remotos con ssh en múltiples máquinas en el fichero "*raft/internal/despliegue/sshClientWithPUBLICKEYAuthAndRemoteExec.go*".

Desarrollar otro paquete simple en "*raft/pkg/cltraft/cltraft.go*"para enviar operaciones a los nodos Raft (a ser posible al líder) a través de llamadas rpc que definái s y que espere su compromiso.

Tenéis disponible código incompleto de test de integración para las 4 pruebas de validación en el fichero "*raft/internal/testintegracionraft1/testintegracionraft1.go*".

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.

4.1. Validación

La mayor parte del desarrollo, se puede trabajar en la máquina local, pero para la validación final debe ejecutarse cada servidor en una máquina física diferente. Comprobar, previamente y con tiempo suficiente, que no hay problemas en ejecución distribuida.

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

1. Arranque y parado de un nodo remoto
2. Se ha elegido a un primer líder correcto.
3. Un líder nuevo toma el relevo de uno caído.
4. Se consigue comprometer 3 operaciones seguidas en 3 entradas, con un líder estable y sin fallos en el sistema.

Teneis ejemplos de tests incompletos en el fichero "*internal/raft/integracionraft1.go*".

Para ejecutar todos los tests del modulo podéis ejecutar "*go test ./....*en el directorio raíz del modulo. Hay disponibles también métodos para ejecutar tests específicos. El comando "go test -v ./....ºs permite tener más salida de información de depuración en la ejecución de tests completos de un modulo.

5. 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 batería 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 gramaticales u ortográficos.
- La memoria debería incluir diagramas de máquina de estados y diagramas de secuencia para explicar los **protocolos de intercambio de mensajes y los eventos de fallo**.
- **Cada nodo**(servidor) **debe ejecutarse en una máquina física diferente en la prueba de evaluación**.

La superación de las pruebas 1 y 2 supone la obtención de una B en la parte correspondiente a test. Para obtener una calificación de A, se deberá superar la prueba 1, 2 y 3. La superación de los test 1, 2, 3 y 4 supone tener una calificación de A+. Para llevar a cabo esta implementación, podéis basaros en el código disponible en el esqueleto.

5.1. 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

6. 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.

La entrega DEBE contener los diferentes ficheros de código Golang 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**.