SISTEMAS DISTRIBUIDOS

PRÁCTICA 3

Pablo Moreno Muñoz 841972

Andrés Yubero Segura 842236

ÍNDICE

Introducción

En la práctica actual y en la siguiente, se llevará a cabo la creación de un servicio de almacenamiento clave/valor en memoria RAM con tolerancia a fallos, utilizando la replicación distribuida basada en el algoritmo Raft. El protocolo Raft es una solución de máquina de estados replicada que se basa en un algoritmo de consenso. La implementación del algoritmo de elección de líder de Raft en esta práctica funciona en escenarios iniciales de fallos básicos. Sin embargo, en prácticas posteriores, se completará la implementación para que funcione en todos los posibles casos de fallos.

Para realizar la comunicación entre las máquinas, se realizan llamadas RPC, el código ha sido basado en el documento oficial de Raft disponible en https://raft.github.io/raft.pdf.

Elección de líder en Raft

En el protocolo Raft, cada nodo (servidor) puede estar en uno de tres estados: líder, candidato o seguidor. Inicialmente, todos los nodos son seguidores. El líder sigue en ese estado mientras no haya fallos. Si ocurre un fallo en el que el líder cae o deja de enviar latidos a los seguidores, se inicia un proceso de elección entre los servidores activos para seleccionar un nuevo líder.

El líder envía heartbeats a las réplicas para mantener la comunicación y notificar su buen funcionamiento. Estos heartbeats se envían regularmente, aproximadamente cada 50 ms, para garantizar que la frecuencia no supere las 20 veces por segundo.

Los nodos seguidores tienen un temporizador de recepción de heartbeats para detectar posibles fallos del líder. Si el temporizador expira, se inicia el proceso de elección de un nuevo líder, y el nodo se convierte en candidato. El candidato aumenta su mandato, emite un voto propio y solicita votos a través de la llamada RPC RequestVote a otros servidores, buscando obtener la mayoría simple de votos.

Si el candidato recibe suficientes votos antes de que su temporizador expire nuevamente, se convierte en el nuevo líder y comienza a enviar heartbeats al resto de réplicas para indicar su estado de liderazgo.

Esquema de los posibles estados de los nodos

- Estado **líder**:
    - Cada 50 ms envía heartbeats al resto de nodos
    - Si se cae, tras una nueva elección de líder al reconectarse pasará a ser SEGUIDOR
    - Si descubre que hay otro nodo con mayor mandato → pasa a ser SEGUIDOR
- Estado **candidato**:
    - Si obtiene la mayoría simple de votos → pasa a ser LÍDER
    - Si hay timeout de elección y sigue sin haber nuevo líder → sigue siendo CANDIDATO
    - Si recibe heartbeat de líder o petición de voto de nodo con mayor mandato → pasa a ser SEGUIDOR

- Estado **seguidor**:
    - Si expira el timeout se inicia una nueva elección → pasa a ser CANDIDATO
    - Recibe latidos del líder

El autómata que resume los estados y sus transiciones es el siguiente:

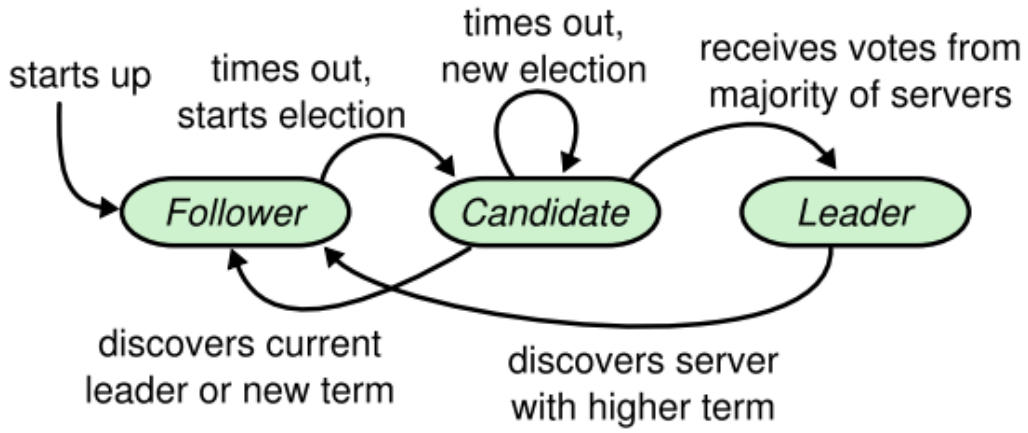
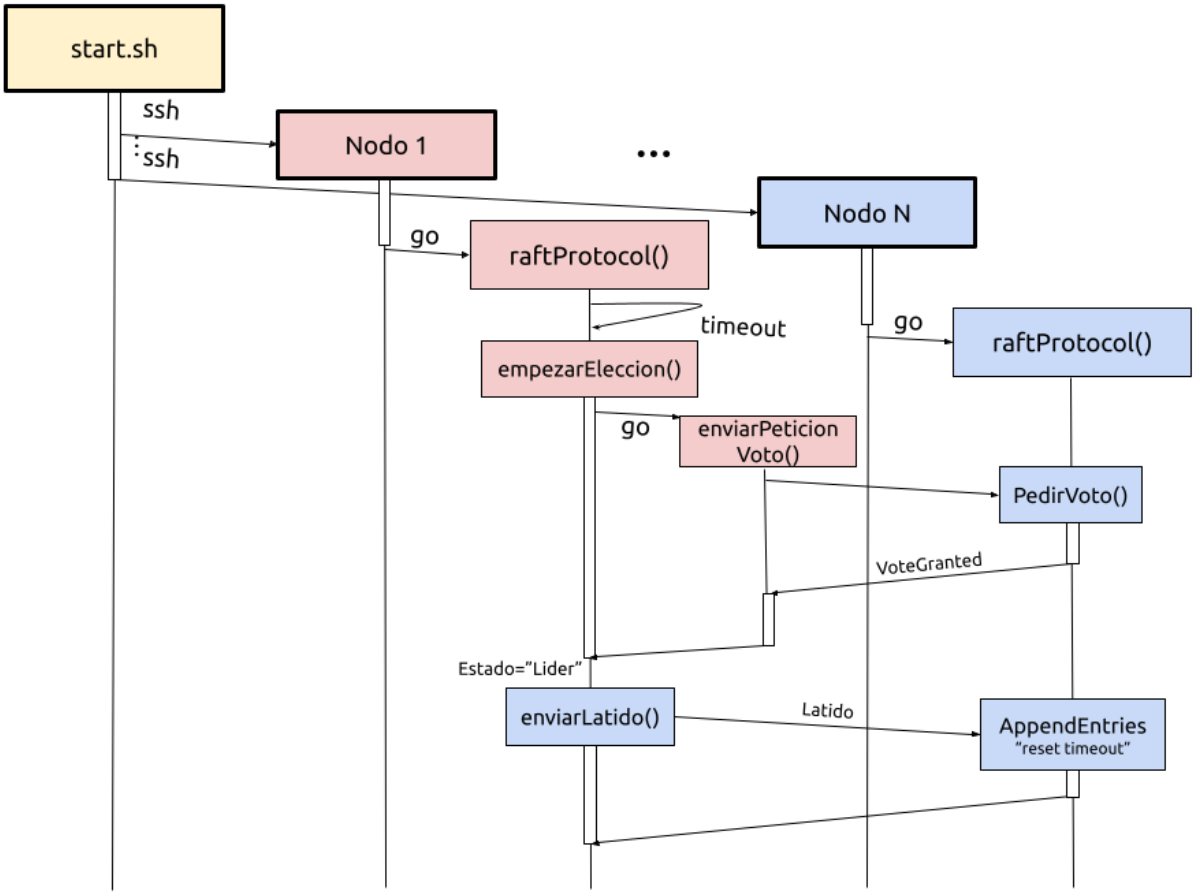


Diagrama de secuencia de elección de líder



Llamada RPC AppendEntries

Hemos implementado la función AppendEntries para que sirva tanto para enviar los latidos (heartbeats) como para comprometer las entradas. A esta función le pasamos primero un struct ArgAppendEntries, que contiene el identificador del líder, el mandato, el índice anterior a las nuevas entradas por comprometer, el término antes de las nuevas entries, un vector de entries a guardar, lo que permite enviar varias entries al mismo tiempo, y el número de entradas comprometidas por el líder.

A esta función también le pasamos por referencia un struct de resultados donde se devuelve el mandato actual del servidor al que se ha realizado la llamada RPC donde si el líder tenía un mandato menor lo actualiza y vuelve a ser seguidor y un booleano que indica si el seguidor que recibe la llamada RPC ha aplicado la entrada a su log.

Cuando AppendEntries se utiliza únicamente para enviar el heartbeat, se envía un true por el canal de Seguidor que indica que este sigue siendo un seguidor.

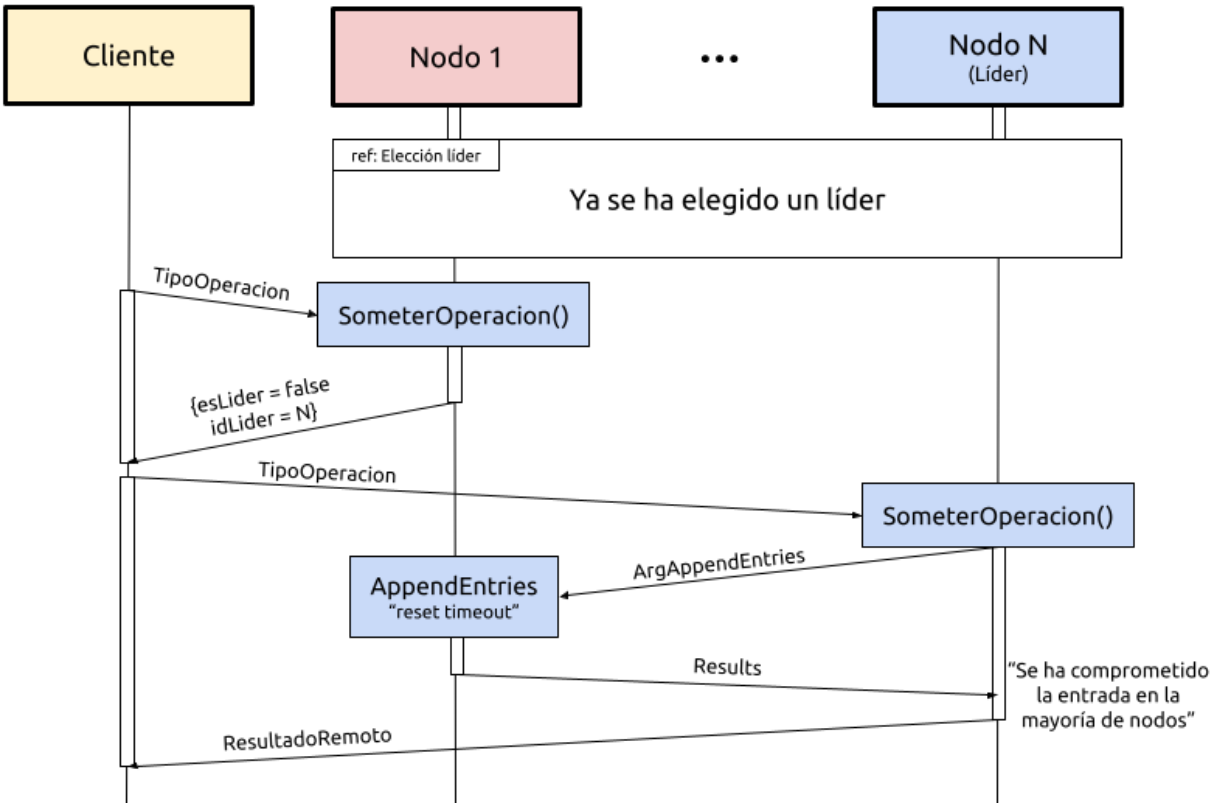
Gestión del cliente

El cliente mantiene una lista con las direcciones IP y puertos de todos los nodos Raft. Si el cliente quiere saber quién es el líder, puede ejecutar la función *ObtenerEstado* en alguno de los nodos, y ésta le devolverá si ese nodo es líder, la id de ese nodo, el mandato y el id del líder, de manera que con la id del líder ya se podría someter una operación directamente al líder.

Por lo tanto, cuando el cliente desea realizar su primera operación, se comunica con un nodo al azar. Si este nodo resulta ser el líder, ejecuta la operación y la confirma. En caso contrario, el nodo envía al cliente la id del líder, ya que la función *SometerOperacionRaft* al igual que *ObtenerEstado* devuelve la id del líder.

SometerOperacionRaft se encarga de guardar la entrada en el log del líder, y posteriormente propagarla al resto de nodos y asegurarse de que al menos la mayoría de nodos ha comprometido la entrada, de forma que una vez se ha hecho esta comprobación se responda al cliente con respuesta “ok” como que la entrada se ha comprometido correctamente.

El diagrama de secuencia sobre cómo se comunica un cliente con el líder y se consiguen comprometer las operaciones se muestra a continuación:



Tests de prueba

Para realizar esta pruebas hemos lanzado 3 nodos réplica, después esperamos cierto tiempo para dejar que se inicializan y desplieguen correctamente.

Test 1: Arranque y parado de un nodo remoto

Iniciar y detener un nodo remoto implica obtener el estado de cada nodo a través de la llamada "ObtenerEstadoNodo" para saber si ha sido arrancado correctamente. El arranque, comprobación del nodo y parado del mismo se hará para 3 nodos.

Test 2: Elegir un primer líder correcto

Se arrancan varios nodos, se espera un tiempo de 2.5s que es el máximo teórico en el que tardarán en elegir un líder, y posteriormente usamos *pruebaUnLider()* para ver si se ha elegido un líder correctamente.

Test 3: Un líder nuevo toma el relevo de uno caído

Al igual que antes, se arrancan varios nodos, se espera el tiempo de elección a que los nodos escojan a un líder, posteriormente con *pruebaUnLider()* se obtiene el identificador del líder y con *paraNodo()* es desconectado Finalmente, se espera otra vez a que los nodos restantes elijan un nuevo líder y se comprueba que se ha elegido un líder correctamente.

Test 4: Se consigue comprometer 3 operaciones seguidas

Una vez se arrancan tres nodos y se elige un líder, enviamos tres operaciones seguidas al líder para que las comprometa junto a una mayoría de nodos. Finalmente comprobamos que las tres operaciones se han registrado correctamente.