Sistemas Distribuidos - Práctica 4
Algoritmo Raft - 2ª Parte

Selivanov Dobrisan, Cristian Andrei - 816456@unizar.es

Wozniak, Dorian Boleslaw - 817570@unizar.es

Índice

1 Introducción

En esta práctica, la segunda en una serie de tres en torno al sistema de consenso distribuido Raft, se ha realizado la implementación completa del algoritmo. En la primera práctica se implementó una versión básica del sistema de elección de líder, y un sistema de replicación de operaciones que asume la ausencia de fallos, por lo que no estaría preparado para caídas de nodos, especialmente el líder.

Al sistema de elección de líder descrito anteriormente se ha añadido una restricción extra para la elección de líder: los nodos candidatos deben tener su registro actualizado respecto al votante, tal que el segundo pueda determinar el mejor líder, evitando así que se elija un líder que no tenga todas las entradas comprometidas (puesto que solo se requiere una mayoría para hacerlo).

Por otro lado, se ha completado la funcionalidad de replicar y comprometer operaciones al pedirlo un cliente mediante la RPC SometerOperacion(). Además, los nodos deberán aplicar operaciones sometidas sobre un sistema de almacenamiento clave-valor en RAM. Los nodos deberán ser capaces de determinar si pueden comprometer una operación o no según el número de nodos activos.

Para probar el funcionamiento de la aplicación, se ha realizado una serie de pruebas de integración que verifiquen que el sistema funciona correctamente, es capaz de someter operaciones cuando se encuentra en funcionamiento, y capaz de detectar caídas y evitar pérdidas de datos ya comprometidos.

2 Diseño del algoritmo

2.1 Elección de líder tolerante a fallos

En el algoritmo presentado originalmente para la elección de líder funcionaba de la siguiente manera: un seguidor, tras pasar su *Election Timeout* sin recibir un latido del líder pasaba al estado Candidato, aumentando su mandato y pidiendo al resto de nodos su voto a través de la RPC PedirVoto(). Cada nodo seguidor, al recibir la petición y siempre que no tengan mayor mandato que el candidato, deciden si le votan o no según si aún no han votado a otro candidato o ya le han votado en el mismo mandato. El candidato, salvo que reciba una respuesta con mandato mayor que el suyo, por cada respuesta positiva se suma un voto y, al obtener una mayoría (siendo que se ha votado a sí mismo), se vuelve líder y envía un latido para mostrarse al resto de nodos. Si no recibe a tiempo una mayoría, el candidato reinicia las elecciones hasta que salga como líder o se vuelva seguidor según su mandato.

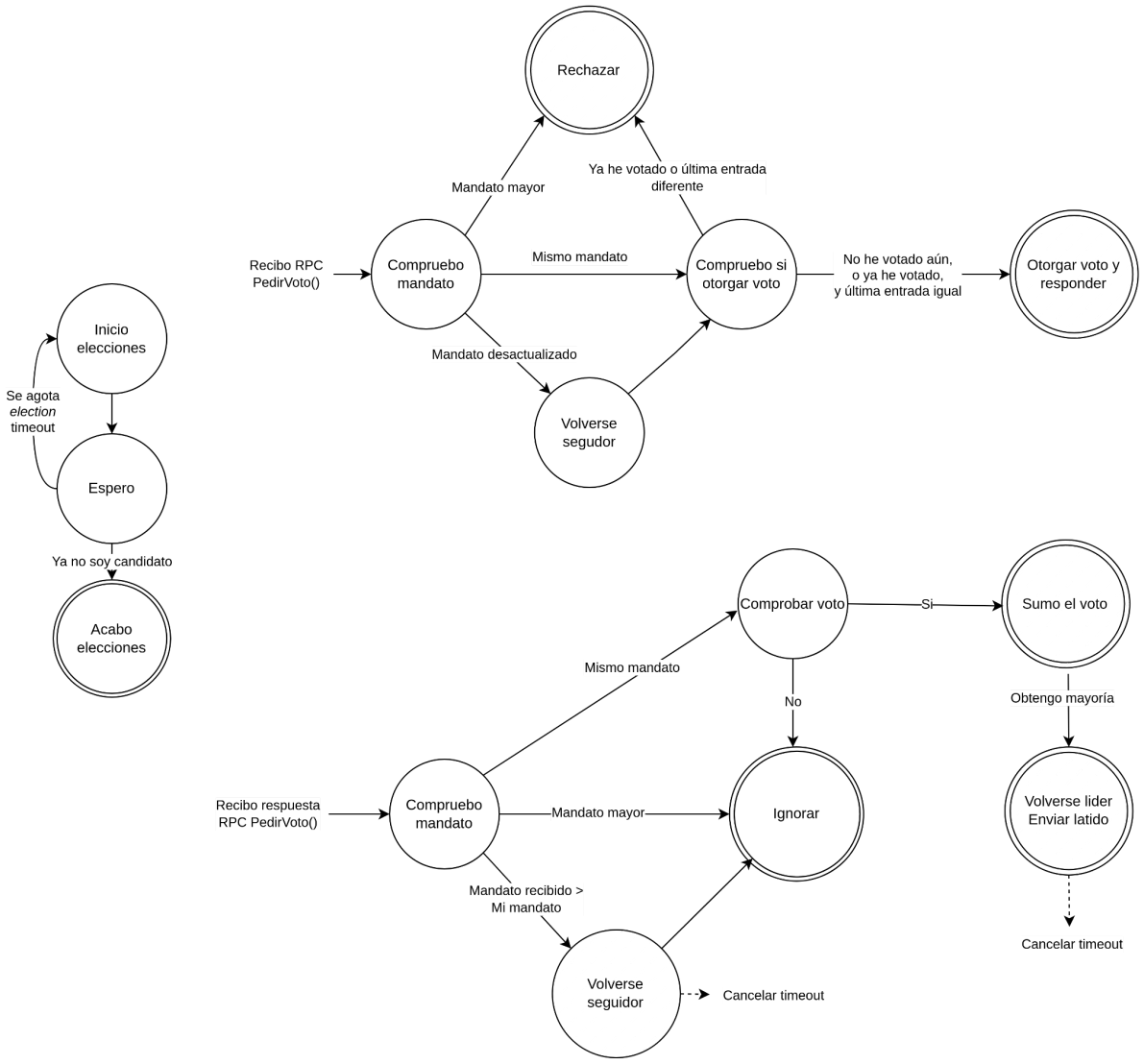

Fig. 1:Máquinas de estado para tratar la elección de líder

El problema que presenta el sistema actualmente se encuentra a la hora de preservar las entradas comprometidas. Puesto que no se requiere unanimidad para comprometer una operación, solo mayoría, existe una posibilidad de que al caer el líder, un nodo cuyas entradas se encuentran

desactualizadas trate de obtener el liderazgo. Si lo consigue las siguientes entradas a la máquina de estados se producirán sobre las entradas ya comprometidas de otros nodos, llevando a la pérdida de consistencia en la información almacenada.

Para solventarlo, se trata siempre de elegir un líder cuya réplica se encuentre al día con las últimas entradas comprometidas. Al requerir que al comprometer una operación una mayoría de nodos deben contener esta, se garantiza que siempre que haya al menos la mitad de los nodos más uno, habrá un nodo con la última entrada comprometida y, dadas las propiedades del sistema, todas las operaciones comprometidas hasta ahora.

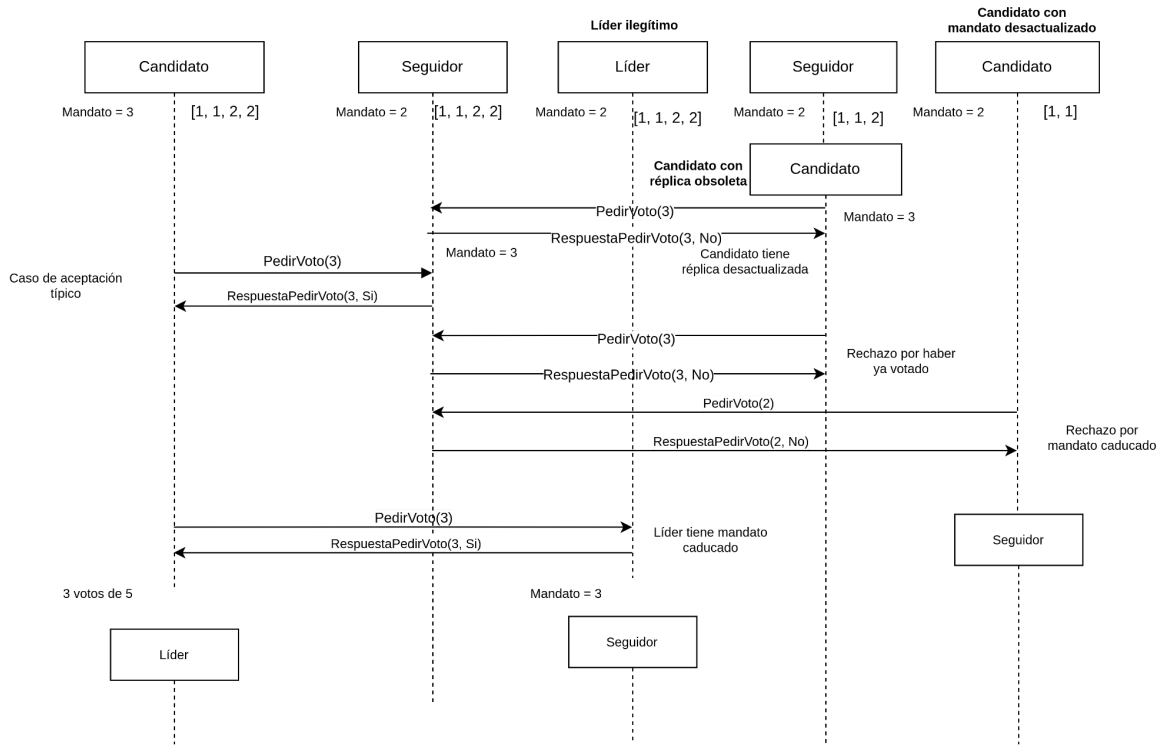

Fig. 2: Diagrama de secuencia con ejemplos de comportamiento durante la elección de un líder

Así, al enviar una RPC PedirVoto(), un candidato además envía el índice y mandato de su última entrada. Además de las condiciones anteriores, el nodo votante sólo otorgará el voto si contiene una entrada en el índice dado con el mismo mandato (y por tanto, que todas las entradas hasta ese punto sean iguales).

2.2 SometerOperacion() y replicación tolerante a fallos

En la versión anterior el sistema de replicación asumía una ausencia de fallos, por lo que el líder, al enviar una RPC AppendEntries(), sólo se encargaba de mandar las nuevas operaciones a los nodos, sin asegurarse de que lo hayan recibido correctamente y sin una traza de en qué punto se encuentran los seguidores en la replicación.

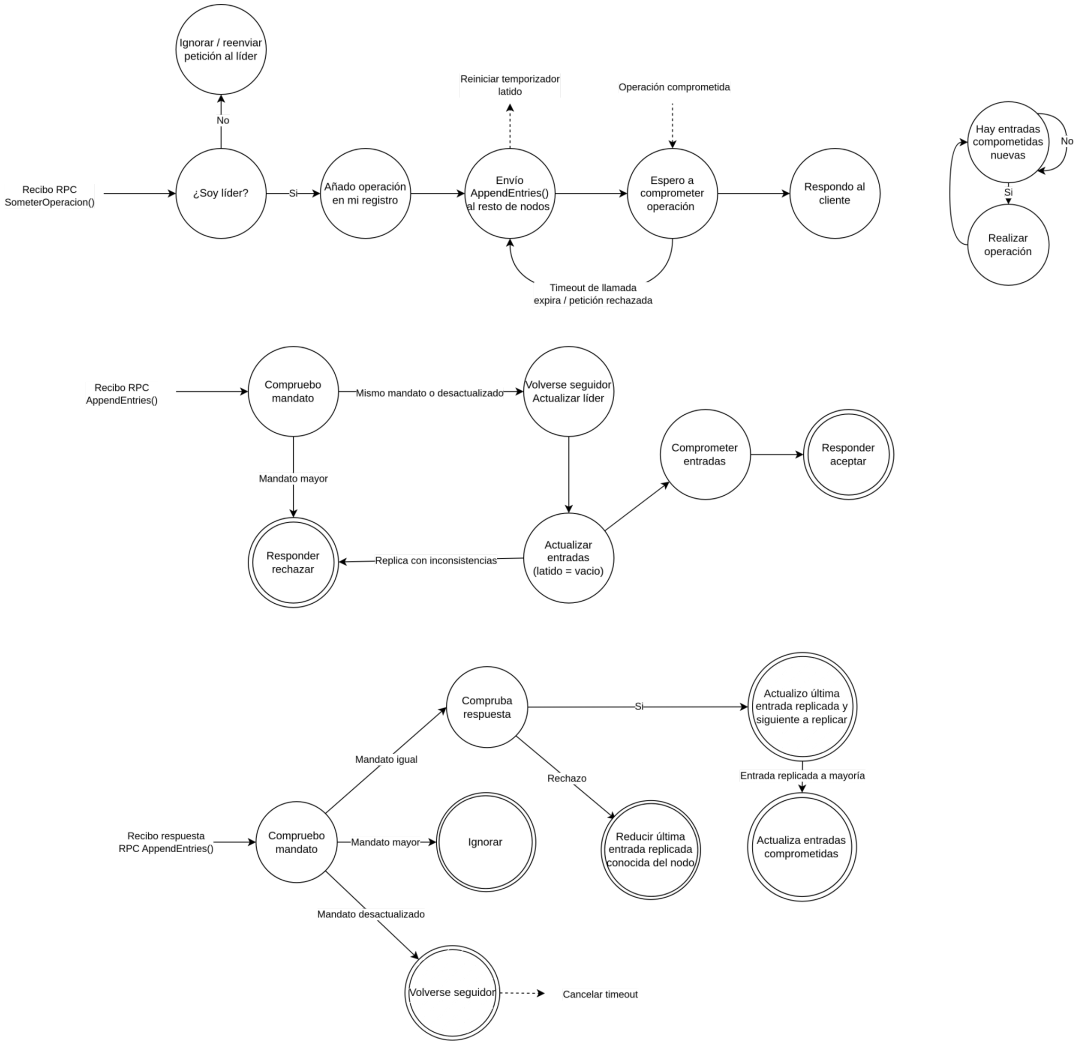

Fig. 3: Máquinas de estado para tratar el sometimiento y replicación de estados entre nodos

Al recibir el líder SometerOperacion(), el líder añade la operación a su réplica, y trata de replicar a todos los nodos el nuevo estado y, una vez obtenida una mayoría de respuestas positivas a la solicitud, debe comprometer la operación y realizarla en su máquina.

Un líder, al ser elegido, debe tener en cuenta para cada seguidor cual es el último índice conocido del nodo que ha sido replicado (actualizado al recibir una respuesta positiva) y el índice de la siguiente entrada a enviarle (se actualiza a medida que el líder recibe nuevas peticiones). Además, todos los nodos, para comprometer y realizar operaciones, deben tener un índice con la última entrada comprometida conocida (actualizada al recibir AppendEntries()) y la última entrada aplicada (se realizarán todas hasta la última entrada comprometida).

Al recibir una solicitud, un seguidor obtiene las entradas a añadir junto al índice anterior a la inserción y su mandato. Si el seguidor no contiene la entrada en el índice dado o el mandato es diferente, significa que la réplica del seguidor no coincide con la del líder hasta ese punto. El líder deberá, en ese caso, reintentar enviando la entrada incorrecta junto al resto hasta que encuentre un punto desde donde actualizar la réplica. Si la entrada dada es igual, entonces se garantiza que todas las entradas anteriores son iguales. En este caso, el seguidor añadirá las entradas, sobreescribiendo si fuera necesario (se garantiza que no se pierden entradas comprometidas). Por otro lado, si el líder le envía un índice mayor para la entrada comprometida de la que tiene, la actualiza.

El líder, al recibir respuestas positivas, actualiza el último índice replicado e índice de la siguiente entrada a enviar. Si comprueba que haya una mayoría de nodos con un índice mayor que el de la entrada comprometida, y esa entrada se realizó en el mandato actual, actualiza la entrada comprometida y manda una respuesta positiva al cliente en caso de ser la operación a someter.

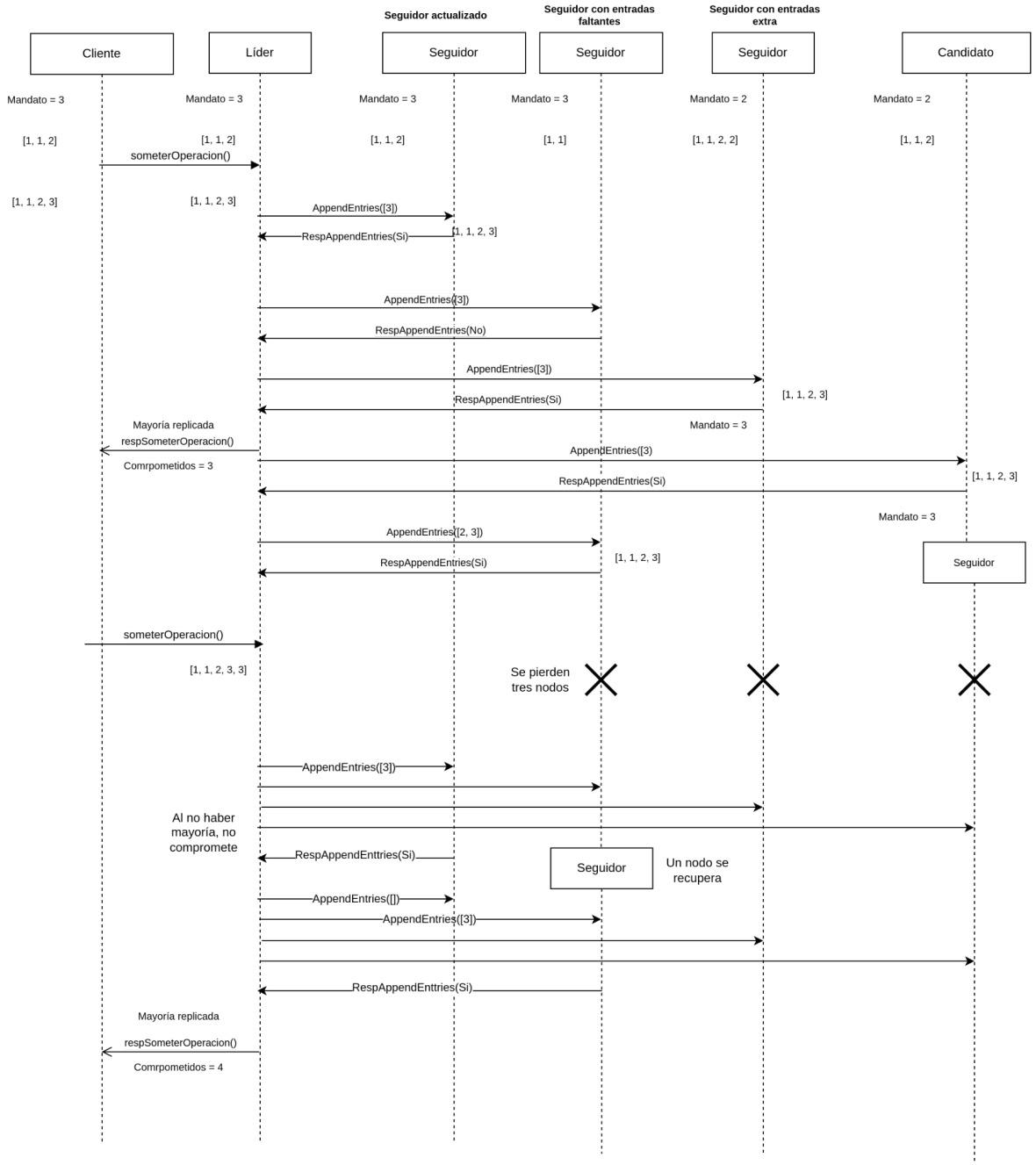

Fig. 4: Diagrama de secuencia con ejemplos de comportamiento para el sometimiento y replicación de estados

3 Implementación

Para el funcionamiento descrito anteriormente, se ha implementado por un lado un sistema de almacenamiento clave-valor en memoria basado en un tipo de datos *map* sobre el que se realizan operaciones a medida que se comprometen entradas de la réplica. Por otro lado, se han implementado los cambios anteriormente descritos para la elección del mejor líder y para la replicación y compromiso correctos de entradas.

Para gestionar el sometimiento de operaciones, habrá gorutinas separadas para gestionar los envíos de las réplicas mediante AppendEntries() para cada nodo tal que, al detectar mayoría, actualice la última entrada comprometida y notifique a la rutina de la RPC SometerOperacion() para contestar, y una gorutina para aplicar nuevas operaciones a medida que se aplican las operaciones.

4 Validación

Se han diseñado una serie de test adicionales a los escritos para la primera parte. Estos tests consisten en:

- El primer test verifica que, teniendo un grupo de tres nodos Raft, si cae un nodo seguidor el líder sigue pudiendo comprometer operaciones. Para ello, primero se realiza una operación de escritura en el almacenamiento RAM. Luego, se encuentra el índice del líder actual y se desconecta un nodo cualquiera salvo este. Se realiza una operación de lectura. En caso de cumplirse, se verifica que se ha cumplido al devolver los nodos restantes el valor en la clave anteriormente añadida. Adicionalmente se verifica que los nodos se pueden reconectar. Si el nodo caído, al reconectarse, realiza la operación de lectura que se comprometió al no estar presente, se verifica el funcionamiento correcto.
- El segundo test es similar al primero, pero en este caso se quiere comprobar que no se compromete una entrada si no hay una mayoría de nodos activos. En este caso, se comprueba que las operaciones sometidas al líder no se realizan hasta que el test reconecte el resto de los nodos.
- El último test verifica que el sistema actúa correctamente incluso en casos de ejecución concurrente. Para ello, se lanzan 5 clientes simultáneos que realizan peticiones al líder. La prueba se considera un éxito si todos los nodos realizan las mismas operaciones en el mismo órden.