RAFT - 2ªPARTE

SISTEMAS DISTRIBUIDOS - PRÁCTICA 4

Lizer Bernad Ferrando - 779035
Lucía Morales Rosa – 81690

---

1 INTRODUCCIÓN

En esta práctica se ha continuado con la implementación del servicio clave-valor empleando Raft. En esta práctica se ha finalizado la implementación de Raft, haciéndolo tolerante a fallos a la hora de tratar la llamada RPC "*AppendEntries*" y la elección del líder.

2 DISEÑO DEL ALGORITMO

Para realizar esta práctica, al igual que para la anterior se ha hecho uso del documento *In Search of an Understandable Consensus Algorithm (Extended Version)* de Ongaro y Ousterhout (2014), desarrollado en la Universidad de Stanford y proporcionado con el guion de la práctica.

La máquina de estados de los nodos se mantiene igual a la versión anterior y se puede observar en la Ilustración 1.

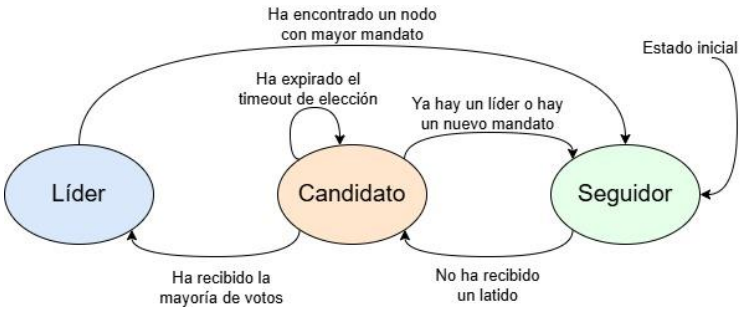

Ilustración 1: Máquina de estados de los nodos en Raft

Sin embargo, a pesar de que la máquina de estados no haya sido modificada, se han debido tener en cuenta nuevos escenarios que conllevan distintos comportamientos de los nodos. Para ello, se han diseñado varios diagramas de secuencia que permiten representar dichos comportamientos.

Para empezar, se ha rediseñado el diagrama original que representa una ejecución sin fallos desde el inicio hasta el compromiso de una entrada tal y como se puede observar en la Ilustración 2. En esta situación, todos los nodos comenzarían con un estado de "Seguidor" y cuando a alguno de ellos les venciera el *timeout* establecido, pasaría a ser "Candidato" y comenzaría una elección. Para este caso, se ha supuesto que todos los nodos responden favorablemente a la petición del candidato y este pasa a ser "Líder".

Tras esto, el nodo líder envía latidos al resto de nodos hasta que el cliente decide someter una operación. Cuando esto ocurre estando en el caso ideal, el líder guarda esta nueva entrada en su *log* y envía el siguiente latido con la nueva entrada como parte del argumento de la llamada RPC "AppendEntries".

Tras esto, los nodos van guardando la entrada en sus respectivos *logs* y cuando el líder detecta que ha sido replicada en más de la mitad de ellos con éxito, compromete la entrada y enviándole a su respectivo servidor la operación que desea realizar el cliente y esperando una confirmación. Después, el servidor responde al cliente, indicando así que la entrada ha sido comprometida con éxito.

Además, una vez que el líder ha comprometido la entrada, modifica su campo *CommitIndex* que posteriormente pasará como argumento al enviar los latidos a los nodos y de esta forma les permitirá saber hasta que entrada pueden comprometer.

Así, cuando el nodo detecte que tiene entradas por comprometer, enviará las operaciones correspondientes a dichas entradas a su servidor y guardará en su campo *LastApplied* el índice de la última entrada aplicada a su máquina de estados.

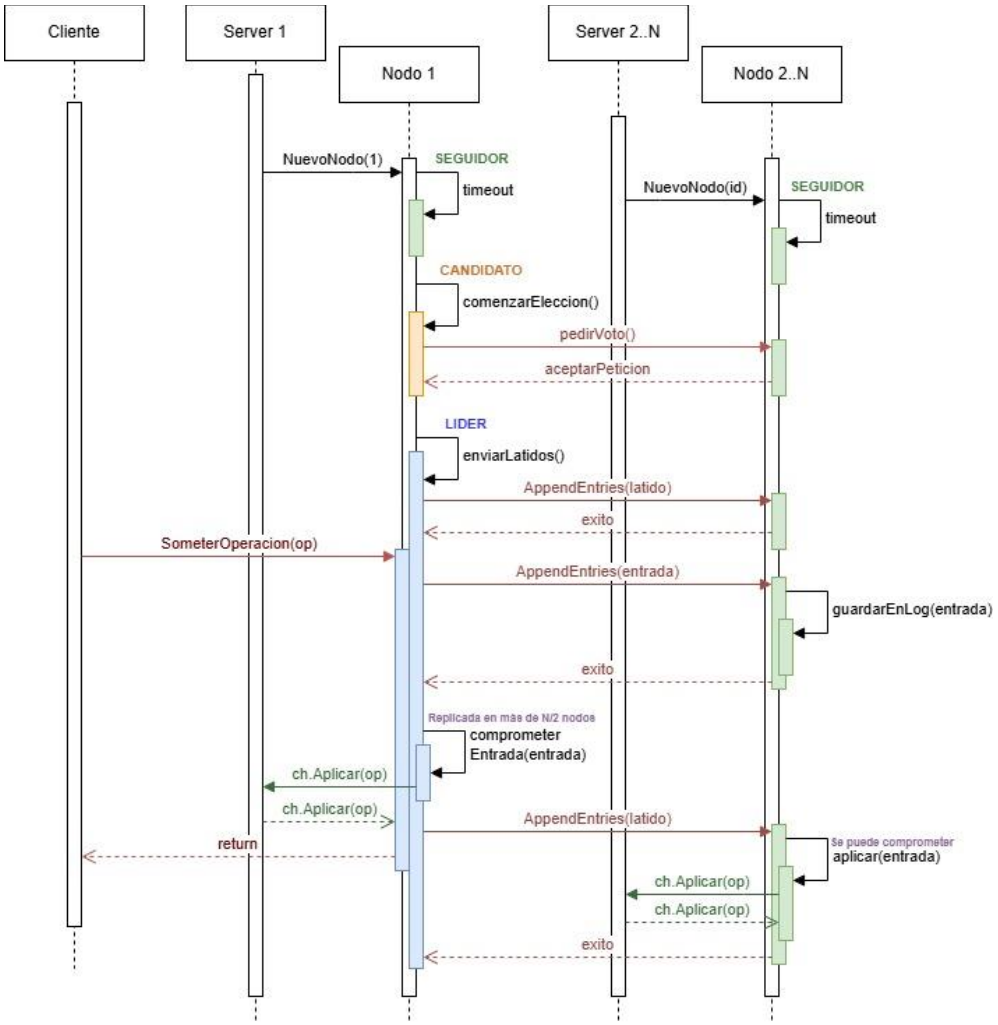

Ilustración 2: Ejecución normal del algoritmo sin fallos

Por otro lado, se ha diseñado un diagrama para representar el funcionamiento del sistema cuando se desea someter una entrada en un nodo caído y lo que ocurre cuando dicho nodo se recupera. Este comportamiento se puede apreciar en la Ilustración 3.

En este caso, se puede ver como el cliente somete una primera operación. Tras esto, el líder añade la nueva entrada a su log y tiene el comportamiento ideal para todos los seguidores menos para

el nodo 2, que está caído. Cuando el nodo 2 se recupera de la caída, el cliente ha sometido una segunda operación aumentando de esta forma el log del líder de forma que recibe una llamada *AppendEntries* con una entrada distinta a la esperada. Debido a esto, el nodo responde con un error al líder y este decrementa en uno el índice de la siguiente entrada a enviar, consiguiendo así que en el próximo latido le envíe la entrada correcta.

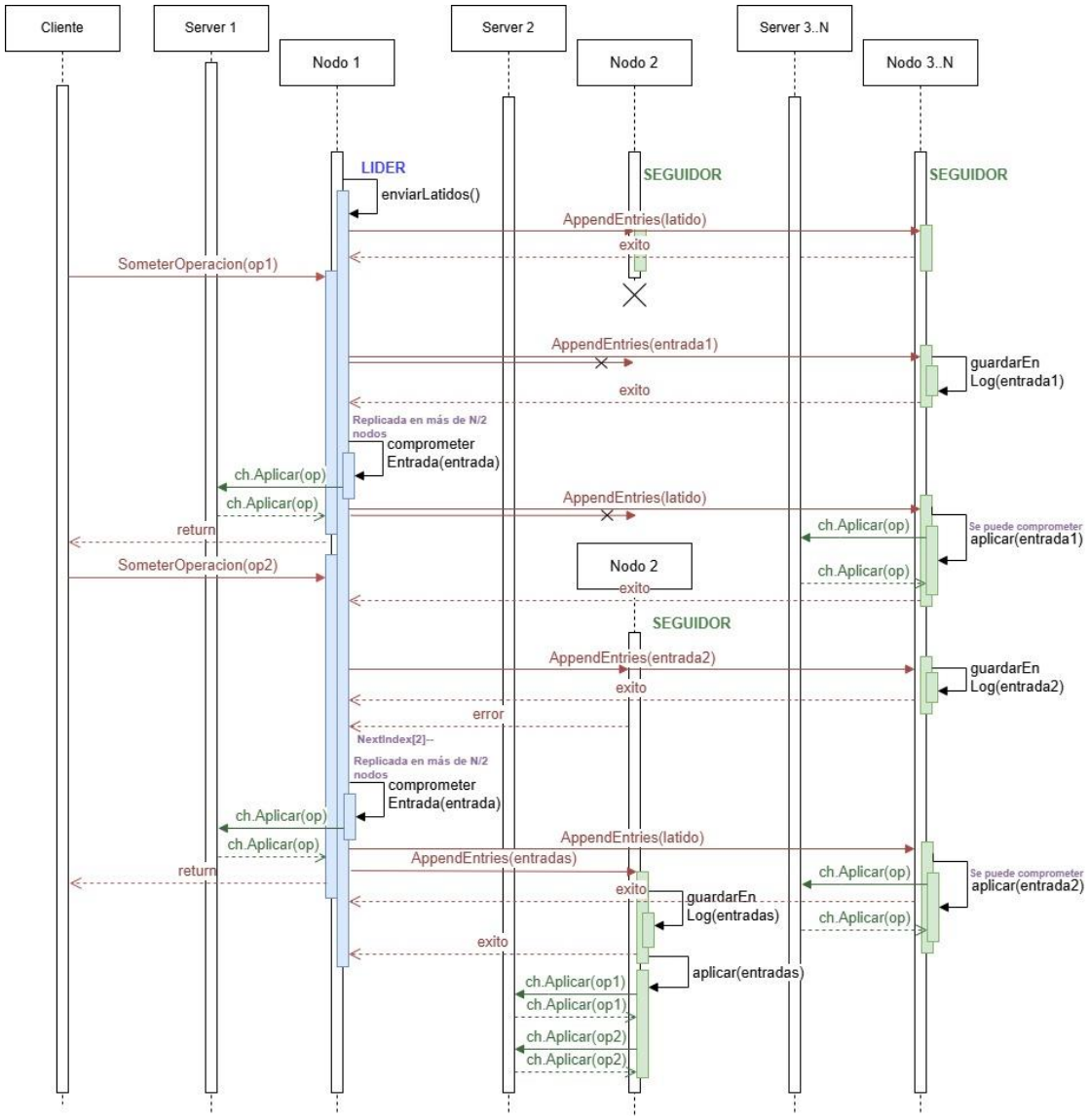

Ilustración 3: Someter operaciones con un nodo caído y su recuperación

Finalmente, se han credo los diagramas de secuencia para representar el comportamiento del sistema cuando se convoca una elección y hay un nodo caído tal y como se puede observar en la Ilustración 4, y cuando los nodos caídos son mas de la mitad, representado en la Ilustración 5**¡Error! No se encuentra el origen de la referencia.**.

En el caso de que haya uno o menos de la mitad de los nodos caídos cuando se convoca una elección, el comportamiento es casi ideal ya que el nodo candidato sigue pudiendo obtener una mayoría de los votos. Cuando un candidato consigue hacerse líder al obtener la mayoría de los votos y ganar la elección, comienza a enviar latidos al resto de nodos incluyendo su identificador de proceso y su mandato.

Lizer Bernad Ferrando - **779035**
Lucía Morales Rosa - **816906**

Con esto, se consigue que, en caso de que los nodos caídos se recuperasen, estos recibirían un latido del nuevo líder con sus datos y les permitiría actualizar sus campos y reconocer al nuevo líder como tal. Incluso en el caso de que un nodo caído creyera ser el líder y enviara un latido, el seguidor no lo reconocería como tal y le mandaría su mandato. De esta forma, el falso líder vería que está desactualizado respecto al resto de nodos y pasaría al estado de "Seguidor", reconociendo al nuevo líder cuando este le envíe el latido correspondiente.

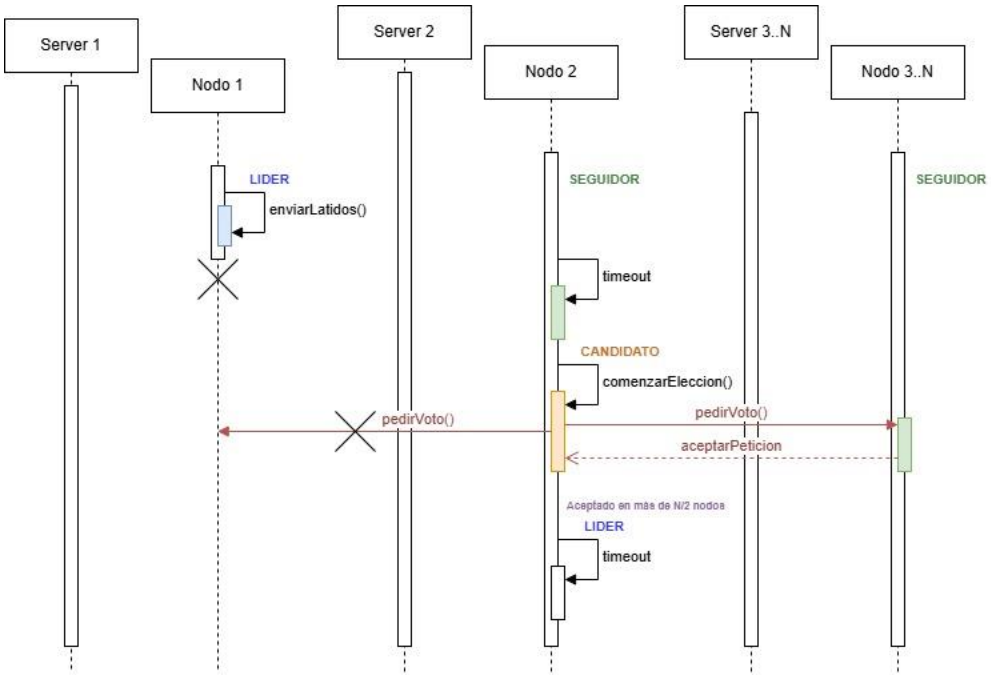

Ilustración 4: Elección de líder con un nodo caído

Por otro lado, en el caso de que más de la mitad de los nodos estén caídos, sería imposible obtener una mayoría de votos. Debido a esto, el candidato volvería a convocar la elección cuando venciera el *timeout* de forma indefinida y no sería posible obtener un nuevo líder hasta que el estado del sistema cambiara.

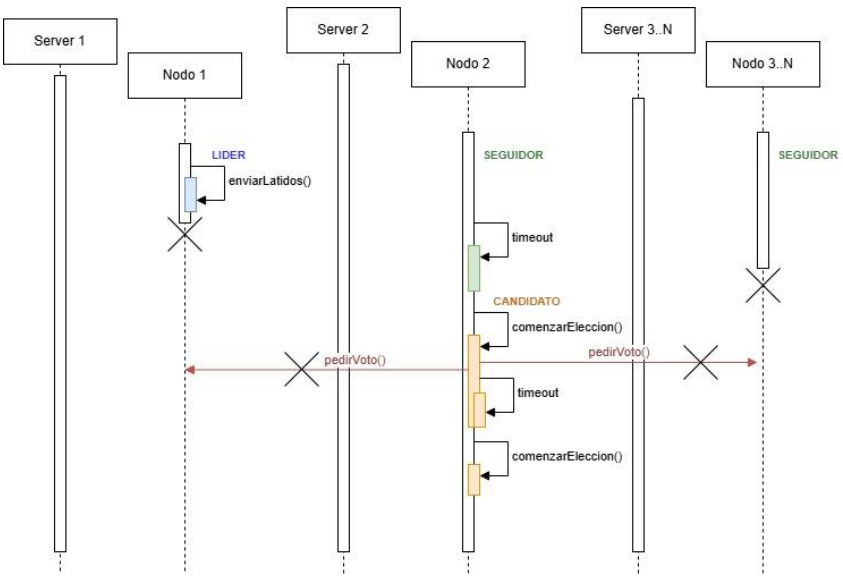

Ilustración 5: Elección de líder con más de la mitad de los nodos caídos

Lizer Bernad Ferrando - **779035**
Lucía Morales Rosa - **816906**

Nota: Para facilitar la comprensión de los diagramas de secuencia, se han representado las llamadas RPC entre los procesos con flechas rojas para diferenciarlas de las llamadas a función representadas con líneas negras. Además se han añadido pequeños textos para indicar estados o situaciones de forma más sencilla.

3 ASPECTOS RELEVANTES EN LA IMPLEMENTACIÓN

Para la correcta implementación de las mejoras propuestas para el algoritmo de Raft se han modificado diversas partes del código. Entre estas modificaciones destacan las siguientes.

3.1 Cambios destacables en las estructuras

A la estructura "NodoRaft" se le han añadido dos campos de tipo mapa de enteros que permitirán al líder conocer los índices de las entrada siguiente a ser enviada a cada nodo y el índice de la entrada más alta replicada en cada nodo. Estos campos son el "*NextIndex*" y el *"MatchIndex"* respectivamente. Además, también cuenta con dos canales de tipo "AplicaOperacion" llamados "AplicarOp" y "Aplicada" empleados para que los nodos se comuniquen con su servidor correspondiente y para que el líder notifique que puede responder al cliente porque la entrada ha sido comprometida.

En la estructura empleada para que el candidato envíe la petición de voto "ArgsPeticionVoto", se han añadido los campos *"LastLogIndex"* y *"LastLogTerm"* que permiten incluir el índice y el mandato de la última entrada del log del candidato para que el nodo receptor pueda determinar su validez como líder.

A la estructura "*ArgAppendEntries*" empleada por el líder al enviar los latidos a los seguidores, también se le han añadido nuevos campos. Para empezar los campos enteros "*PrevLogIndex*" y "*PrevLogTerm*" permiten enviar al seguidor el índice y el mandato de la entrada inmediatamente anterior a la nueva que se está enviando para que el seguidor pueda ver si su log es consistente con el del líder. En esta nueva versión, el líder también envía su *CommitIndex* para que el nodo sepa si puede o no comprometer entradas. Finalmente, se ha añadido un campo de tipo *array* de "EntradaLog" para que el líder pueda enviar todas las entradas faltantes al nodo en un único latido, ahorrando de esta forma varias llamadas.

3.2 Tratamiento de las llamadas RPC AppendEntry

3.2.1 Envío de latidos y entradas

En la versión anterior diseñada de Raft, al no estar pensada para tolerar fallos, se había establecido que de manera periódica el líder enviara únicamente latidos, de forma que si recibía una entrada nueva, enviaba un latido extra a todos los nodos con dicha entrada para que la incorporaran a su log.

En esta versión, sin embargo, se ha diseñado una versión tolerante a fallos por lo que la aproximación anterior no funcionaría correctamente. Para solucionarlo, el líder decide para cada nodo si debe o no enviar entradas a cada nodo dependiendo de los valores almacenados en su mapa *"NextIndex"*. Si el nodo tiene un *NextIndex* menor a la longitud del log del líder, implica que le faltan algunas entradas por lo que el líder le envía todas las entradas de su log desde la componente *NextIndex* hasta el final. Si por el contrario, el nodo está actualizado, el líder envía una entrada vacía para indicar que es un latido.

3.2.2 Replicación de las entradas en los logs y compromiso de entradas

Cuando a un seguidor le llega un latido, el seguidor comprueba la validez del líder comparando sus mandatos, de forma que si el líder no está actualizado, le responde con un su propio mandato para indicar al líder que debe dejar de serlo.

Si por el contrario, el líder es válido, el seguidor evalúa que los logs de ambos nodos sean consistentes empleando los argumentos "*PrevLogIndex*" y *"PrevLogTerm"* enviados por el líder. En caso de no serlo, el nodo responderá al líder con un no éxito y este decrementará en uno su *NextIndex* hasta encontrar el punto en el que el seguidor deja de ser consistente.

Cuando esto ocurra, el seguidor establecerá en qué posición deberá copiar las nuevas entradas, conservando las entradas coincidentes con el líder. Finalmente, comprobará si el *CommitIndex* del líder es mayor al suyo y de serlo comprometerá las entradas correspondientes en su máquina de estados enviando la notificación a su servidor a través del canal "AplicarOp" y responderá con éxito al líder. Sin embargo, para que el seguidor pueda aplicar la entrada, el líder debe haberla comprometido previamente.

El líder comprueba si puede comprometer alguna entrada tras el envío de un latido no vacío a un nodo. Cuando le llega la respuesta del nodo con un éxito, significa que existe la posibilidad de que la entrada se haya replicado en más de la mitad de los nodos, por lo que lo comprueba. De ser ese el caso, el líder notifica a su propio servidor y emplea el canal "Aplicada" para poder responder al cliente.

3.3 Elección de líder

A la hora de elegir un líder en la versión de Raft anterior, únicamente se necesitaba mayoría de votos y los nodos solo miraban que el candidato no estuviera desactualizado. Ahora, para que un nodo vote a un candidato, este debe tener un log al menos tan actualizado como el del votante. Es decir, el candidato comprobará si el *"LastLogTerm"* del candidato es mayor al del propio nodo o, en caso que sean iguales, el *"LastLogIndex"* del candidato sea mayor o igual que el suyo.

De esta forma, se asegura que no habrá un líder electo con un log desactualizado.

3.4 Comunicación con servidores y respuesta a clientes

La comunicación de los nodos con los servidores, como se ha adelantado anteriormente, se realiza por medio del canal "AplicarOp". Este canal es pasado como argumento desde el servidor al crear un nuevo nodo y permite el paso de mensajes "AplicaOperacion". De esta forma, cuando un nodo aplica una operación a su máquina de estados, lo que hace es mandar la operación a través del canal para que el servidor la ejecute y espera la notificación de que se ha recibido.

Por otro lado, el cliente se comunica únicamente con el nodo líder a través de una función "SometerOperacion" a través de la cual le envía al líder la operación que desea realizar. Cuando al líder le llega una entrada, este espera la llegada de una notificación a través del canal "Aplicada" que indique que la entrada ha sido comprometida en el nodo y aplicada en su máquina de estados. Esto significa que puede responder al cliente.

4 TESTS DE PRUEBA

Se han diseñado tres tests de prueba para verificar el correcto comportamiento del algoritmo de Raft. Para ejecutar los tests se lanzan tres nodos replica en las direcciones definidas previamente

Lizer Bernad Ferrando - **779035**
Lucía Morales Rosa - **816906**

en el fichero "internal/testintegracionraft1/testintegracionraft1_test.go" y tras realizar las comprobaciones se detienen los nodos.

El primer test comprueba que se consiguen acuerdos de varias entradas de registro a pesar de que una réplica de las tres ejecutadas inicialmente se desconecta del grupo. Esto se cumple ya que al ser tres nodos, sigue habiendo mayoría de nodos que pueden comprometer la entrada.

En el segundo test se prueba lo contrario. En este caso habrá dos replicas caídas y de esta forma no se conseguirá mayoría para comprometer la entrada.

Finalmente, se someterán 5 operaciones de forma concurrente y se comprobará el avance del índice del registro en el log.