Práctica 4: RAFT

Jorge Grima Terrén 801324
Guilermo Enguita Lahoz 801618

Introducción

A lo largo de la práctica 4, vamos a implementar las funcionalidades básicas del algoritmo Raft, utilizado para consenso y replicación de aplicaciones distribuidas con estado. En concreto, implementaremos una versión del algoritmo de elección basado en mayorías simples, y en una versión de la operación “AppendEntry”, que supone un funcionamiento sin fallos.

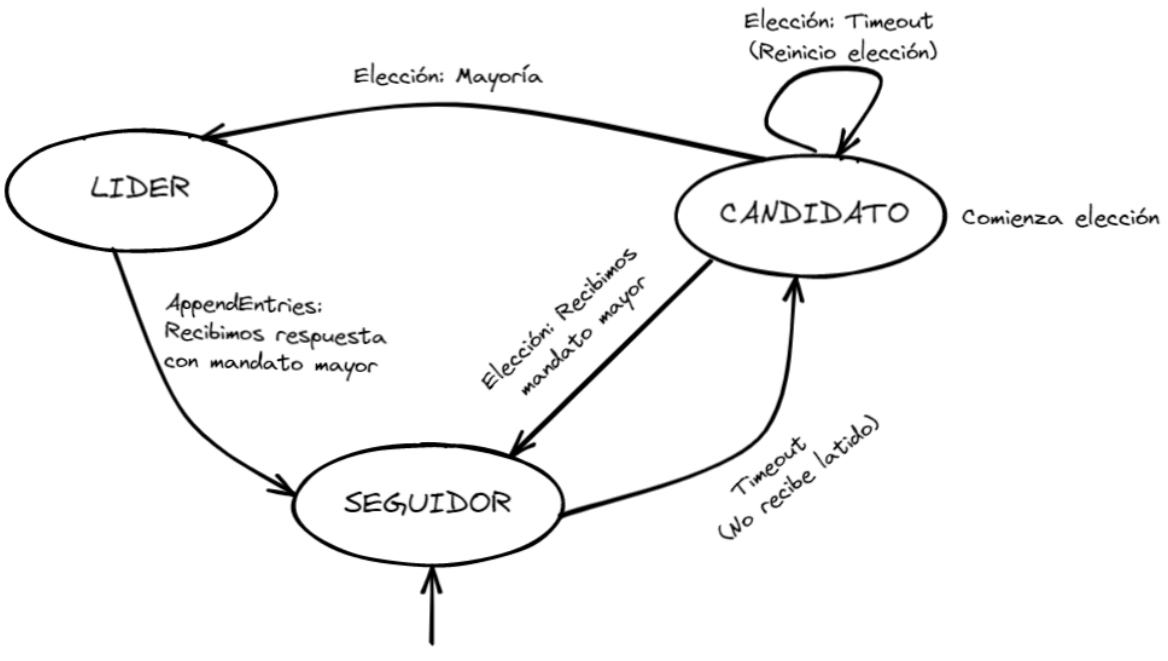
Funcionamiento básico

El algoritmo Raft nos permite ofrecer un servicio o aplicación distribuido de forma replicada, aplicando (o comprometiendo) las operaciones que modifiquen el estado únicamente cuando haya un consenso en el sistema.

Para conseguir esta funcionalidad, una de las réplicas actuará como líder, recibiendo peticiones de los clientes y comunicándolas al resto de nodos. El líder únicamente comprometerá la operación de un cliente en caso de que la mayoría de las réplicas hayan aprobado dicha acción.

Cada una de las réplicas almacenará un registro con las entradas comprometidas hasta el momento, guardando también el mandato en el que fueron comprometidas. Un mandato indica el período de tiempo durante el cuál un nodo específico fue líder. En un mandato solo puede haber un líder, en caso de que el líder fallara y hubiera una elección, el mandato aumentaría.

El funcionamiento básico de un nodo Raft viene descrito por el siguiente diagrama de estados:



Un nodo empezará su ejecución actuando como seguidor, recibiendo operaciones y latidos de su líder a través de la operación “AppendEntry”.
Si transcurre un tiempo determinado en el que el seguidor no ha recibido ningún mensaje del líder, el nodo detectará que este ha sufrido un fallo, pasando a ser un candidato y comenzando una nueva elección.

Si el nodo recibiera la mayoría de votos (la mitad más uno), pasaría a ser el nuevo líder, en caso de no recibirla iniciaría una nueva elección. A lo largo de la elección, si el candidato encuentra a un líder (o un nodo con un mandato mayor), volvería a estado de seguidor.

Finalmente, la réplica líder utilizará la llamada “AppendEntry” para comunicar las operaciones de los clientes al resto de nodos, y para enviar latidos, de forma que todas las réplicas sean conscientes de que el líder sigue en funcionamiento.

Funcionamiento de una elección

Las elecciones darán comienzo cuando un seguidor detecte que el líder ha sufrido un error. Para detectar dicho error, el seguidor establecerá un período de timeout. Cuando reciba un mensaje del líder, el seguidor reiniciará dicho período, e iniciará la elección únicamente cuando el timeout venza.

Para evitar que cuando el líder caiga se inicien varias elecciones simultáneas, los seguidores establecerán los valores de timeout de forma aleatoria (entre 150 y 300ms), evitando que varias réplicas detecten la caída a la vez.

Cuando comience la elección, el candidato incrementará su mandato y se votará a sí mismo. A continuación, enviará una petición “PedirVoto” a cada una de las réplicas esperando su respuesta por un tiempo determinado.
Al pedir el voto, el candidato indicará a los nodos su mandato, así como el índice y el mandato de la última entrada de su registro.

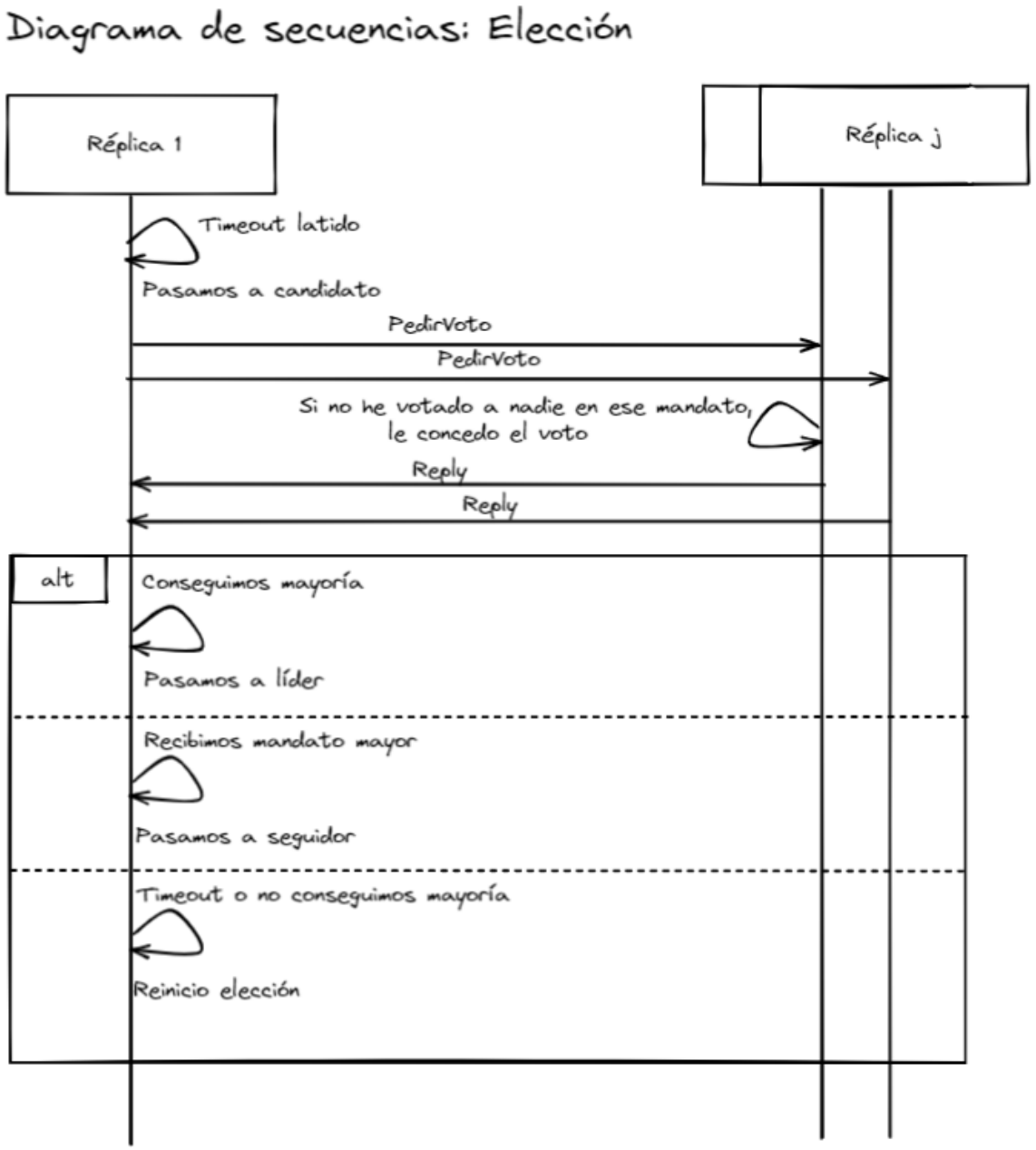
Los nodos concederán su voto al candidato siempre y cuando no hayan recibido ninguna otra petición de voto en el término de la elección. Destacamos que sí el nodo concede el voto, reiniciará su propio timeout de elección.
Como respuesta a la llamada “PedirVoto”, le indicarán al candidato si le han concedido o no el voto, y su propio mandato.

Nuestra implementación del candidato pondrá en ejecución tantas goroutine como réplicas, cada una encargada de realizar la petición a una de ellas. Cuando reciban

la respuesta, indicarán a la goroutine principal el resultado de la petición, comunicándose a través de canales síncronos.

La goroutine principal irá contando la cantidad de votos recibidos, y en caso de obtener la mayoría, el nodo pasará a ser líder. Se analizará también el mandato de cada réplica encuestada. En caso de que este sea mayor al propio, el nodo candidato pasará a ser seguidor.

Si no se reciben los suficientes votos o vence un timeout de elección, el candidato comenzará una nueva, incrementando su mandato.



Funcionamiento de AppendEntry

La llamada AppendEntry permite al líder comunicarse con sus seguidores, permitiéndole establecer su autoridad a través de latidos y alcanzar consenso en el sistema comunicando las operaciones de los clientes.

La implementación de la operación para la práctica tiene en cuenta un funcionamiento sin fallos, por lo que no ofrece la funcionalidad completa.

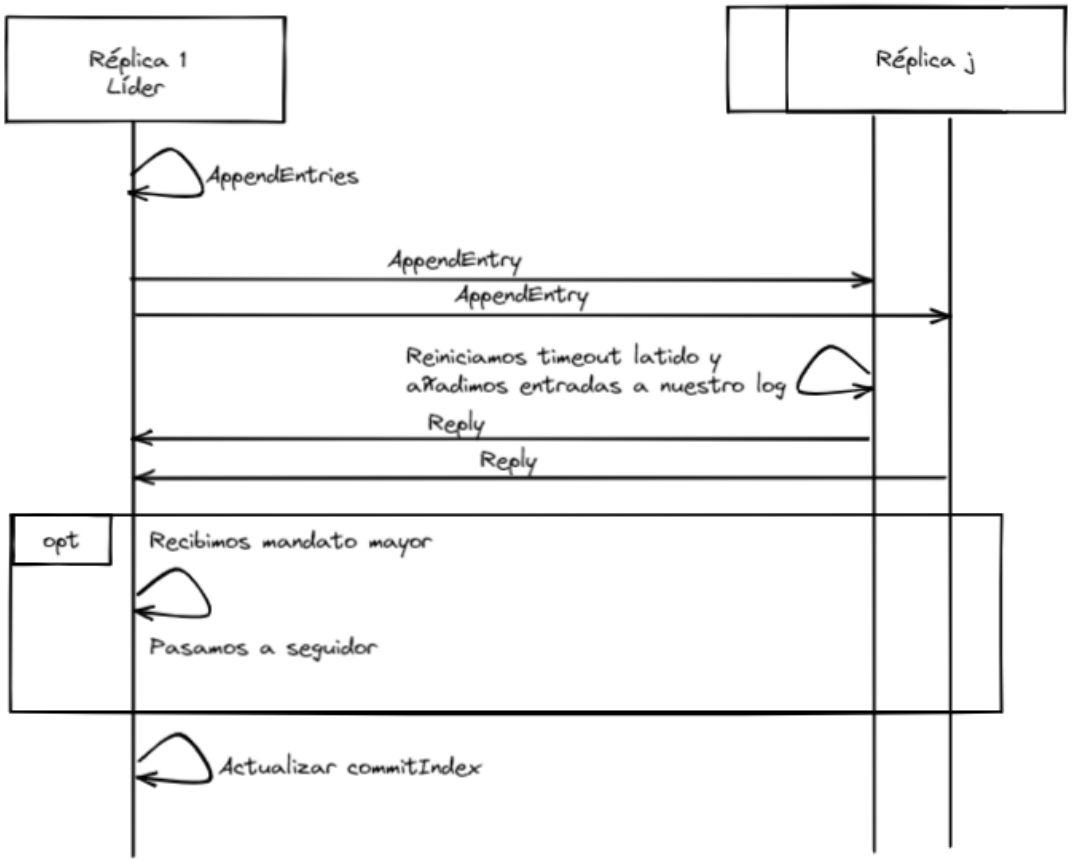
Al realizar un AppendEntry, el líder enviará al nodo su mandato, su índice de compromiso (que índica la última entrada comprometida del registro), las entradas a añadir, además del índice y el mandato de la entrada del registro a partir de la cuál se añadirán nuevas entradas.

La réplica, al recibir la petición, actualizará su mandato para estar actualizado con el líder, y reiniciará su timeout de elección. A continuación, comprobará que las entradas que ha recibido del líder son correctas, y pueden ser añadidas en su registro.
Para ello, comprobará que la última entrada de su registro coincide en índice y mandato con el índice y mandato indicado por el líder. En tal caso, añadirá las entradas recibidas a su registro y actualizará su índice de compromiso, poniéndolo al día con el del líder.

Para realizar los AppendEntry, el líder funcionará de forma similar al candidato durante la elección, creando una goroutina para cada réplica, que realice la petición y le comunique los resultados a través de canales síncronos. En caso de que durante la ejecución, el líder encontrara una réplica con mayor mandato, este pasaría a estado seguidor.

Al finalizar, el líder incrementará el valor de su índice de compromiso (commitIndex), según la longitud de la lista de operaciones enviadas, ya que si trabajamos suponiendo que no hay fallos, podemos considerar que cualquier operación enviada será comprometida.

Diagrama de secuencias: AppendEntries



Organización del código y otros

Para mantener la modularidad y la legibilidad del código, se ha dividido la implementación de Raft en varios ficheros:

- *raft.go* define las funciones relacionadas con la inicialización de un nodo y la gestión de su estado, además de la estructura común de todos los nodos.
- *raftEleccion.go* define todas las estructuras y funciones relacionadas con los procesos de elección.
- *raftAppendEntries.go* define todas las estructuras y funciones relacionadas con la llamada AppendEntry.
- *raftMisc.go* define funciones que permiten comunicarnos con las diferentes réplicas, que serán detalladas a continuación.

Hemos definido una serie de funciones que nos permiten conocer el estado de cada nodo e interactuar con ellos:

- *Para* finaliza la ejecución del nodo en cuestión-
- *ObtenerEstado* nos informa sobre el mandato del nodo, su identificador, el identificador de quién cree que es el líder, y si cree ser líder o no.
- *SometerOperacion* nos permite introducir una operación en el registro.

Para cada una de estas funciones hemos creado también una versión RPC, que nos permitirá interactuar y monitorizar las réplicas remotamente de manera sencilla.

Finalmente, hemos creado también un programa *crlcraft.go* que a partir de un menú textual por consola, nos permite controlar las réplicas Raft de forma remota, usando las llamadas RPC previamente definidas.