SISTEMAS DISTRIBUIDOS

Práctica III

Algoritmo RAFT

Viernes B Mañana Pareja 2

09/11/2023

GUILLERMO BAJO LABORDA, 842748@unizar.es

Universidad Zaragoza
1542

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Índice

1. Introducción

En esta práctica se ha implementado el algoritmo RAFT de algoritmo de consenso en sistemas distribuidos. Raft es ampliamente utilizado para garantizar que un solo nodo asuma el papel de líder, lo que es esencial para coordinar las operaciones en un sistema distribuido.

A través de un proceso de elección de líder basado en términos y votos, Raft asegura que el liderazgo sea único y coherente en el tiempo. Esta característica fundamental es clave para mantener la coherencia y la consistencia en las decisiones y operaciones en el sistema.

2. Implementación del algoritmo RAFT

Nuestra implementación del sistema RAFT se divide fundamentalmente en el autómata que realiza las operaciones pertinentes a la lógica de cada nodo y en las funciones que ofrecen distintas llamadas RPC usadas para comunicar los distintos nodos entre ellos. Por tanto, cada nodo tendrá una serie de atributos propios y operaciones que expone al resto del sistema.

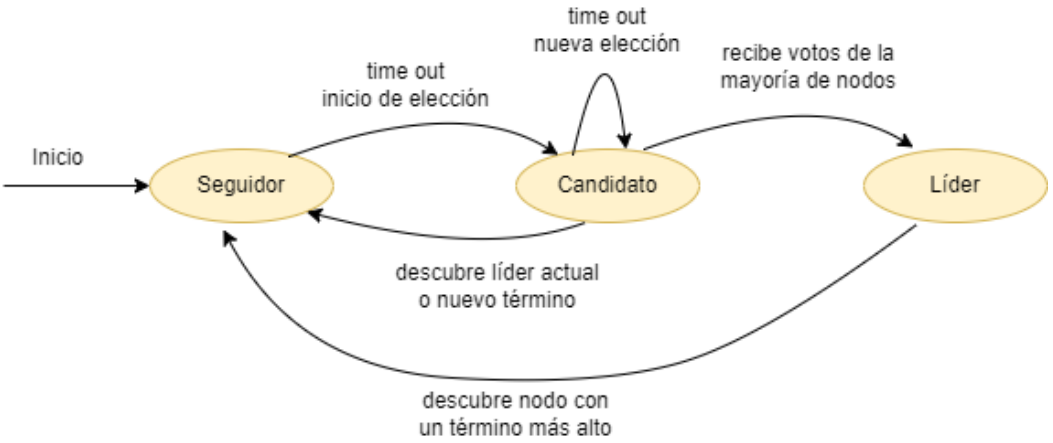
Los atributos que componen el estado de un nodo son:

- Lista de **nodos** que forman parte del sistema
- **Identificador del líder** en ese momento, pudiendo ser -1 en caso de que no exista uno.
- **Número de votos recibidos**, en caso de que el nodo sea candidato.
- **Canales de comunicación** para comunicar cambios en los roles en el propio sistema, como *FollowerChan, LeaderChan* o *Heartbeat*.
- **Rol** del nodo, pudiendo ser el mismo “Follower”, “Candidate” o “Leader”.
- El **último mandato** conocido por el nodo.
- Índice del **nodo al que se ha votado** en el mandato.
- Índice del registro más alto conocido que se ha **comprometido** e índice más alto que se ha **aplicado** a la máquina de estados.
- Lista del siguiente registro a **enviar** para cada nodo. Se utilizará en estado “Leader” para enviar a los distintos nodos su log.
- Lista del último registro **aplicado** a la máquina de estados para cada nodo.

En la siguiente práctica incluimos el Log, un vector de operaciones que forman el estado del sistema. La finalidad del algoritmo RAFT es mantener el log consistente en los distintos nodos del sistema distribuido.

2.1. Coordinador del RAFT

El coordinador se ejecuta como una gorutina y se encarga de gestionar las acciones que realiza el nodo. Estas son distintas en función del rol que tenga asignado en ese momento. El rol que tiene el nodo en el sistema distribuido está descrito por el siguiente autómata.



Observamos como en un comienzo todos los nodos son *Followers*, cambiando a estado candidato una vez venza un temporizador. Si este gana la elección será el nuevo *Leader*. Las distintas operaciones correspondientes a los estados se describen a continuación:

Para todos los roles, si vemos que el índice más alto comprometido es mayor que el último índice aplicado, lo aumentaremos en una unidad. A partir de aquí, el comportamiento cambia para cada rol.

2.1.1. Follower

El nodo en estado follower escucha dos canales distintos:

1. **Heartbeat:** recibe *true* si el líder hace una llamada a RPC *AppendEntries* sin contenido en el campo *Entry*, siempre y cuando el término del nodo que hace la llamada RPC se mayor que el propio.
2. **Timeout:** generamos un timeout aleatorio que indica el tiempo máximo que el nodo esperará antes de comenzar una nueva elección y presentarse como candidato.

Por tanto, si el timeout vence antes de que llegue un nuevo heartbeat, cambiamos de rol a "*Candidate"* y pondremos el identificador del líder como falso (-1).

2.1.2. Candidate

El nodo candidato realizará un procedimiento en el que pide el voto de los otros nodos para convertirse en líder. Primero aumenta su término, se vota a sí mismo e invoca a la RPC *PedirVoto* de los nodos del sistema con argumentos: su propio término e identificador así como el índice y término del último registro que conoce.

Una vez hecha la petición, escucha tres posibles mensajes: que siga siendo *Follower*, ya sea por resultado de la elección o por recibir un Heartbeat; que se ha convertido en *Leader*; o que ha vencido el timeout de la elección sin una decisión, por lo que tendrá que comenzar una nueva elección.

A medida que el candidato recibe respuestas de los nodos comprueba que el término indicado en la respuesta es menor o igual que el propio. Si se cumple esta condición, suma un voto y cuando el nº de votos sea mayoría enviará por un canal un mensaje que indica al proceso principal que ha ganado la elección. En caso de que el término sea menor, enviará un mensaje usando el canal *FollowerChan* en el que se le indicará que cambie su rol a *Follower*.

2.1.3. Leader

El nodo líder envía periódicamente un heartbeat al resto de nodos al tiempo que comprueba que es el *"líder legítimo"*. Para ello utilizará la llamada RPC *AppendEntries*. Si en la respuesta ve que el término del otro nodo es mayor que el suyo, cambiará su término y enviará un mensaje utilizando el canal *FollowerChan,* por el que el líder estará escuchando. Cuando reciba el mensaje, el nodo cambiará su rol a *Follower.*
Si el líder se cayera, al volver a comenzar la ejecución entraría con rol *Follower.*

2.2. Llamadas RPC

2.2.1. PedirVoto

Los nodos votan cuando un candidato invoca su función RPC *PedirVoto*. Lo harán bajo una premisa de *first-come-first-served*, es decir, concederán el voto al primer candidato que se lo pida. Concederán únicamente un voto por periodo electoral, y sólo si el término del candidato es mayor que el suyo. La respuesta del nodo contendrá su término y si se ha concedido el voto o no. En el caso de que el rol del votante sea distinto de *Follower* enviará un mensaje al proceso RAFT principal mediante un canal en el que se le indicará que cambie su rol a *Follower*.

2.2.2. AppendEntries

Las llamadas en *AppendEntries* distinguen dos casos, el heartbeat y la operación. Cuando el nodo reciba una operación, la guardará en su log (o en este caso, la mostrará por pantalla).
En el caso de recibir un heartbeat, el nodo siempre responderá con su término. Si este es igual o menor que el del nodo invocador, lo reconocerá como líder, guardará el término y enviará *True* al canal *Heartbeat*. Además, si es menor y no es *Follower*, mandará un mensaje por el canal *FollowerChan* para indicar al nodo que cambie su rol.

2.2.3. SometerOperacionRaft

SometerOperacionRaft es el servicio que ofrece Raft para que los clientes envíen operaciones. La llamada está soportada por todos los nodos, pero sólo tendrá efecto si el nodo que la recibe es *Leader*. Si este es el caso, guardará en su log el índice en el que se encontrará y el mandato en el que se recibió, además de la propia operación.
Después, utilizando la llamada RPC *AppendEntries* con un timeout determinado, mandará esta misma información a todos los nodos de forma concurrente. Si la respuesta de la mayoría de los nodos es favorable, aumentaremos *CommitIndex* y la operación formará parte de la máquina de estados del sistema distribuido.

3. Descripción del sistema

Este sistema consta de **3 máquinas del clúster**, cada una de las cuales correrá un nodo con los que se ha verificado la correcta implementación del algoritmo de consenso RAFT.

Para poner a punto este sistema, se han generado un par de **claves criptográficas** asimétricas, compuesto por una clave privada y una clave pública. La clave pública es compartida entre las máquinas. Consideraremos máquina principal a aquella desde la que se lanzará el sistema, cada una de las dos máquinas restantes deben tener la **clave pública** de dicha máquina para que la principal pueda conectarse a ellas y viceversa. Estas claves se generan utilizando algoritmos de criptografía.

Se ha implementado una arquitectura distribuida en la que los tres procesos se ejecutan en máquinas distintas y se comunican entre sí a través de la red utilizando la biblioteca **Remote Procedure Call** (RPC). Cada proceso tiene la capacidad de invocar las llamadas de procedimiento remoto (RPC) de los otros nodos, permitiendo la ejecución de funciones y métodos en máquinas remotas de manera transparente. El proceso de comunicación se basa en el paradigma de llamada a procedimiento remoto, donde los programas pueden invocar funciones o procedimientos en máquinas remotas como si fueran locales. La **comunicación** entre los procesos se realiza a través de **puertos** específicos. Cada proceso utiliza un puerto designado para recibir solicitudes RPC entrantes y para enviar respuestas.

4. Testing

Para la validación del correcto funcionamiento del sistema, se ha diseñado un módulo de pruebas *testintegracionalraft1_test,* el cual incluye diversas pruebas para verificar que el comportamiento es el esperado y que no hay problemas en la ejecución distribuida.

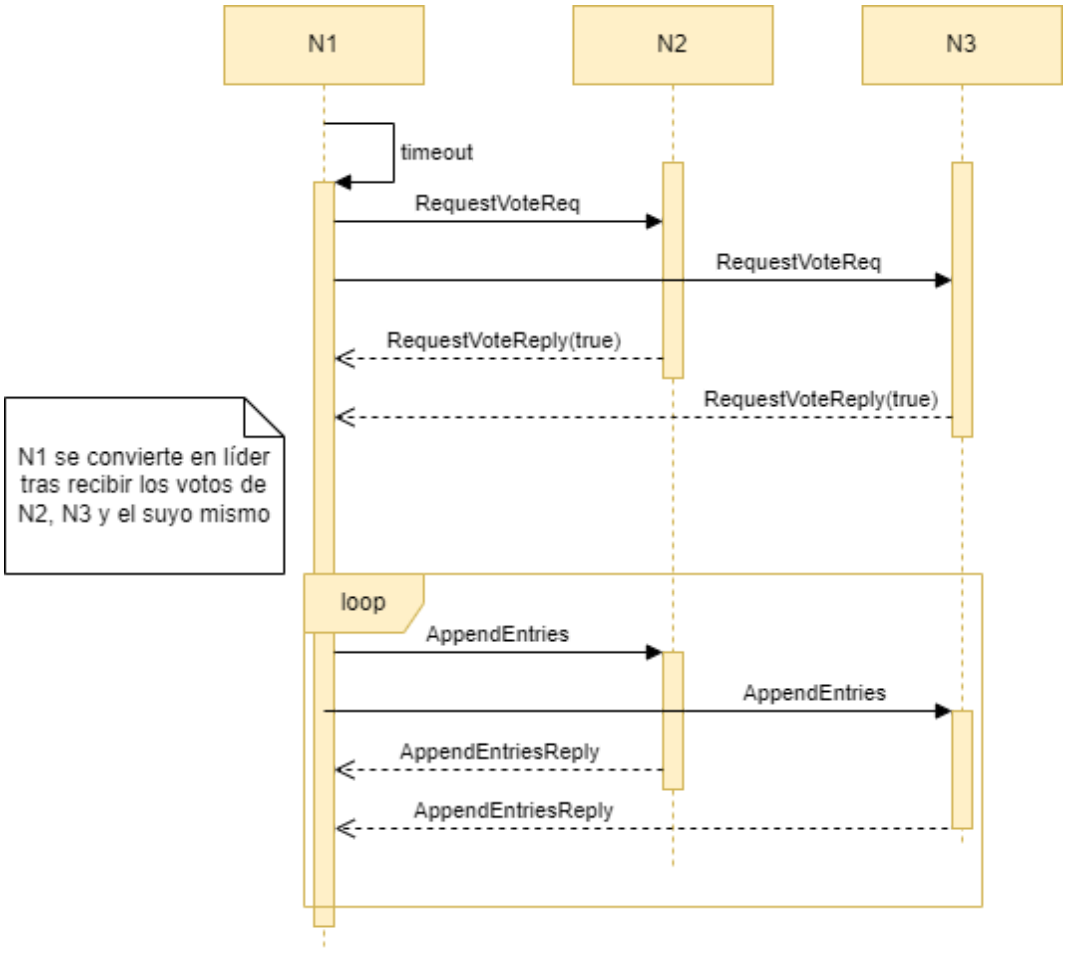
Para esta tercera práctica de la asignatura, se han de superar un total de cuatro test:

1. **Arranque y parada de un nodo:** En este primer test, inicialmente se realiza el despliegue u arranque de procesos distribuidos en los nodos Raft, para posteriormente detener dichos procesos. Es decir, se comprueba el correcto arranque y la parada de los nodos del sistema.
2. **Elección del primer líder:** En este test se inicia el sistema distribuido, se comprueba que hay un único líder y finalmente se detiene el sistema distribuido.
3. **Elección de nuevo líder tras un fallo anterior:** Tras iniciar el sistema distribuido y comprobarse que hay un líder único y válido, se provoca la caída de este con la llamada a la función *pararNodo.* Posteriormente, se relanza el líder y se esperan 5 segundos para garantizar que ha sido lanzado correctamente. Finalmente, se vuelve a comprobar que hay un líder y que este es único, para a continuación parar el sistema distribuido.
4. **Comprometimiento de tres operaciones estable:** Este último test busca verificar el correcto funcionamiento de las operaciones enviadas al líder, así como el comprometimiento de las mismas. Para ello, inicialmente se inicia el sistema distribuido, se verifica que haya un líder único, y mediante la función auxiliar del módulo de tests *checkOp,* se someten 3 operaciones. Finalmente se paran los procesos distribuidos.

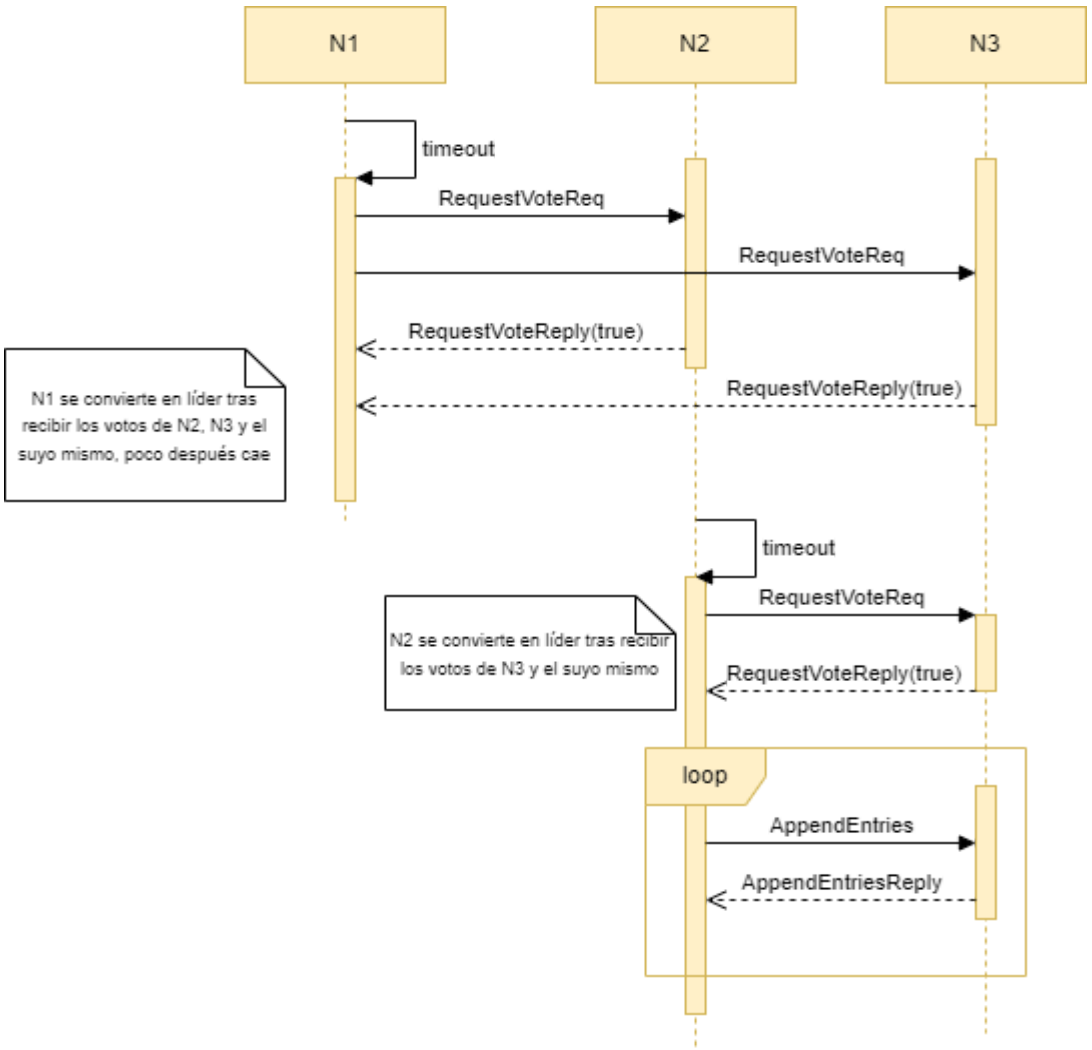
5. Diagramas de secuencia

Para representar de una forma más sencilla el comportamiento del algoritmo de consenso RAFT, se han diseñado distintos diagramas de secuencia para mostrar aquellos casos más representativos:

Funcionamiento estándar de RAFT



Caída de un líder y expiración de timeout para nueva elección



Proceso de elección con dos candidatos