Práctica 4: Raft (2ª Parte)

Sistemas Distribuidos (30221/39521)
Universidad de Zaragoza

Pedro Chaves – 844390

Beatriz Fetita – 840269

Curso 2025/2026

Índice

1. Resumen

En la práctica 4, se plantea construir un servicio de almacenamiento clave/valor, en memoria RAM, tolerante a fallos utilizando replicación distribuida basada en Raft, una solución de máquina de estados replicada mediante un algoritmo de consenso. La llamada RPC *raft.SometerOperacion*, enviada al líder, distribuye la operación al resto de réplicas mediante invocaciones RPC *AppendEntries*. Las referencias son el tema 5 de teoría y el documento original de Raft. La especificación de estado de las réplicas y las llamadas RPC de referencia se encuentran disponibles en el anexo al final del guión de la práctica.

2. Objetivo de la práctica

Esta práctica tiene como objetivo, partiendo de la implementación realizada en la práctica 3 con el algoritmo de elección de líder de Raft—que funciona en escenarios iniciales con fallos básicos—y el tratamiento de la llamada RPC `AppendEntry` en un funcionamiento sin fallos, avanzar e implementar:

- La solución completa de elección de líder, incluyendo la restricción de número de mandato y número de índice para seleccionar al mejor líder.
- La llamada RPC `SometerOperacion`, utilizada por los clientes del sistema replicado, que desencadena la replicación mediante `AppendEntries` desde el líder a los seguidores con avance del índice de entradas comprometidas.

Ambos objetivos deben funcionar correctamente en diferentes escenarios de fallos. Además, se aplicarán operaciones sobre una máquina de estados simple, con soporte para operaciones de lectura y escritura sobre un almacén de datos en RAM utilizando un `map` en Go.

Raft es un protocolo de consenso diseñado para garantizar consistencia en sistemas distribuidos. Su propósito principal es coordinar réplicas de datos entre nodos de un clúster distribuido, asegurando que todos acuerden un único estado incluso en presencia de fallos.

Raft se utiliza para:

1. Mantener la consistencia de datos en sistemas distribuidos.
2. Elegir un líder seguro para coordinar operaciones.
3. Tolerancia a fallos.
4. Replicación consistente entre réplicas.

El servicio de elección de líder implementado en esta práctica permite obtener siempre el mejor líder posible gracias a las restricciones aplicadas al otorgar votos. La llamada RPC `AppendEntry` permitirá tanto enviar operaciones a los seguidores para replicarlas como enviar *heartbeats*, notificando que el líder sigue activo. Si existe alguna operación previa, se enviarán también el índice y mandato anteriores para verificar consistencia.

Se han realizado pruebas tanto manuales como mediante *tests*, para validar el correcto funcionamiento.

3. Diseño e implementación del servicio de elección de líder

3.1. Descripción general

En esta práctica se ha desarrollado un algoritmo de elección de líder por votación mayoritaria. Este algoritmo permite elegir:

- el primer líder al arrancar los nodos;
- un nuevo líder en caso de fallo del líder actual.

El líder se convierte en aquel nodo que obtiene más votos. Los nodos votan de manera positiva o negativa según si el término del candidato es mayor, igual o menor que el suyo propio.

Un nodo seguidor pasa a estado *candidato* cuando transcurre un tiempo sin recibir un *heartbeat* del líder. Entonces inicia una votación enviando una RPC `PedirVoto` al resto de nodos con:

- su identificador,
- su término actual,
- el índice y mandato de la última operación del log (si lo hubiera).

El nodo que recibe la petición comprueba:

1. Si el término del candidato es mayor, actualiza su término.
2. Si el candidato está al menos tan actualizado como él, comparando mandato e índice.
3. Si ya ha votado o no en este término.

El algoritmo garantiza que el líder es siempre el nodo más actualizado entre los que obtienen mayoría.

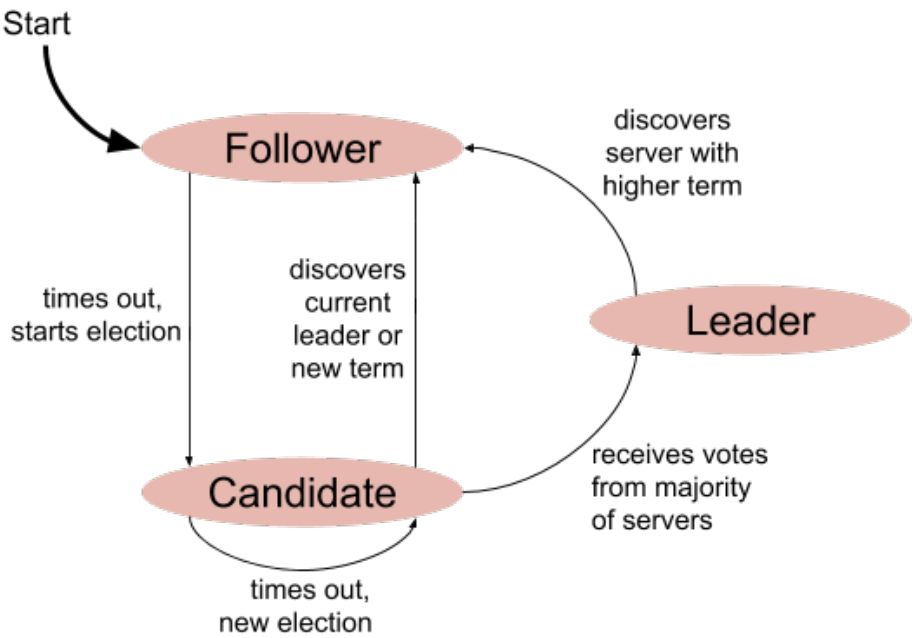

Figura 1: Diagrama de máquina de estados de Raft.

3.2. Mecanismo de temporización

El sistema utiliza el paquete estándar `time` y `math/rand` para establecer tiempos de expiración aleatorios. El intervalo de espera entre latidos es inferior a 20 por segundo, y el periodo máximo de elección es de 2.5 segundos, con pequeñas variaciones aleatorias entre nodos.

Esto evita sincronizaciones simultáneas que impedirían la elección de un líder.

4. Implementación de la llamada RPC AppendEntry (sin fallos)

4.1. Descripción del proceso

La llamada RPC `AppendEntry` es utilizada por el líder tanto para enviar *heartbeats* como para replicar operaciones.

El líder enviará *heartbeats* cada cierto tiempo con un `LogEntry` vacío. Cuando haya operaciones pendientes de enviar a un nodo, se enviará el siguiente `LogEntry` correspondiente a su `NextIndex`.

Cuando existen operaciones en el log, se enviarán también:

- el índice previo,
- el mandato previo,

para que el nodo receptor verifique si la operación es consistente con su propio log. En caso contrario, la operación será rechazada y el líder deberá retroceder en el log hasta encontrar una posición consistente.

El *heartbeat*, al no incluir operaciones, simplemente notifica al seguidor que el líder sigue activo, evitando que expire su temporizador de elección.

La interfaz `SometerOperacionRaft` permite a los clientes solicitar la ejecución de una operación. El líder la añade a su log y la replica entre los nodos hasta obtener confirmación de la mayoría, actualizando su `CommitIndex`.

func (nr *NodoRaft) AppendEntries(args *ArgAppendEntries, results *Results) error

Esta funcion la ejecuta un nodo cuando le llega una RPC NodoRaft.AppendEntries por dos motivos:

1. Para enviarle un heartbeat con el LogEntry vacio
2. Para enviarle una operacion para que la aplique en su maquina de estados, con una operacion en el LogEntry
   Respondo enviandole mi termino, y si he hecho la operacion con **Success=true** porque soy un follower de el.a
3. Se comprueba si tiene un termino menor que el mio
   i. El termino de la respuesta sera el mio **currentTerm** , y le digo que no he hecho la operacion **results.Success=false** porque si tiene un término menor que el mío no puede ser mi lider.
4. Si tiene un termino igual o mayor que el mio: puede ser mi lider
   i. Actualizo mi termino al suyo, y me guardo que es mi lider y devuelvo en la respuesta mi término actualizado.
   ii. Compruebo si en el AppendEntries en el LogEntry hay una operacion
      a. Si la hay, añado las entradas a mi Log
   iii. Miro si el termino del emisor es mayor que el mio
      a. Miro si era leader, que entonces me tengo que conveirtir en follower porque yo no puedo ser lider. Me notifico con **nr.serFollower<-true**
      b. Si no era lider
         a. Miro si el lider me envía un commit mayor al mio
            a. Si es mayor, actualizo mi commitIndex al suyo
         b. Notifico que he recibido un heartbeat con **nr.heartbeatReceived<-true**
         c. Respondo que he hecho la operacion con **results.Success=true**
   iv. Si no lo es, me mantengo como estoy y notifico que he recibido un heartbeat con **nr.heartbeatReceived<-true** y respondo que he hecho la operacion con **results.Success=true**

Figura 2: Diagrama del proceso AppendEntries.

4.2. Consideraciones de sincronización

Durante el envío de latidos, cada gorutina asociada a un nodo remoto utiliza un *timeout* configurado mediante la función auxiliar `CallTimeout`. Si un nodo no responde dentro del intervalo, se considera inactivo, y el sistema continúa con el resto de réplicas.

Esta gestión concurrente garantiza la disponibilidad del servicio incluso ante caídas temporales.

5. Validación y pruebas realizadas

5.1. Pruebas básicas

Para verificar el correcto funcionamiento se realizaron:

- **Pruebas manuales**: ejecutando nodos en distintas máquinas, observando mediante *prints* el comportamiento de cada nodo, incluso bajo fallos provocados.
- **Tests automáticos**: implementados en el archivo `testintegracionraft1test.go`.

Los tests verifican:

1. Acuerdo de varias entradas del log aunque un nodo se desconecte (grupo de 3).
2. Imposibilidad de acuerdo si se desconectan 2 nodos de 3.
3. Envío concurrente de 5 operaciones y avance correcto del índice del registro.

Una vez implementados los componentes faltantes del esqueleto entregado por el profesorado, se realizaron depuraciones hasta superar todos los tests.

6. Conclusiones

La implementación completa del algoritmo Raft desarrollada en esta práctica ha permitido profundizar en el funcionamiento interno de los sistemas de consenso y en los mecanismos necesarios para garantizar la coherencia en entornos distribuidos.

En primer lugar, la realización del algoritmo de elección de líder con las restricciones de mandato e índice ha demostrado la importancia de seleccionar siempre el nodo más actualizado para evitar inconsistencias en el sistema. Esto garantiza que el nuevo líder pueda continuar de manera segura con la replicación del log sin riesgo de sobrescribir operaciones válidas.

En segundo lugar, la implementación y análisis detallado de la llamada RPC `AppendEntries` ha puesto de manifiesto el papel fundamental que juegan tanto los *heartbeats* como la replicación de operaciones en la estabilidad del clúster. El manejo adecuado del `NextIndex`, la verificación de consistencia y la actualización del `CommitIndex` han resultado esenciales para lograr un comportamiento correcto incluso en presencia de fallos.

Las pruebas manuales y los tests de integración han permitido validar el funcionamiento global del sistema, evidenciando que los mecanismos implementados funcionan adecuadamente bajo distintos escenarios, incluyendo desconexión de nodos, concurrencia en las operaciones y variaciones de términos. Estas pruebas han sido clave para depurar y reforzar la robustez de la solución.

En conjunto, esta práctica ha facilitado una comprensión más profunda de los sistemas distribuidos y ha mostrado cómo un protocolo relativamente compacto como Raft integra conceptos complejos de consenso, replicación y tolerancia a fallos de manera clara y modular.

7. Referencias

- Diego Ongaro and John Ousterhout, *In Search of an Understandable Consensus Algorithm.*
- Documentación oficial de Raft: `https://raft.github.io/`
- Material docente de la asignatura Sistemas Distribuidos, Universidad de Zaragoza.