SISTEMAS DISTRIBUIDOS

PRÁCTICA 4

874055 - Ariana Porroche Llorén
871627 - Irene Pascual Albericio

ÍNDICE

Introducción

En esta práctica se continúa el código de Raft para poder proporcionar consistencia y disponibilidad de fallos, incluso en escenarios donde existen fallos o caídas de nodos. Esta solución permite que múltiples nodos trabajen de forma coordinada.

Se permiten procesar operaciones tanto de lectura como de escritura, las cuales son enviadas al nodo líder, que actúa como coordinador principal del sistema. Una vez el líder recibe las operaciones, las replica entre los nodos seguidores a través de llamadas RPC AppendEntries, y compromete las entradas del registro una vez que se haya obtenido una aprobación de una mayoría simple del total de nodos.

Es importante destacar que cuando una entrada no llega a ser comprometida, se persiste para intentar que se acabe comprometiendo esa entrada (aunque sigan llegando nuevas operaciones para someter, se intentarán comprometer las que no estaban comprometidas previamente y las nuevas operaciones). Por tanto, se asegura que las entradas pendientes no se pierdan y sean eventualmente aplicadas.

Para controlar los cambios de líder, cuando un nuevo líder es elegido, este debe encargarse de replicar las entradas que existan en su registro (e intentar que sean comprometidas esas entradas). Con esto logramos que el sistema tenga un registro íntegro y sincronizado.

A continuación vamos a explicar partes importantes del código donde se realizan todas estas partes mencionadas.

Diseño Raft:

Primero de todo, vamos a explicar los índices:

- **SigIndice**: es un vector donde el líder puede ver el siguiente índice que debe enviar a cada seguidor.
  Al principio, el líder asume que cada seguidor tiene un registro vacío, así que SigIndice[i] apunta al final del registro del líder.
  Si el seguidor rechaza una entrada, el líder retrocede su índice para enviarle entradas anteriores y alinear los registros.
- **MatchIndice**: es un vector donde el líder puede ver el índice más alto que ha sido replicado con éxito de cada seguidor.
  Este índice se actualiza cuando un seguidor confirma que ha aceptado una entrada específica.
- **IndiceAntReg**: es un índice que es enviado como parte del mensaje AppendEntries, y representa el índice anterior a la entrada que el líder está intentando replicar. Sirve para que el seguidor verifique que su registro es consistente hasta ese punto.
- **IndiceComprometido**: es un índice que indica la entrada más reciente que ha sido comprometida (es decir, replicada por una mayoría simple de nodos). Todas las entradas con índices menores o iguales a este, se consideran aplicadas en la máquina de estados.

Cuando el líder recibe una **operación para someter**, utiliza la operación de "**enviarLatidos**" para gestionar el envío de las entradas del registro al resto de nodos (si no hay entradas que enviar, simplemente enviará un latido para avisar a los seguidores que no se ha caído el líder).

```
func (nr *NodoRaft) enviarLatidos() {
    var results Results
    for i := 0; i < len(nr.Nodos); i++ {
        if i != nr.Yo {
            if nr.SigIndice[i] <= nr.IndiceReg {
                entrada := TipoReg{
                    Indice:    nr.SigIndice[i],
                    Mandato:   nr.Reg[nr.SigIndice[i]].Mandato,
                    Operacion: nr.Reg[nr.SigIndice[i]].Operacion,
                }
                args := ArgAppendEntries{
                    Mandato:         nr.MandatoActual,
                    IdLider:         nr.Yo,
                    Entradas:        entrada,
                    IndiceAntReg:    nr.SigIndice[i] - 1,
                    MandatoAntReg:   nr.Reg[nr.SigIndice[i]-1].Mandato,
                    IndiceCompromiso: nr.IndiceComprometido,
                }
                go nr.enviar1Latido(i, &args, &results)
            } else {
                // Solo envía un latido si no hay entradas pendientes
                args := ArgAppendEntries{
                    Mandato:         nr.MandatoActual,
                    IdLider:         nr.Yo,
                    Entradas:        TipoReg{},
                    IndiceCompromiso: nr.IndiceComprometido,
                }
                go nr.enviar1Latido(i, &args, &results)
            }
        }
    }
}
```

Los seguidores reciben el latido de corazón mediante **AppendEntries**, y si se detecta que los datos del latido de corazón contienen alguna operación, se procede a **verificar**: si los datos recibidos muestran que el líder está desactualizado y si el índice anterior proporcionado por el líder es válido, si el índice actual del registro del seguidor no está desactualizado o incompleto, y si el registro es inconsistente (el mandato de la entrada del seguidor no coincide con el mandato enviado por el líder).

En cualquier caso donde esto se cumpla, no se guardará el registro, y por tanto no se notificará al líder de ello para que tenga un voto más para su mayoría simple.

```
func (nr *NodoRaft) AppendEntries(args *ArgAppendEntries, results *Results) error {
    if args.Entradas != (TipoReg{}) { // Operación a replicar
        nr.Logger.Println("AppendEntries1: recibida la ENTRADA:", args.Entradas)

        if args.Mandato < nr.MandatoActual { // El líder está desactualizado
            results.Exito = false
        } else if args.IndiceAntReg >= 0 &&
                (nr.IndiceReg < args.IndiceAntReg ||
                 nr.Reg[args.IndiceAntReg].Mandato != args.MandatoAntReg) {
            results.Exito = false // Registro inconsistente
        } else {
            nr.Reg = append(nr.Reg[:args.IndiceAntReg+1], args.Entradas)
            nr.IndiceReg++
            results.Exito = true
        }
    }
    return nil
}
```

Una vez el **líder** va recibiendo **confirmaciones** de que el resto de nodos seguidores han guardado la entrada, aumenta el número de confirmaciones. Cuando haya llegado a una mayoría simple, comprometerá la entrada y restaurará los valores de nuevo ("**confirmacionesCompromiso**" se pone a 1 ya que tiene su propia confirmación).

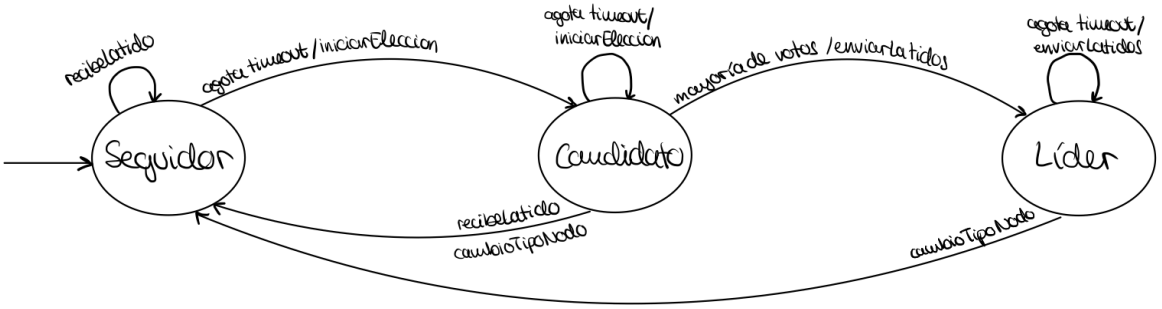
```
if results.Exito {
    nr.MatchIndice[nodo]++
    nr.SigIndice[nodo]++

    if nr.IndiceComprometido+1 == args.Entradas.Indice {
        nr.ConfirmacionesCompromiso++
        if nr.ConfirmacionesCompromiso > len(nr.Nodos)/2 {
            nr.IndiceComprometido++
            nr.ConfirmacionesCompromiso = 1
            nr.CanalAplicarOperacion <- AplicaOperacion{
                Indice:    args.Entradas.Indice,
                Operacion: args.Entradas.Operacion,
            }
        }
    }
}
```

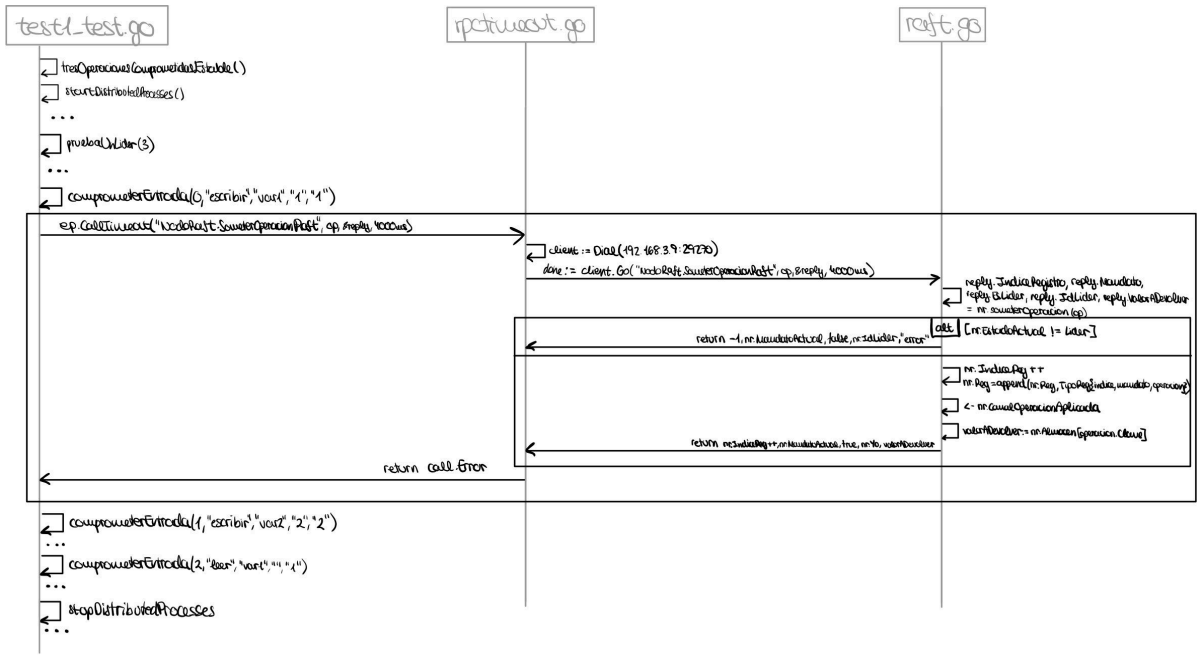
El **almacén** en Raft es una máquina de estados replicada, implementada como un mapa clave/valor (**map[string]string**), donde se aplican las operaciones comprometidas. Solo las entradas que han sido replicadas y comprometidas por una mayoría simple de nodos se aplican al almacén, lo que garantiza consistencia en todo el sistema. Las operaciones de escritura actualizan o añaden valores en el almacén, mientras que las lecturas devuelven el valor asociado a una clave específica. Cuando un nodo recibe una operación comprometida, esta se procesa a través del CanalAplicarOperacion y se ejecuta en su almacén, asegurando que el estado sea el mismo en todas las réplicas. Si un nodo falla y se reincorpora, su almacén se sincroniza con el registro del líder mediante la replicación de las entradas pendientes, manteniendo la integridad del sistema.

Diagramas de secuencia:

La **máquina de estados** que hemos seguido para esta práctica es la misma que para la anterior. Únicamente hemos corregido un fallo en el líder, que cuando le llegaba la señal de cambioTipoNodo, se iba a Candidato, pero en realidad se tiene que ir a Seguidor.

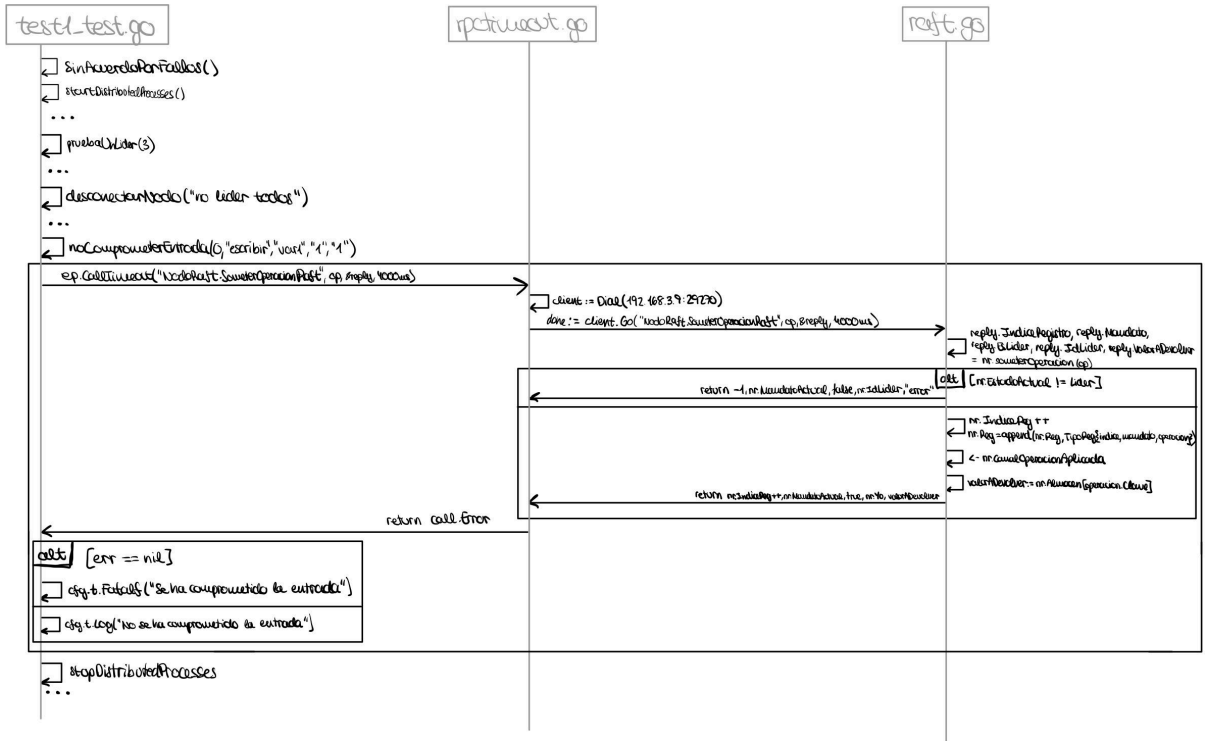


Ahora vamos a presentar los distintos diagramas de secuencia, uno por cada test. El **Test4**, **tresOperacionesComprometidasEstable**, se encarga de someter 3 operaciones y comprobar que han sido comprometidas y aplicadas a la máquina de estados correctamente. Para ello, primero inicia los procesos distribuidos en las 3 réplicas, para lo cual crea la sesión ssh y el nodo en sí, tal y como explicamos en los diagramas de secuencia de la práctica 3. Una vez creados los 3 nodos, y puestos en marcha con su máquina de estados, comprobamos que se haya elegido un líder. Seguidamente, compromete las entradas una a una, enviándoselas al líder. Hemos decidido comprometer 2 operaciones de escritura y una de lectura. Para comprometer una entrada, hace la llamada SometerOperacionRaft, a la que le pasa la operación que desea someter, en nuestro caso, (0, “escribir”, “variable1”, “1”), una variable reply dónde nos devuelve el resultado, y el tiempo máximo, en nuestro caso, el tiempo de 4 latidos, por si algún nodo está caído y necesita reenviar hasta comprometer la entrada. Una vez el líder nos ha devuelto el resultado, comprobamos si era el esperado, y si no, devolvemos un error.



El **Test5**, **AcuerdoApesarDeSeguidor**, es similar al Test4, pero una vez se ha elegido el líder, desconecta un nodo de los que no sean líder, el primero que encuentre. Ahora sólo habrá 2 nodos activos, pero suficientes para que pueda comprometer las entradas. Le mandamos a comprometer las mismas entradas que en el Test4.

El **Test6**, **SinAcuerdoPorFallos**, es también similar a los anteriores, pero en este caso, en lugar de desconectar sólo un nodo, desconecta a todos los nodos que no sean el líder, por lo que nunca habrá mayoría y no podrá comprometer las entradas. Aquí probamos a someter una única operación, y si el CallTimeout nos ha devuelto error, es porque no ha podido someter la operación y el test ha sido correcto. En el caso de que no nos devuelva error, lanzamos nosotros un error en el test.



El **Test7**, **SometerConcurrentementeOperaciones**, en lugar de someter una operación y esperar a que sea comprometida, lanza una goroutine por cada comprometerEntrada, de tal manera que el líder mete varias entradas en el registro, pero tarda varios latidos a comprometer las entradas. Si comprobamos los logs de cada máquina, podemos ver cómo se han comprometido correctamente.