Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

sisdis-pr-raft

RAFT

Autor 1:	Toral Pallás, Héctor - 798095
Autor 2:	Pardos Blesa, Javier - 698910
Grado:	Ingeniería Informática
Curso:	2022-2023

15 de noviembre de 2022

Índice

1. Introducción

En esta práctica se plantea realizar una sistema de almacenamiento tolerante a fallos aplicando una serie de mejoras que consisten en completar el algoritmo de raft para hacerlo robusto frente a varios casos de fallo. Este sistema de almacenamiento de clave/valor se va a implementar añadiendo una maquina de estados propia a cada nodo del sistema que acumulará las operaciones que se vayan comprometiendo, todo ello haciendo uso de las llamadas a procedimientos remoto.

2. Pruebas de validación

Para verificar la correcta ejecución del algoritmo de consenso RAFT, se han realizado varias pruebas que se enumeran a continuación:

- Test 'soloArranqueYparada': Comprueba que el nodo Raft puede arrancar.
- Test 'pruebaUnLider': Comprueba que una vez inicializado el cluster sin fallos se consigue un lider y se mantiene estable.
- Test 'falloAnteriorElegirNuevoLiderTest': Comprueba que tras haber eliminado/parado un lider previamente creado el cluster consigue encontrar un nuevo lider.
- Test 'tresOperacionesComprometidasEstable': Comprueba que tras haber elegido un lider, se pueden someter 3 operaciones de forma secuencial y son comprometidas con exito.
- Test 'AcuerdoApesarDeSeguidor': En el cluster de 3 nodos en funcionamineto tras la caída de un nodo seguidor se consigue someter y comprometer un conjunto de operaciones gracias al otro seguidor y, tras la recuperación de este, todos los nodos consiguen el estado estable correcto.
- Test 'SinAcuerdoPorFallos': Se comprueba que durante la caida de los 2 nodos seguidores, el lider no es capaz de comprometer nuevas entradas.
- Test 'SometerConcurrentementeOperaciones': Comprueba que las entradas en el índice de registro se comprometen de forma correcta.

3. Protocolos de intercambio de mensajes

En Raft se hace uso del mecanismo RPC (Remote Procedure Call).Para esta práctica, las RPC más importantes son PedirVoto y AppendEntries ya que en ellas reside todo el peso de la elección de lider y, tanto el añadir nuevas entradas a los nodos replicas como el notificar que el líder sigue con vida.

Elección de Lider

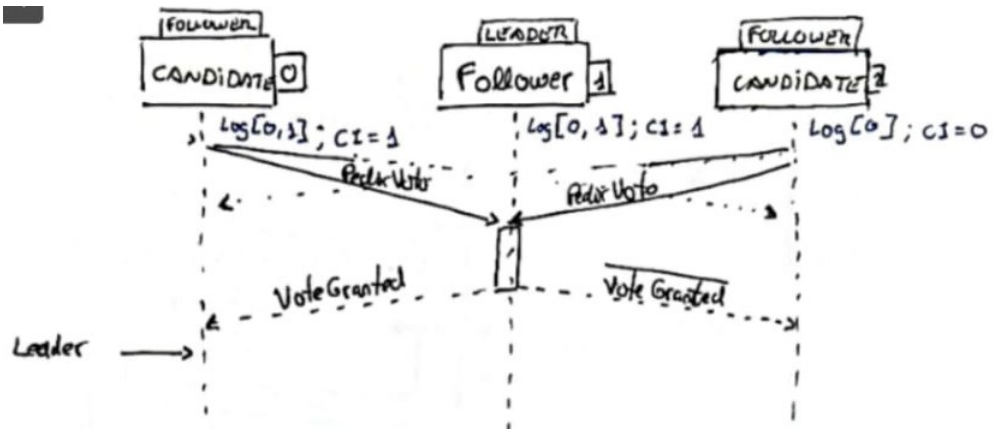

Figura 1: Nodo 0 gana la votación. TermOK y LogOK

AppendEntries

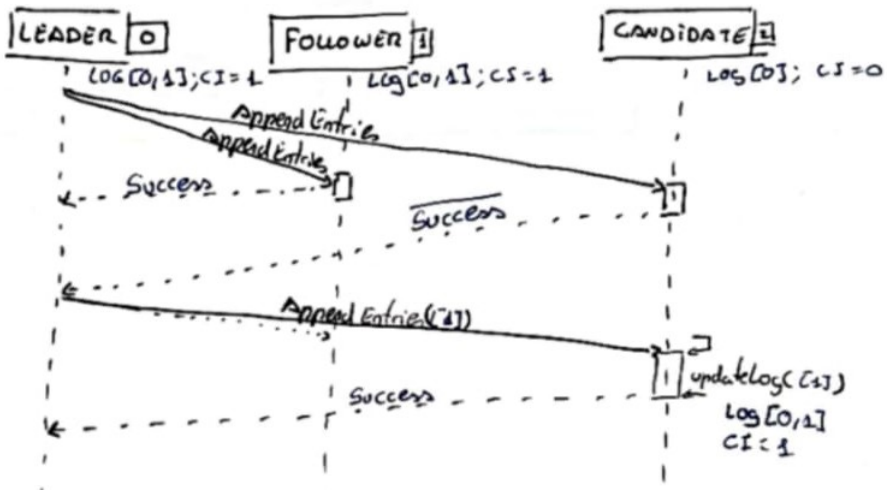

Figura 2: Candidate1 se vuelve Follower y compromete el elemento 1.

SometerOperaciónRaft

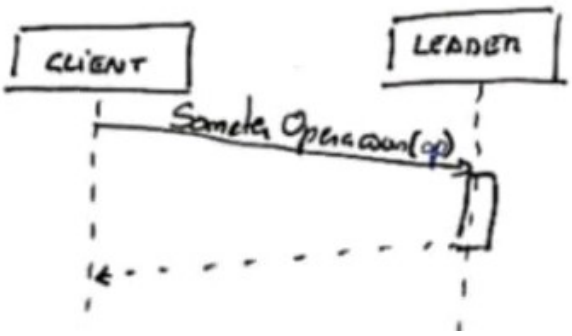

Figura 3: Un cliente somete una operación al nodo líder.

4. Eventos de fallo

Elección de Líder

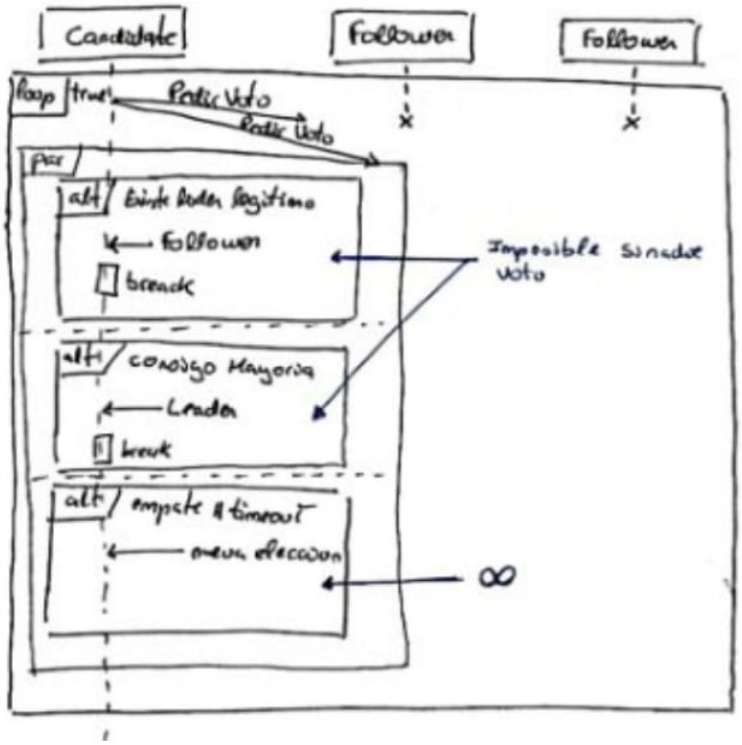

Figura 4: Candidato desde Term[0..N] hasta que algún nodo despierte y desbloquee la situación.

AppendEntries

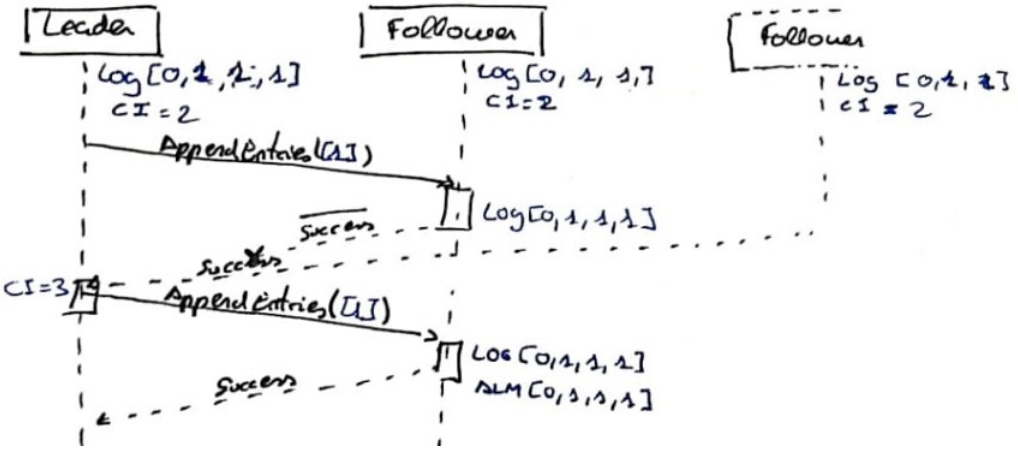

Figura 5: Se pierde un mensaje de confirmación y se repite la operación.

En esta situación se muestra un cliente sometiendo una operación y como el líder va enviando appendEntries en una situación normal.

Cuando el líder detecta una nueva entrada la manda a los followers, la actualizan en su registro y esperan a la siguiente envío del líder para comprobar si ha sido correctamente comprometida. En caso de algún fallo en este primer AppendEntries, el Líder comienza con la sincronización del follower para llegar a un estado estable y poder restaurar el estado bueno dentro del nodo follower. (ver en zona naranja).

Si por un casual el líder no es el bueno y recibe un Term ¿TermLider entonces este líder volverá a follower y la operación no podrá ser comprometida.

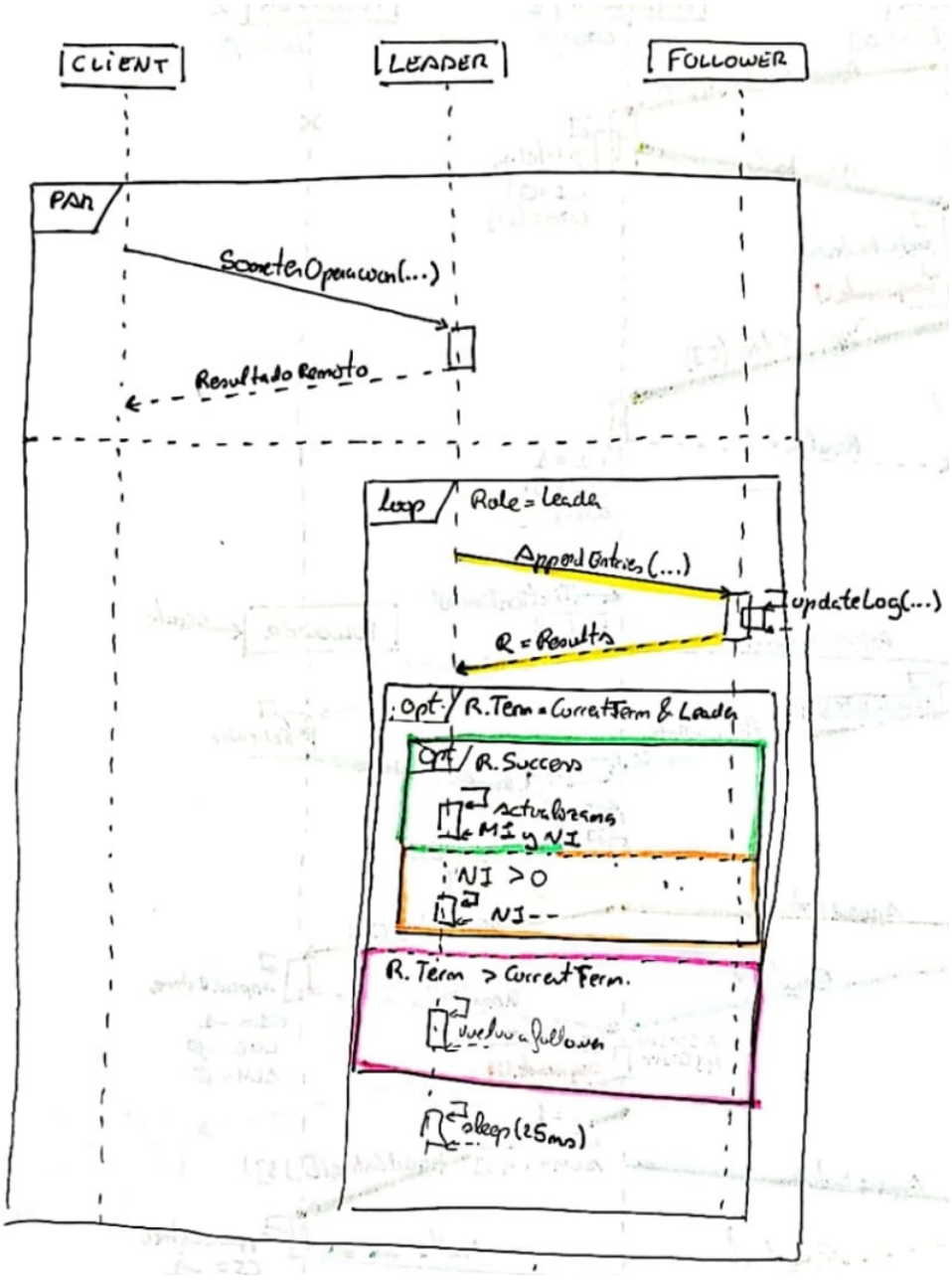

Figura 6: AppendEntries con intento de someter.

En la siguiente imagen se puede ver como va aumentando NextIndex (NI), MatchIndex(MI), CommitIndex (CI), el almacen o maquina de estado de las operaciones aplicadas además del log para cada uno de los nodos.

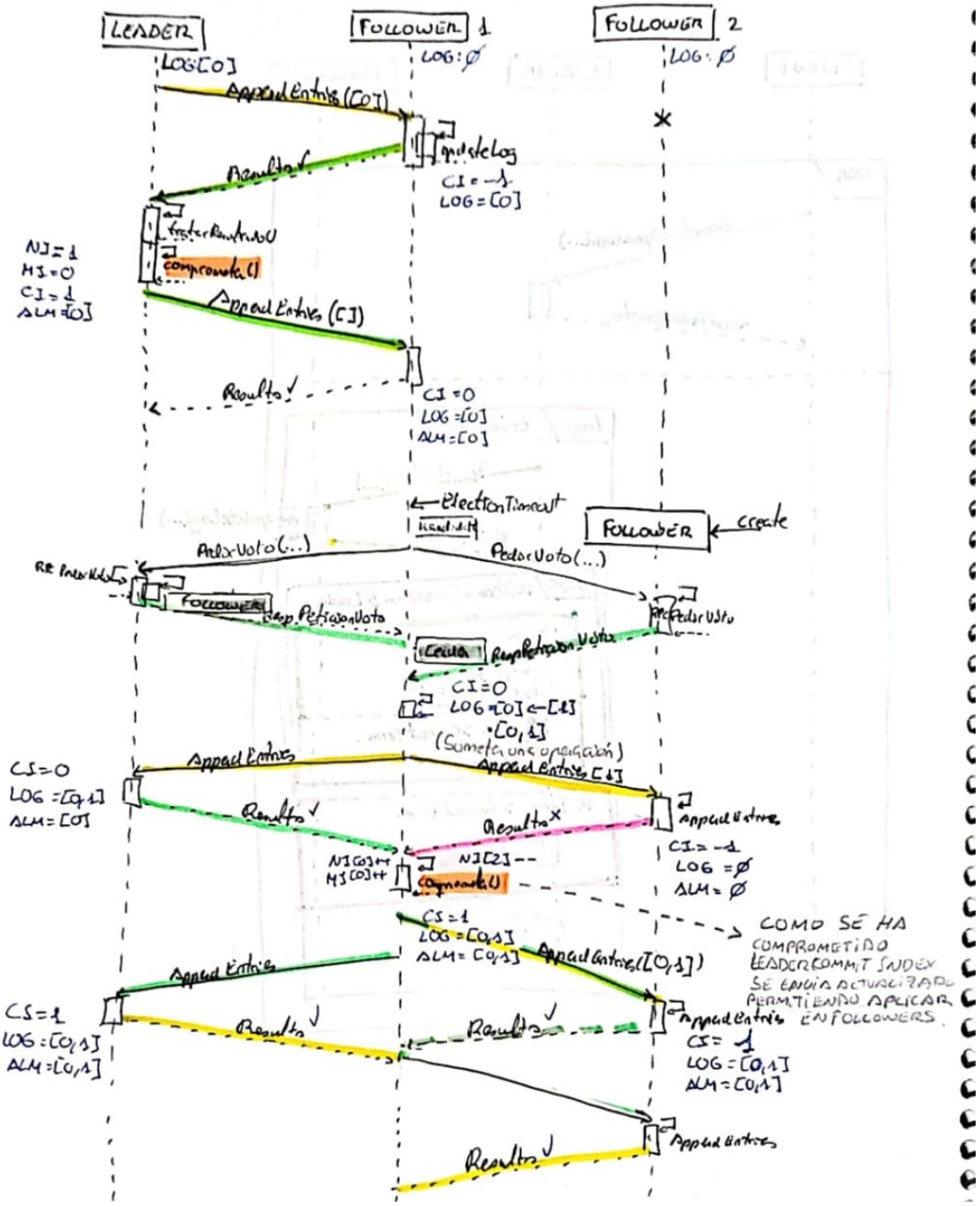

Figura 7: Append Entries sometiendo y comprometiendo