SISTEMAS DISTRIBUIDOS

PRÁCTICA 4

Pablo Moreno Muñoz 841972

Andrés Yubero Segura 842236

ÍNDICE

Introducción

En esta práctica se ha construido una máquina de estados distribuida a base de un diccionario clave valor y gracias al algoritmo tolerante a fallos de replicación distribuida de raft.

Elección del mejor líder en Raft completo

En esta práctica se ha desarrollado la solución completa para la elección de líder, incorporando la limitación del número del mandato y el número de índice para seleccionar al mejor líder.

Ahora los candidatos incluyen en la solicitud de voto en la función *enviarPeticionVoto()* tanto el mandato actual como el último índice de la última entrada del log.

Esto se debe a la necesidad de asegurar que un candidato que se postula como líder tenga una visión completa y actualizada del estado del registro distribuido antes de ser elegido.

Si el registro de un seguidor es más reciente que el del candidato, se le denegará el voto a dicho candidato. Para determinar cuál de dos registros es más reciente entre ellos, se contrastan el índice y el término de sus últimas entradas. Si estas últimas entradas tienen términos distintos, se considerará más reciente el registro con el término más alto. En caso de que los términos sean iguales, el registro cuya última entrada tenga un índice mayor será considerado más reciente.

Llamada RPC AppendEntries tolerante a fallos

El método *AppendEntries()* se utiliza para replicar registros entre nodos y mantener la consistencia del registro en todo el clúster. Esta función es llamada en *enviarLatido()*. Vamos a describir los aspectos clave de la implementación relacionados con la tolerancia a fallos:

1. Actualización de mandato y conversión a seguidor:
    - Cuando el líder envía un *AppendEntries*, el receptor verifica si el término del líder es mayor que el suyo.
    - Si es así, el receptor actualiza su mandato y se convierte en seguidor mediante el canal *chSeguidor*.
2. Verificación de Entradas Previas:
    - Antes de agregar nuevas entradas al registro, se verifica la existencia de las entradas previas.
    - Si no hay entradas previas en el registro o si las entradas coinciden, se procede.
3. Inserción de Nuevas Entradas:
    - Se determina el punto de inserción en el registro local para las nuevas entradas del líder y se insertan a partir de la última entrada
4. Compromiso de Entradas y Actualización del Índice de Commit:
    - Se verifica si el líder especifica un índice de compromiso (*LeaderCommit*) mayor al índice de compromiso actual (*CommitIndex*).

- Si es así, se actualiza el *CommitIndex* y se aplican entradas en la máquina de estados de cada nodo

5. Replicación de entradas aplicadas a la máquina de estados:
    - Se guarda cual es la última entrada aplicada y se comprueba cuántas entradas había sin aplicar a la máquina de estados
    - Para cada una de las entradas se manda la solicitud de aplicación por el canal *chAplicar* y esperamos a la confirmación.

Este método es crucial para garantizar la coherencia del registro y tolerar fallos, ya que aborda la replicación y la consistencia de las entradas entre los nodos

Todas las operaciones se aplican sobre una máquina de estados simple, que soporta operaciones de lectura y escritura sobre un almacén de datos implementado mediante un mapa de Golang.

Una vez el líder aplica la entrada a la máquina de estados mediante el canal de aplicaciones,devuelve el resultado al cliente y notifica a los seguidores que se ha comprometido la entrada y en casas latido que reciben comprueban el estado de su log.

Mantenimiento de la Consistencia:

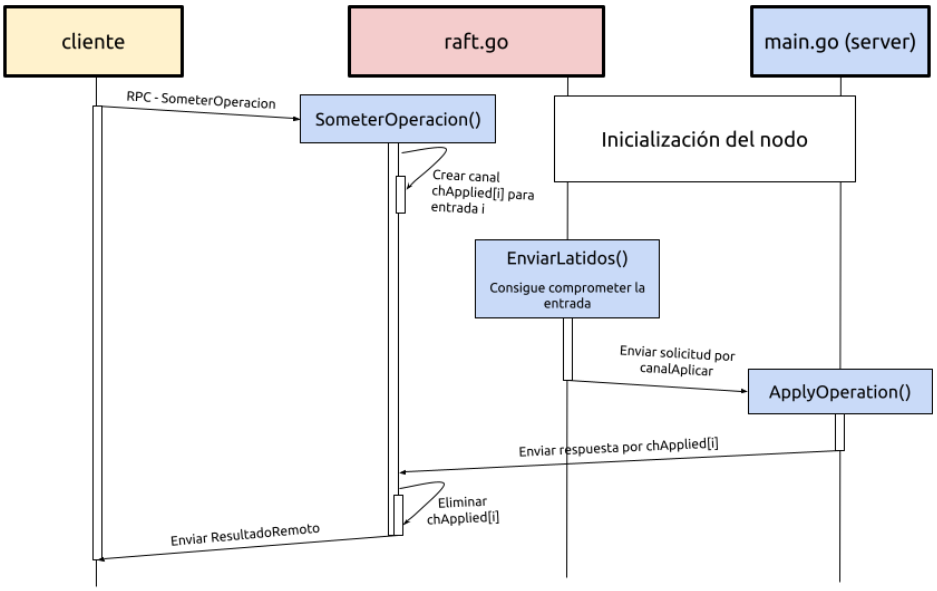
Para mantener la consistencia de los logs, el líder almacena, para cada servidor, el índice de la siguiente entrada del log que le debe enviar (*NextIndex*) y el índice de la entrada replicada más alta (MatchIndex)=NextIndex-1.

Se añaden a la llamada RPC "*AppendEntries()*" los argumentos *PrevLogIndex* y *PrevLogTerm*. Si el candidato no tiene esta entrada en su log, rechaza nuevas entradas devolviendo un valor falso.

Resolución de Inconsistencias:

Si el líder tiene nuevas entradas que enviar a un servidor, envía "*AppendEntries()*" con la entrada del log presente en *NextIndex*. Si el resultado es exitoso, se actualizan *NextIndex* y *MatchIndex* para ese servidor. En caso contrario, se decrementa `NextIndex` para ese servidor y se reintenta hasta que se resuelva la inconsistencia.

Aplicación de entradas a la máquina de estados



El cliente envía al líder una solicitud de operación y este crea el canal *chApplied* que servirá para aplicar a la máquina de estados esa operación en caso de que sea comprometida.

Luego en la operación *EnviarLatido* si hay mayoría se compromete este entrada y se envía la solicitud de aplicación a la máquina de estados al server por el canal *chAplicar*.

Este tras aplicarla responderá por el canal *chApplied[i]* donde i es el índice de la entrada a aplicar. Posteriormente se elimina el canal y se envía el resultado al cliente.

Tests de prueba

Para realizar esta pruebas hemos lanzado 3 nodos réplica, después esperamos cierto tiempo para dejar que se inicializan y desplieguen correctamente.

Test 5: Acuerdo a pesar de desconexión del seguidor

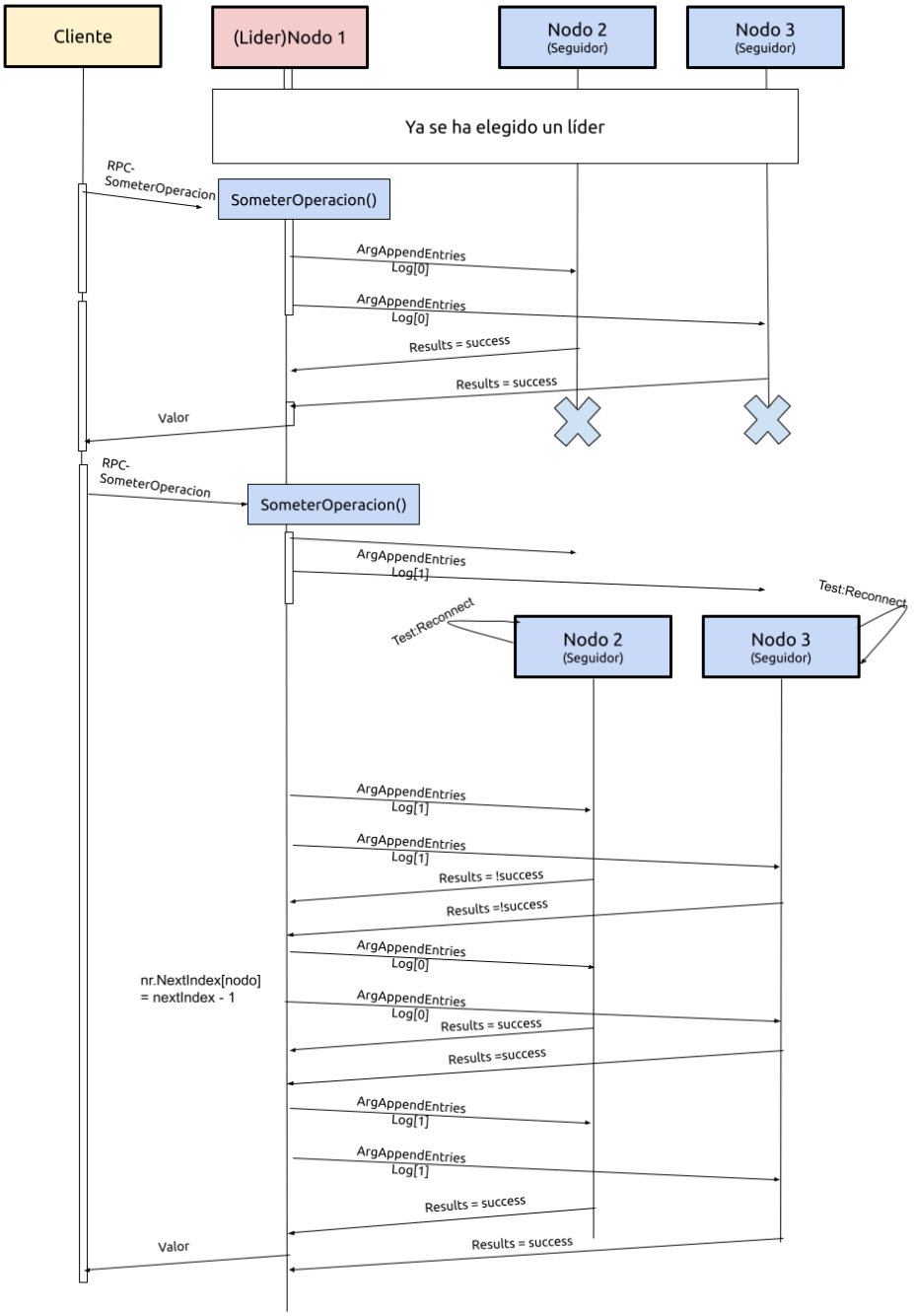
Se comprueba que hay un líder luego se somete una operación. Después se provoca un fallo de una de las réplicas. Con una réplica caída, se someten varias operaciones y se comprueba que se han consolidado.

Comprobamos el correcto avance del index de registro y que el valor devuelto es el correspondiente. Después se vuelve a reconectar la réplica caída, y se someten nuevas operaciones comprobando que se produce un consenso

Test 6: Sin acuerdo por fallos

Se comprueba que hay un líder luego se somete una operación. Después se provoca un fallo de dos seguidores.Luego sometemos varias operaciones y vemos que no se han logrado comprometer ya que no se puede llegar a una mayoría.

Después volvemos a reconectar a los seguidores desconectados, y volvemos a someter unas operaciones y comprobamos que estas se comprometen y como avanzan los índices del registro,



Test 7: Someter concurrentemente operaciones

Primero se comprueba que hay líder. Luego sometemos una operación y justo después sometemos 5 operaciones concurrentemente. Esperamos y comprobamos que se han consolidado todas, comprobamos el mandato y que el índice del registro coincide con todos los nodos.