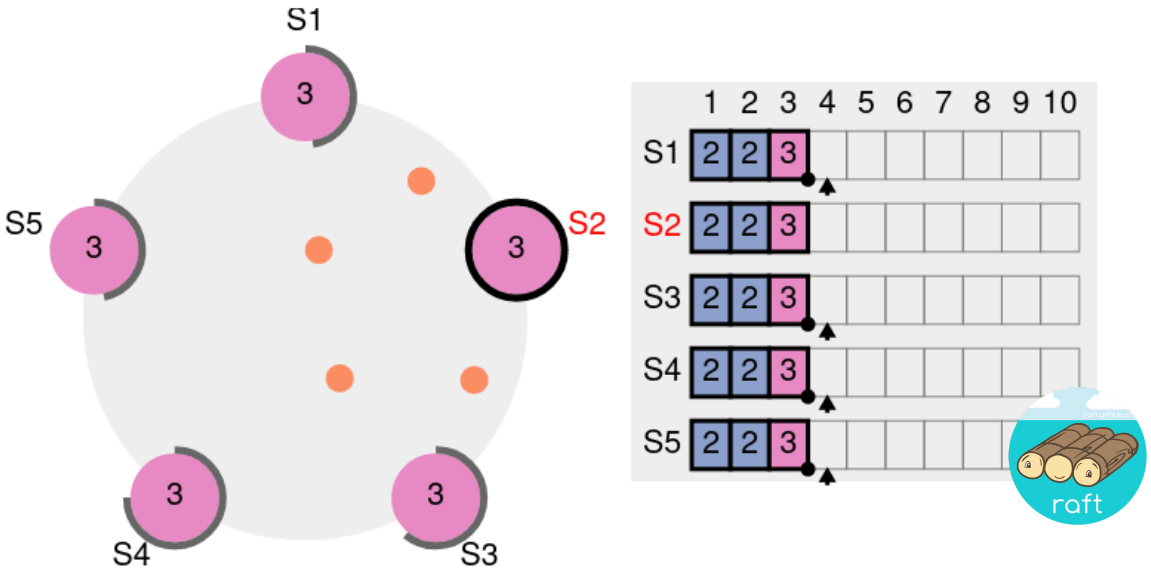
Práctica 3:
Raft 1ª parte



Sistemas Distribuidos

Universidad de Zaragoza, curso 2022/2023

Ayelen Nuño Gracia 799301

Loreto Matinero Angusto 796598

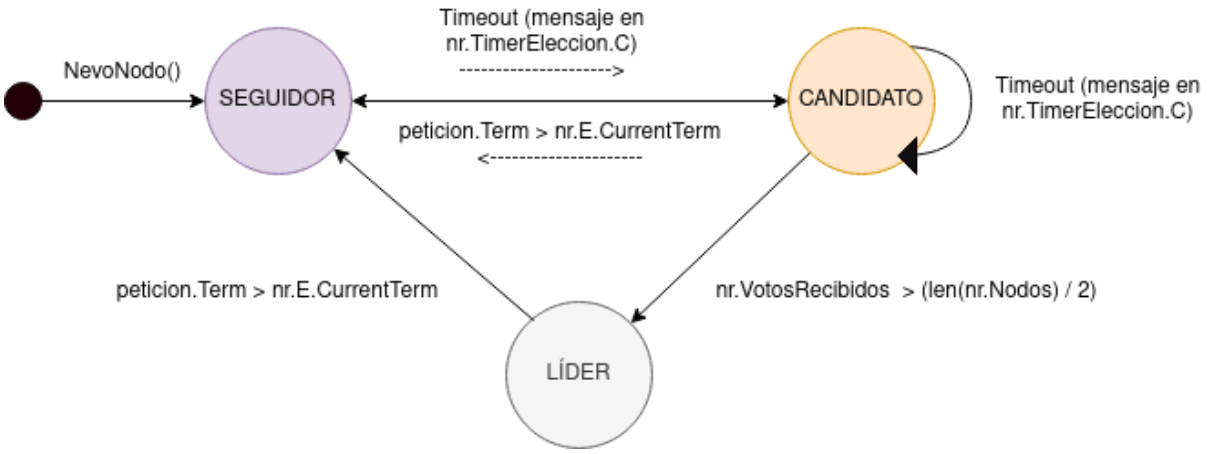
Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Descripción del problema:
En esta práctica se va a proceder a implementar una base del algoritmo de raft para la tolerancia a fallos, para ello se ha seguido el siguiente guión:
Para la implementación inicial se van a obviar los casos de fallo existentes para añadir entradas nuevas en las réplicas.
Algoritmo Raft Implementado:

En primer lugar se ha decidido crear las siguientes variables:

LIDER = “lider”
CANDIDATO = “candidato”
SEGUIDOR = “seguidor”

Las cuales nos permitirán gestionar el Roll de cada uno de nuestros nodos, en función de la situación en la que nos encontremos, en primera instancia todos los nodos que se creen se iniciaran como seguidores.
Esto lo harán a siguiendo el siguiente diagrama de estados:



Además se ha hecho uso de diversos structs, los cuales se comentan en el *Anexo 1*:

Ya conocemos todas las estructuras de datos que se han creado y utilizado a lo largo de este programa, ahora se van a explicar las funciones existentes y la labor que realizan cada una de ellas.

func **NuevoNodo**(nodos []rpctimeout.HostPort, yo int,
canalAplicarOperacion chan AplicaOperacion) *NodoRaft

En esta función se incializan todos los parámetros de nodoRaft, siguiendo las indicaciones de la tabla mostrada al principio de este documento en los casos que se indican. Esta función hace uso de:

func **inicializarEstado**(nr *NodoRaft)

la cual inicializa nr.E, parámetro definido anteriormente del nodoRaft.

La siguiente función definida es:

```
func (nr *NodoRaft) obtenerEstado() (int, int, bool, int)
```

que nos devolverá el id del nodo con el que ejecuta, el mandato en el que se encuentra, un booleano que indica si es el líder o no y el id del proceso que considera líder.

```
func (nr *NodoRaft) someterOperacion(operacion TipoOperacion) (int, int,
        bool, int, string)
```

Esta función devolverá false en caso de que no sea el líder, en caso contrario comenzará el procedimiento de consenso para saber si la operación puede comprometerse. Además devolverá los siguientes valores por parámetro:

- Primer valor indica el índice del registro se va a colocar la operación si consigue someterse.
- Segundo parámetro mandato en curso
- Tercero, tendrá el valor de true en caso de que el nodo crea ser el líder
- Cuarto, el id del lider en caso de que no sea el

La siguiente operación que se va a explicar es Pedir Voto, a continuación se muestra la cabecera:

```
func (nr *NodoRaft) PedirVoto(peticion *ArgsPeticionVoto,
        reply *RespuestaPeticionVoto)
```

Esta se ejecuta cuando un candidato nos ha solicitado que le votemos, lo primero que se realiza es comprobar si el candidato cumple las condiciones necesarios para darle nuestro voto, en caso de que así sea se le enviará un reply, con el campo voteGranted igualado a true. En caso contrario no se le concederá el voto.

```
func (nr *NodoRaft) enviarPeticionVoto(nodo int, args *ArgsPeticionVoto,
   reply *RespuestaPeticionVoto) bool
```

Aquí se observa la cabecera de la función que solicita la petición del voto, la ejecuta siempre un nodo en estado de candidato, y es donde comprobaremos si tenemos los funcion centes votos como para llegar a convertirnos en lider, en caso de que no sea asi, seguiremos esperando votos del resto de nodos, hasta tener los votos necesarios, o bien hasta que expira el timeout de TiempoElecicon. También es posible que nos llegue una petición con un mandato superior al nuestro, en cuyo caso pasaremos directamente al estado de seguidor.

A continuación explicaremos la funcionalidad de las funciones que envían y reciben los latidos, comenzamos por la que los recibe:

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

Lo primero que se realiza es comprobar que el término es igual o superior al nuestro, en caso contrario se responderá falso. los pasos de comprobación 2 y 3 indicados en la tabla del comienzo la cual explica el algoritmo se implementaran en pasos posteriores. El cuarto

paso consiste en añadir las nuevas entradas que nos envía el líder a nuestro Log, por último comprometemos las entradas que lo requieran, aquellas que nos ha dicho el líder que podemos comprometer.

La función que envía estos latidos, y por tanto que reexpide las solicitudes de los clientes a los diversos nodos existentes tiene la siguiente cabecera:

```
func (nr *NodoRaft) enviarAppendEntries(nodo int, args *ArgAppendEntries,
   reply *Results) bool
```

El primer paso a realizar es preparar los datos que le tenemos que enviar a los nodos para que estos puedan realizar las comprobaciones necesarias, al igual que actualizar su Log. Tras esto con la información disponible que se ha ido actualizando a través de los diversos mensajes de latido y respuesta que se intercambian entre los procesos a lo largo de la ejecución del programa, se calculará el tamaño que debe tener el Entries a enviar a este nodo, al igual que el contenido del mismo. Una vez realizado este paso se procederá a enviar el mensaje, y esperar la respuesta del mismo. Comprobaremos si somos capaces de comprometer la entrada, en función del número de nodos que nos han respondido de forma afirmativa. Si el número de nodos que han respondido positivamente a la actualización de su Log es mayor a la mitad del número de nodos comprometeremos la entrada.

```
func (nr *NodoRaft) Latir()
```

En esta función lanzaremos las gorutinas que se encargaran de ejecutar enviarAppendEntries para cada uno de los nodos activos y calcularán los datos que necesitan cada uno de ellos. En esta función es donde se tiene en cuenta que nosotros mismos (que somos el líder) no debemos enviarnos ningún mensaje.

Además se han implementado una serie de funciones que nos ayudan a realizar los cambios de Roll (Seguidor, Líder y Candidato).
La cabecera de la función convertirse a líder es la siguiente:

```
func (nr *NodoRaft) ConvertirseEnLider()
```

En esta función, al igual que en todas las funciones de conversion el primer paso a seguir es actualizar nuestro nr.Roll al correspondiente. Además, el servidor que se convierte en seguidor tendrá que actualizar su término con el que le mande el líder (este se pasa por parámetro en la función) y resetear tu variable nr.E.VotedFor.
También se tendrá que resetear el tiempo de elección y parar el tiempo de latido, esto último se hace por si anteriormente el servidor era líder..

Continuamos explicando la función de ConvertirseEnCandidato:

```
func (nr *NodoRaft) ConvertirseEnCandidato()
```

En este caso actualizamos nuestro Roll a CANDIDATO, y lo primero que se hace es votar por nosotros mismos, es decir el número de votos que tenemos para convertirnos en líder es igual a 1. Tras esto prepararemos los parámetros para la solicitud del voto e iniciaremos

la votación, enviandiole a cada uno de los servidores la petición a través de la función enviarPeticionVoto, cada uno de las funciones de solicitud se lanzarán en una gorutina diferente.
En prácticas posteriores en esta funcions de comprobará si la partición de voto ha llegado sin problemas y en caso contrario se reexpedira la solicitud. En última instancia se gestionan los timers por si fuese necesario comenzar una nueva elección.

Por último tenemos la función de ConvertirseEnLider cuya cabecera es la siguiente:

func (nr *NodoRaft) **ConvertirseEnLider**()

En primer lugar al convertirse uno de los nodos en líder tendrá que parar su nr.TimerEleccion, ya que habrá dejado de ser CANDIDATO o SEGUIDOR, y resetear el nr.TimerLatido para que comience a mandar latidos. También se tendrán que inicializar las siguientes variables: nr.Roll, con el roll correspondiente, LIDER en este caso; nr.IdLider con nr.Yo, ya que el nodo que se convierte es el que pasa a ser lider; nr.E.NextIndex apuntando al proximo indice de nuestro registro en el que se añadiran posibles nuevas entradas; y nr.E.MatchIndex, inicializado a 0 como indica la documentación.
Además, una vez un nodo se ha convertido en LIDER tiene que comenzar a enviar latidos a los demás servidores, esto se hace llamando a la función Latir(), explicada anteriormente.

En último lugar se va a explicar la función de gestión que se encarga de responder cuando un timer marca timout, la cabecera se muestra a continuación:

func (nr *NodoRaft) **Gestion**()

esta función se basa en un bucle for infinito, en el que encontramos una separación en dos casos principales, el primero de ellos es que nos llegue un mensaje a través del canal TimerEleccion.C, en cuyo caso comprobaremos que tenemos asignados el Roll de seguidor, o el de candidato y si es así, nos convertiremos en candidatos y comenzaremos el proceso de elección. El segundo caso que se trata es que llegue un mensaje a través del canal TimerLatido.C. En este caso comprobamos que el estado asociado al nodo que ha entrado en esta sección es líder, y tras ello reseteamos el tiempo de latido e invocamos a la función Latir.

Pruebas realizadas.

1. Arranque y parado de un nodo remoto

```
155.210.154.204_29001 -> 04:35:57.559543 raft.go:176: logger initialized
155.210.154.204_29001 -> 04:35:57.609795 raft.go:626: Gestion: Tiempo latido TIMEOUT
155.210.154.204_29001 -> 04:35:57.741147 raft.go:615: Gestion: Tiempo eleccion TIMEOUT
155.210.154.204_29001 -> 04:35:57.741155 raft.go:618: Gestion: Somos: SEGUIDOR - TimerEleccion ha expirado
155.210.154.204_29001 -> 04:35:57.741158 raft.go:584: ConvertirseEnCandidato: Nos convertimos en CANDIDATO
155.210.154.204_29001 -> 04:35:57.741162 raft.go:384: enviarPeticionVoto: procedo a enviar la peticion de voto
155.210.154.204_29001 -> 04:35:57.741170 raft.go:388: enviarPeticionVoto: gano por mayoria
155.210.154.204_29001 -> 04:35:57.741172 raft.go:563: ConvertirseEnLider: Nos convertimos en LIDER
155.210.154.204_29001 -> 04:35:57.741256 raft.go:531: Latir: Soy LIDER:  0  con mandato  1 y Log [{ 0}]  y empiezo a enviar latidos
155.210.154.204_29001 -> 04:35:57.791412 raft.go:626: Gestion: Tiempo latido TIMEOUT
155.210.154.204_29001 -> 04:35:57.791419 raft.go:630: Gestion: Somos: LIDER - TimerLatido ha expirado
155.210.154.204_29001 -> 04:35:57.791424 raft.go:531: Latir: Soy LIDER:  0  con mandato  1 y Log [{ 0}]  y empiezo a enviar latidos
155.210.154.204_29001 -> 04:35:57.841598 raft.go:626: Gestion: Tiempo latido TIMEOUT
155.210.154.204_29001 -> 04:35:57.841605 raft.go:630: Gestion: Somos: LIDER - TimerLatido ha expirado
155.210.154.204_29001 -> 04:35:57.841612 raft.go:531: Latir: Soy LIDER:  0  con mandato  1 y Log [{ 0}]  y empiezo a enviar latidos
```

Teniendo en cuenta que todos los nodos comienzan con el rol de SEGUIDOR, lo que ocurre en este caso en el que solo se dispone de un nodo remoto es que el tiempo de elección de ese nodo expira y pasa a convertirse en candidato e iniciar una elección. Como el único nodo activo es el que se vota a sí mismo, saldrá mayoría en la votación y pasará a ser el nuevo líder.

2. Se ha elegido a un primer líder correcto.

Esta segunda prueba podría justificarse con el caso de arriba, pero vamos a probarla con al menos tres nodos en ejecución, el resultado sería el que se muestra a continuación:

```
155.210.154.201_29001 -> 04:51:01.895903 raft.go:621: Gestion: Somos: CANDIDATO - TimerEleccion ha expirado
155.210.154.201_29001 -> 04:51:01.895905 raft.go:584: ConvertirseEnCandidato: Nos convertimos en CANDIDATO
155.210.154.201_29001 -> 04:51:01.895912 raft.go:384: enviarPeticionVoto: procedo a enviar la peticion de voto
155.210.154.201_29001 -> 04:51:01.895926 raft.go:384: enviarPeticionVoto: procedo a enviar la peticion de voto
155.210.154.201_29001 -> 04:51:01.895937 raft.go:384: enviarPeticionVoto: procedo a enviar la peticion de voto
155.210.154.201_29001 -> 04:51:02.046037 raft.go:615: Gestion: Tiempo eleccion TIMEOUT
155.210.154.201_29001 -> 04:51:02.046050 raft.go:621: Gestion: Somos: CANDIDATO - TimerEleccion ha expirado
155.210.154.201_29001 -> 04:51:02.046057 raft.go:584: ConvertirseEnCandidato: Nos convertimos en CANDIDATO
155.210.154.201_29001 -> 04:51:02.046063 raft.go:384: enviarPeticionVoto: procedo a enviar la peticion de voto
155.210.154.201_29001 -> 04:51:02.046073 raft.go:384: enviarPeticionVoto: procedo a enviar la peticion de voto
155.210.154.201_29001 -> 04:51:02.046078 raft.go:384: enviarPeticionVoto: procedo a enviar la peticion de voto
155.210.154.201_29001 -> 04:51:02.047185 raft.go:400: enviarPeticionVoto: recibo voto
155.210.154.201_29001 -> 04:51:02.047205 raft.go:403: enviarPeticionVoto: gano por mayoria
155.210.154.201_29001 -> 04:51:02.047207 raft.go:563: ConvertirseEnLider: Nos convertimos en LIDER
155.210.154.201_29001 -> 04:51:02.047230 raft.go:531: Latir: Soy LIDER:  0  con mandato  55 y Log [{ 0}]  y empiezo a enviar latidos
155.210.154.201_29001 -> 04:51:02.047234 raft.go:541: Latir: Soy LIDER:  0  con mandato  55 y Log [{ 0}]  y empiezo a envio latido a: 1
155.210.154.201_29001 -> 04:51:02.047239 raft.go:541: Latir: Soy LIDER:  0  con mandato  55 y Log [{ 0}]  y empiezo a envio latido a: 2
155.210.154.201_29001 -> 04:51:02.047247 raft.go:494: enviarAppendEntries: valor de Entries:  [] con nextIndex:  1
155.210.154.201_29001 -> 04:51:02.047583 raft.go:494: enviarAppendEntries: valor de Entries:  [] con nextIndex:  1
155.210.154.201_29001 -> 04:51:02.097587 raft.go:626: Gestion: Tiempo latido TIMEOUT
155.210.154.201_29001 -> 04:51:02.097594 raft.go:630: Gestion: Somos: LIDER - TimerLatido ha expirado
155.210.154.201_29001 -> 04:51:02.097599 raft.go:531: Latir: Soy LIDER:  0  con mandato  55 y Log [{ 0}]  y empiezo a enviar latidos
155.210.154.201_29001 -> 04:51:02.097602 raft.go:541: Latir: Soy LIDER:  0  con mandato  55 y Log [{ 0}]  y empiezo a envio latido a: 1
155.210.154.201_29001 -> 04:51:02.097606 raft.go:541: Latir: Soy LIDER:  0  con mandato  55 y Log [{ 0}]  y empiezo a envio latido a: 2
155.210.154.201_29001 -> 04:51:02.097613 raft.go:494: enviarAppendEntries: valor de Entries:  [] con nextIndex:  1
155.210.154.201_29001 -> 04:51:02.097903 raft.go:494: enviarAppendEntries: valor de Entries:  [] con nextIndex:  1
```

En este caso el servidor .201 es quien se convierte en líder en primer lugar, esto es debido a que al ser el primero en lanzarse su tiempo de elección expiró antes que el de los demás, y cuando este les solicita que le voten, los demás nodos le conceden su voto y por eso pasa a ser el líder. Como vemos, no consigue ser líder hasta que obtiene la mayoría, antes de eso es candidato e inicia elecciones constantemente.

3. Un líder nuevo toma el relevo de uno caído.

En esta tercera prueba buscamos que tras la caída del líder otro nodo salga seleccionado y tome su lugar, para ello se ejecutarán diversos servidores, se observará cuál de ellos era el líder, y se detendrá dicho nodo, el resultado de esta prueba es el siguiente:

```
155.210.154.205_29002 -> 04:51:50.189279 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.205_29002 -> 04:51:50.240155 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.205_29002 -> 04:51:50.291187 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.205_29002 -> 04:51:50.365285 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.205_29002 -> 04:51:50.400206 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.205_29002 -> 04:51:50.568574 raft.go:615: Gestion: Tiempo eleccion TIMEOUT
155.210.154.205_29002 -> 04:51:50.568604 raft.go:618: Gestion: Somos: SEGUIDOR - TimerEleccion ha expirado
155.210.154.205_29002 -> 04:51:50.568606 raft.go:584: ConvertirseEnCandidato: Nos convertimos en CANDIDATO
155.210.154.205_29002 -> 04:51:50.568635 raft.go:384: enviarPeticionVoto: procedo a enviar la peticion de voto
155.210.154.205_29002 -> 04:51:50.568666 raft.go:384: enviarPeticionVoto: procedo a enviar la peticion de voto
155.210.154.205_29002 -> 04:51:50.568664 raft.go:384: enviarPeticionVoto: procedo a enviar la peticion de voto
155.210.154.205_29002 -> 04:51:50.569722 raft.go:400: enviarPeticionVoto: recibo voto
155.210.154.205_29002 -> 04:51:50.569728 raft.go:403: enviarPeticionVoto: gano por mayoria
155.210.154.205_29002 -> 04:51:50.569731 raft.go:563: ConvertirseEnLider: Nos convertimos en LIDER
155.210.154.205_29002 -> 04:51:50.569737 raft.go:531: Latir: Soy LIDER:  1  con mandato  62 y Log [{ 0}]  y empiezo a enviar latidos
155.210.154.205_29002 -> 04:51:50.569741 raft.go:541: Latir: Soy LIDER:  1  con mandato  62 y Log [{ 0}]  y empiezo a envio latido a: 0
155.210.154.205_29002 -> 04:51:50.569744 raft.go:541: Latir: Soy LIDER:  1  con mandato  62 y Log [{ 0}]  y empiezo a envio latido a: 2
155.210.154.205_29002 -> 04:51:50.569751 raft.go:494: enviarAppendEntries: valor de Entries:  [] con nextIndex:  1
155.210.154.205_29002 -> 04:51:50.570438 raft.go:494: enviarAppendEntries: valor de Entries:  [] con nextIndex:  1
155.210.154.205_29002 -> 04:51:50.620798 raft.go:626: Gestion: Tiempo latido TIMEOUT
155.210.154.205_29002 -> 04:51:50.620835 raft.go:630: Gestion: Somos: LIDER - TimerLatido ha expirado
155.210.154.205_29002 -> 04:51:50.620860 raft.go:531: Latir: Soy LIDER:  1  con mandato  62 y Log [{ 0}]  y empiezo a enviar latidos
155.210.154.205_29002 -> 04:51:50.620863 raft.go:541: Latir: Soy LIDER:  1  con mandato  62 y Log [{ 0}]  y empiezo a envio latido a: 0
155.210.154.205_29002 -> 04:51:50.620883 raft.go:541: Latir: Soy LIDER:  1  con mandato  62 y Log [{ 0}]  y empiezo a envio latido a: 2
155.210.154.205_29002 -> 04:51:50.620894 raft.go:494: enviarAppendEntries: valor de Entries:  [] con nextIndex:  1
155.210.154.205_29002 -> 04:51:50.622014 raft.go:494: enviarAppendEntries: valor de Entries:  [] con nextIndex:  1
```

Como vemos en la imagen este nodo comienza siendo seguidor ya que recibe AppendEntries constantemente. Cuando el nodo que actuaba como líder se cae este pasa a ser el líder, esto es posible ya que recibe el voto del otro nodo restante, obteniendo así mayoría. En la siguiente imagen se muestra además cómo actúa el nodo restante:

```
155.210.154.208_29003 -> 04:51:49.885736 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:49.930787 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:49.981366 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.032521 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.095436 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.136210 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.186950 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.237799 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.288801 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.358414 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.397799 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 61  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.568753 raft.go:328: Me ha llegado una solicitud para que vote a:  1
155.210.154.208_29003 -> 04:51:50.568758 raft.go:334: PedirVoto: nos convertimos a seguidor
155.210.154.208_29003 -> 04:51:50.568761 raft.go:550: ConvertirseEnSeguidor: Nos convertimos en SEGUIDOR
155.210.154.208_29003 -> 04:51:50.568763 raft.go:345: PedirVoto: voto a  1
155.210.154.208_29003 -> 04:51:50.568784 raft.go:550: ConvertirseEnSeguidor: Nos convertimos en SEGUIDOR
155.210.154.208_29003 -> 04:51:50.569517 raft.go:434: AppendEntries: recibido latido del lider:  1 con mandato 62  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.621040 raft.go:434: AppendEntries: recibido latido del lider:  1 con mandato 62  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.671578 raft.go:434: AppendEntries: recibido latido del lider:  1 con mandato 62  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.722241 raft.go:434: AppendEntries: recibido latido del lider:  1 con mandato 62  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.798802 raft.go:434: AppendEntries: recibido latido del lider:  1 con mandato 62  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.823462 raft.go:434: AppendEntries: recibido latido del lider:  1 con mandato 62  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.873877 raft.go:434: AppendEntries: recibido latido del lider:  1 con mandato 62  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.923897 raft.go:434: AppendEntries: recibido latido del lider:  1 con mandato 62  y Entries:  []  LastApplied:  0
155.210.154.208_29003 -> 04:51:50.974566 raft.go:434: AppendEntries: recibido latido del lider:  1 con mandato 62  y Entries:  []  LastApplied:  0
155.210.154.208 29003 -> 04:51:51.025028 raft.go:434: AppendEntries: recibido latido del lider:  1 con mandato 62  y Entries:  []  LastApplied:  0
```

4. Se consigue comprometer 3 operaciones seguidas en 3 entradas, con un líder estable y sin fallos en el sistema.

Para la realización de esta prueba se ha modificado el fichero cmd/srvraft/main.go y completado el pkg/cltraft/main.go. Estas modificaciones permiten abrir una nueva conexión tcp, a través de la cual escucharemos las solicitudes de los clientes. Además para poder someter las operaciones y asegurar el correcto funcionamiento del código implementado se ha creado una función llamada LecEsc, en la que se somete la operación por parte del nodo líder.

En este último caso se puede ver como se comprometen 3 operaciones con dos nodos activos. En la primera imagen se muestra al líder, al cual le llegan las tres operaciones de someter, en la imagen se muestra una de ellas:

```
01:40.697660 raft.go:494: enviarAppendEntries: valor de Entries:  [] con nextIndex:  3
01:40.747824 raft.go:626: Gestion: Tiempo latido TIMEOUT
01:40.747865 raft.go:630: Gestion: Somos: LIDER - TimerLatido ha expirado
01:40.747894 raft.go:531: Latir: Soy LIDER:  0  con mandato  19 y Log [{ 0} {escritura 19} {escritura 19}]  y empiezo a enviar lat
01:40.747922 raft.go:541: Latir: Soy LIDER:  0  con mandato  19 y Log [{ 0} {escritura 19} {escritura 19}]  y empiezo a envio lati
01:40.747965 raft.go:494: enviarAppendEntries: valor de Entries:  [] con nextIndex:  3
01:40.756643 raft.go:245: SometerOperacion: nuevo log  [{ 0} {escritura 19} {escritura 19} {escritura 19}]
01:40.798379 raft.go:626: Gestion: Tiempo latido TIMEOUT
01:40.798451 raft.go:630: Gestion: Somos: LIDER - TimerLatido ha expirado
01:40.798495 raft.go:531: Latir: Soy LIDER:  0  con mandato  19 y Log [{ 0} {escritura 19} {escritura 19} {escritura 19}]  y empie
01:40.798515 raft.go:541: Latir: Soy LIDER:  0  con mandato  19 y Log [{ 0} {escritura 19} {escritura 19} {escritura 19}]  y empie
01:40.798554 raft.go:491: log: jjjjj {escritura 19}
01:40.798572 raft.go:494: enviarAppendEntries: valor de Entries:  [{escritura 19}] con nextIndex:  3
```

Por otro lado, en esta segunda imagen lo que se muestra es a un nodo seguidor, el cual ya ha actualizado su log con las tres nuevas entradas.

```
0.647009 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 19  y Entries:  []  LastApplied:  2
0.698271 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 19  y Entries:  []  LastApplied:  2
0.748393 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 19  y Entries:  []  LastApplied:  2
0.799281 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 19  y Entries:  [{escritura 19}]  LastApplied:  2
0.799324 raft.go:461: AppendEntries: Soy servidor y he modificado mi log:  [{ 0} {escritura 19} {escritura 19} {escritura 19}]
0.849674 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 19  y Entries:  []  LastApplied:  3
0.849704 raft.go:471: AppendEntries: Soy servidor y he comprometido mi entrada  3
0.900950 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 19  y Entries:  []  LastApplied:  3
0.951053 raft.go:434: AppendEntries: recibido latido del lider:  0 con mandato 19  y Entries:  []  LastApplied:  3
```

Anexo 1:

```
type TipoOperacion struct {
        Operacion  string        // La operaciones posibles son "leer" y "escribir"
        Clave      string
        Valor      string        // en el caso de la lectura Valor = ""
}
```

Este se utilizará para guardar las operaciones que realizan los clientes, estas pueden ser o bien lectura, o bien escritura, en caso de ser lectura se devolverá en el campo valor los carácteres que se han leido en la ejecución de la operación.

```
type AplicaOperacion struct {
        Indice      int  // en la entrada de registro
        Operacion   TipoOperacion
}
```

El struct AplicaOperacion por su parte se utilizará posteriormente en la práctica 5 para enviar cada una de las operaciones comprometidas a través del canal “canalAplicar”, actualizando así la máquina de estados de los nodos de repetición activos.

```
type RegistroOp struct {
        Comando  string
        Mandato  int
}
```

RegistroOp constituye el tipo de dato que almacenará el registro de Log de cada uno de los nodos que se creen, además formará parte del tipo de dato explicado anteriormente AplicaOperacion, está constituido por dos campos, en los que el primero de ellos indicará el mandato en el cual se ha registrado la información, y el comando el cual coincidirá con la solicitud realizada por los clientes, ya sea de escritura, o de lectura.
El siguiente struct definido es NodoRaft:

```
type NodoRaft struct {
        Mux                 sync.Mutex // Mutex para proteger acceso a estado compartido

        // Host:Port de todos los nodos (réplicas) Raft, en mismo orden
        Nodos               []rpctimeout.HostPort
        Yo                  int // indice de este nodos en campo array "nodos"
        IdLider             int
        // Utilización opcional de este logger para depuración
        // Cada nodo Raft tiene su propio registro de trazas (logs)
        Logger              *log.Logger

        // Estado.
        E                   Estado

        Roll                string  // Roll (SEGUIDOR, LIDER, CANDIDATO)
```

```
	// contador de votos en caso de ser candidato
	VotosRecibidos      int

	// cuenta el numero de nodos que guardan la entrada
	NodosLogCorrecto int

	TimerEleccion       *time.Timer
	TimerLatido         *time.Timer
}
```

Este registro es con el que se va a gestionar todo el algoritmo de raft, cada nodo existente seguira esta estructura, se utilizaran dos datos de tipo time.Time los cuales incluyen su propio canal, para la gestión de los timeouts, el primero de ellos TiempoEleccion tan solo estará activo en los seguidores y en los candidatos, cuando el tiempo establecido en estos, el cual será un número aleatorio entre 50 y 150 milisegundos, se termine comenzará una nueva elección de líder, para lo cual primero nos votaremos a nosotros mismos (el proceso en el que ha saltado el timeout) y solicitaremos la votación del resto de los nodos. El segundo time por su parte estará activo tan solo en el líder, y vencera en un periodo de tiempo fijado en 50 milisegundos, para que el número de latidos enviados no sea suoerior a 20.

por otro laod tenemos dos contadores que nos ayudarán a saber en qué casos tenemos mayoría, el primero de ellos para saber si nos podemos convertir en líderes, el segundo para saber si podemos comprometer una entrada.

EL apartado de Roll almacenará el estado en el que se encuentra el nodo, puede ser “seguidor”, “candidato” o bien “lider”

albergará un E del tipo Estado el cual se explicará posteriormente, por último diversas variables para almacenar nuestro id, el del líder y un vector con las direcciones del todos los nodos que se han lanzado.

```
type Estado struct {
	CurrentTerm    int       // ultimo mandato que ha visto el servidor
	VotedFor       int       // candidato al que se ha votado
	Log            []RegistroOp
	CommitIndex    int       // indice de la ultima entrada comprometida
	LastApplied    int
	// respecto a las peticiones de los clientes
	NextIndex      []int     //indice de la sig entrada de reg para enviar
	MatchIndex     []int     //indice de la entrada de reg mas alta
}
```

Este struct almacena los campos que se solicitan en la tabla, en la que se ha basado el algoritmo en la cual se puede encontrar la explicación de cada una de las variables.

```
type ArgsPeticionVoto struct {
	Term            int
```

```
        CandidateId    int
        LastLogIndex   int
        LastLogTerm    int
}
```

Este struct se enviará por parte del candidato a todos los nodos a los que se solicite su voto, con la información del nodo candidato, siendo Term el mandato del candidato, CandidateId el id del cantidado que solicita la votacion, LastLogIndex el ultimo valor del índice del log del candidato y LastLogTerm el mandato en el que se registró el último índice del log.

```
type RespuestaPeticionVoto struct {
        Term           int      // currentTerm
        VoteGranted bool   // TRUE si recivimos el voto, false en caso contrario
}
```

Los nodos que reciban dicha solicitud de voto por su parte responderán con un RespuestaPeticionVoto el cual indicará el término de los servidores, y la respuesta al voto que podrá ser true o false.

```
type ArgAppendEntries struct {
        Term              int
        LeaderId          int
        PrevLogIndex   int
        PrevLogTerm    int
        Entries           []RegistroOp
        LeaderCommit  int
}
```

Uno de los últimos structs definidos es ArgAppendEntries, este se enviará a cada uno de los seguidores cada vez que se realice un latido. En él enviaremos el término del líder en el instante en el que se ha realizado el latido, el LeadLerId para que los seguidores puedan redirigir al cliente en caso de les envie una solicitud, PrevLogIndex que es el índice del último log realizado por el líder, PrevLogTerm valor del término apuntado por el PrevLogIndex, Entries log que rellena el líder para cada uno de los nodos, en el que enviará las solicitudes de los clientes que los servidores que lo reciben no tienen en su Log según los datos que tenemos registrados en el propio líder. En caso de que el nodo esté al corriente en su registro de entradas será nil. El último valor será LeaderCommit que indicará la última entrada del Log que esté comprometida.

A este los servidores le responderan con el siguiente tipo:

```
type Results struct {
        Term       int
        Success bool
}
```

En el que le indiarán al líder a través del parámetro Success si han almacenado en el Log el parámetro recibido en Entries.