SISTEMAS DISTRIBUIDOS

PRÁCTICA 2:

SISTEMAS DISTRIBUIDOS LECTORES Y ESCRITORES DISTRIBUIDOS

Alejandro Terron 761069
Óscar Anadón 760628

8-noviembre-2021

RESUMEN

Para esta práctica se ha planteado un problema en el cual se ha tenido que traducir y adaptar el algoritmo de Ricart-Agrawala, que es una implementación en Algol del mutex distribuido. Se ha usado su versión generalizada la cual permite tener en cuenta la concurrencia de varias operaciones sobre la misma sección crítica.

A la hora de implementarlo en go se ha intentado ser lo más fiel posible al código original.

El código cuenta con un constructor el cual inicializa todos los campos necesarios para la implementación del algoritmo. También se lanza el proceso esperarMensaje() encargado de captar todos aquellos mensajes que lleguen al nodo, dependiendo de la operación indicada en el tipo del mensaje este evaluará si tiene que ser respondido o almacenado para enviar el ack una vez terminada la sesión crítica (caso READ y WRITE), actualizar la base de datos “personal” (caso CopyDB) o simplemente restar uno al contador global de respuestas pendientes (caso REPLY).

También presenta la función PreProtocol() la cual es la encargada de una vez se tiene acceso a través del mutex a modificar los valores ReqCS, el cual indica si el nodo en cuestión quiere acceder a la sección crítica, y OurSeqNum, el cual indica la prioridad mínima para la cual debemos responder inmediatamente cualquier petición. (La acción de responder no depende únicamente de este parámetro), desde esta función también se lanza otro proceso encargado de difundir la intención de acceder a la sección crítica a todos los nodos excepto a sí mismo.

Por último también se cuenta con una operación PostProtocol(), esta es llamada desde el proceso principal una vez ha salido de la sección crítica. Su finalidad es avisar a todos aquellos nodos cuyas peticiones han sido catalogadas como diferibles, es decir, todos aquellos nodos que se encuentren almacenados como true en el vector RepDefd[ ].

Para la comunicación se ha hecho uso del módulo ms(messagesystem). En los mensajes definidos en el ra se han añadido el elemento Logger del módulo govector para facilitar el debug y comprensión del funcionamiento del código.

A la hora de escribir se ha creado una función análoga a la de sendMessage() la cual, en vez de notificar la intención de entrar en la sección crítica, envía el texto introducido en la base de datos por los escritores.

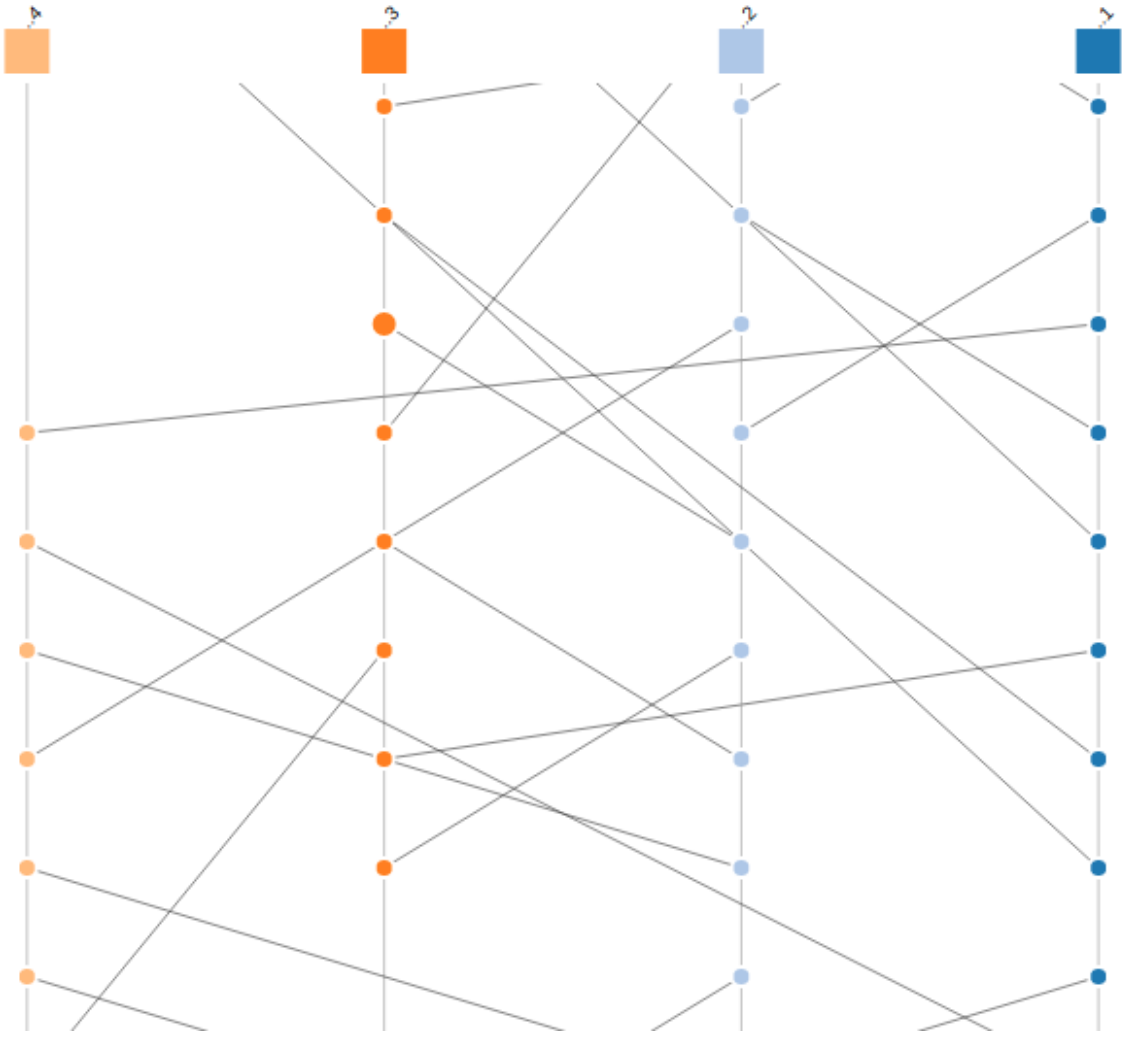
Por último se han creado varios shell-script para facilitar la comprobación del funcionamiento del código y la automatización a la hora de crear y gestionar los ficheros de log generados a través del módulo govector.

PRUEBAS

A lo largo del desarrollo de esta práctica se han ido realizando una serie de pruebas mínimas para comprobar el funcionamiento del código, como por ejemplo pruebas con dos nodos para comprobar el correcto funcionamiento de la exclusión entre lectores y escritores, el correcto paso de los mensajes entre los distintos nodos que componen el sistema. Se han lanzado tres test compuestos por cuatro nodos además de visualizar los log a través de la herramienta ShiViz.

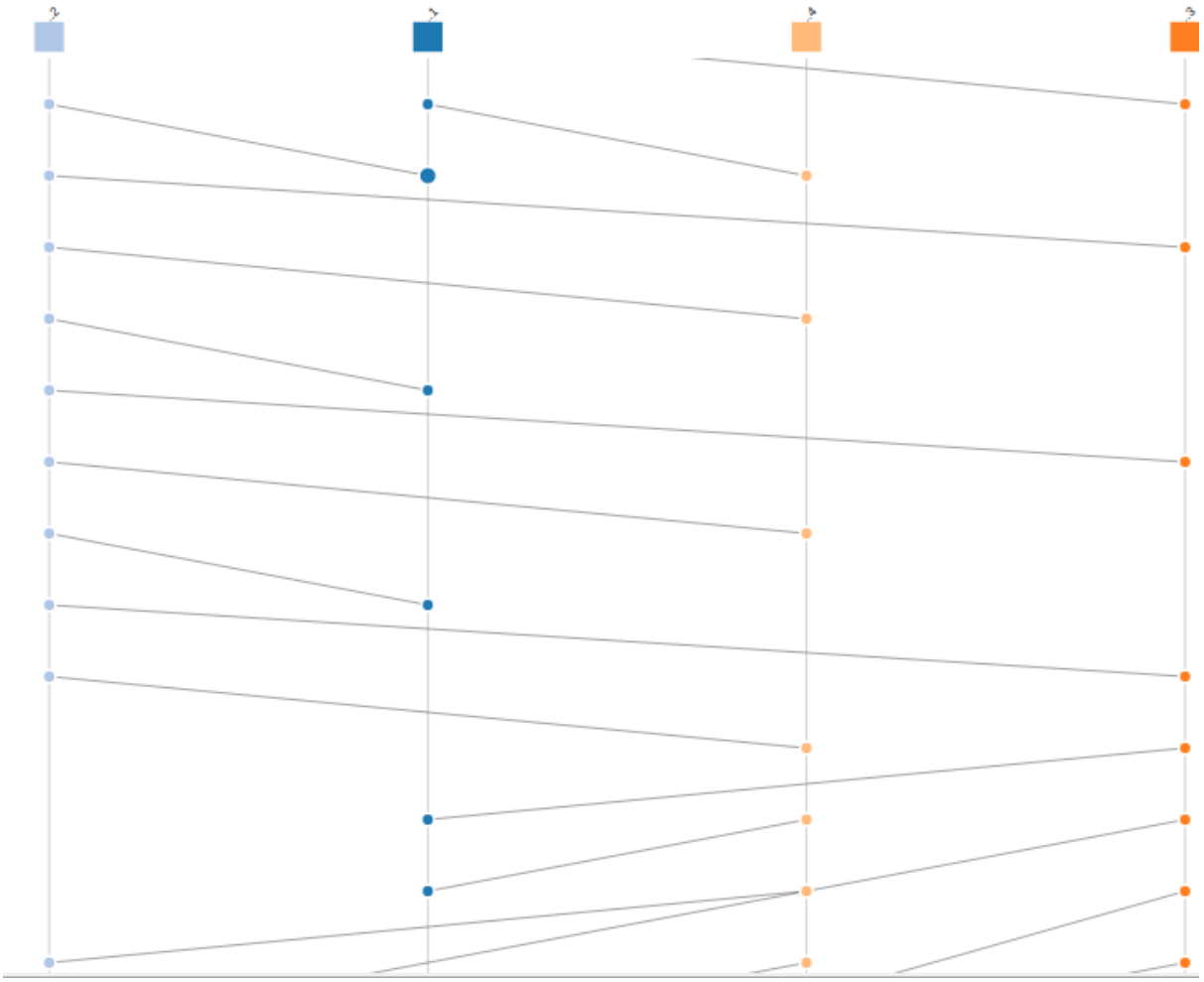
TEST_1

Este test está compuesto por cuatro lectores. Ha sido diseñado específicamente para comprobar la concurrencia de este tipo de nodos sobre la base de datos. Como se puede observar existe un gran flujo de mensajes a lo largo de la ejecución, el seguimiento de las trazas del log junto con los test aislados de concurrencia nos reafirma en que no existe problemas de concurrencia entre los lectores.



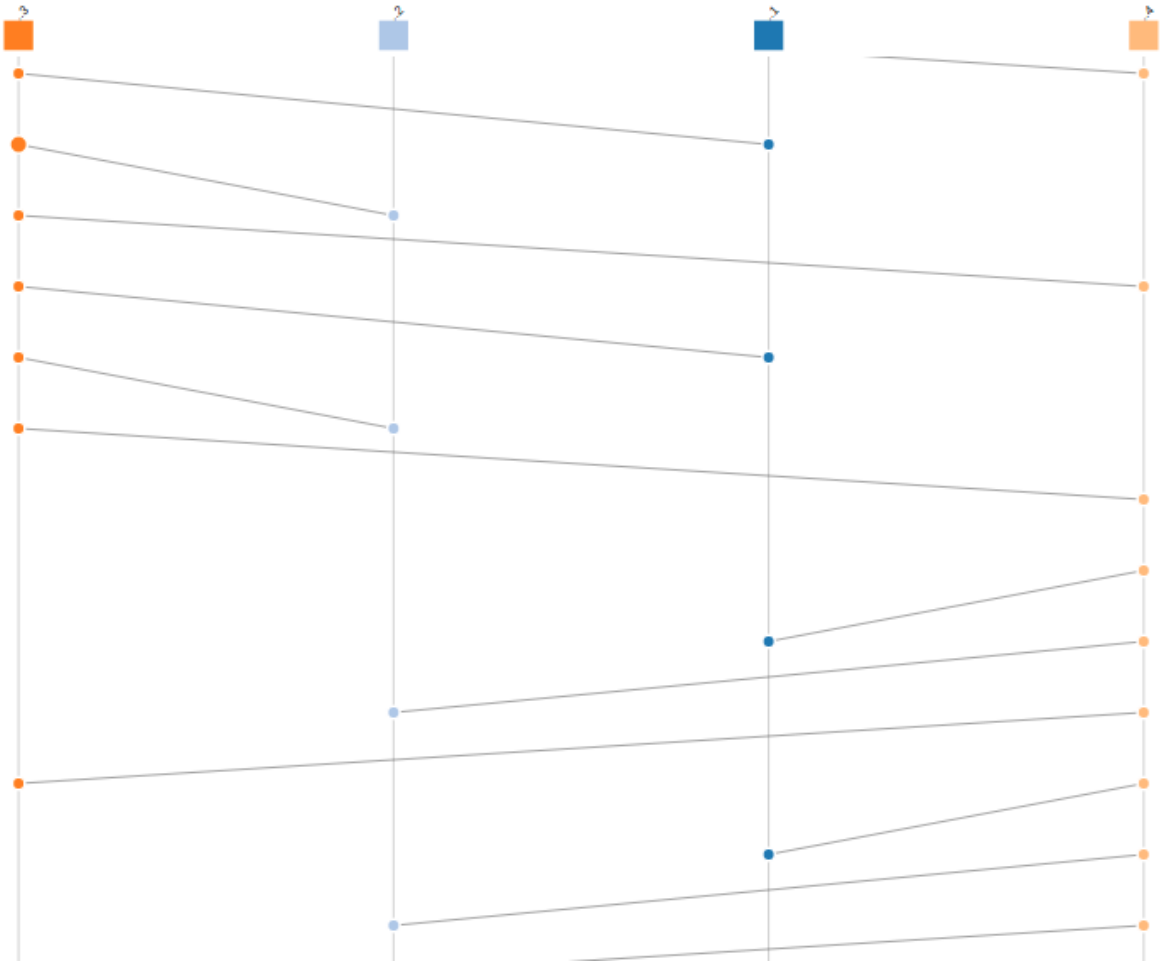
TEST_2

Este test está compuesto de dos escritores (1 y 2) y dos lectores (3 y 4). Ha sido diseñado para comprobar que para distintas operaciones (lectura, escritura) no hay concurrencia. Como se puede ver en el esquema cuando el nodo 2 está escribiendo, una vez que le da el mensaje de CopyDB a todos y responde es el nodo 3 de lectura el que lee y posteriormente empieza a realizar los reply.



TEST_3

Este test está compuesto por 4 escritores. Se ha diseñado para comprobar que no existe concurrencia entre ninguno de ellos cuando 1 está en proceso de escritura (dentro de la sección crítica) . Se puede observar que el nodo 4 no responde al 3 mientras están leyendo de la base, una vez termina empieza el proceso de notificar a todos la escritura en la DB.



Conclusión

Con esta práctica hemos logrado entender los fundamentos de los relojes lógicos de Lamport, el algoritmo de Ricart-Agrawala, así como obtener una mayor soltura a la hora de programar sistemas distribuidos en Golang. Una de las cosas que más nos han ayudado y nos han llamado la atención es la facilidad con la que se pueden seguir las ejecuciones de un sistema distribuido, apoyándonos de una herramienta gráfica tan potente como shiviz.