Práctica 1
Comunicación de grupo causal, relojes lógicos, estados globales

Autor: Unai Arronategui

Límite entrega memoria y código fuente en moodle : 22 de octubre de 2018

Resumen

Este trabajo plantea el diseño, implementación y validación de una librería de comunicación de grupo causal, mediante la utilización de relojes vectoriales.

La validación se efectuará con un simple motor de simulación de videojuegos.

Esta práctica incluye ***redactar*** *una* ***memoria y escribir todo el código fuente (salvo el propocionado en la asignatura)****. El texto de la memoria y código debe ser original y bien explicado. Se puede comentar sobre la práctica con otros alumnos, pero no está permitido mirar o copiar el código de otros.*

Recursos Go

Adjunto al guión debeis tener disponible los códigos de la librería "*vclock*" y el código "*go-multicast*" como apoyos para esta práctica.

Teneis diferentes opciones de editores de código (sublime text, vim, etc) o entornos de programación IDE para Golang, como LiteIDE o GoLand (JetBrain).

Hay referencias y anexos de apoyo y comprehensión del lenguaje al final de este guión.

1. Objetivos

Los objetivo de esta 1ª parte del mini proyecto de sistemas distribuidos es el diseño e implementación de una librería para un servicio Paxos que utilice un número fijo de nodos para llegar a acuerdos.

2. Comunicación de grupo causal

Diseñar una librería de comunicación de grupo causal que utilice relojes vectoriales para garantizar la consistencia causal de los mensajes entregados a cada uno de los participantes distribuidos de la aplicación. Junto al guión teneis disponible una librería básica de relojes vectoriales, "*vclock*", que podeis utilizar. La comunicación de grupo supone enviar el mismo mensaje a todos los procesos distribuidos que forman la aplicación, pero su entrega debe garantizar una consistencia causal. El conjunto de procesos distribuidos es estático y conocido al inicio de la ejecución. Cada proceso del grupo será identificado por un *string* **"IP:puerto"**. Inicialmente se puede utilizar la IP localhost (127.0.0.1) para test en una máquina, pero posteriormente debeis derivar a **tests en distribuido** con cada proceso del grupo en un ordenador diferente. El laboratorio 1.02 dispone de ordenadores con direcciones IP consecutivas desde el 144.210.154.191 hasta el 155.210.154.210. Para no interferir entre vosotros en las comunicaciones de red, cada alumno utilizará, en recepción, el rango de puertos correspondiente a un número de 1 cifras de modelo 1*XYZ?*, donde XYZ corresponden a las cifras de menor peso de vuestro NIP y la *?* es la cifra diferenciada del proceso dentro del grupo.

En primera instancia, la implementación de las funciones **sendGroup** y **receiveGroup** de comunicación de grupo debe ser compuesta mediante comunicación punto a punto (*send*, *receive*), como la planteada en el ejemplo de las transparencias de clase sobre el lenguaje Go (suponemos comunicación fiable con TCP). Se pueden simular los retardos de red (desordenación de mesajes) con la función de librería estándar *time.Sleep*(X * time Milliseconds), introducida de forma voluntaria en algunos envios punto a punto, para alterar el orden de llegada. Parametrizar los envios con retardos explicitos destinados a receptores específicos. Esto nos permite inducir, en los tests, una recepcion de los mensajes en orden incorrecto (inspiraros de la diapositiva 29 del tema 1), que permita validar vuestra implementación de la reordenación causal de

mensajes recibidos, mediante una bateria de tests que utilice la infraestructura de tests de Golang.

Pensar de forma detenida sobre el problema de la gestión de la ordenación local de los mensajes recibidos desde la red, previo a la entrega definitiva al proceso local de la aplicación, para garantizar las relaciones causales apoyandose en los relojes vectoriales.

3. Comunicación de grupo causal fiable con IP multicast

Diseñar, implementar y validar mediante tests una comunicación de grupo causal que utilice UDP con IPmulticast. Apoyaros en el código "*go-multicast*" puesto a disposición con el guión para ver detalles de la implementación en Go del envio (sendGroupM) o y recepción (receiveGroupM) IP/UDP multicast.

Añadir un solución simple de entrega fiable de mensajes dado que UDP no garantiza ni entrega fiable ni ordenación, como es el caso de TCP. La ordenación será garantizada con los relojes vectoriales.

Tanto el envio como la recepción multicast deben incorporar la serialización "gob" de datos que teneis como ejemplo en el send y receive basicos mostrados en diapositiva en clase sobre el lenguaje Go.

La dirección IP multicast/puerto utilizada por cada alumno corresponde al modelo "229.0.0XY.00Z:9999" con los valores XYZ especificados en la sección 2.

4. Algoritmo Chandy-Lamport para determinar estados globales en sistemas distribuidos

Diseñar, implementar y validar mediante test el algoritmo de Chandy-Lamport para la obtención de fotos congeladas de una ejecución distribuida, tal como se ha visto en clase.

Incluir los métodos de comunicación de grupo que habeis desarrollado en las secciones previas, en los casos que puedan ser utiles.

Teneis en moodle el articulo de investigación que explica en detalle este método.

5. Requisitos de implementación

- El código implementado debe ser compatible con la versión 1.13.1 de Go.
- La solución planteada debe utilizar únicamente los paquetes de la **librería estándar** de Go y, eventualmente, los paquetes de código provistos junto a este guión.
- El estilo del código desarrollado debe ser comprobado o adaptado mediante la herramienta **gofmt**.

6. Opcional

Opcionalmente, podeis introducir intrumentación de depuración en vuestro código con las librería "GoVector" (https://github.com/DistributedClocks/GoVector) para utilizar la herramienta de visualización de trazas de mensajes "*ShiViz*" (https://bestchai.bitbucket.io/shiviz/).

7. Memoria

En la memoria de prácticas a entregar debe aparecer claramente explicado el diseño de alto nivel de vuestras soluciones, los protocolos de interacción detallados entre los procesos distribuidos y los elementos más importantes de la implementación en el código.

8. Evaluación

Para la evaluación de la práctica, se debe entregar una memoria de su diseño e implementación y todo el código fuente necesario para ejecutar vuestro programa, el cual debera pasar los tests que hayais habilitado a tal efecto. Estos tests deberan ejecutarse con éxito en máquinas del laboratorio 1.02.

La entrega se realizará, mediante un solo fichero comprimido (tar.gz o zip), en una sección de **entrega** que se habilitará en la página moodle de la asignatura, siendo la fecha **límite el 22 de octubre a las 10h**.

Referencias

Lenguaje GO :

- Especificacion : https://golang.org/ref/spec
- Tutorial : https://tour.golang.org/
- Resumen operativa básica de desarrollo incluida la validación (testing) : https://golang.org/doc/code.html
- Consideraciones de estilo en detalle : https://golang.org/doc/effective_go.html
- Ejemplos de código Go : https://gobyexample.com/

Relojes Vectoriales y estados globales (articulos disponibles en moodle) :

- Baldoni and Raynal. Fundamentals of Distributed Computing: A Practical Tour of Vector Clock Systems. IEEE Distributed Systems Online 2002.
- Chandy and Lamport. Distributed Snapshots: Determining the Global States of a Distributed System. TOCS 1985.

Anexo

El comando go es la herramienta base para manipular los programas, con subcomandos especificos para diferentes tareas .

- Compilación : **go build** fichero.go
- Compilación y ejecución : **go run** fichero.go
- Comprobación y formateo si necesario : **go fmt** fichero.go
- Ejecutar tests asociados al progra.a princicpal o a paquete de librería : **go test**
- etc