Programación Distribuida y Tiempo Real

Ulises Jeremias Cornejo Fandos[1]

[1] *Licenciatura en Informática, Facultad de Informática, UNLP*

compiled: November 16, 2019

1. Programar un agente para que periódicamente recorra una secuencia de computadoras y reporte al lugar de origen

a) El tiempo total del recorrido para recolectar la información

b) La carga de procesamiento de cada una de ellas.

c) La cantidad de memoria total disponible.

d) Los nombres de las computadoras.

Para este ejercicio se crea un agente el cual crea 10 containers, y agrega sus ID a una lista de tipo `ArrayList` con el fin de recordar los mismos, agregando al final al container inicial.

Luego simplemente se itera sobre la lista, juntando la información del container en el que se encuentra. Para almacenar la información, se creo una clase anidada (`ContainerInfo`) la cual se utiliza para que, una vez conseguida la información se almacene en otra lista de tipo `ArrayList`. Al terminar la iteración, tenemos acceso al container origen, con una lista de instancias de `ContainerInfo`.

Luego se recorre esa lista imprimiendo la información correspondiente.

Para ejecutar, se debe levantar el `jade.Boot`, esto se puede hacer mediante el comando *make*.

Simplemente se puede ejecutar `make start-gui` Luego, para ejecutar la implementación para este ejercicio, se puede ejcutar `make run-ex1`

2. Programe un agente para que calcule la suma de todos los números almacenados en un archivo de una computadora que se le pasa como parámetro. Comente cómo se haríalo mismo con una aplicación cliente/servidor. Comente que pasaría si hubiera otros sitios con archivos que deben ser procesados de manera similar

Para este ejercicio, se programa un agente que recibe el nombre de una computadora, y migra el agente. Una vez en la computadora destino, se realiza la lectura del archivo, junto con la suma. Al finalizar, se retora al container origen, y se muestra el resultado.

Para ejecutar, se debe levantar el `jade.Boot`, esto se puede hacer mediante el comando *make*.

Simplemente se puede ejecutar *make start-gui* Luego, para ejecutar la implementación del ejercicio, se puede ejcutar `make run-ex2`.

Para realizar una implementación similar en el modelo cliente/servidor, el servidor debe poder acceder al filesystem del cliente. De poder hacer, puede copiarlo y trabajarlo localmente, para retornan un valor final.

De no poder acceder al filesystem, se puede trabajar exponiendo un servicio que acepte un archivo, y retorne su suma.

De todas formas, la interacción puede llegar a ser un poco menos directa, aunque más simple.

3. Defina e implemente con agentes un sistema de archivos distribuido similar al de las prácticas anteriores.

Para este punto, se programa un agente que pueda recibir argumentos, y en base a lo recibido, realiza diferentes acciones.

Al ser métodos java, se reutiliza gran parte del la lógica de la práctica anterior.

El agente comprende 3 acciones:

- list: La acción más simple. Recibe un parámetro extra a la acción, que marca cual es el directorio a listar. Una vez parseados los argumentos, se realiza la migración al servidor (recordemos que nos encontramos en el cliente). Al terminar la migración, se leen los archivos del directorio, y se almcenan en una variable de instancia del agente. Luego se vuelve al cliente y se informa lo leído.

- read: Recibe 2 argumentos extra a la acción, que marcan el nombre del nuevo archivo y el nombre del archivo remoto (en ese orden) Una vez parseados los argumentos, se realiza la migración al serviodr. Al termionar la migración, se leen los primeros 2000 bytes del archivo, y el tamaño total del archivo a leer y se almacenan en variables de instancia. En el cliente, se realiza la copia de los 2000 bytes, y se consulta si el tamaño total del archivo es menor o igual al tamaño copiado (en la primer vuelta va a ser igual a 2000), de no ser asi, se suman a una variable de `bytes copiados`, y se realiza la migración al servidor. En el servidor se

copian los siguientes 2000 bytes `a partir` de los bytes copiados.(Esta acción se repite tantas veces como sea necesaria)

- write: Trabaja de igual manera que el read, pero a manera inversa, siendo el cliente del item anterior, el servidor, y el servidor del item anterior el cliente.

Para ejecutar, se debe levantar el `jade.Boot`, esto se puede hacer mediante el comando `make`

Simplemente se puede ejecutar `make start-gui`

Luego, para interactuar con el servidor, se cuenta con el script `run`

Para ver la sección de ayuda del comando, se puede utilizar la flag `-h`

3.A. Comparación con prácticas anteriores

Esta solución, plantea más comodidad al momento de manejar los datos (se cuenta con una clase con atributos accesibles tanto en el cliente, como en el servidor). Aunque veo que el migrado de código en repetidas ocaciones puede llegar a generar un overhead mayor.