Programación Distribuida y Tiempo Real

Ulises Jeremias Cornejo Fandos[1]

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

compiled: November 18, 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 IDs a una lista de tipo `ArrayList<String>` con el fin de recordar los mismos, agregando al final al container inicial.

Luego se itera sobre la lista, juntando la información del contenedor en el que se encuentra. Para almacenar la información, se crea una clase anidada (`ContainerInfo`) la cual se utiliza para que, una vez conseguida la información, se almacene en otra lista de tipo `ArrayList<ContainerInfo>`. 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.

Ver como ejecutar la implementación en la sección 4.A del Apéndice.

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 haría lo 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 el agente `SumAgent` que recibe el nombre de una computadora, y migra el agente.

Una vez migrado en la computadora destino, se realiza la lectura del archivo, junto con la suma. Al finalizar, se retorna la suma al container origen, y se muestra el resultado.

Ver como ejecutar la implementación en la sección 4.B del Apéndice.

Para realizar una implementación similar en el modelo cliente/servidor, el servidor debe poder acceder al filesystem del cliente. De poder hacerlo, 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 el agente `FTPAgent` que puede recibir argumentos, y en base a lo recibido, realiza diferentes acciones.

El agente comprende 3 acciones:

- **list**

  Recibe como parámetro el directorio a listar. Una vez parseados los argumentos, se realiza la migración al servidor. Al terminar la migración, se leen los archivos del directorio, y se almacenan 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 en orden sen:

  - Nombre del nuevo archivo.
  - Nombre del archivo remoto.

  Una vez parseados los argumentos del lado del cliente, se realiza la migración al serviodor. Al termionar la migración, se leen los primeros 1024 bytes del archivo, y el tamaño del mismo y se almacenan en variables de instancia.

  En el cliente, se realiza la copia de los 1024 bytes, y se consulta si el tamaño total del archivo es menor o igual al tamaño copiado siendo en la primer

vuelta igual a 1024. 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 1024 bytes `a partir` de los bytes copiados, repitiendo esta acción tantas veces como sea necesario.

- **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.

Ver como ejecutar la implementación en la sección 4.C del Apéndice.

4. Apéndice

4.A. Ejecución Ejercicio 1

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

Luego, para ejecutar la implementación para este ejercicio, se puede ejcutar `make run-info`.

```
$ make
$ make start &
$ make run-info
```

4.B. Ejecución Ejercicio 2

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

Luego, para ejecutar la implementación del ejercicio, se puede ejcutar `make run-sum`.

```
$ make
$ make start &
$ make run-sum
```

4.C. Ejecución Ejercicio 3

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

Luego, para interactuar con el servidor, se cuenta con el script `run` dentro del directorio *FTP*. Para ver la sección de ayuda del comando, se puede ejecutar `make run-ftp`.

```
$ make
$ make start &
$ make run-ftp # imprime ayuda
$ ./FTP/run ... # en la ayuda se muestra como utilizar el comando
```