Practica 4 - JADE

Alumno: Cazorla, Ignacio Nicolás
N° de alumno: 13926/1

Contenidos

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

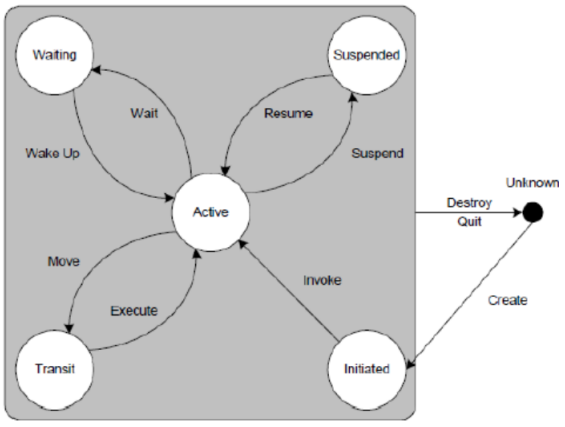
a) El tiempo total del recorrido para recolectar la información.
b) La carga de procesamiento de cada una de ellas.
c) La cantidad de memoria total disponible.
d) Los nombres de las computadoras.

Comente la relación entre este posible estado del sistema distribuido y el estado que se obtendría implementando el algoritmo de instantánea.

La clase **AgenteMóvil**, es la implementación del agente que recorrerá las computadoras y reportará información sobre cada una de ellas. Dicha clase hereda de Agent todo lo relacionado a su interacción con la plataforma JADE.
Entre los métodos más relevantes utilizados tenemos:

- **setup()** : realiza toda la configuración inicial del agente. Se llama a este método únicamente cuando el agente se ejecuta por primera vez.
- **doMove(Location location)** : que permite especificar un destino para mover el agente. Según la definición de movilidad débil, del agente sólo se envía el código y los datos, pero no su estado (de ejecución sobre la máquina), entonces una implementación de ese tipo debe funcionar como una máquina de estados finitos. Por lo tanto, la estructura del código está basada en la definición de movilidad débil. Esto es porque el contador de programa no es transmitido, haciendo imposible que el agente continúe la ejecución desde la línea siguiente a la llamada a **doMove()**
- **afterMove()** : operaciones a realizar apenas el agente se mueve, es decir, en el momento en el que comienza a ejecutarse en el destino al cual fue migrado.

En el proceso de migración del código, el agente atraviesa una serie de estados.



El agente pasa al estado AP_TRANSIT, cuando se hace un **doMove()**, hasta que se mueve a otro container. Una vez esté corriendo sobre el destino, vuelve a tener el estado activo.

Inicialización del entorno

1. Levantar al Main-Container.

```
java -cp lib/jade.jar jade.Boot -gui -local-host 127.0.0.1
```

Una vez que se lanza al Main-Container, se puede comenzar a agregar más containers a la plataforma. El resto de los contenedores se vinculan al Main-Container.

2. Levantar los contenedores necesarios.

3. Compilar y correr al agente que realiza lo solicitado.

```
$ javac -classpath lib/jade.jar -d classes/ ejercicio1/AgenteMovil.java

$ java -cp lib/jade.jar:classes jade.Boot -gui -container -host
localhost -agents mov:AgenteMovil
```

Ejecución de un AgenteMovil

En resumen, la clase AgenteMóvil en su inicialización usa el archivo de texto “containers_list.txt”, del cual obtendrá los nombres de los destinos a los que deberá migrar, agregandose a sí mismo al final de los destinos para poder volver a su container de origen.
También es posible como alternativa, realizar una consulta al agente AMS, el cual podría brindarnos los ContainerID de los containers disponibles en la plataforma.
Los agentes no interactúan mediante la invocación de métodos (como en los casos de gRPC o RMI) sino más bien intercambiando mensajes asincrónicos.
Para el intercambio de información dentro de la plataforma es útil tener presente la manera de formatear mensajes que dispone JADE, las ontologías[1].

Continuando con el agente, al final de la inicialización realiza el primer movimiento, entonces pasará al estado de **tránsito**. Una vez que migra al destino, cambiará al estado **activo** y ejecutará el método **afterMove()**. Allí recupera el estado (memoria, carga de trabajo,etc) de la máquina en la que se está ejecutando actualmente y vuelve a moverse, si aún no terminó el recorrido, o mostrará en pantalla el reporte resultante de desplazarse por las otras computadoras al llegar al punto de partida.

[1] Vocabulario y semántica para el contenido de los mensajes que pueden intercambiar los agentes.

Algoritmo de instantánea

El algoritmo de instantánea consiste en mantener un estado global de un sistema. La manera de hacer esto posible para sistemas distribuidos es que cada proceso guarde su estado (y eventualmente que haya un proceso que recolecta todos esos estados para evaluar todo como uno solo).
Esto se lleva a cabo mediante envío de mensajes. Estos mensajes pueden ser los relacionados a la aplicación o mensajes especiales, llamados “marcadores” , los cuales se usan como indicación para guardar estado.
El hecho de almacenar estados con este algoritmo, le da la capacidad a un sistema distribuido de poder mantener consistencia (si miramos que ocurre en un punto específico de un sistema en funcionamiento tendremos consistencia).

En el caso de nuestro agente el estado es consistente. Para el estado de nuestra aplicación en particular podría no ser relevante el concepto de este algoritmo con dichas características, pero para una aplicación de tipo productores/consumidores por ejemplo, podrían generarse inconsistencias en el sistema si los agentes que interactúan no mantienen un estado global como en el caso del algoritmo presentado.

Tener presente que el estado de este algoritmo es consistente porque **se asume** que “*Ni los canales ni los procesos fallan - la comunicación es confiable de manera tal que cada mensaje enviado es eventualmente recibido intacto, exactamente una vez.*”

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 qué pasaría si hubiera otros sitios con archivos que deben ser procesados de manera similar.

Este agente en principio, será similar al anterior. La diferencia se encontrará en que recibe un parámetro (el nombre del archivo) y en esta ocasión realiza una suma de todos los valores que encuentre en dicho archivo y vuelve al contenedor de origen para informar el resultado.

La forma de indicarle parámetros al agente, es la que se muestra a continuación:

```
java -cp lib/jade.jar:classes jade.Boot -gui -container -container-name
ejercicio2 -host localhost -agents "mov:AgenteMovil(numeros.txt)"
```

En caso de necesitar pasar más de un parámetro separarlos con espacios o comas, según la versión de JADE utilizada.

En una aplicación cliente/servidor, la manera más simple de llevar a cabo esta operatoria, sería solicitar a un host (mediante una petición) que sume los números del archivo indicado, y recibir la respuesta en el cliente; o bien, el cliente solicita que el servidor le mande el archivo para proceder a realizar la suma.

Las situaciones a las que se pueden enfrentar los agentes al migrar a otra parte, pueden ser muy adversas. A continuación se mencionan algunas de ellas:

1. El archivo no existe en el destino
2. No se disponen de los permisos requeridos para poder interactuar con el archivo
3. Se produce un error durante el procesamiento del archivo.

Todas estas situaciones deben ser consideradas para la implementación de este tipo de soluciones.

En caso de que haya varios archivos que deben ser procesados, el agente debe realizar el proceso de migración, cálculo y entrega de resultado, tantas veces como sea necesario.

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

a.- Debería tener como mínimo la misma funcionalidad, es decir las operaciones (definiciones copiadas aquí de la práctica anterior):
leer: dado un nombre de archivo, una posición y una cantidad de bytes a leer, retorna
1) la cantidad de bytes del archivo pedida a partir de la posición dada o en caso de haber menos bytes, se retornan los bytes que haya y
2) la cantidad de bytes que efectivamente se retornan leídos.

Escribir: dado un nombre de archivo, una cantidad de bytes determinada, y un buffer a partir del cual están los datos, se escriben los datos en el archivo dado. Si el archivo existe, los datos se agregan al final, si el archivo no existe, se crea y se le escriben los datos. En todos los casos se retorna la cantidad de bytes escritos.

b.- Implemente un agente que copie un archivo de otro sitio del sistema distribuido en el sistema de archivos local y genere una copia del mismo archivo en el sitio donde está originalmente. Compare esta solución con la de los sistemas cliente/servidor de las prácticas anteriores.

Este agente realiza las operaciones leer y escribir con un objeto **Lector**. El agente por su parte, se encarga de migrar el código según la operación a realizar.
Los parámetros que recibe son **op**, **fuente** y **destino**, dónde:

- **op** es un caracter que indica la operación que se quiere llevar a cabo (w para escribir fuente en destino y r para leer en destino desde fuente)
- **destino** y **fuente**, son las rutas de los archivos involucrados en las operaciones.

```
-- Lectura
java -cp lib/jade.jar:classes jade.Boot -gui -container -container-name
ejercicio3 -host localhost -agents "mov:AgenteMovil(r,
ejercicio3/database_remota/prac4.pdf,
ejercicio3/database_cliente/prac4.pdf)"

-- Escritura
java -cp lib/jade.jar:classes jade.Boot -gui -container -container-name
ejercicio3 -host localhost -agents "mov:AgenteMovil(w,
ejercicio3/database_cliente/prac4.pdf,
ejercicio3/database_remota/prac4_copia.pdf)"
```

Respecto de las soluciones cliente/servidor de las prácticas anteriores, la migración de código parece ser más “pesada”. Para el caso de sockets, RMI y gRPC con enviar mensajes alcanza para intercambiar datos. Desde esa perspectiva la transferencia es mucho más “liviana” ya que solo se envían los datos solicitados, mientras que en JADE, además de los datos, se envía todo el proceso cada vez que se necesita, al menos en esta implementación. De todas maneras podría ser posible implementar agentes que envíen mensajes como se comentó antes en este documento.

La característica interesante que provee JADE respecto a las otras alternativas de comunicación para sistemas distribuidos, es la posibilidad de implementar un solo agente (como en este caso), y de esta manera no se necesitaría tener que instalar/correr clientes y/o servidores en el resto de las computadoras que están en la plataforma, como si se hace con las herramientas utilizadas en las prácticas anteriores.

Bibliografía

JADE API v3.5
https://depinfo.u-cergy.fr/~pl/wdocs/sma/doc/api/jade/core/package-summary.html [Último acceso: 21/11/2021]

Developing Multi Agent Systems, *Fabio Luigi Bellifemine, Giovanni Caire, Dominic Greenwood.*

Monitoreo para la JVM:
https://docs.oracle.com/javase/6/docs/api/java/lang/management/package-summary.html
https://stackoverflow.com/questions/25552/get-os-level-system-information/61727 [Último acceso: 21/11/2021]

Capítulo 6, *Distributed systems - Concepts and design.*