

Practica 4

Inti María Tidball 17612/3

Juan Pablo Sanchez Magariños 13238/3

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.

Se crea una clase que se llama **MultipleContainers** que crea 10 containers de manera programática, se encarga de configurar una red de diez contenedores, simulando una serie de computadoras interconectadas en una red. El Main-Container actúa como el núcleo de esta red, mientras que los otros nueve contenedores representan nodos individuales en la red, que son los que se va a recorrer juntando información.

De la misma manera, el AgenteMovil tiene una lista de nombres de nodos con el fin de poder recorrerlos. Está programado para desplazarse secuencialmente entre los contenedores, su función es recolectar datos críticos de cada nodo. Estos datos incluyen la carga de procesamiento, la memoria disponible y los nombres de los nodos. La movilidad del agente se logra mediante el método `doMove()`, que le permite visitar todos los contenedores de manera eficiente y ordenada, gracias a una lógica modular basada en el índice del contenedor y el tamaño total de la lista de contenedores.

Se considera simplemente imprimir la información en cada contenedor, por consejo de una profesional en esta area, se decidió guardar la información usando una clase interna serializable al Agente llamada **ContainerInfo**, Durante su recorrido, el AgenteMovil almacena la información recolectada en instancias de la clase interna **ContainerInfo**. Esta clase, diseñada para ser serializable, facilita la gestión eficiente de los datos recogidos en cada nodo. La función `getInfoAsString()` de esta clase permite la presentación estandarizada de la información recogida, que luego es reportada al Main-Container al finalizar el recorrido del agente.

Por último, se recorre la última lista imprimiendo toda la información solicitada y se imprime el tiempo de recorrido. El tiempo total que el agente toma para completar su recorrido se calcula utilizando marcas de tiempo generadas por `System.currentTimeMillis()` al comienzo y al final del ciclo de recorrido.

Nota: Para compilar y ejecutar todos los ejercicios de esta práctica es necesario obtener y colocar el .jar de JADE en el directorio lib/. No se incluye en la entrega como pedía la consigna. Para cada ejercicio, se tienen 3 scripts, para compilar, correr la gui y ejecutar los agentes, los cuales se explican en cada punto.

En este caso el código se encuentra en el directorio 1/

Instrucciones para compilar (./1.compilacion.sh):

```
javac -classpath ../lib/jade.jar -d ./classes ./myexamples/AgenteMovil.java  
./myexamples/MultipleContainers.java
```

Instrucciones para ejecutarlo:

Terminal 1 (./2.gui.sh)

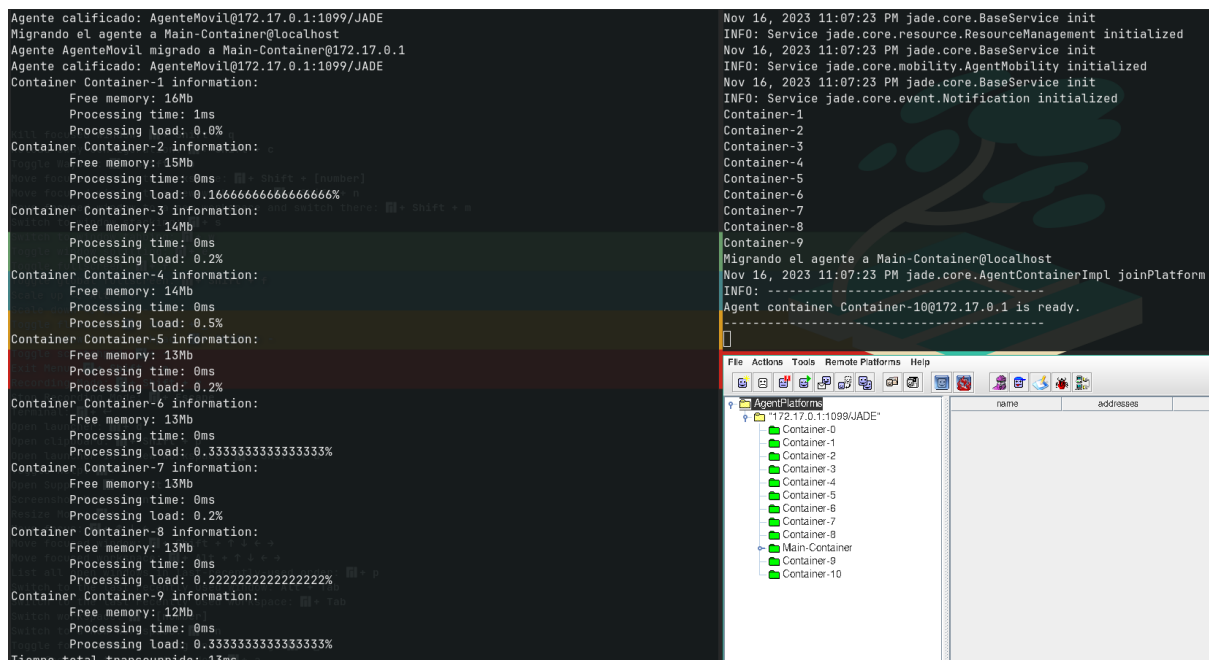
```
java -cp ../lib/jade.jar:classes myexamples.MultipleContainers
```

Terminal 2 (./3.agente.sh)

```
java -cp ../lib/jade.jar:classes jade.Boot -container -host localhost -port  
1099 -agents "AgenteMovil:myexamples.AgenteMovil()"
```

Si se utilizara el algoritmo de instantánea, cada uno de los containers tendría un proceso y estos deberían sincronizarse entre ellos para ejecutarse al mismo tiempo.

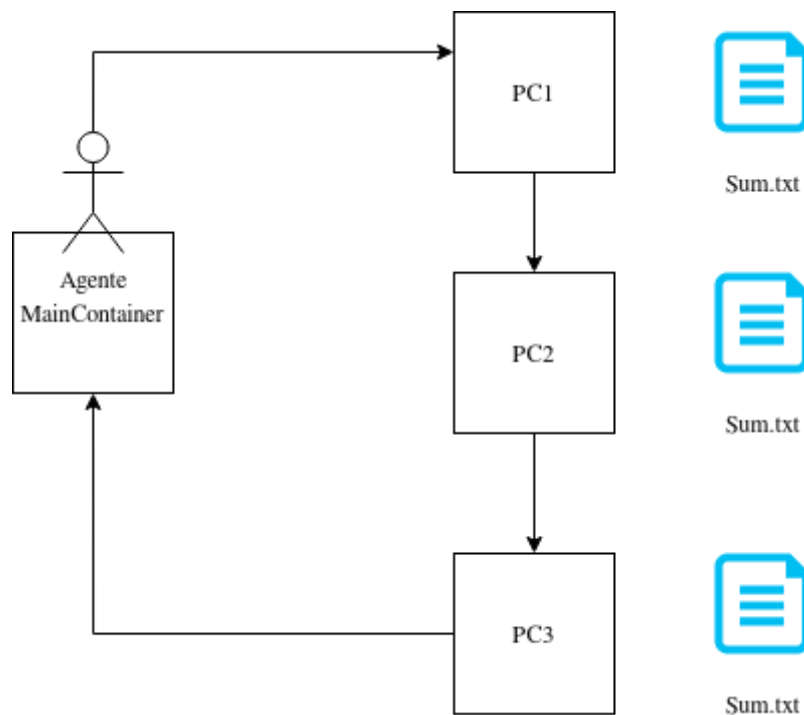
Es por ello que la principal diferencia se daría en el tiempo de ejecución, el algoritmo de instantánea tendrá un tiempo menor al original.



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.

Se toma la decisión de resolver el problema con múltiples contenedores para poder también entender lo que se plantea en la pregunta final. Similarmente al ejercicio anterior, se crea una clase que se llama **MultipleContainers** que crea 3 containers de manera programática, se encarga de configurar una red de 3 contenedores, simulando una serie de computadoras interconectadas en una red.

Se crea adicionalmente una clase de tipo agente, **AgenteMovil**, que recibe por parámetro el *path* del archivo, y luego de migrar a la primera computadora, lee el archivo, realiza la suma de los números almacenados en dicho archivo, se guarda el resultado, migra a la próxima computadora, vuelve a leer el archivo, realiza la suma de los números almacenados en ese archivo, guarda la suma, y así en cada computadora hasta retornar al *Main-Container*, adonde suma todas las sumas almacenadas y muestra el resultado.



Instrucciones para compilar (**./1.compilacion.sh**):

```
javac -classpath ../lib/jade.jar -d classes myexamples/AgenteMovil.java
myexamples/MultipleContainers.java -Xdiags:verbose
```

Instrucciones para ejecutarlo

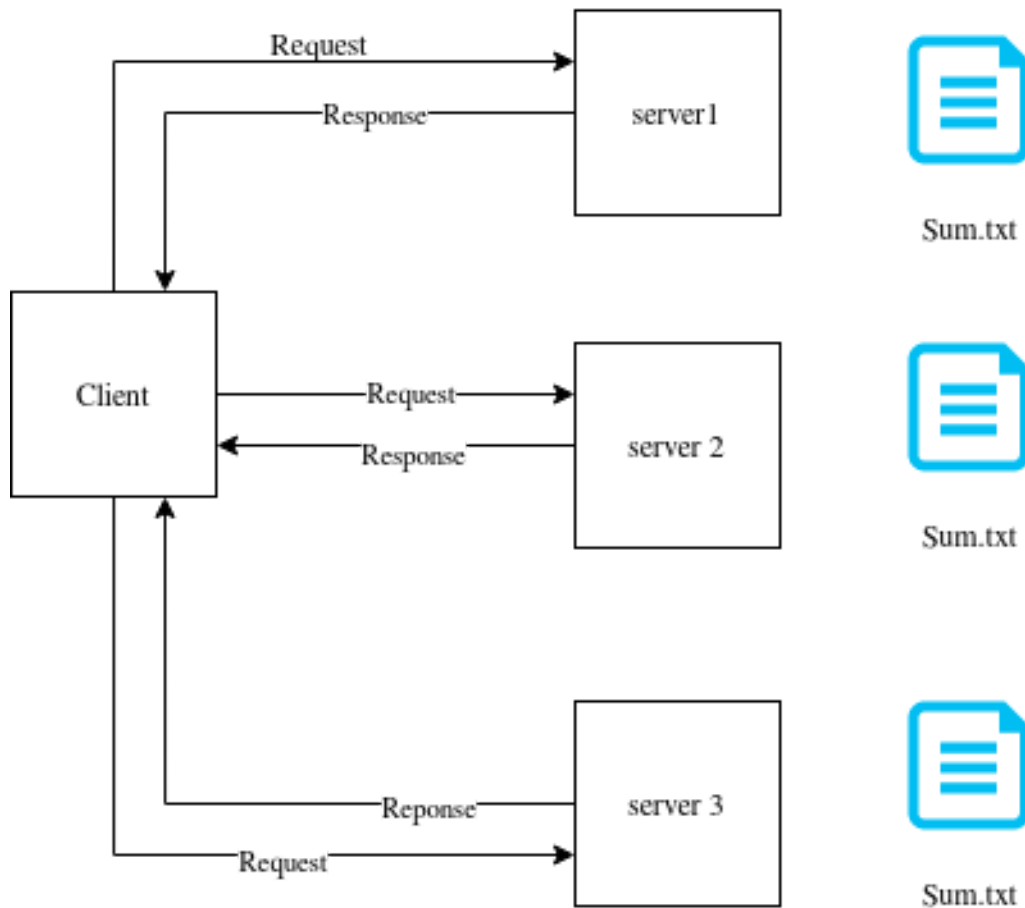
Terminal 1 (**./2.gui.sh**)

```
java -cp ../lib/jade.jar:classes myexamples.MultipleContainers
```

Terminal 2 (**./3.agente.sh**)

```
java -cp ../lib/jade.jar:classes jade.Boot -container -host localhost  
-port 1099 -agents "AgenteMovil:myexamples.AgenteMovil($PWD/sum.txt)"
```

Si se quisiera implementar este funcionamiento en una app de arquitectura cliente/servidor, el servidor debería tener acceso al filesystem en el cual se almacena el archivo, y se debería exponer un servicio que lee los contenidos del archivo. El servidor se encargará de procesar y retornar la suma solicitada, o a lo sumo retornar los números en el archivo y delegar al cliente el procesamiento. En el caso de haber que leer varios archivos distribuidos, debería haber varios servidores con servicios similares expuestos al cual el cliente pueda acceder. En este caso, sería necesario que el cliente se encargue de hacer la suma final.



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.

En este proyecto, se planteó un agente llamado FPTAgent, con una clase auxiliar que procesa los comandos. La clase del FPTAgent ejecuta operaciones de lectura, escritura y lectura-escritura sobre archivos en un sistema de archivos entre contenedores. Este agente se comporta tanto como cliente como servidor.

Funcionalidad:

- Lectura (read): El agente lee una porción del archivo remoto, específicamente 200,000 bytes cada vez, y los almacena en una variable. Si el archivo es más grande, se repite este proceso hasta completar la lectura.
- Escritura (write): Similar a la lectura, pero en sentido inverso. Aquí, el agente escribe los datos en el servidor, leyendo del archivo local.
- Lectura y Escritura (readwrite): Esta función combina las dos anteriores, copiando un archivo del sistema local al distribuido y luego creando una copia en su ubicación original.

Para la ejecución, se reutilizaron los scripts de ejemplo:

1.compile.sh

```
javac -classpath ../lib/jade.jar -d classes FPTAgent.java FTPUtils.java
```

2.gui.sh

```
java -cp ../lib/jade.jar:classes jade.Boot -gui
```

Y al **3.agente.sh** se le agrega una logica para recibir paramentros:

```
#!/bin/bash

if [ "$#" -ne 3 ]; then
    echo "Usage: $0 [read|write|readwrite] localPath remotePath"
    exit 1
fi

op=$1
local_path=$2
remote_path=$3

java -cp ../lib/jade.jar:classes jade.Boot -gui -container -host
localhost -agents "mol:FTPAgent($op,$local_path,$remote_path)"
```

Ejemplos con archivos binarios (una imagen):

READ:

```
> ./3.agente.sh read files/dst/new.jpg files/src/pika.jpg
```

```
> ./3.agente.sh read files/new.jpg files/pika.jpg
Nov 16, 2023 1:31:06 PM jade.core.Runtime beginContainer
INFO: -----
This is JADE 4.3.2 - revision 6708 of 2014/03/28 15:19:44
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
-----
```

```
Nov 16, 2023 1:31:06 PM jade.core.BaseService init
INFO: Service jade.core.resource.ResourceManagement initialize
d
Nov 16, 2023 1:31:06 PM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
Nov 16, 2023 1:31:06 PM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
Nov 16, 2023 1:31:06 PM jade.core.AgentContainerImpl startBoot
strapAgents
SEVERE: Cannot create agent rma: Name-clash Agent rma@172.17.0
.1:1099/JADE already present in the platform
Nov 16, 2023 1:31:06 PM jade.core.AgentContainerImpl joinPlatf
orm
INFO:
Agent container Container-1@172.17.0.1 is ready.
-----
Migrating agent to Main-Container@<Unknown Host>
Written 6965 bytes to local
Read operation completed successfully.
```

name	addresses	state	owner
NAME	ADDRESSES	STATE	OWNER

```
oot
INFO: MTP addresses:
http://konoa:7778/acc
Nov 16, 2023 1:31:04 P
one
INFO:
Agent container Main-Container@172.17.0.1 is ready.
-----
Nov 16, 2023 1:31:06 PM jade.core.PlatformManagerImpl localAdd
Node
INFO: Adding node <Container-1> to the platform
Nov 16, 2023 1:31:06 PM jade.core.PlatformManagerImpl$1 nodeAd
ded
INFO: --- Node <Container-1> ALIVE ---
Nov 16, 2023 1:31:06 PM jade.intp.leap.NodeSkel executeCommand
WARNING: Error serving H-Command jade.core.management.AgentMan
agement/4: jade.core.NameClashException: Name-clash Agent rma@
172.17.0.1:1099/JADE already present in the platform
Read 6965 bytes from remote
```

Screenshot copied to clipboard

```
~/Dev/unlp-pdytr/tp4/3 main*
> diff files/new.jpg files/pika.jpg
Toggle focus between tiling and floating: [F]
```

WRITE:

```
> ./3.agente.sh write files/new.jpg files/new-new.jpg
Nov 16, 2023 1:32:56 PM jade.core.Runtime beginContainer
INFO: This is JADE 4.3.2 - revision 6708 of 2014/03/28 15:19:44
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
```

The screenshot shows the JADE IDE interface. On the left, the 'AgentPlatforms' tree is expanded, showing a hierarchy: '172.17.0.1:1099/JADE' -> 'Main-Container' -> 'Container-1'. The 'Container-1' node is selected, and its details are shown in the table below:

name	addresses	state	owner
NAME	ADDRESSES	STATE	OWNER

On the right, the console output shows the following messages:

```
oot
INFO: MTP addresses:
http://konoha:7778/acc
Nov 16, 2023 1:32:54 PM jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Main-Container@172.17.0.1 is ready.
-----
Nov 16, 2023 1:32:56 PM jade.core.PlatformManagerImpl localAddNode
INFO: Adding node <Container-1> to the platform
Nov 16, 2023 1:32:56 PM jade.core.PlatformManagerImpl$1 nodeAdded
INFO: --- Node <Container-1> ALIVE ---
Nov 16, 2023 1:32:56 PM jade.imtp.leap.NodeSkel executeCommand
WARNING: Error serving H-Command jade.core.management.AgentManagement/4: jade.core.NameClashException: Name-clash Agent rma@172.17.0.1:1099/JADE already present in the platform
Written 6965 bytes to remote
```

```
~/Dev/unlp-pdytr/tp4/3 main*
> diff files/new.jpg files/new-new.jpg
```

READWRITE:

The screenshot shows the JADE IDE interface. On the left, the console output shows the following messages:

```
Agent container Main-Container@172.19.0.1 is ready.
-----
Nov 18, 2023 10:44:45 AM jade.core.PlatformManagerImpl localAddNode
INFO: Adding node <Container-1> to the platform
Nov 18, 2023 10:44:45 AM jade.core.PlatformManagerImpl$1 nodeAdded
INFO: --- Node <Container-1> ALIVE ---
Nov 18, 2023 10:44:45 AM jade.imtp.leap.NodeSkel executeCommand
WARNING: Error serving H-Command jade.core.management.AgentManagement/4: jade.core.NameClashException: Name-clash Agent rma@172.19.0.1:1099/JADE already present in the platform
I am at: Main-Container@172.19.0.1
Reading from origin!
Read 6965 bytes from origin
I am at: Main-Container@172.19.0.1
Writing copy!
Written 6965 bytes to local copy
Read-write operation completed successfully.
```

On the right, the file explorer shows the following files:

- algo.txt
- one.txt
- pika.jpg
- pika-remote.jpg
- pika.jpg.cpy
- zero.txt

```
~/Dev/unlp-pdytr/tp4/3 main* :~  
) diff files/src/pika.jpg files/src/pika.jpg.cpy  
Switch to the last recently used workspace: ~+ Tab  
~/Dev/unlp-pdytr/tp4/3 main* :~  
) diff files/src/pika.jpg files/dst/pika-remote.jpg  
Toggle focus between tiling and floating: ~+ _  
~/Dev/unlp-pdytr/tp4/3 main* :~+ a
```

Este enfoque, aunque unifica el cliente y el servidor en un solo agente, resulta en una mayor complejidad y potencialmente en una menor escalabilidad y mantenimiento. Además, el uso intensivo de migraciones puede incrementar el overhead en comparación con las implementaciones cliente/servidor tradicionales.