

# Tarea 6

## Multiplicación de matrices

Usando objetos distribuidos – RMI | Sistemas distribuidos

Jueves, 12 de noviembre de 2020

Joel Harim Hernández Javier

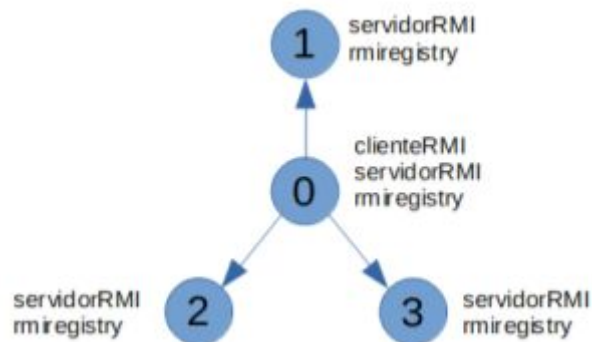
## Descripción

Cada alumno deberá desarrollar un sistema que calcule el producto de dos matrices cuadradas utilizando Java RMI, tal como se explicó en clase.

Se **deberá** ejecutar dos casos:

- N = 4, se deberá desplegar las matrices A, B y C y el checksum de la matriz C.
- N = 500, deberá desplegar el checksum de la matriz C.

El servidor RMI se ejecutará en cuatro máquinas virtuales Ubuntu (nodo 0, nodo 1, nodo 2 y nodo 3). El programa `rmiregistry` se ejecutará en cada nodo donde se ejecute el servidor RMI.



El cliente RMI, el cual ejecutará en el nodo 0, inicializará las matrices A y B, obtendrá la transpuesta de la matriz B, invocará el método remoto `multiplica_matrices()`, calculará el checksum de la matriz C, y en su caso (N = 4) desplegará las matrices A, B y C.

Se **deberá** utilizar las funciones que vimos en clase: `parte_matriz()`, `multiplica_matrices()` y `acomoda_matriz()`.

Se **deberá** subir a la plataforma un archivo ZIP que contenga el código fuente de los programas desarrollados y el reporte de la tarea en formato PDF. El reporte de la tarea deberá incluir portada, desarrollo, captura de pantallas de la compilación y ejecución, y conclusiones.

## Desarrollo

En esta práctica se implementó, usando las funciones y los programas ejemplos ya desarrollados en clase y en prácticas pasadas, un sistema distribuido que calcula la multiplicación de 2 matrices cuadradas con dimensión par (para que puedan ser divididas) que utiliza varios nodos servidor Java RMI.

El sistema cuenta con dos programas base, uno es el `ServerRMI.java`, que es el programa que sirve como enlace entre la clase `ClassRMI` que contiene el método remoto y el servidor `rmiregistry` de Java.

El servidor recibe como parámetros el tamaño a utilizar de la matriz y el puerto que usará el servidor `rmiregistry`, esto con el propósito de soportar casos donde se usa el servidor `rmiregistry` con un puerto distinto al default que es 1099, por ejemplo, en una prueba local donde se tienen que correr varias instancias del mismo con distinto número de puerto.

La otra parte fundamental del sistema es el programa `ClientRMI.java`, el cual es el encargado de inicializar las matrices y hacer la invocación del método remoto que finalmente utiliza para reconstruir la matriz resultante.

El cliente recibe como parámetros el tamaño de la matriz a utilizar y 4 direcciones IPv4 junto con el puerto a utilizar, lo que le permite soportar casos donde se utilicen servidores `rmiregistry` con un número de puerto distinto al default (1099).

Cómo se realizarán 4 invocaciones al método remoto de multiplicar, se lanzan 4 hilos, los cuales se encargan de multiplicar los segmentos de las matrices A y B y el resultado lo guardan en una matriz resultante temporal, que posteriormente será copiada a la matriz resultante final en el hilo principal:

```
public static void main(String[] args) throws Exception {
    validateArguments(args);

    initializeMatrix();

    Worker[] workers = new Worker[4];

    for (int i = 0; i < NODES; ++i) {
        workers[i] = new Worker(args[i + 1], i);
        workers[i].start();
    }

    for (int i = 0; i < NODES; ++i) {
        workers[i].join();
    }

    showResults();
}
```

Código del método `run()` de los hilos:

```
public void run() {
    try {
        String url = "rmi://" + ip + "/prueba";

        InterfaceRMI remote = (InterfaceRMI) Naming.lookup(url);
    }
}
```

```

        int a = node == 0 || node == 1 ? 0 : 1;
        int b = node == 0 || node == 2 ? 0 : 1;

        Cs[node] = remote.multiplica_matrices(As[a], Bs[b]);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

```

Finalmente, el método `showResults()` reconstruye la matriz resultante, calcula el checksum y muestra la matriz C si es necesario:

```

static void showResults() {

    acomoda_matriz(C, Cs[0], 0, 0);
    acomoda_matriz(C, Cs[1], 0, N / 2);
    acomoda_matriz(C, Cs[2], N / 2, 0);
    acomoda_matriz(C, Cs[3], N / 2, N / 2);

    long checksum = 0;

    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            checksum += C[i][j];
        }
    }

    System.out.println("The checksum is: " + checksum);

    if (N != 4) {
        return;
    }

    System.out.println("C:");
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            System.out.print(C[i][j] + " ");
        }
        System.out.println("");
    }
}

```

## Despliegue

Para hacer el *deployment* del sistema distribuido, primero se crearon las máquinas virtuales Ubuntu en Azure liberando el puerto 1099. Se instaló el jdk de Java y también la utilidad Make.

Para subir los archivos a sus respectivos servidores, utilicé un script escrito en Python para auxiliarme para ese fin. Esto con el propósito de poder desplegar cambios en los archivos si fuese necesario de forma más fácil (lo cual sucedió):

```
distribuidos/Joel/Practicas/06_matrices_objetos_distribuidos on p Joel [!?] took 9s
→ ./upload.py
[nodo: 0 , ip: 13.68.181.122 ] Enviado ServerRMI.java a ~/practice_06/
[nodo: 0 , ip: 13.68.181.122 ] Enviado InterfaceRMI.java a ~/practice_06/
[nodo: 0 , ip: 13.68.181.122 ] Enviado Makefile a ~/practice_06/
[nodo: 0 , ip: 13.68.181.122 ] Enviado ConsoleColors.java a ~/practice_06/
[nodo: 0 , ip: 13.68.181.122 ] Enviado ClientRMI.java a ~/practice_06/
[nodo: 1 , ip: 13.82.215.91 ] Enviado ServerRMI.java a ~/practice_06/
[nodo: 1 , ip: 13.82.215.91 ] Enviado InterfaceRMI.java a ~/practice_06/
[nodo: 1 , ip: 13.82.215.91 ] Enviado Makefile a ~/practice_06/
[nodo: 1 , ip: 13.82.215.91 ] Enviado ConsoleColors.java a ~/practice_06/
[nodo: 2 , ip: 13.92.225.140 ] Enviado ServerRMI.java a ~/practice_06/
[nodo: 2 , ip: 13.92.225.140 ] Enviado InterfaceRMI.java a ~/practice_06/
[nodo: 2 , ip: 13.92.225.140 ] Enviado Makefile a ~/practice_06/
[nodo: 2 , ip: 13.92.225.140 ] Enviado ConsoleColors.java a ~/practice_06/
[nodo: 3 , ip: 13.82.45.250 ] Enviado ServerRMI.java a ~/practice_06/
[nodo: 3 , ip: 13.82.45.250 ] Enviado InterfaceRMI.java a ~/practice_06/
[nodo: 3 , ip: 13.82.45.250 ] Enviado Makefile a ~/practice_06/
[nodo: 3 , ip: 13.82.45.250 ] Enviado ConsoleColors.java a ~/practice_06/

distribuidos/Joel/Practicas/06_matrices_objetos_distribuidos on p Joel [!?] took 19s
→
```

## Compilación

Utilicé un archivo Makefile auxiliar para la compilación y la ejecución de los programas en cada nodo. Aquí está la compilación en el nodo 0:

```
(joel) 13.68.181.122 — Konsole

joel@ubuntu0:~/practice_06$ make
javac ConsoleColors.java
javac InterfaceRMI.java
javac ServerRMI.java
javac ClientRMI.java
joel@ubuntu0:~/practice_06$
```

Aquí la compilación del nodo 1, nótese que no se compilo el cliente:

```
(joel) 13.82.215.91 — Konsole

joel@ubuntu1:~/practice_06$ make ServerRMI.class
javac ConsoleColors.java
javac InterfaceRMI.java
javac ServerRMI.java
joel@ubuntu1:~/practice_06$
```

La compilación de los demás nodos fue parecida.

## Ejecución N = 4

En otras sesiones ssh de cada nodo se ejecutó el servidor `rmiregistry`:

```
(joel) 13.68.181.122 — Konsole <2>
joel@ubuntu0:~/practice_06$ rmiregistry
█
```

```
(joel) 13.82.215.91 — Konsole <2>
joel@ubuntu1:~/practice_06$ rmiregistry
█
```

```
(joel) 13.92.225.140 — Konsole <2>
joel@ubuntu2:~/practice_06$ rmiregistry
█
```

```
(joel) 13.82.45.250 — Konsole
joel@ubuntu3:~/practice_06$ rmiregistry
█
```

Después, en la sesión donde se compilaron los programas, se ejecutaron los servidores, aquí muestro la captura con **N = 4** (usando el archivo Makefile auxiliar) en el nodo 0:

```
(joel) 13.68.181.122 — Konsole
joel@ubuntu0:~/practice_06$ make run_remote_
server_4
java ServerRMI 4 1099
rmi://localhost:1099/prueba
█
```

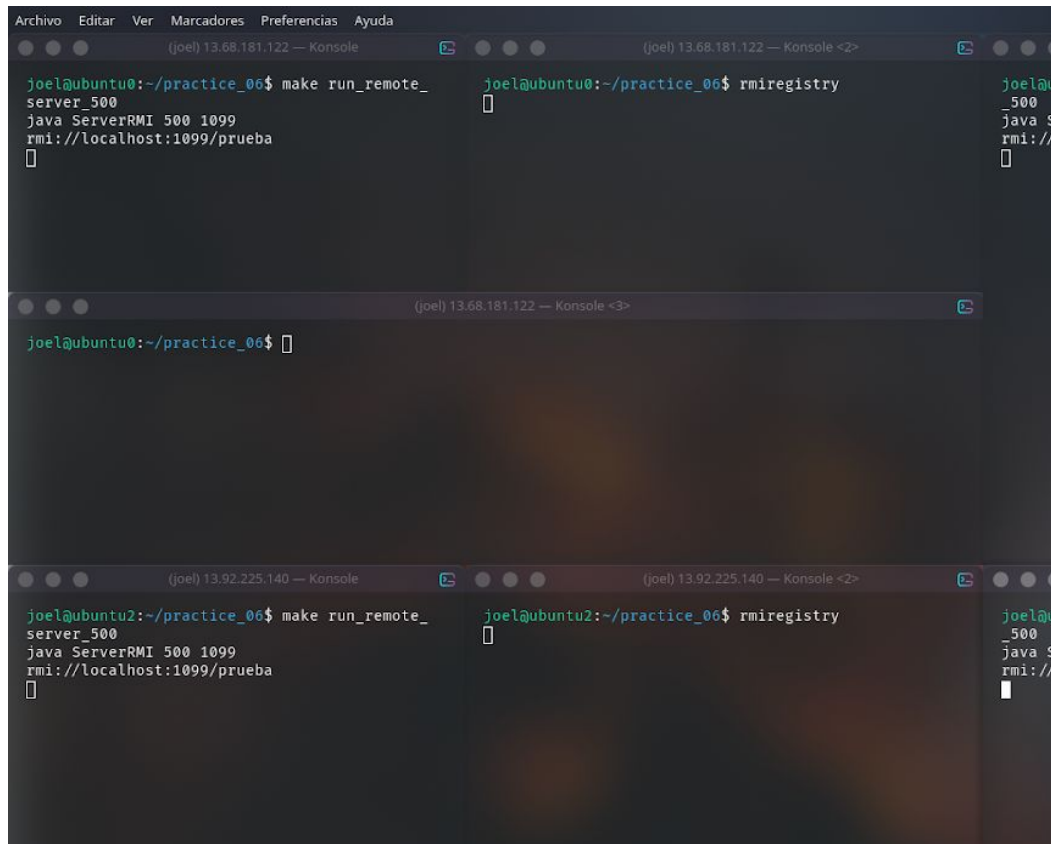
Los demás servidores en los nodos fueron ejecutados de forma parecida.

Finalmente, en otra sesión del nodo 0, ejecutamos el cliente con **N = 4** como parámetro:

```
(joel) 13.68.181.122 — Konsole <3>
joel@ubuntu0:~/practice_06$ make run_client_4
java ClientRMI 4 13.68.181.122:1099 13.82.215.91:1099 13.92.225.140:1099 13.82.45.250:1099
A:
0 -1 -2 -3
2 1 0 -1
4 3 2 1
6 5 4 3
B:
0 1 2 3
2 3 4 5
4 5 6 7
6 7 8 9
The checksum is: 272
C:
-28 -34 -40 -46
-4 -2 0 2
20 30 40 50
44 62 80 98
joel@ubuntu0:~/practice_06$ █
```

## Ejecución N = 500

Para la ejecución del cliente con **N = 500**, se tuvo que parar los procesos servidores (no los rmi) y ejecutarlos con el parámetro **N = 500**. Captura de todas las terminales usadas:

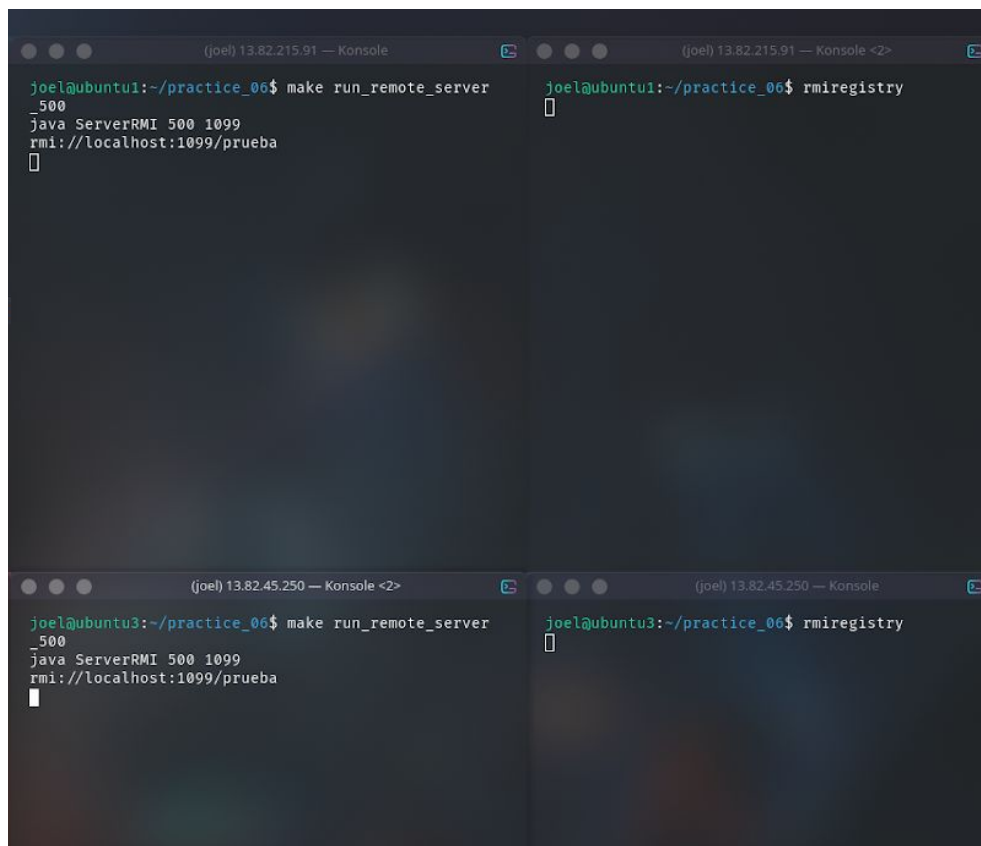


The screenshot shows two terminal windows. The top window is titled '(joel) 13.68.181.122 — Konsole' and contains the following commands and output:

```
joel@ubuntu0:~/practice_06$ make run_remote_server_500
java ServerRMI 500 1099
rmi://localhost:1099/prueba
```

The bottom window is titled '(joel) 13.92.225.140 — Konsole' and contains the following commands and output:

```
joel@ubuntu2:~/practice_06$ make run_remote_server_500
java ServerRMI 500 1099
rmi://localhost:1099/prueba
```



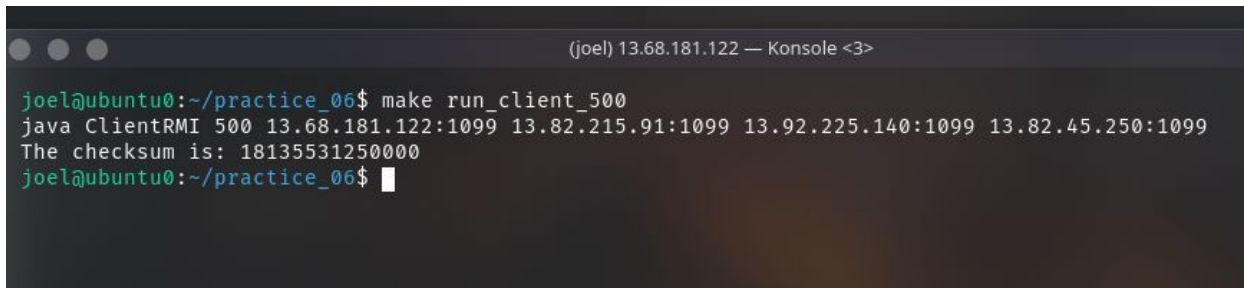
The screenshot shows two terminal windows. The top window is titled '(joel) 13.82.215.91 — Konsole' and contains the following commands and output:

```
joel@ubuntu1:~/practice_06$ make run_remote_server_500
java ServerRMI 500 1099
rmi://localhost:1099/prueba
```

The bottom window is titled '(joel) 13.82.45.250 — Konsole' and contains the following commands and output:

```
joel@ubuntu3:~/practice_06$ make run_remote_server_500
java ServerRMI 500 1099
rmi://localhost:1099/prueba
```

Finalmente ejecuté el cliente de nuevo, pero indicando que **N = 500**:

A terminal window titled "(joel) 13.68.181.122 — Konsole <3>". The prompt is "joel@ubuntu0:~/practice\_06\$". The user enters "make run\_client\_500". The terminal shows the command "java ClientRMI 500 13.68.181.122:1099 13.82.215.91:1099 13.92.225.140:1099 13.82.45.250:1099" and the output "The checksum is: 18135531250000". The prompt returns to "joel@ubuntu0:~/practice\_06\$".

```
(joel) 13.68.181.122 — Konsole <3>
joel@ubuntu0:~/practice_06$ make run_client_500
java ClientRMI 500 13.68.181.122:1099 13.82.215.91:1099 13.92.225.140:1099 13.82.45.250:1099
The checksum is: 18135531250000
joel@ubuntu0:~/practice_06$
```

## Conclusiones

El desarrollo de este sistema fue desafiante. Uno de los primeros problemas que tuve fue el hacer el enlace entre la instancia de la clase `ClassRMI` y el servidor RMI (el cuál resultó ser un problema de la Interface más que de la clase en sí).

Otro fue el resolver cómo hacer las pruebas en mi entorno local, con lo que tuve que modificar los programas y desarrollar scripts escritos en Bash.

Finalmente, el último problema que tuve respecto al sistema y a su despliegue fue darme cuenta que el servidor `rmiregistry` tiene que ser ejecutado dentro de la carpeta raíz de mi proyecto, para que así detecte las clases a utilizar.

Esta práctica me permitió practicar el desarrollo de algunos scripts escritos en Bash (pruebas locales) y en Python (para el despliegue del sistema) que utilicé para la automatización de ciertas tareas repetitivas y las cuales son susceptibles a errores de escritura, como el ingreso a mano de los parámetros de cada programa.