INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

"Tarea 6. Multiplicación de matrices utilizando objetos distribuidos"

Alumno:
Lara Cázares Jaime Arturo

Materia:
Desarrollo de sistemas distribuidos

Grupo:
4CM3

Profesor: Pineda Guerrero Carlos

Índice general

1. Desarrollo

Para desarrollar esta práctica es necesario desarrollar los siguientes programas en Java:

- Interfaz RMI
- Clase RMI
- Servidor RMI
- Cliente RMI

1.1. Interfaz RMI

Una interfaz es la responsable de definir el comportamiento de la clase que la implementa (en este caso ClaseRMI), la interfaz desarrollada es la siguiente:

```
import java.rmi.RemoteException;
import java.rmi.*;


public interface InterfaceRMI extends Remote{
    public int[][] multiplica_matrices(int[][] A,int[][] B) throws RemoteException;
}
```

Como vemos la interfaz define únicamente el método *multiplica_matrices()* la cual es la encargada de multiplicar las dos matrices que se le pasen por argumentos y maneja las excepciones *java.rmi.RemoteException*. Hay que notar que el método aún no se implementa.

1.2. ClaseRMI

Como anteriormente se dijo, la interfaz solo define el comportamiento de la clase pero no implementa los métodos, por ello debemos crear una clase que si implemente los métodos (en este caso solo *multiplica_matrices()*). Dicha clase en esta practica es *ClaseRMI* y se presenta a continuación:

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ClaseRMI extends UnicastRemoteObject implements InterfaceRMI{
    // es necesario que el contructor ClaseRMI() invoque el constructor de la superclase
    public ClaseRMI() throws RemoteException{
        super();
    }

    public int[][] multiplica_matrices(int[][] A,int[][] B) throws RemoteException{
```

```
        int N = A[0].length;
        int[][] C = new int[N/2][N/2];
        for (int i = 0; i < N/2; i++)
            for (int j = 0; j < N/2; j++)
            for (int k = 0; k < N; k++)
                C[i][j] += A[i][k] * B[j][k];
        return C;
    }

}
```

Esta clase debe heredar de *java.rmi.server.UnicastRemoteObject*, implementar la interfaz *InterfaceRMI*, manejar las excepciones *java.rmi.RemoteException* y su constructor debe invocar al constructor de la súper clase. Para este programa sabemos que las matrices son cuadradas y de las mismas dimensiones, entonces es fácil obtener la variable "N" a través de la longitud de la matriz A que llega al método.

1.3. ServidorRMI

Ahora es turno de implementar el servidor, el cual es el encargado de instancias un objeto de la clase *ClaseRMI* y registrarlo en el **rmiregistry** para que pueda ser ocupado remotamente. El servidor desarrollado es el siguiente;

```
import java.rmi.Naming;
import java.lang.Exception;


public class ServidorRMI{
    public static void main(String[] args) throws Exception{
        String url = "rmi://localhost/multiplicaMatriz";
        try {
            if(esNumero(args[0])){
                //Cuando se desea utilizar en localhost y tener multiples servidores
                //se agrega un identificador al nombre que en este caso es un entero
                //que representa el número del nodo.
                url = "rmi://localhost/multiplicaMatriz" + args[0];
            } else{
                //Cuando se utiliza en maquinas virtuales se utiliza la ip que corresponde
                //a dicha maquina virtual
                url = "rmi://" + args[0] + "/multiplicaMatriz";
            }

        } catch (Exception e) {
            //TODO: handle exception
            System.err.println("Uso:");
```

```
            System.err.println("java ServidorRMI <ip de server>");
            System.err.println("java ServidorRMI <nodo>");
            System.exit(0);
        }

        System.out.println("Url: " + url);
        ClaseRMI obj = new ClaseRMI();
        // registra la instancia en el rmiregistry
        Naming.bind(url,obj);
        System.out.println("Servidor "+ args[0] +" iniciado con exito!");
    }

    public static boolean esNumero(String strNum) {
        if (strNum == null) {
            return false;
        }
        try {
            double d = Integer.parseInt(strNum);
        } catch (NumberFormatException nfe) {
            return false;
        }
        return true;
    }
}
```

Para este servidor tenemos dos tipos de uso:

1. **Para uso en localhost:** el cual se le debe pasar como argumento el número del nodo que se trate, así su url sería la siguiente

   rmi://localhost/multiplicaMatriz(Nodo).

2. **Para uso en maquina virtual:** el cual se le debe pasar como argumento la dirección IP a la cual pertenece la máquina virtual, así su url quedaría así **rmi://dirección IP/multiplicaMatriz**.

Para ello se implementa el método *esNumero* el cual regresa regresa **Falso** si se trata de una dirreción IP o **Verdadero** si es un número entero que se interpreta como el nodo.

La clase instancia un objeto de la clase *ClaseRMI* y lo registra en el *rmiregistry* con el método *bind()* de la clase *Naming*.

1.4. ClienteRMI

Por último se debe desarrollar la clase del cliente que es la encargada de inicializar las matrices, partirlas y acomodarlas en la matriz resultante C. También debe de instanciar a los objetos remotos de los servidores. El código del cliente desarrollado es el siguiente:

```java
import java.io.IOException;
import java.rmi.Naming;
import java.lang.Exception;

public class ClienteRMI_N4{
    static int N = 4;//Tamanio de matrices
    static int[][] A = new int[N][N];//Declaracion dematriz de NxN
    static int[][] B = new int[N][N];//Declaracion dematriz de NxN
    static int[][] C = new int[N][N];//Declaracion dematriz de NxN
    static long checksum = 0;

    public static void main(String args[]) throws Exception{
        String[] urls = crearUrls(args);

        System.out.println("Urls:");
        for(String url : urls){
            System.out.println(url);
        }

        //Obtiene una referencia que "apunta" al objeto remoto asociado a la URL
        //Dado a que el servidor 0 se ejecuta en el mismo nodo que el cliene
        //su url se mantiene constante en localhost

        InterfaceRMI r0 = (InterfaceRMI)Naming.lookup("rmi://localhost/multiplicaMatriz0");
        InterfaceRMI r1 = (InterfaceRMI)Naming.lookup(urls[0]);
        InterfaceRMI r2 = (InterfaceRMI)Naming.lookup(urls[1]);
        InterfaceRMI r3 = (InterfaceRMI)Naming.lookup(urls[2]);

        //Inicializamos las matrices
        inicializaMatrices();


        //Trasnponemos la matriz B
        B = transponerMatriz(B);

        //Obtenemos las matrices A1, A2, B1 y B2
        int[][] A1 = parte_matriz(A,0);
        int[][] A2 = parte_matriz(A,N/2);
        int[][] B1 = parte_matriz(B,0);
        int[][] B2 = parte_matriz(B,N/2);

        //Objetenemos la multiplicación de matrices
        int[][] C1 = r0.multiplica_matrices(A1,B1);
        int[][] C2 = r1.multiplica_matrices(A1,B2);
        int[][] C3 = r2.multiplica_matrices(A2,B1);
        int[][] C4 = r3.multiplica_matrices(A2,B2);
```

```
        acomoda_matriz(C,C1,0,0);
        acomoda_matriz(C,C2,0,N/2);
        acomoda_matriz(C,C3,N/2,0);
        acomoda_matriz(C,C4,N/2,N/2);

        if(N == 4){
            imprimirResultados();
        }

        System.out.println("Checksum="+calcularChecksum(C));
    }

    static void inicializaMatrices(){
        for (int i = 0; i < N; i++){
            for (int j = 0; j < N; j++){
                A[i][j] = 2 * i - j;
                B[i][j] = 2 * i + j;
                C[i][j] = 0;
            }
        }

    }

    static int[][] transponerMatriz(int[][] matriz){
        for (int i = 0; i < N; i++){
            for (int j = 0; j < i; j++){
                int x = matriz[i][j];
                matriz[i][j] = matriz[j][i];
                matriz[j][i] = x;
            }
        }
        return matriz;
    }

    static int[][] parte_matriz(int[][] A,int inicio){
        int[][] M = new int[N/2][N];
        for (int i = 0; i < N/2; i++)
            for (int j = 0; j < N; j++)
                M[i][j] = A[i + inicio][j];
        return M;
    }

    static void acomoda_matriz(int[][] C,int[][] A,int renglon,int columna){
        for (int i = 0; i < N/2; i++)
            for (int j = 0; j < N/2; j++)
```

```
            C[i + renglon][j + columna] = A[i][j];
}

static void imprimirMatriz(int[][] matriz){
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            System.out.print(matriz[i][j] + "\t");
        }
        System.out.println("");
    }
}

static void imprimirResultados(){
    System.out.println("Matriz A");
    imprimirMatriz(A);
    System.out.println("Matriz B transpuesta");
    imprimirMatriz(B);
    System.out.println("Matriz C");
    imprimirMatriz(C);
}

static String[] crearUrls(String args[]){
    String[] urls = new String[3];

    //Cuando se desea utilizar en localhost y tener multiples servidores
    //se agrega un identificador al nombre que en este caso es un entero
    //que representa el número del nodo.
    if(args.length == 0){
        for(int i=0; i<3; i++){
            urls[i] = "rmi://localhost/multiplicaMatriz" + (i+1);
        }

        return urls;
    }

    try {
        for(int i=0; i<3; i++){
            //Cuando se utiliza en maquinas virtuales se utiliza la ip que corresponde
            //a dicha maquina virtual
            urls[i] = "rmi://" + args[i] + "/multiplicaMatriz";
        }

    } catch (Exception e) {
        //TODO: handle exception
        System.err.println("Uso:");
        System.err.println("java ClienteRMI <ip de server 1>
```

```
                                <ip de server 2> <ip de server 3>");
            System.err.println("java ClienteRMI
                            (para uso en localhost con nodo 1, 2 y 3)");
            System.exit(0);
        }

        return urls;
    }

    public static long calcularChecksum(int[][] matriz) {
        long checksum = 0;
        for (int i = 0; i < N; i++){
            for (int j = 0; j < N; j++){
                checksum += matriz[i][j];
            }
        }

        return checksum;
    }
}
```

Para esta práctica se tienen dos formas de uso:

1. **Para uso en localhost:** en el cual no se pasan parámetros y se generan automáticamente las urls con el siguiente formato.
   - **rmi://localhost/multiplicaMatriz1**.
   - **rmi://localhost/multiplicaMatriz2**.
   - **rmi://localhost/multiplicaMatriz3**.
2. **Para uso en máquina virtual:** en el cual se pasan las direcciones IP de los nodos 1, 2 y 3 por los parámetros, las urls quedan con el siguiente formato.
   - **rmi://direcciónIP nodo 1/multiplicaMatriz**.
   - **rmi://direcciónIP nodo 2/multiplicaMatriz**.
   - **rmi://direcciónIP nodo 3/multiplicaMatriz**.

La creación de las urls se realiza en el método *crearUlrs()* donde verifica la longitud del arreglo de los argumentos y si es 0 genera las urls para el uso en localhost, de lo contrario genera las urls con las direcciones IP de las máquinas virtuales. Dado a que en el nodo 0 se ejecuta el cliente y el servidor comparten la misma dirección, podemos utilizar *localhost* y la url es constante.

En los objetos r0, r1, r2 y r3 se asocian los apuntadores a los objetos remotos de los servidores 0, 1, 2 y 3, respectivamente. Para ello el método *lookup()* de la clase *Naming* busca las urls en los directorios de *rmiregistry* para obtener el apuntador al objeto remoto.

Posteriormente se inicializan las matrices, se transpone la matriz B, se parten las matrices y se llama el método *multiplica_matrices()* de los objetos remotos, se acomoda la matriz y se calcula el checksum, si N es igual a 4 se imprimen las matrices.

Cabe mencionar que RMI se encarga de controlar la comunicación entre las clases y facilita la implementación de la programa comparada que el envío por mensajes de la práctica 3.

Para el caso de N=500 se desarrolla otro cleinte donde solo se cambia el valor N. Dicho programa se muestra a continuación:

```
import java.io.IOException;
import java.rmi.Naming;
import java.lang.Exception;

public class ClienteRMI_N500{
    static int N = 500;//Tamanio de matrices
    static int[][] A = new int[N][N];//Declaracion dematriz de NxN
    static int[][] B = new int[N][N];//Declaracion dematriz de NxN
    static int[][] C = new int[N][N];//Declaracion dematriz de NxN
    static long checksum = 0;

    public static void main(String args[]) throws Exception{
        String[] urls = crearUrls(args);

        System.out.println("Urls:");
        for(String url : urls){
            System.out.println(url);
        }

        //Obtiene una referencia que "apunta" al objeto remoto asociado a la URL
        //Dado a que el servidor 0 se ejecuta en el mismo nodo que el cliene
        //su url se mantiene constante en localhost

        InterfaceRMI r0 = (InterfaceRMI)Naming.lookup("rmi://localhost/multiplicaMatriz0");
        InterfaceRMI r1 = (InterfaceRMI)Naming.lookup(urls[0]);
        InterfaceRMI r2 = (InterfaceRMI)Naming.lookup(urls[1]);
        InterfaceRMI r3 = (InterfaceRMI)Naming.lookup(urls[2]);


        //Inicializamos las matrices
        inicializaMatrices();


        //Trasnponemos la matriz B
        B = transponerMatriz(B);
```

```
    //Obtenemos las matrices A1, A2, B1 y B2
    int[][] A1 = parte_matriz(A,0);
    int[][] A2 = parte_matriz(A,N/2);
    int[][] B1 = parte_matriz(B,0);
    int[][] B2 = parte_matriz(B,N/2);

    //Instanciamos a un objeto ClaseRMI para que el cliente ejecute
    //el metodo multiplica_matrices de A1 x B1
    ClaseRMI objCliente = new ClaseRMI();

    //Objetenemos la multiplicación de matrices
    int[][] C1 = objCliente.multiplica_matrices(A1,B1);
    int[][] C2 = r1.multiplica_matrices(A1,B2);
    int[][] C3 = r2.multiplica_matrices(A2,B1);
    int[][] C4 = r3.multiplica_matrices(A2,B2);

    acomoda_matriz(C,C1,0,0);
    acomoda_matriz(C,C2,0,N/2);
    acomoda_matriz(C,C3,N/2,0);
    acomoda_matriz(C,C4,N/2,N/2);

    if(N == 4){
        imprimirResultados();
    }

    System.out.println("Checksum="+calcularChecksum(C));
}

static void inicializaMatrices(){
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            A[i][j] = 2 * i - j;
            B[i][j] = 2 * i + j;
            C[i][j] = 0;
        }
    }

}

static int[][] transponerMatriz(int[][] matriz){
    for (int i = 0; i < N; i++){
        for (int j = 0; j < i; j++){
            int x = matriz[i][j];
            matriz[i][j] = matriz[j][i];
            matriz[j][i] = x;
        }
```

```
    }
    return matriz;
}

static int[][] parte_matriz(int[][] A,int inicio){
    int[][] M = new int[N/2][N];
    for (int i = 0; i < N/2; i++)
        for (int j = 0; j < N; j++)
            M[i][j] = A[i + inicio][j];
    return M;
}

static void acomoda_matriz(int[][] C,int[][] A,int renglon,int columna){
    for (int i = 0; i < N/2; i++)
        for (int j = 0; j < N/2; j++)
            C[i + renglon][j + columna] = A[i][j];
}

static void imprimirMatriz(int[][] matriz){
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            System.out.print(matriz[i][j] + "\t");
        }
        System.out.println("");
    }
}

static void imprimirResultados(){
    System.out.println("Matriz A");
    imprimirMatriz(A);
    System.out.println("Matriz B transpuesta");
    imprimirMatriz(B);
    System.out.println("Matriz C");
    imprimirMatriz(C);
}

static String[] crearUrls(String args[]){
    String[] urls = new String[3];

    //Cuando se desea utilizar en localhost y tener multiples servidores
    //se agrega un identificador al nombre que en este caso es un entero
    //que representa el número del nodo.
    if(args.length == 0){
        for(int i=0; i<3; i++){
            urls[i] = "rmi://localhost/multiplicaMatriz" + (i+1);
        }
```

```
            return urls;
        }

        try {
            for(int i=0; i<3; i++){
                //Cuando se utiliza en maquinas virtuales se utiliza la ip que corresponde
                //a dicha maquina virtual
                urls[i] = "rmi://" + args[i] + "/multiplicaMatriz";
            }

        } catch (Exception e) {
            //TODO: handle exception
            System.err.println("Uso:");
            System.err.println("java ClienteRMI <ip de server 1>
                                <ip de server 2> <ip de server 3>");
            System.err.println("java ClienteRMI
                        (para uso en localhost con nodo 1, 2 y 3)");
            System.exit(0);
        }

        return urls;
    }

    public static long calcularChecksum(int[][] matriz) {
        long checksum = 0;
        for (int i = 0; i < N; i++){
            for (int j = 0; j < N; j++){
                checksum += matriz[i][j];
            }
        }

        return checksum;
    }
}
```

2. Capturas de pantallas.

2.1. Compilación

Dada a la portabilidad de Java, podemos compilar los archivos en una máquina personal y utilizarlos en las máquinas virtuales. La Figura 1 muestra la compilación de los archivos.

```
D:\Documentos\ESCOM\Distribuidos\Desarrollo-de-sistemas-distribuidos\Tarea6>javac InterfaceRMI.java

D:\Documentos\ESCOM\Distribuidos\Desarrollo-de-sistemas-distribuidos\Tarea6>javac ClaseRMI.java

D:\Documentos\ESCOM\Distribuidos\Desarrollo-de-sistemas-distribuidos\Tarea6>javac ServidorRMI.java

D:\Documentos\ESCOM\Distribuidos\Desarrollo-de-sistemas-distribuidos\Tarea6>javac ClienteRMI_N4.java

D:\Documentos\ESCOM\Distribuidos\Desarrollo-de-sistemas-distribuidos\Tarea6>javac ClienteRMI_N500.java

D:\Documentos\ESCOM\Distribuidos\Desarrollo-de-sistemas-distribuidos\Tarea6>
```

Figura 1: Compilación de los archivos Java.

2.2. Creación maquinas virtuales y grupo de recursos

Se crean 4 máquinas virtuales en un mismo grupo de recursos. En la Figura 2 se muestran las máquinas virtuales y en la Figura 3 el grupo de recursos.

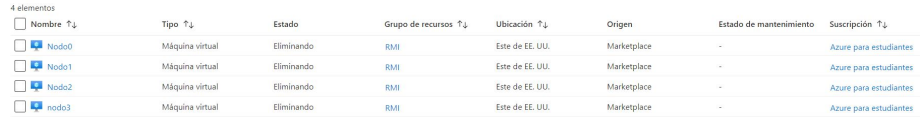

Figura 2: Máquinas virtuales creadas en Azure.

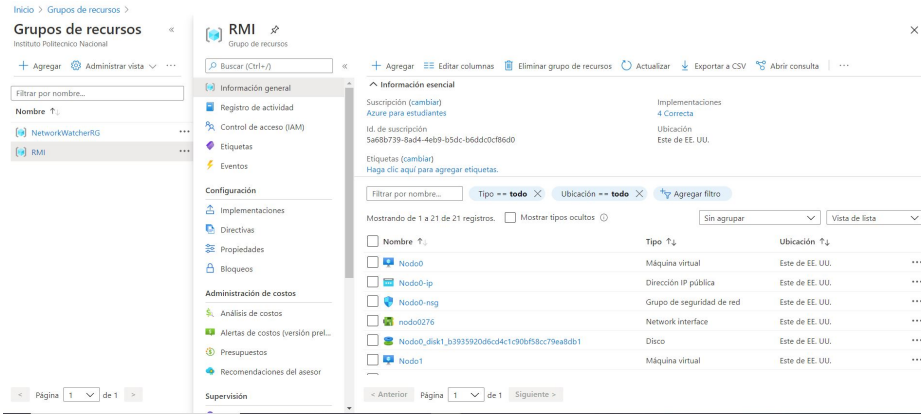

Figura 3: Grupo de recursos en Azure.

2.3. Acceso a máquinas virtuales

Se accesa con ayuda de **Putty** a las maquinas virtuales. La Figuras 4, 5, 6 y 7 muestran el acceso a cada maquina virtual.

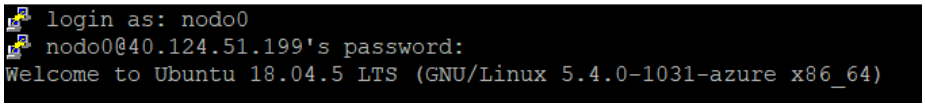

Figura 4: Acceso a la maquina virtual 0 (nodo 0).

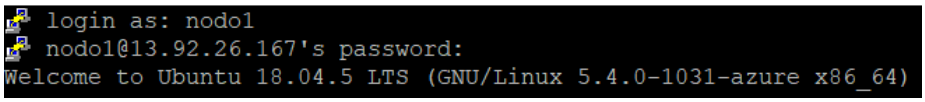

Figura 5: Acceso a la maquina virtual 1 (nodo 1).

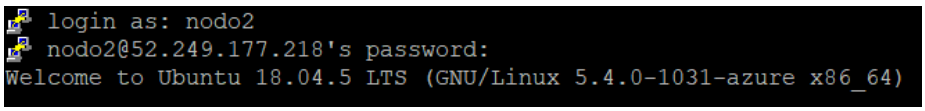

Figura 6: Acceso a la maquina virtual 2 (nodo 2).

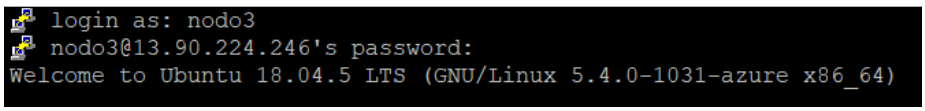

Figura 7: Acceso a la maquina virtual 3 (nodo 3).

2.4. Preparación de las máquinas virtuales

Una vez teniendo las 4 maquinas virtuales creadas se procede a instalar el JDK, para ello en cada una se hace una actualización, Figura 8, y después se instala como se muestra en la Figura 9. Este proceso se realiza en todas las maquinas virtuales.

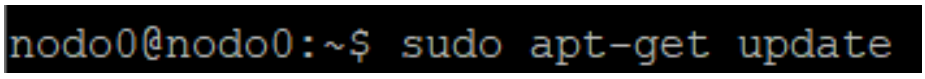

Figura 8: Actualización.

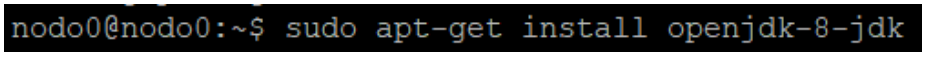

Figura 9: Instalación del JDK.

Con ayuda de github se clona el repositorio, se utiliza el comando git clone para poder obtener el programa directamente de mi repositorio personal. En la Figura 10 se aprecia la clonación en la máquina virtual del nodo2, este proceso se hizo en todas las máquinas virtuales.

```
git clone https://github.com/artLara/Desarrollo-de-sistemas-distribuidos.git
```

Figura 10: Obtención del programa clonando el repositorio github.

2.5. Ejecución de rmiregistry y servidores

Ahora se debe abrir otra terminal en Putty para los nodos 0, 1, 2 y 3, así tendrémos 2 terminales una para ejecutar *rmiregistry* y otra para ejecutar el *servidor*. En la Figura 11, Figura 12 y Figura 13 muestran las ejecuciones de *rmiregistry* y *servidores* en sus nodos correspondiente. En este caso se pasa como parámetro su dirección IP para registrar el objeto remoto.

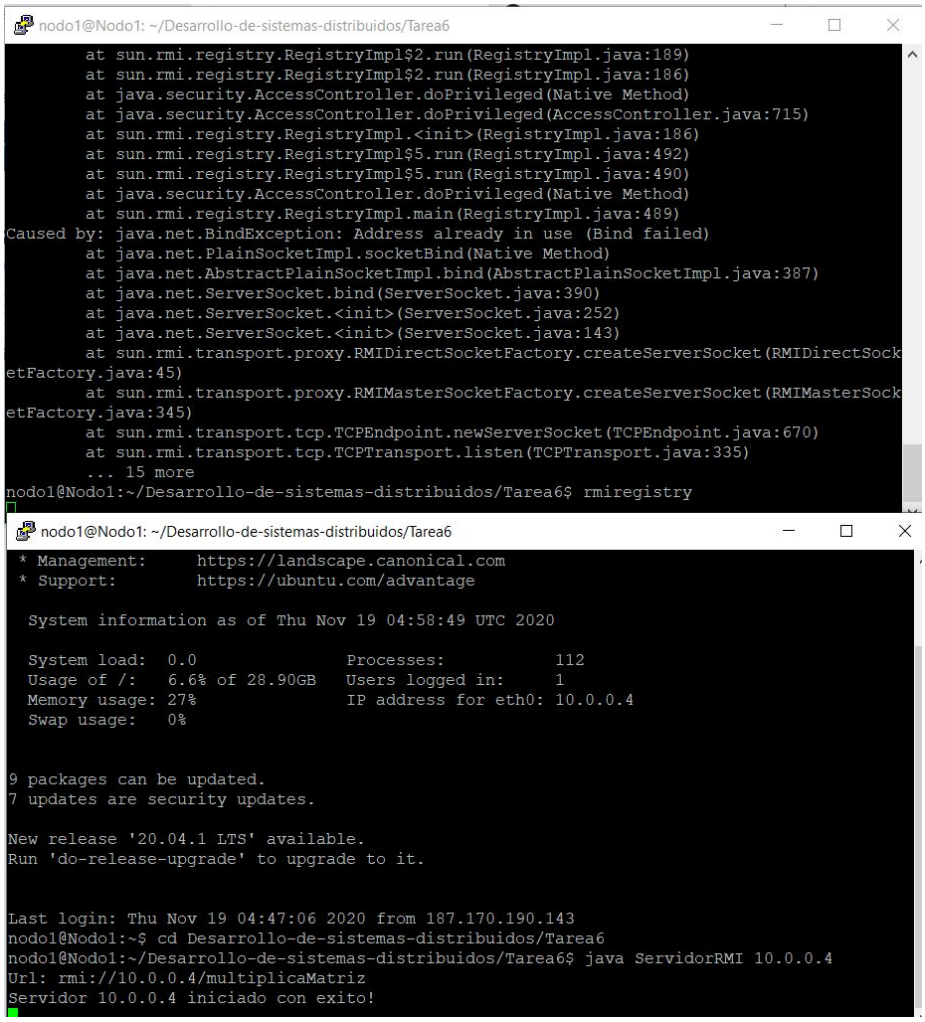

Figura 11: Ejecución de *rmiregistry* y *servidor 1* en el nodo 1.

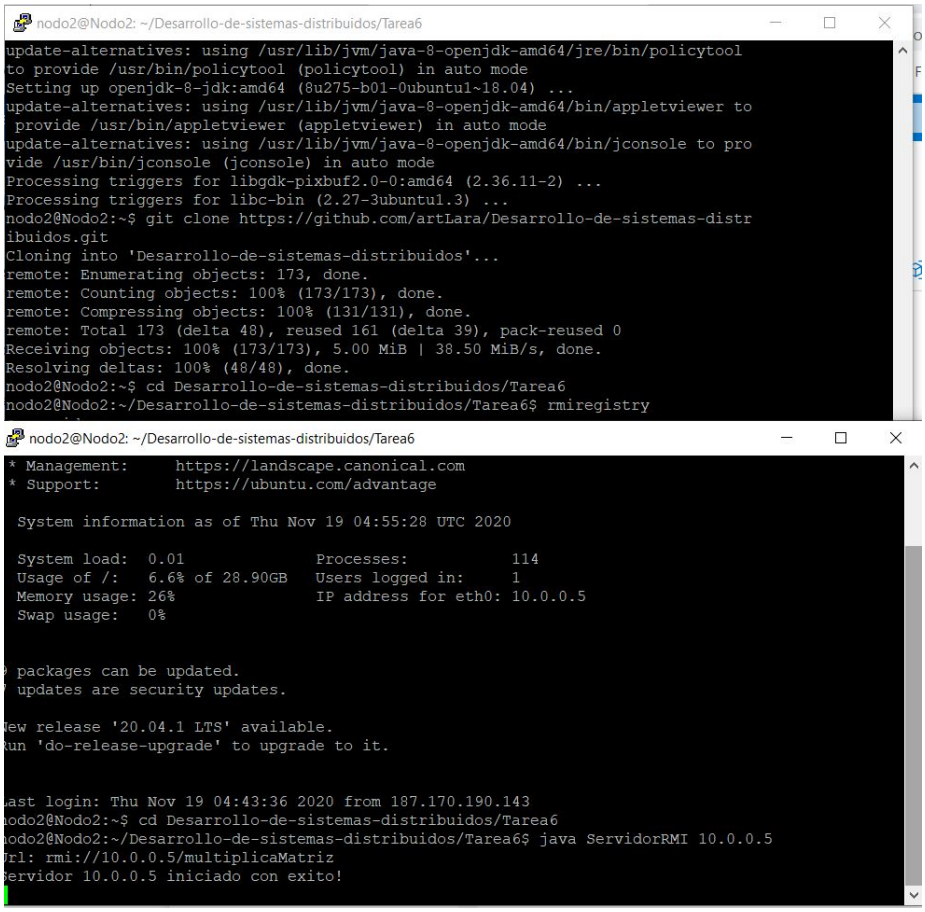

Figura 12: Ejecución de *rmiregistry* y *servidor 2* en el nodo 2.

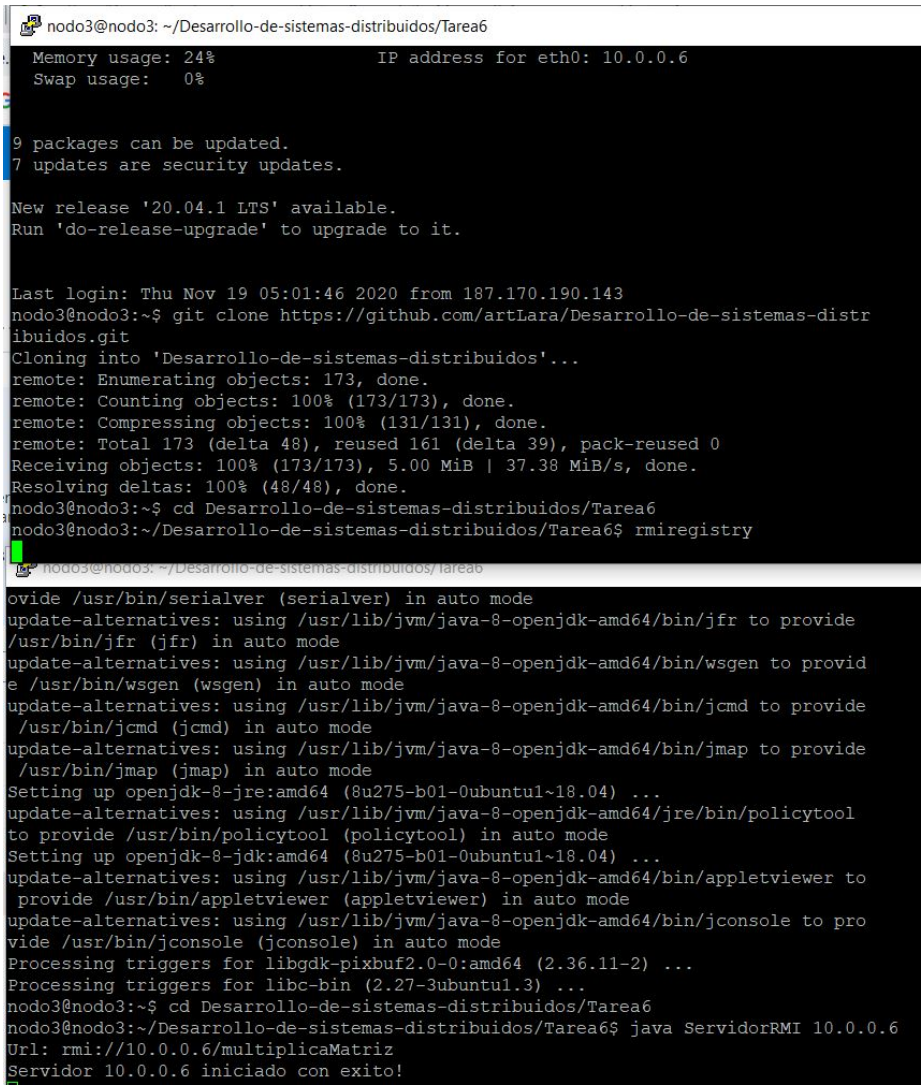

Figura 13: Ejecución de *rmiregistry* y *servidor 3* en el nodo 3.

En la Figura 14 se muestran las terminales de la máquina virtual 1, 2 y 3 ejecutando rmiregistry al mismo tiempo.

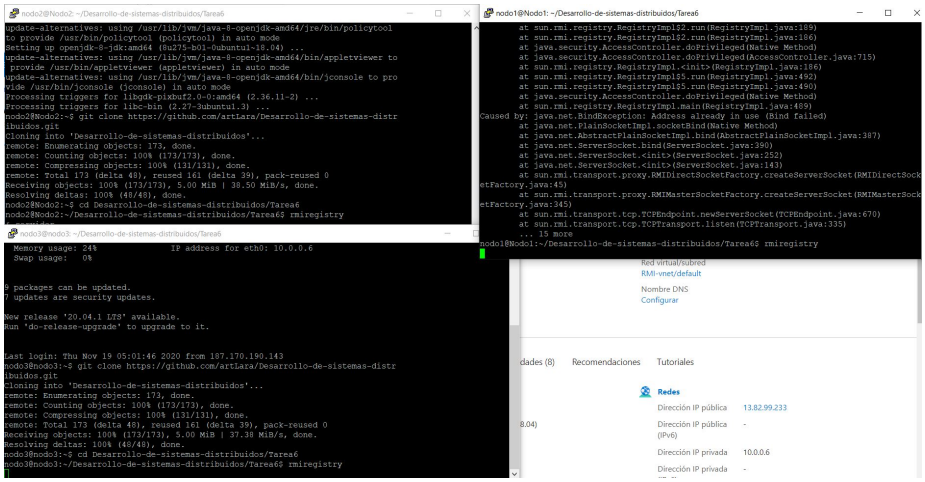

Figura 14: Ejecución de *rmiregistry* en el nodo 1, 2 y 3.

Para el nodo 0 se deben abrir dos teminales extras, así tenemos una terminal para ejecutar *rmiregistry*, otra para el *Servidor 0* y la tercera para el *Cliente*. La Figura 15 muestra la ejecución de *rmiregistry* y *Servidor 0*.

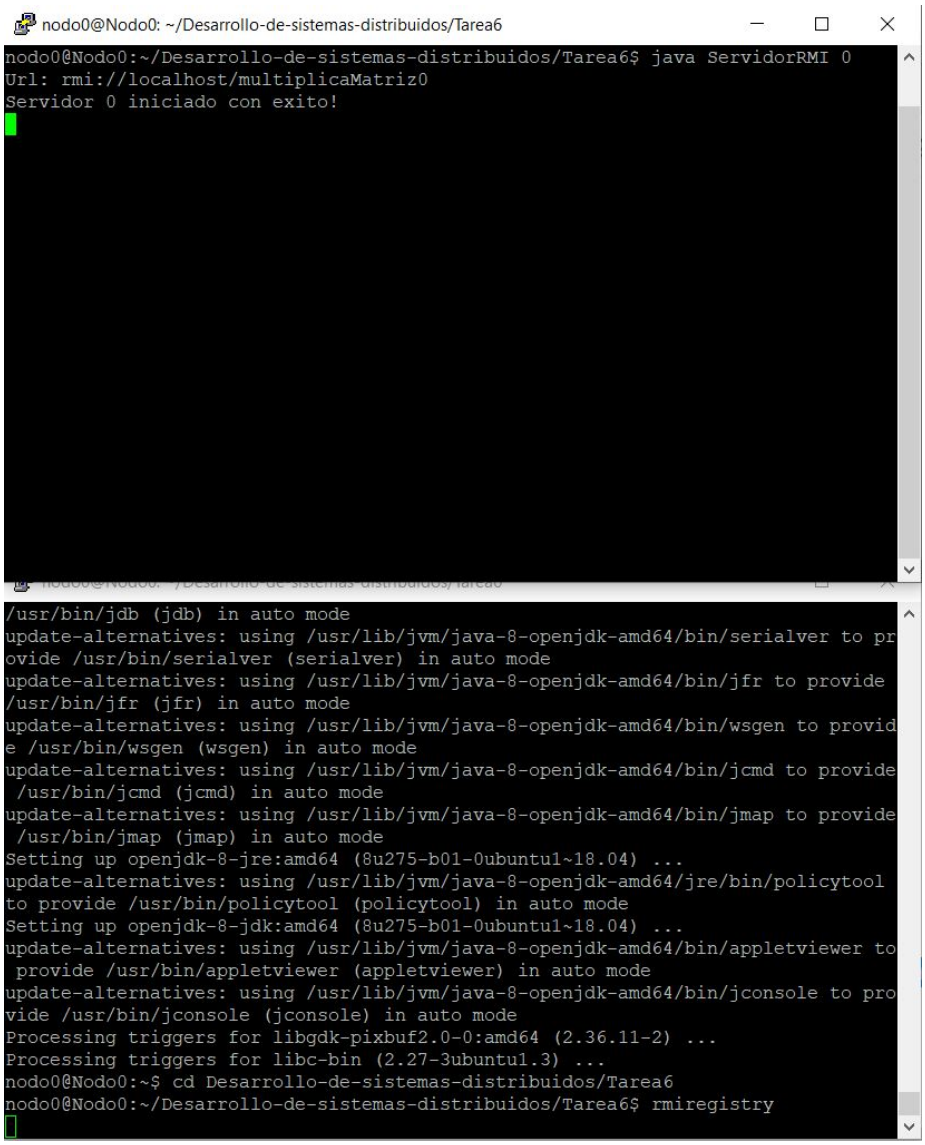

Figura 15: Ejecución de *rmiregistry* y *servidor 0* en el nodo 0.

Hay que notar que a este servidor se le pasa el número de nodo como parámetro y así se registre el objeto con la url **rmi://localhost/multiplicaMatriz0**. Hay que notar que se en este servidor se puede utilizar *localhost* dado a que es la misma máquina virtual.

2.6. Ejecución de cliente

Ya tenemos la terminal para ejecutar el cliente, ahora corresponde correrlo pasando las direcciones IP del servidor 1, 2 y 3 en ese orden.

2.6.1. N=4

Para el caso de N igual a 4 corremos el programa *ClienteRMI_N4.java* y pasamos las direcciones IP, en este caso los nodos 1, 2 y 3 tienen las direcciones IP 10.0.0.4, 10.0.0.5 y 10.0.0.6, respectivamente. La ejecución se aprecia en la Figura 16

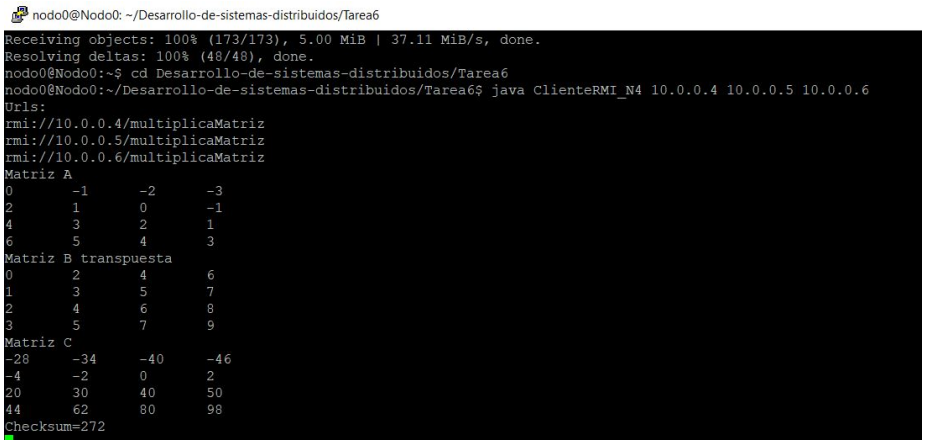

Figura 16: Ejecución del cliente con N=4.

En la Figura 17 se muestran las terminales de los nodos 1, 2 y 3 corriendo los servidores correspondientes y a la terminal del cliente con N=4.

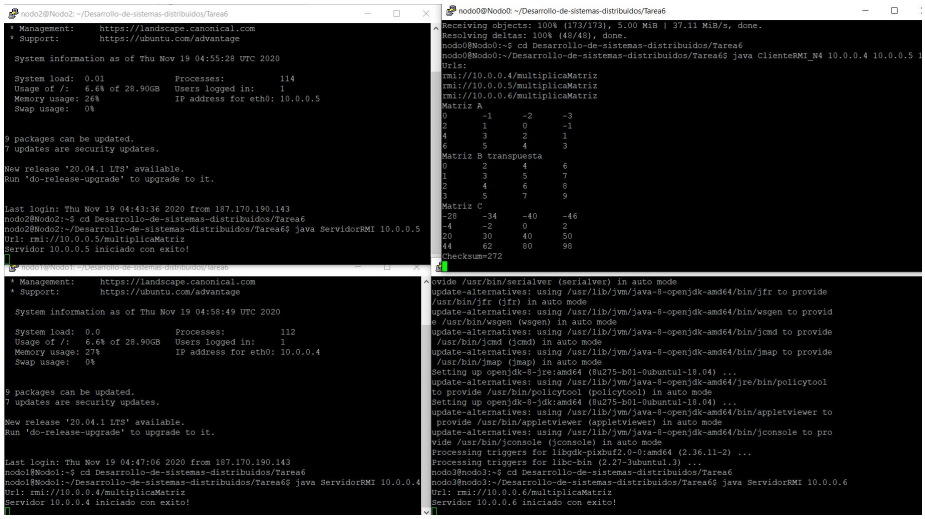

Figura 17: Ejecución del cliente con N=4 y servidores 1, 2 y 3.

2.6.2. N=500

Para el caso de N igual a 500 se sigue el mismo proceso, corremos el programa *ClienteRMI_N500.java* y pasamos las direcciones IP, en este caso los nodos 1, 2 y 3 tienen las direcciones IP 10.0.0.4, 10.0.0.5 y 10.0.0.6, respectivamente. La ejecución se aprecia en la Figura 18

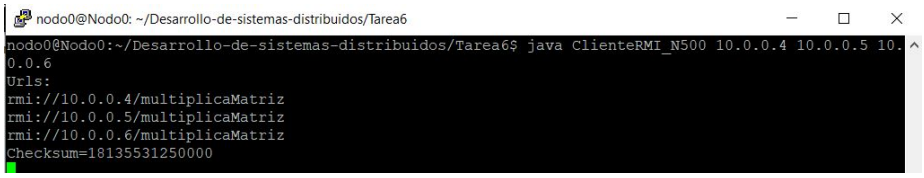

Figura 18: Ejecución del cliente con N=500.

En la Figura 19 se muestran las terminales de los nodos 1, 2 y 3 corriendo los servidores correspondientes y a la terminal del cliente con N=500.

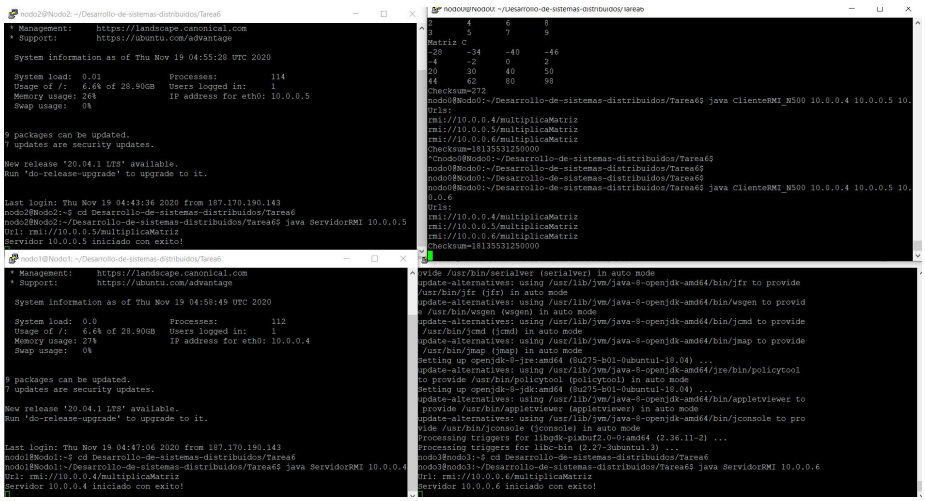

Figura 19: Ejecución del cliente con N=500 y servidores 1, 2 y 3.

3. Conclusiones

El uso de objetos remotos ofrece diversas ventajas como poder compartir la carga de la ejecución y de transferencia de datos, que a comparación a un sistema local es más estable dado a que en este recae toda la responsabilidad de ejecución.

El uso de RMI facilita mucho la implementación de la práctica, a comparación del envió por mensajes de la práctica 3, por ello el código es más corto y limpio. Por seguridad rmiregistry se debe ejecutar en la misma máquina virtual en la que se aloja el servidor.

En el uso de Azure se debe configurar un grupo de trabajo para que las máquinas virtuales habiten en la misma red virtual, en este caso se asignó la red 10.0.0.0, así estas se pueden comunicar sin necesidad de configurar firewall.