

Planteamiento del problema

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

Se deberá ejecutar dos casos:

- $N = 8$, se deberá desplegar las matrices A, B y C y el checksum será de tipo float.
- $N=4000$, se deberá desplegar el checksum de la matriz C.

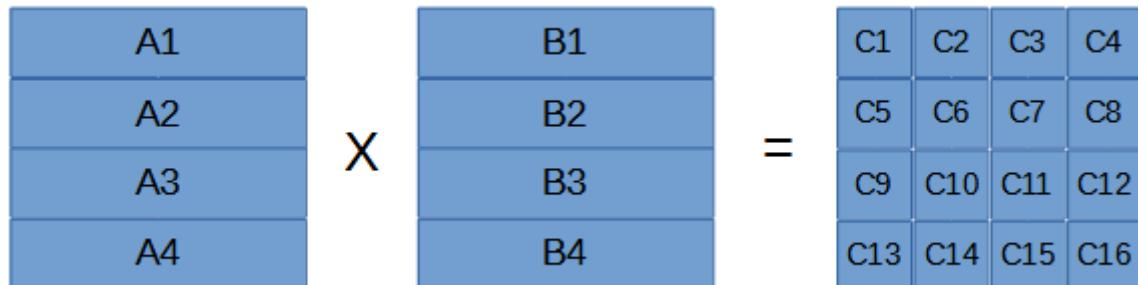
Los elementos de las matrices A, B y C serán de tipo float y el checksum será de tipo float.

Se deberá inicializar las matrices A y B de la siguiente manera:

$$A[i][j] = i + 2 * j$$

$$B[i][j] = 3 * i - j$$

Se deberá dividir las matrices A y B en cuatro partes, tanto la matriz C estará dividida en 16 partes:

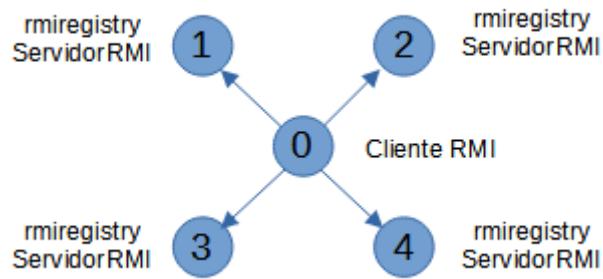


El cliente RMI ejecutará en una máquina virtual con Ubuntu en Azure (nodo 0).

El servidor RMI ejecutará en cuatro máquinas virtuales (nodo 1, nodo 2, nodo 3 y nodo 4) con Ubuntu en Azure.

El programa rmiregistry ejecutará en cada nodo dónde ejecute el servidor RMI.

El nodo 1 calculará los productos C1, C2, C3 y C4, el nodo 2 calculará los productos C5, C6, C7 y C8, el nodo 3 calculará los productos C9, C10, C11 y C12, y el nodo 4 calculará los productos C13, C14, C15 y C16.

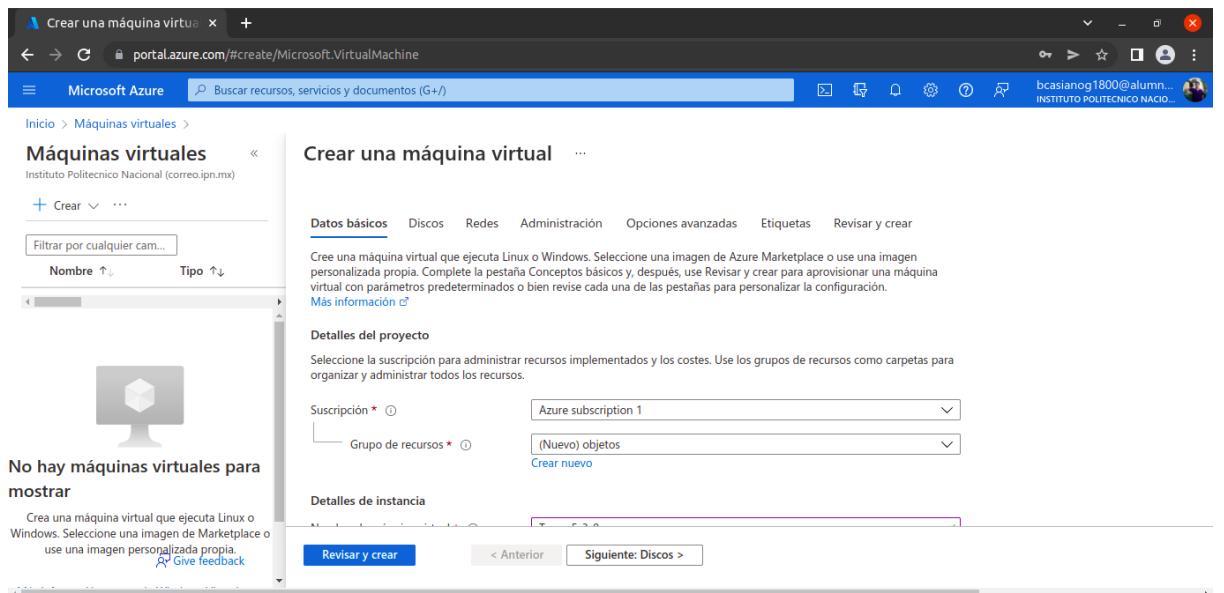


El cliente RMI inicializará las matrices A y B, realizará la transpuesta de la matriz B, obtendrá las matrices A1, A2, A3, A4, B1, B2, B3 y B4, invocará el método remoto multiplica_matrices(), obtendrá la matriz C a partir de las matrices C1 a la 16, y calculará el checksum de la matriz. En el caso N=8 el cliente RMI deberá desplegar las matrices A, B y C (la matriz B antes de transponer).

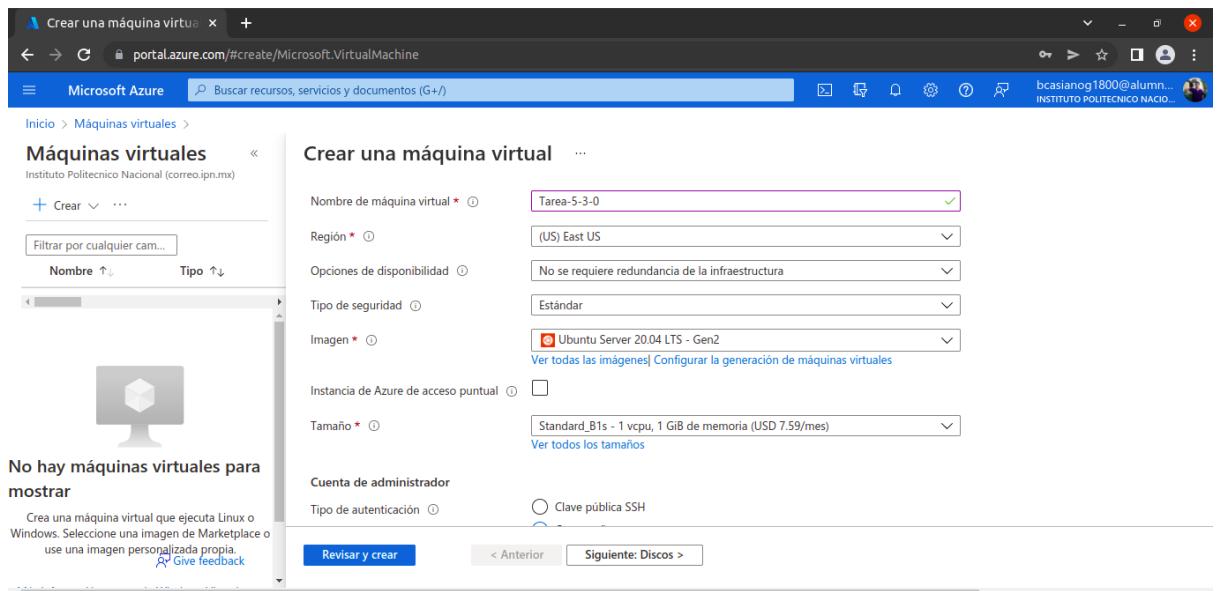
Se deberá utilizar threads en el cliente RMI para invocar el método remoto multiplica_matrices() de manera que los servidores RMI calculen los productos en paralelo.

Desarrollo de la tarea

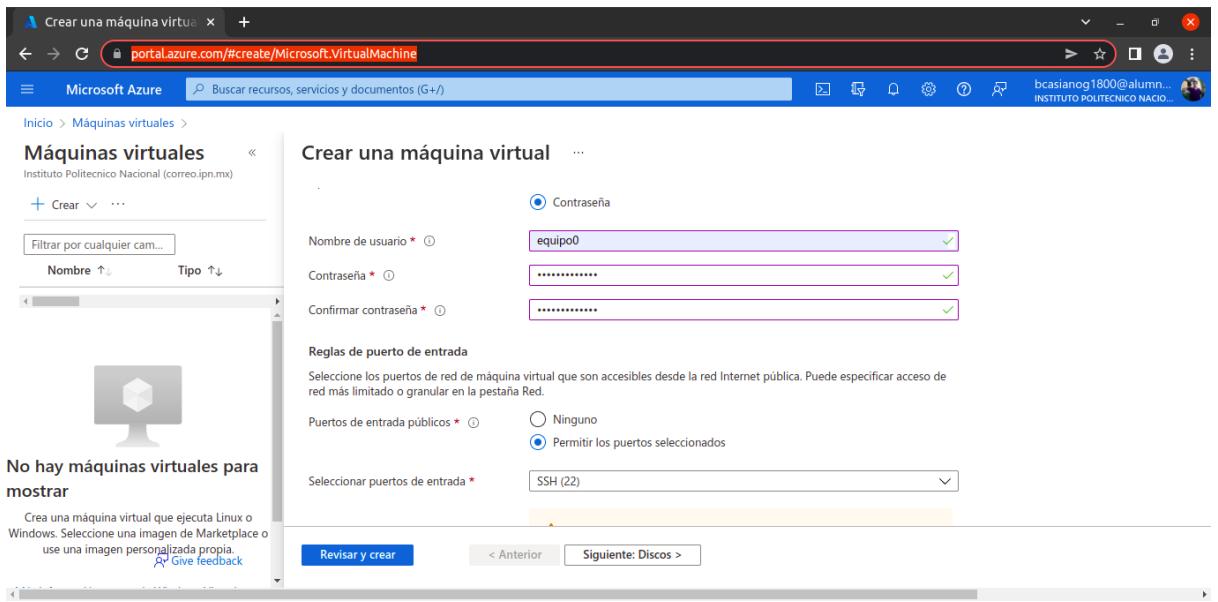
- Creación de la máquina virtual del nodo 0 con Ubuntu en Azure



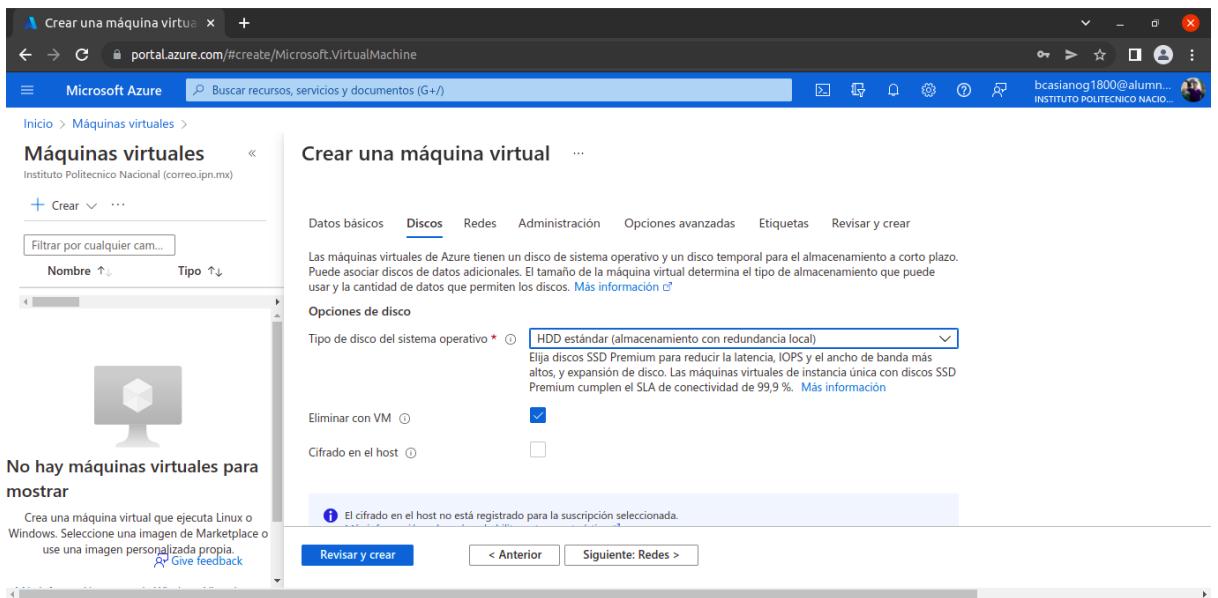
- En esta imagen ya escogimos el recurso a crear que en éste caso fue una máquina virtual.



- Se coloca el nombre de la máquina virtual en este caso es “Tarea-5-3-0”
- la dejamos en Estados Unidos y escogemos Ubuntu Server.
- Nota “Ya está seleccionado 1 Gb de Ram como se especifica en el momento de entrega”.



- En esta imagen colocamos el nombre del usuario y la contraseña.



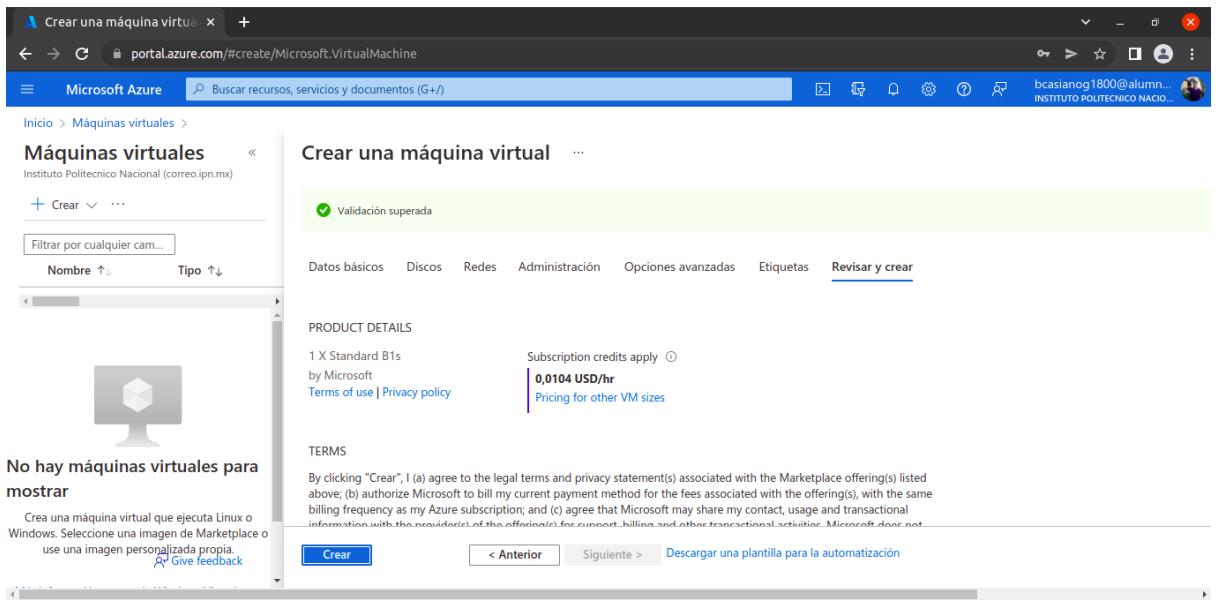
- Aquí colocamos el tipo de disco que se utilizará para el sistema operativo de la máquina virtual.

The screenshot shows the Microsoft Azure portal interface for creating a new virtual machine. The main title is 'Crear una máquina virtual'. The 'Redes' (Network) tab is selected. The configuration pane describes setting up the network interface (NIC) to define connectivity. It includes fields for 'Red virtual' (new item 'objetos-vnet'), 'Subred' (new item 'default (10.0.0.0/24)'), and 'IP pública' (new item 'Tarea-5-3-0-ip'). A note at the bottom says 'Al crear una máquina virtual, se crea una interfaz de red automáticamente.' (When creating a virtual machine, a network interface is automatically created). At the bottom right are buttons for 'Revisar y crear' (Review + Create), '< Anterior' (Previous), and 'Siguiente: Administración >' (Next: Administration).

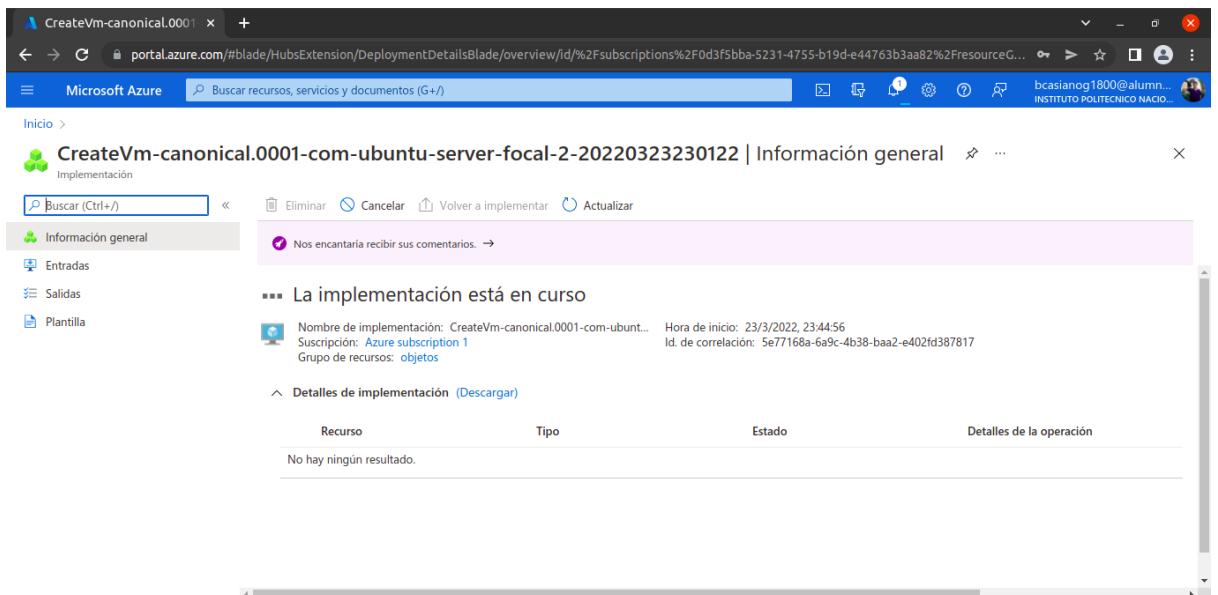
- Se configura la interfaz de red.

The screenshot shows the 'Administración' (Administration) step of the 'Create a virtual machine' wizard. The title is 'Crear una máquina virtual'. The 'Administración' tab is selected. It includes a note about Azure Security Center providing unified security and protection across hybrid workloads. Under 'Supervisión' (Monitoring), there is a section for 'Diagnósticos de arranque' (Boot Diagnostics) with three options: 'Habilitar con la cuenta de almacenamiento administrada (recomendado)' (Enable with managed storage account (recommended)), 'Habilitar con la cuenta de almacenamiento personalizada' (Enable with custom storage account), and 'Deshabilitar' (Disable). The 'Deshabilitar' option is selected. At the bottom right are buttons for 'Revisar y crear' (Review + Create), '< Anterior' (Previous), and 'Siguiente: Opciones avanzadas >' (Next: Advanced Options).

- Deshabilitamos el diagnóstico de arranque, damos clic en “Crear”.



- Imagen donde se puede ver la creación de la máquina virtual en Azure.



- Podemos ver que la máquina se creó correctamente y está corriendo.

Tarea-5-3-0 - Microsoft Azure

portal.azure.com/#@correo.ipn.mx/resource/subscriptions/0d3f5bba-5231-4755-b19d-e44763b3aa82/resourcegroups/objetos/providers/Microsoft.Compute/virtualmachines/Tarea-5-3-0

Tarea-5-3-0 | Máquina virtual

Buscar (Ctrl+ /)

Conectar Iniciar Reiniciar Detener Captura Eliminar Actualizar Abrir en dispositivos móviles CLI / PS Comentarios

⚠️ Tarea-5-3-0 estado del agente de máquina virtual no está listo. Solucionar el problema →

Información general Registro de actividad Control de acceso (IAM) Etiquetas Diagnosticar y solucionar problemas

Configuración Redes Conectar Discos Tamaño Seguridad Recomendaciones de Advisor Extensiones + aplicaciones

Propiedades Supervisión Funcionalidades (7) Recomendaciones Tutoriales

Máquina virtual Nombre del equipo: Tarea-5-3-0 Estado de mantenimiento: - Sistema operativo: Linux

Redes Dirección IP pública: 20.120.19.35 Dirección IP pública (IPv6): - Dirección IP privada: 10.0.0.4

Agregar regla de seguridad

portal.azure.com/#@correo.ipn.mx/resource/subscriptions/0d3f5bba-5231-4755-b19d-e44763b3aa82/resourcegroups/objetos/providers/Microsoft.Compute/virtualmachines/Tarea-5-3-0/networkInterfaces/tarea-5-3-0-0682

Tarea-5-3-0 | Redes

Buscar (Ctrl+ /)

Adjuntar interfaz de red Desasociar interfaz de red Comentarios

tarea-5-3-0-0682 Configuración de IP: ipconfig1 (Principal)

Interfaz de red: tarea-5-3-0-0682 Reglas de seguridad vigentes Sol Red virtual/subred: objetos-vnet/default IP pública de NIC: 20.120.19.35

Reglas de puerto de entrada Reglas de puerto de salida Grupos de seguridad

Grupo de seguridad de red Tarea-5-3-0-nsg (se conectó a la interfaz de red) Impactos 0 subredes, 1 interfaces de red

Prioridad	Nombre	Puerto	Protocolo
300	SSH	22	TCP
65000	AllowVnetInbound	Cualquiera	Cualquier
65001	AllowAzureLoadBalanc...	Cualquiera	Cualquier
65500	DenyAllInbound	Cualquiera	Cualquier

Agregar regla de seguridad de entrada

Origen: Any Intervalos de puertos de origen: *

Destino: Any Intervalos de puertos de destino: 1099

Servicio: Custom Protocolo: TCP

Agregar Cancelar

The screenshot displays two side-by-side views of the Microsoft Azure portal, specifically for a virtual machine named "Tarea-5-3-0".

Left Window (Screenshot 1):

- URL:** portal.azure.com/#@correo.ipn.mx/resource/subscriptions/0d3f5bba-5231-4755-b19d-e44763b3aa82/resourcegroups/objetos/providers/Microsoft.Compute/virtualMachines/tarea-5-3-0
- Section:** Tarea-5-3-0 | Redes
- Sub-section:** Redes
- Task Bar:** Buscar (Ctrl+ /), Adjuntar interfaz de red, Desasociar interfaz de red, Comentarios
- Content:**
 - Configuración de IP:** ipconfig1 (Principal)
 - Reglas de puerto de entrada:**
 - Tarea-5-3-0-0682:** Grupo de seguridad de red Tarea-5-3-0-nsg (se conectó a la interfaz de red: ipconfig1). Impactos 0 subredes, 1 interfaces de red.
 - Reglas de seguridad vigentes:** Red virtual/subred: objetos-vnet/default, IP pública de NIC: 20.120.19.35
 - Reglas de puerto de entrada:** Prioridad 300 (SSH), Puerto 22, Protocolo TCP.
 - Reglas de puerto de salida:** Prioridad 65000 (AllowVnetInbound), Puerto 0, Protocolo TCP.
 - Grupos de seguridad de aplicación:** Prioridad 65001 (AllowAzureLoadBalancedOutbound), Puerto 0, Protocolo TCP.
 - Equilibrio de carga:** Prioridad 65500 (DenyAllInbound), Puerto 0, Protocolo TCP.

Right Window (Screenshot 2):

- URL:** portal.azure.com/#@correo.ipn.mx/resource/subscriptions/0d3f5bba-5231-4755-b19d-e44763b3aa82/resourcegroups/objetos/providers/Microsoft.Compute/virtualMachines/tarea-5-3-0
- Section:** Tarea-5-3-0 | Redes
- Sub-section:** Redes
- Task Bar:** Buscar (Ctrl+ /), Adjuntar interfaz de red, Desasociar interfaz de red, Comentarios
- Content:**
 - Configuración de IP:** ipconfig1 (Principal)
 - Reglas de puerto de entrada:**
 - Tarea-5-3-0-0682:** Grupo de seguridad de red Tarea-5-3-0-nsg (se conectó a la interfaz de red: tarea-5-3-0-0682). Impactos 0 subredes, 1 interfaces de red.
 - Reglas de seguridad vigentes:** Red virtual/subred: objetos-vnet/default, IP pública de NIC: 20.120.19.35, IP privada de NIC: 10.0.0.4, Redes aceleradas: Deshabilitado
 - Reglas de puerto de entrada:** Prioridad 300 (SSH), Puerto 22, Protocolo TCP, Origen Cualquiera, Destino Cualquiera, Acción Permitir.
 - Reglas de puerto de salida:** Prioridad 65000 (AllowVnetInbound), Puerto 0, Protocolo TCP, Origen Cualquiera, Destino VirtualNetwork, Acción Permitir.
 - Grupos de seguridad de aplicación:** Prioridad 65001 (AllowAzureLoadBalancedOutbound), Puerto 0, Protocolo TCP, Origen AzureLoadBalancer, Destino Cualquiera, Acción Permitir.
 - Equilibrio de carga:** Prioridad 65500 (DenyAllInbound), Puerto 0, Protocolo TCP, Origen Cualquiera, Destino Cualquiera, Acción Denegar.

Explicacion del codigo

A continuación procederemos a explicar cada uno de las clases utilizados para la multiplicación de las matrices, como lo son la clase clienteRMI, servidorRMI, InterfaceRMI y ClaseRMI. Además de explicar los métodos de los que hace uso cada clase para cumplir con sus respectivas funciones.

Clase ClienteRMI.java

Variables globales

Las variables globales a,b y c se tratan de las matrices principales, siendo a y b las matrices a multiplicar y c la matriz resultante de esta multiplicación. De acuerdo a las propiedades de las matrices, si a y b son matrices de dimensiones $N \times N$, que es la matriz resultante tendrá las mismas dimensiones. Como podemos ver estas variables las declaramos como arreglos bidimensionales sin especificar sus dimensiones, ya que este dato lo recibimos como parámetro al momento de iniciar el cliente y que guardamos en la variable "n".

Los arreglos a1,a2,a3,a4,b1,b2,b3 y b4 son las submatrices que resultan de dividir las matrices A y B en 4 partes iguales del lado de las filas. Estas submatrices tienen las mismas dimensiones que las matrices A y B no les especificamos sus dimensiones, ya que depende de las dimensiones de las matrices de origen (A y B). Mientras que los arreglos bidimensionales c1, c2, c3, ,..... , C16 son las matrices resultantes de multiplicar cada submatriz de A por cada submatriz de B, por lo que el tamaño de estas submatrices, depende de las dimensiones de las matrices de origen.

El arreglo de tipo string "ip" lo usamos para leer la dirección ip de cada servidor con el que establecerá conexión el cliente, debido a que en la máquina virtual cada servidores independiente uno de otros. Para obtener dichas ip's la variable leer de la clase Scanner nos servirá para obtener desde teclado la ip de cada servidor.

Por último las variables nodo1,nodo2,nodo3 y nodo4, variables de la clase worker las necesitamos debido a que la clase worker hereda una extensión de la clase Thread nos permite que estas variables nodo sean hilos multithreaded, permitiendo que los servidores puedan realizar las multiplicaciones de las matrices de forma paralela, es decir, que se puedan ejecutar al mismo tiempo sin la necesidad de los nodos esperen a que los demás terminen su ejecución para poder empezar su ejecución.

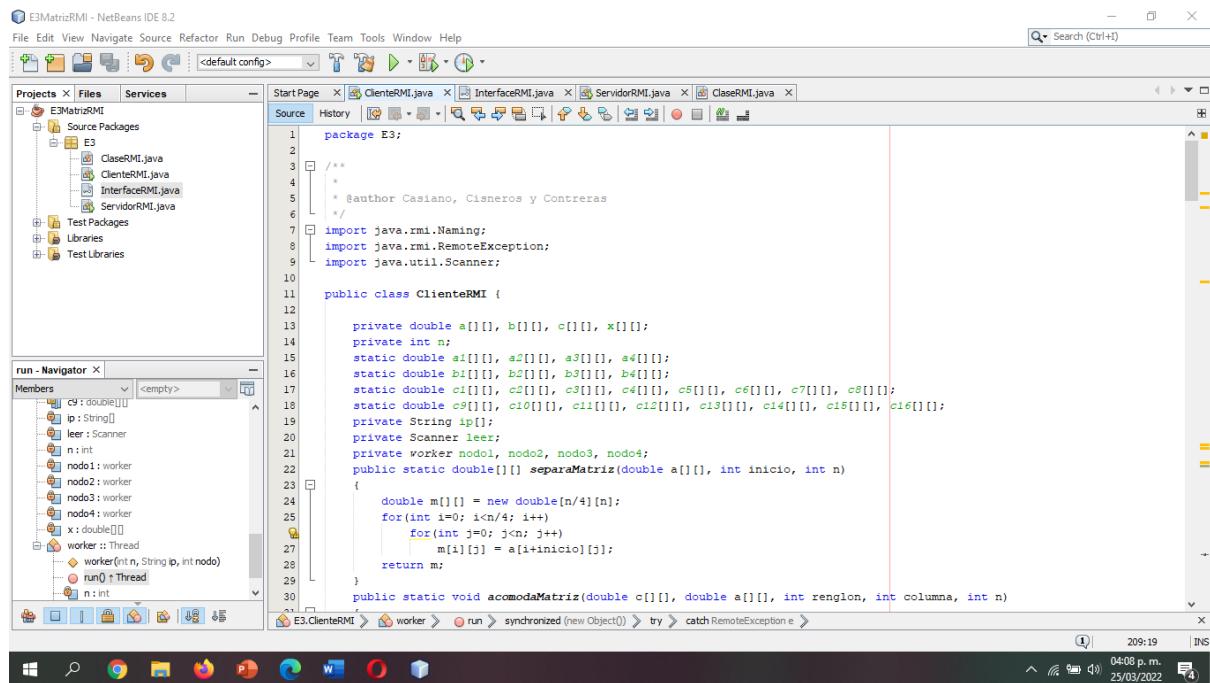


Imagen 1. Captura de pantalla de las variables globales de la clase ClienteRMI.

Método separaMatriz()

El método recibe como tres datos como parámetro, el primero se trata de la tabla a dividir, el segundo una variable de tipo entero que indica la fila por la que se iniciara la división, y el último parámetro se trata del tamaño de las dimensiones de la matriz. Al ser la matriz A y B de dimensiones ($N \times N$) solo pedimos una variable para saber el tamaño de estas.

Usamos un arreglo bidimensional auxiliar para obtener una de las cuartas partes ya sea de la matriz A o By es por eso que podemos ver que en el tamaño de las filas de este arreglo se trata del tamaño (n) dividido en 4, mientras que el número de columnas no se ve afectado.

Para asignar los valores que corresponden a nuestra matriz auxiliar empleamos un ciclo anidado para recorrer cada columna y fila de la matriz mientras le asignamos el valor correspondiente a la matriz original (sea A o B) de acuerdo a la posición de la columna y en el caso de las filas se trata de la posición inicial que pasamos como parámetro mas el número de vuelta del ciclo for que recorre las filas. Además con el fin de evitar un desbordamiento por parte de las filas ya que nuestra matriz auxiliar solo tiene una cuarta parte de filas que la original el ciclo que maneja las filas se asegura que el número de vueltas que se de sea equivalente al número de filas de la matriz original entre 4, haciendo que se empiece desde cero hasta $(N/4)-1$ por ejemplo si la matriz A es de dimensiones 8*8 el ciclo para las filas iría de 0 hasta 3. Por último una vez asignados los valores a

nuestra matriz auxiliar, procedemos a mandar como valor de retorno este arreglo bidimensional de tipo double.

```

1 package E3;
2
3 /**
4 * 
5 * @author Casiano, Cisneros y Contreras
6 */
7
8 import java.rmi.Naming;
9 import java.rmi.RemoteException;
10 import java.util.Scanner;
11
12 public class ClienteRMI {
13
14     private double a[][], b[][], c[][], x[][];
15     private int n;
16     static double a1[][], a2[][], a3[][], a4[][];
17     static double b1[][], b2[][], b3[][], b4[][];
18     static double c1[][], c2[][], c3[][], c4[][], c5[][], c6[][], c7[][], c8[][];
19     static double c9[][], c10[][], c11[][], c12[][], c13[][], c14[][], c15[][], c16[][];
20     private String ip[];
21     private Scanner leer;
22     private worker nodo1, nodo2, nodo3, nodo4;
23     public static double[][] separaMatriz(double a[][], int inicio, int n)
24     {
25         double m[][] = new double[n/4][n];
26         for(int i=0; i<n/4; i++)
27             for(int j=0; j<n; j++)
28                 m[i][j] = a[i+inicio][j];
29         return m;
30     }
31     public static void acomodaMatriz(double c[][], double a[][], int renglon, int columna, int n)
32     {
33         ...
34     }
35 }

```

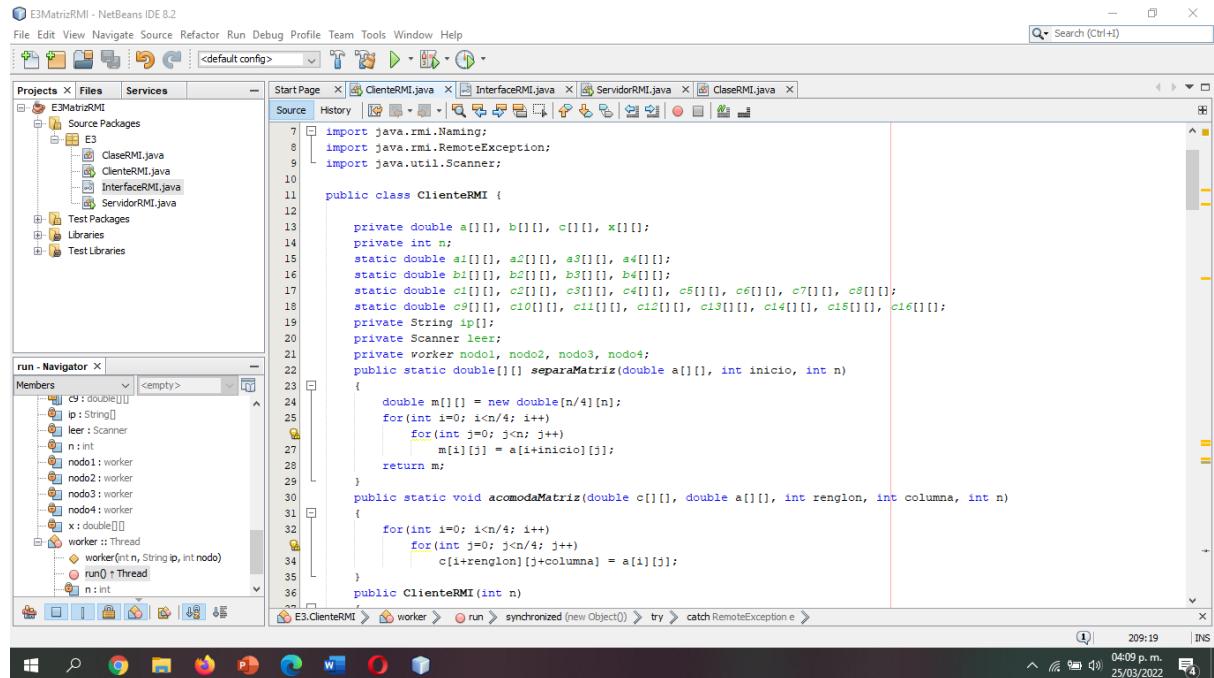
Imagen 2. Captura del método separaMatriz.

Método acomodaMatriz()

Este método lo utilizaremos para acomodar los valores de las submatrices c1,c2,c3,.....,c16 en las posiciones que corresponden de la matriz A, otras palabras acomodamos estas submatrices de tal forma de obtener la matriz C como si hubiéramos multiplicado de forma directa A y B. Para lo que necesitamos recibir como parámetro la matriz C a la que le acomodaremos los valores de la segunda matriz que pasamos de parámetro (a), la posición en la que se inicia la asignación de los valores en los renglones y otro para las columnas, y por último la variable n que representa el tamaño de la matriz que vamos a asignarle los valores, para que no haya un desbordamiento al acceder a los valores de la submatriz (a).

Teniendo en cuenta que las submatrices son de dimensiones ($[N/4]*N$) la matriz resultante de multiplicar 2 submatrices tiene dimensiones ($[N/4]*[N/4]$). Debido a haber sacado la transpuesta de B. Por esta razón es necesario que el ciclo anidado que usaremos para acomodar la submatriz en la matriz C, tanto para las filas y columnas ponemos de condición que el ciclo se detenga una vez que haya recorrido lo equivalente a una cuarta parte de las columnas y filas ($i < N/4$). También esto evita tratar consultar un índice que no existe.

Para acomodar los valores en la matriz que pasamos como primer parámetro le asignamos como índice el valor de renglón y columna respectivamente, que pasamos como parámetro más el valor del contador de cada ciclo esto con la finalidad de asignar el valor de la submatriz a la posición de la matriz original que le corresponde.



```

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.util.Scanner;

public class ClienteRMI {

    private double a[][], b[][], c[][][], x[][];

    private int n;
    static double a1[][], a2[][], a3[][], a4[][];
    static double b1[][], b2[][], b3[][], b4[][];
    static double c1[][], c2[][], c3[][], c4[][], c5[][], c6[][], c7[][], c8[][];
    static double c9[][], c10[][], c11[][], c12[][], c13[][], c14[][], c15[][], c16[][];

    private String ip[];
    private Scanner leer;
    private worker nodo1, nodo2, nodo3, nodo4;
    public static double[][] separarMatriz(double a[][], int inicio, int n)
    {
        double m[][] = new double[n/4][n];
        for(int i=0; i<n/4; i++)
            for(int j=0; j<n; j++)
                m[i][j] = a[i+inicio][j];
        return m;
    }
    public static void acomodaMatriz(double c[][], double a[][], int renglon, int columna, int n)
    {
        for(int i=0; i<n/4; i++)
            for(int j=0; j<n/4; j++)
                c[i+renglon][j+columna] = a[i][j];
    }
    public ClienteRMI(int n)
    {
        ip = new String[n];
        leer = new Scanner(System.in);
        nodo1 = new worker();
        nodo2 = new worker();
        nodo3 = new worker();
        nodo4 = new worker();
        x = new double[n][n];
        worker[] workers = new worker[n];
        for(int i=0; i<n; i++)
            workers[i] = new worker(i, ip[i], leer.nextLine(), i);
        run();
    }
}

```

Imagen 3. Captura del método acomodaMatriz.

Clase worker()

Recordemos que la clase worker es una extensión de la clase `Thread`, tanto lo que realmente hacemos con la clase worker es modificar el método `run`, porque al mandar a llamar al método `start` de esta clase se ejecute las instrucciones que hayamos puesto.

Vamos a necesitar una variable de la clase `InterfaceRMI`, variable de tipo entero para obtener el tamaño de las dimensiones de las matrices originales y otra variable de tipo entero para guardar el número de nodo. Además modificamos el método constructor de esta clase para poder recibir como parámetros el tamaño del servidor y el número de nodo. La ip la utilizamos para crear la referencia al objeto remoto que en este caso se trata del método `multiplicaMatrices` (`InterfaceRMI`) que está a cargo de ejecutarse por los servidores.

Modificamos el método `run` de la clase agregando un switch para verificar el número de nodo el cual debe de ser 1,2,3 o 4, esto ya que recordemos que dependiendo del número de nodos son las submatrices que se deben multiplicar así como las submatrices de C que se obtienen por lo que es necesario verificar el número de nodo y a partir de esto

mandar a llamar al método multiplicar matrices un total de 4 veces que recuerden que cada nodo hace 4 multiplicaciones. Solo es necesario especificar la submatriz de A, submatriz de B y la submatriz de C resultante de esta multiplicación. Como podemos ver en cualquiera de los casos establecidos en el switch, usamos como parámetros nuestras variables globales. El switch está dentro de una cláusula try-catch en caso de que ocurra una excepción con nuestro objeto remoto.

```

public static class worker extends Thread
{
    private InterfaceRMI r;
    private int n, nodo;
    public worker(int n, String ip, int nodo) throws Exception
    {
        this.n = n;
        this.nodo = nodo;
        r = (InterfaceRMI) Naming.lookup("rmi://" + ip + "/server");
    }
    public void run()
    {
        synchronized(new Object())
        {
            try {
                switch(nodo)
                {
                    case 1:
                        //CALCULO DE C1
                        c1 = r.multiplicaMatrices(a1, b1, n);
                        //CALCULO DE C2
                        c2 = r.multiplicaMatrices(a1, b2, n);
                        //CALCULO DE C3
                        c3 = r.multiplicaMatrices(a1, b3, n);
                        //CALCULO DE C4
                        c4 = r.multiplicaMatrices(a1, b4, n);
                    break;
                    case 2:
                        //CALCULO DE C5
                        c5 = r.multiplicaMatrices(a2, b1, n);
                    break;
                    case 3:
                        //CALCULO DE C6
                        c6 = r.multiplicaMatrices(a2, b2, n);
                        //CALCULO DE C7
                        c7 = r.multiplicaMatrices(a2, b3, n);
                        //CALCULO DE C8
                        c8 = r.multiplicaMatrices(a2, b4, n);
                    break;
                    case 4:
                        //CALCULO DE C9
                        c9 = r.multiplicaMatrices(a3, b1, n);
                        //CALCULO DE C10
                        c10 = r.multiplicaMatrices(a3, b2, n);
                        //CALCULO DE C11
                        c11 = r.multiplicaMatrices(a3, b3, n);
                        //CALCULO DE C12
                        c12 = r.multiplicaMatrices(a3, b4, n);
                    break;
                    case 5:
                        //CALCULO DE C13
                        c13 = r.multiplicaMatrices(a4, b1, n);
                        //CALCULO DE C14
                        c14 = r.multiplicaMatrices(a4, b2, n);
                        //CALCULO DE C15
                        c15 = r.multiplicaMatrices(a4, b3, n);
                        //CALCULO DE C16
                        c16 = r.multiplicaMatrices(a4, b4, n);
                    break;
                }
            } catch (RemoteException e) {}
        }
    }
}

```

```

182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212

```

Imágenes 4 y 5. Capturas de la clase worker y las modificaciones a su método run.

Método conectar()

Este método tiene como función establecer las conexiones con los servidores para proceder a enviar las matrices a multiplicar, recibir las matrices resultantes de las multiplicaciones que realiza cada servidor y acomodar los valores de dichas matrices en su posición correspondiente en la matriz C.

Primero procedemos a llamar un total de 8 veces al método separaMatrizsto para obtener las 4 submatrices de la matriz A y B respectivamente, pero para esto, es necesario que en cada llamada al método especifiquemos de que matriz deseamos obtener la submatriz (primer argumento), también a partir de que posición de fila empezar a obtener los valores de la matriz original, teniendo en cuenta que tanto la matriz A como la B la debemos dividir en 4 partes iguales, usamos estos valores para obtener los siguientes valores para obtener los valores correspondientes:

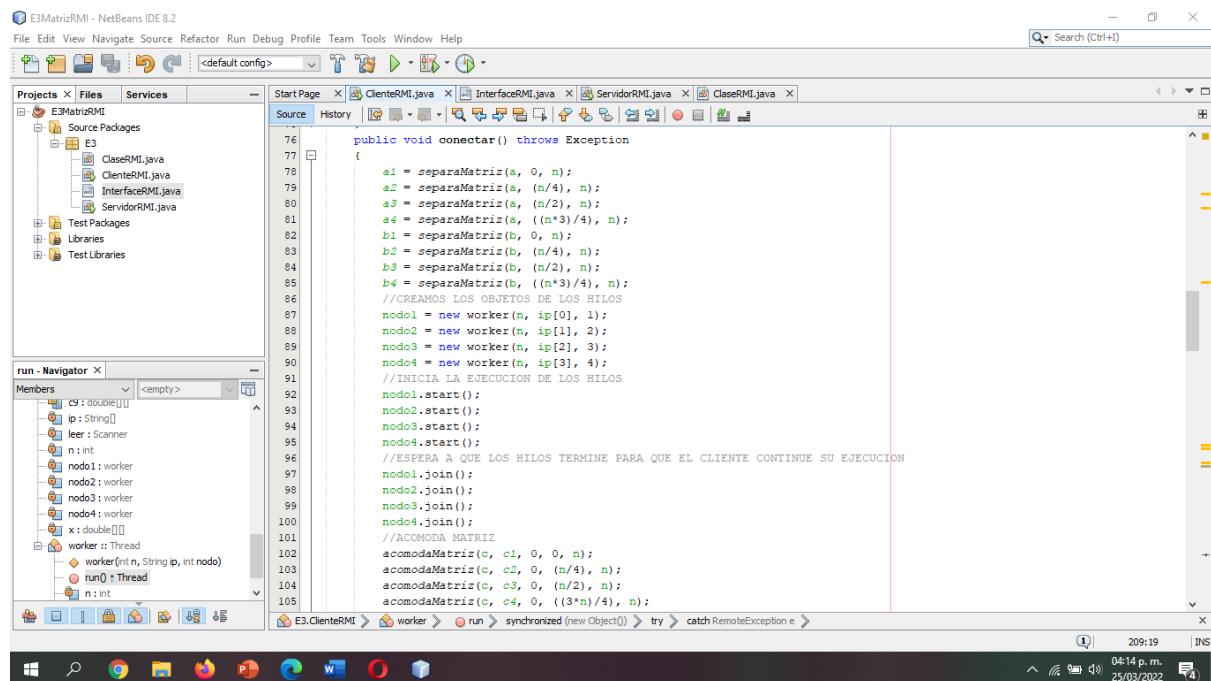
- 0 Para obtener la primera cuarta parte de la matriz.
- $(n/4)$ Para empezar una fila después a la última de la primera submatriz (0) y así obtener la segunda cuarta parte de la matriz .
- $(n/2)$ Esta empezaría en la mitad de filas de la matriz original, que recordando lo que hace el método separaMatrizsto nos dará como resultado la tercera cuarta parte de la matriz.
- $([n*3]/4)$ Este valor nos permite empezar a obtener los valores una fila después de la última fila de la tercera parte de la matriz, obteniendo la última cuarta parte de la matriz.

En cuanto al último parámetro, tratarse del tamaño de las dimensiones de la matriz, en todas las llamadas ponemos como parámetro nuestra variable global n.

Posterior a esto procedemos a inicializar nuestros cuatro variables globales nodo1, nodo2, nodo3 y nodo4 pasando como parámetros el tamaño de las dimensiones de las matrices la ip del servidor correspondiente y el número de nodo. Una vez inicializadas nuestras variables mandamos a llamar el método start de la clase worker para que cada nodo realice sus procedimientos de forma paralela, sin esperar a ningún otro nodo.

Habiendo obtenido las matrices resultantes de las multiplicaciones que realizó cada nodo servidor, procedemos a acomodar los valores de cada submatriz en su lugar correspondiente en la matriz C, por lo que es necesario mandar a llamar un total de 16 veces al método acomodaMatrizsto especificando que la matriz donde se acomodaran los valores es la matriz C (primer argumento), matriz de la que deseamos acomodar sus valores en la matriz C (segundo argumento), el tercer y cuarto argumento se tratan de posición de filas y columnas a iniciar el acomodamiento de los valores, que dependiendo de

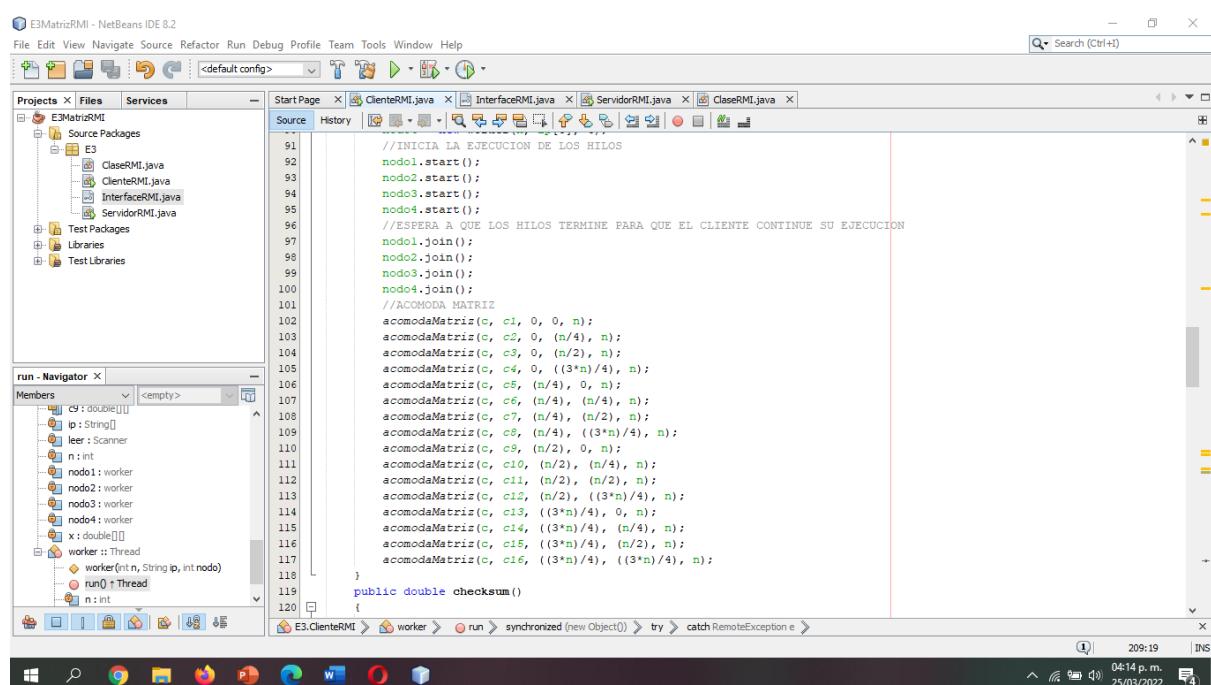
qué cuarta parte se trate será el valor que pondremos, si no fijamos se tratan de los mismos que pusimos en las llamadas al método separaMatriz, que ahora también se aplica a las columnas. El último parámetro se trata de n para poder establecerla condicional en el ciclo anidado y evitar un error por tratar de acceder a una posición de la submatriz que no existe o excede el tamaño de la submatriz.



```

public void conectar() throws Exception
{
    a1 = separaMatriz(a, 0, n);
    a2 = separaMatriz(a, (n/4), n);
    a3 = separaMatriz(a, (n/2), n);
    a4 = separaMatriz(a, ((n3)/4), n);
    b1 = separaMatriz(b, 0, n);
    b2 = separaMatriz(b, (n/4), n);
    b3 = separaMatriz(b, (n/2), n);
    b4 = separaMatriz(b, ((n3)/4), n);
    //CREAMOS LOS OBJETOS DE LOS HILOS
    nodo1 = new worker(n, ip[0], 1);
    nodo2 = new worker(n, ip[1], 2);
    nodo3 = new worker(n, ip[2], 3);
    nodo4 = new worker(n, ip[3], 4);
    //INICIA LA EJECUCION DE LOS HILOS
    nodo1.start();
    nodo2.start();
    nodo3.start();
    nodo4.start();
    //ESPERA A QUE LOS HILOS TERMINE PARA QUE EL CLIENTE CONTINUE SU EJECUCION
    nodo1.join();
    nodo2.join();
    nodo3.join();
    nodo4.join();
    //ACOMODA MATRIZ
    acomodaMatriz(c, c1, 0, 0, n);
    acomodaMatriz(c, c2, 0, (n/4), n);
    acomodaMatriz(c, c3, 0, (n/2), n);
    acomodaMatriz(c, c4, 0, ((3*n)/4), n);
}

```

```

//INICIA LA EJECUCION DE LOS HILOS
nodo1.start();
nodo2.start();
nodo3.start();
nodo4.start();
//ESPERA A QUE LOS HILOS TERMINE PARA QUE EL CLIENTE CONTINUE SU EJECUCION
nodo1.join();
nodo2.join();
nodo3.join();
nodo4.join();
//ACOMODA MATRIZ
acomodaMatriz(c, c1, 0, 0, n);
acomodaMatriz(c, c2, 0, (n/4), n);
acomodaMatriz(c, c3, 0, (n/2), n);
acomodaMatriz(c, c4, 0, ((3*n)/4), n);
acomodaMatriz(c, c5, (n/4), 0, n);
acomodaMatriz(c, c6, (n/4), (n/4), n);
acomodaMatriz(c, c7, (n/4), (n/2), n);
acomodaMatriz(c, c8, (n/2), 0, n);
acomodaMatriz(c, c10, (n/2), (n/4), n);
acomodaMatriz(c, c11, (n/2), (n/2), n);
acomodaMatriz(c, c12, (n/2), ((3*n)/4), n);
acomodaMatriz(c, c13, ((3*n)/4), 0, n);
acomodaMatriz(c, c14, ((3*n)/4), (n/4), n);
acomodaMatriz(c, c15, ((3*n)/4), (n/2), n);
acomodaMatriz(c, c16, ((3*n)/4), ((3*n)/4), n);
}

public double checksum()
{
}

```

Imágenes 6 y 7. Capturas del método conectar.

Constructor de la clase ClienteRMI(int n)

Este constructor recibe como parámetro el tamaño de las dimensiones de la matriz A y B, ya que lo necesita para asignar las dimensiones a nuestras matrices A, ~~B~~ y C. ~~Como~~ a las submatrices ya que en el caso de las submatrices de A y B, estas tienen una dimensión de $([n/4]*n)$. Una vez habiendo asignado las dimensiones procedemos a llenar las matrices A y B con ayuda de un ciclo anidado ~~asignando~~ a cada matriz y en su respectiva celda, valor que le corresponde de acuerdo a las siguientes fórmulas:

- ❖ Caso de A: $i+(2*j)$
- ❖ Caso de B: $(3*i)-j$

Posterior a llenar las matrices, con ayuda de otro ciclo anidado sacamos la transpuesta de B la cual sustituirá a la matriz ~~B~~. Esto claro con ayuda de una matriz auxiliar “x”, para poder realizar la transpuesta a partir de la diagonal.

La siguiente instrucción es pedir la ip de cada nodo servidor con ayuda de un ~~ciclo~~ for, que los servidores en la máquina virtual son independientes unos de otros.

Las últimas instrucciones sería la llamada ~~método~~ conectar que sabemos que manda a llamar a los métodos necesarios para realizar la multiplicación de las matrices ~~caso~~ recibir la multiplicación de estas submatrices por parte de los nodos servidor, y una vez que todo esto se haya realizado ~~procedemos~~ a mandar a llamar al ~~método~~ checksum para desplegar el resultado de la sumatoria de los valores de la matriz C (matriz resultante de la multiplicación de las submatrices de A y B). En caso de que $n=8$ el ~~método~~ checksum también desplegará en pantalla la matriz A, B y C.

E3MatrizRMI - NetBeans IDE 8.2

```

public ClienteRMI(int n)
{
    this.n = n;
    leer = new Scanner(System.in);
    ip = new String[4];
    a = new double[n][n];
    b = new double[n][n];
    c = new double[n][n];
    x = new double[n][n];
    a1 = new double[n/4][n];
    b1 = new double[n/4][n];
    a2 = new double[n/4][n];
    b2 = new double[n/4][n];
    a3 = new double[n/4][n];
    b3 = new double[n/4][n];
    a4 = new double[n/4][n];
    b4 = new double[n/4][n];
    for(int i=0; i<this.n; i++)
        for(int j=0; j<this.n; j++)
    {
        a[i][j] = i+2*j;
        b[i][j] = 3*i-j;
    }
    for(int i=0; i<this.n; i++)
        for(int j=0; j<this.n; j++)
            x[i][j] = b[i][j];
    b = x;
    System.out.println("ingresa las ip");
    for(int i=0; i<4; i++)
    {
        ip[i] = leer.nextLine();
    }
}

```

E3MatrizRMI - NetBeans IDE 8.2

```

public void multiplicar()
{
    b2 = new double[n/4][n];
    a3 = new double[n/4][n];
    b3 = new double[n/4][n];
    a4 = new double[n/4][n];
    b4 = new double[n/4][n];
    for(int i=0; i<this.n; i++)
        for(int j=0; j<this.n; j++)
    {
        a[i][j] = i+2*j;
        b[i][j] = 3*i-j;
    }
    for(int i=0; i<this.n; i++)
        for(int j=0; j<this.n; j++)
            x[i][j] = b[i][j];
    b = x;
    System.out.println("ingresa las ip");
    for(int i=0; i<4; i++)
    {
        System.out.println("ingresa la ip del nodo :"+(i+1));
        ip[i] = leer.nextLine();
    }
    try {
        conectar();
        System.out.println("checksum cuando n == "+n+"es igual a :"+checksum());
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
public void conectar() throws Exception
{
}

```

Imágenes 8 y 9 capturas de constructor de la clase ClienteRMI **hdo le pasamos como parametro el tamaño de las dimensiones de las matrices A y B.**

Método checksum()

El método checksum es el encargado de realizar la sumatoria del valor de cada una de las celdas de la matriz C (matriz resultante de la multiplicación de A y B) por lo que es necesario una variable de tipo double que servirá como acumulador (s) con el fin de poder realizar la sumatoria.

Lo primero que se hace es verificar si el tamaño de las dimensiones de A y B es igual a 8, ya que en caso de serlo se procederá a usar ciclos for para poder desplegar las matrices A,B y C, además de esto se utilizara un ciclo anidado que recorre cada elemento de la matriz C empezando por recorrer cada columna de la matriz para poder pasar a la siguiente fila y volver a recorrer cada columna hasta haber recorrido cada celda de la matriz y haber acumulado cada uno de estos valores en nuestra variable auxiliar dandonos así la sumatoria. En caso de no cumplirse la condición de que el tamaño de las dimensiones de las matrices A y B sea 8, solo se realizará el nodo anidado que realiza la sumatoria, independientemente si cumple la condición o no, terminar la sumatoria esta retorna como valor de tipo double.

```

118 }
119     public double checksum()
120     {
121         double s = 0;
122         if(n == 8)
123         {
124             System.out.println("MATRIZ A");
125             for(int i = 0; i<n; i++)
126             {
127                 for(int j = 0; j < n; j++)
128                     System.out.printf("%f \t", a[i][j]);
129                 System.out.print("\n");
130             }
131             System.out.println("MATRIZ B");
132             for(int i = 0; i<n; i++)
133             {
134                 for(int j = 0; j < n; j++)
135                     System.out.printf("%f \t", b[i][j]);
136                 System.out.print("\n");
137             }
138             System.out.println("MATRIZ C");
139             for(int i = 0; i<n; i++)
140             {
141                 for(int j = 0; j < n; j++)
142                     System.out.printf("%f \t", c[i][j]);
143                 System.out.print("\n");
144             }
145             for(int i = 0; i<c[0].length; i++)
146                 for(int j = 0; j < c[0].length; j++)
147                     s += c[i][j];
148         }
149     }
150     synchronized (new Object())
151     {
152         try
153         {
154             catch( RemoteException e )
155         }
156     }
157 }

```

```

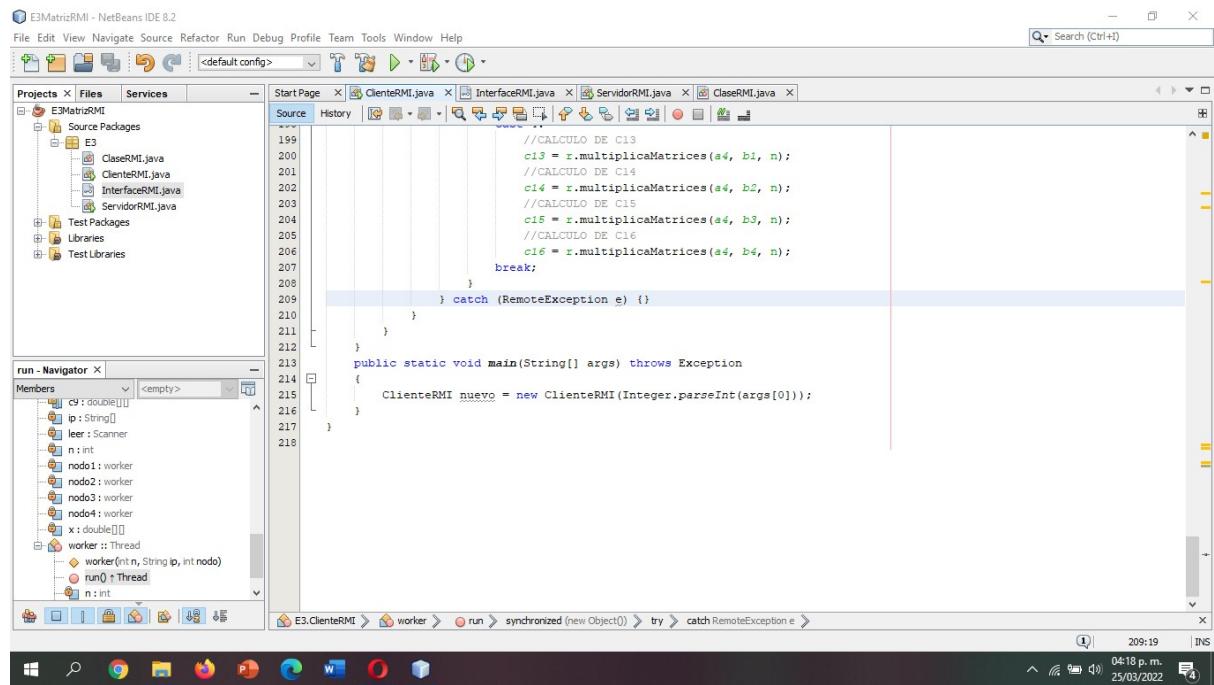
121     double s = 0;
122     if(n == 0)
123     {
124         System.out.println("MATRIZ A");
125         for(int i = 0; i<n; i++)
126         {
127             for(int j = 0; j < n; j++)
128                 System.out.printf("%f \t", a[i][j]);
129             System.out.print("\n");
130         }
131         System.out.println("MATRIZ B");
132         for(int i = 0; i<n; i++)
133         {
134             for(int j = 0; j < n; j++)
135                 System.out.printf("%f \t", b[i][j]);
136             System.out.print("\n");
137         }
138         System.out.println("MATRIZ C");
139         for(int i = 0; i<n; i++)
140         {
141             for(int j = 0; j < n; j++)
142                 System.out.printf("%f \t", c[i][j]);
143             System.out.print("\n");
144         }
145         for(int i = 0; i<c[0].length; i++)
146             for(int j = 0; j < c[0].length; j++)
147                 s += c[i][j];
148     }
149     return s;
150 }

```

Imágenes 10 y 11. Capturas del método checksum

Metodo main()

El método main se encarga de crear una instancia de la clase ClienteRMI usando el constructor que pide como parámetro el tamaño de las dimensiones de la matriz, esto teniendo como efecto que se inicializan nuestras variables y se proceda a realizar todo el procedimiento para la multiplicación de las matrices y el checksum, ya que como vimos este constructor será el encargado de empezar este procedimiento, al que terminarlo al mandar a llamar a la función checksum.



capturamos el arreglo regresado en una de las submatrices de C que también son variables globales de la clase ClienteRMI.

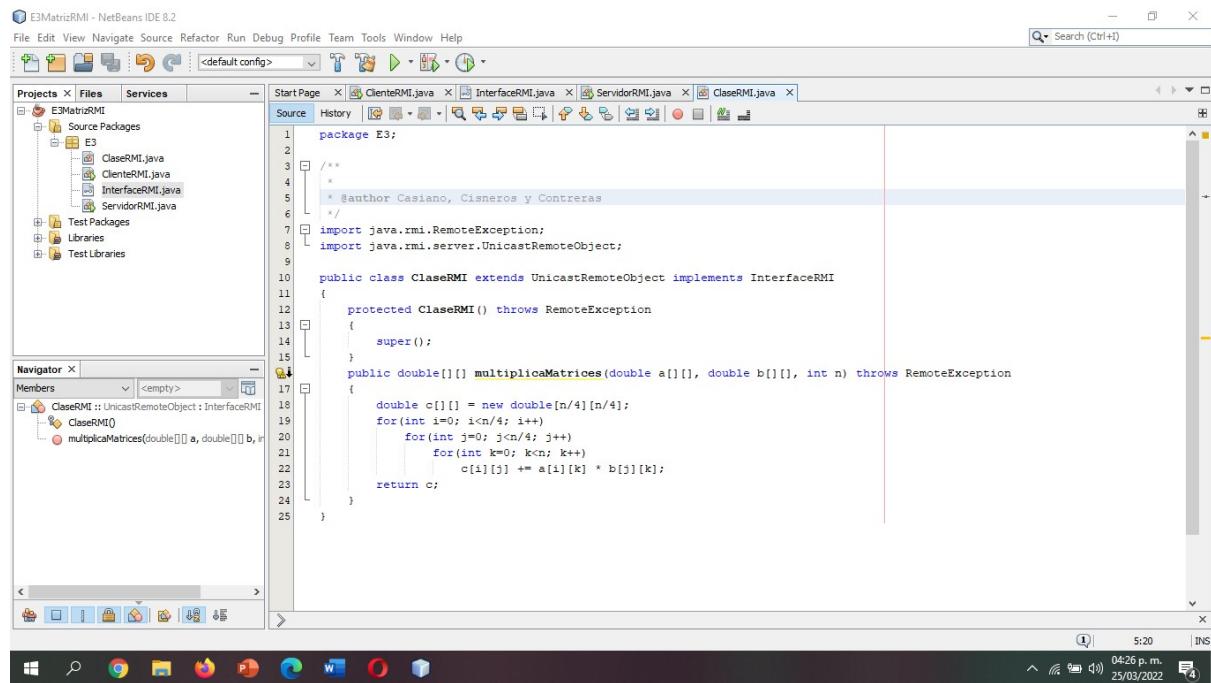


Imagen 13. Captura de ClaseRMI.

Clase InterfaceRMI.java

La clase InterfaceRMI se trata de un objeto remoto donde declaramos los métodos a exportar, que en este caso solo declaramos el método multiplicaMatrices de la clase ClaseRMI, ya que nuestra intención es dejar la realización del método más ~~pesada~~ en este caso se trata de la multiplicación de las submatrices, a cargo de un objeto remoto.

```

package E3;
/*
 * @author Casiano, Cisneros y Contreras
 */
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface InterfaceRMI extends Remote
{
    //public double checksum(int m[][])
    public double[] multiplicaMatrices(double a[][], double[][] b, int n) throws RemoteException;
}

```

Imagen 14. Captura de la interfaz RMI “InterfaceRMI”.

Clase ServidorRMI.java

La clase ServidorRMI registra en el servidor rmiregistry una instancia de la clase “ClaseRMI” con ayuda del método rebind, que además de esta instancia también le pasamos la variable de tipo String url, la cual contiene la ip del servidor y el nombre con el que identificamos al objeto.

```

package E3;
/*
 * @author Casiano, Cisneros y Contreras
 */
import java.rmi.Naming;

public class ServidorRMI {
    public static void main(String[] args) throws Exception
    {
        String url = "rmi://localhost/server";
        ClaseRMI obj = new ClaseRMI();
        //REGISTRA LA INSTANCIA EN EL RMIREGISTRY
        Naming.rebind(url, obj);
    }
}

```

Imagen 15. Captura de la clase ServidorRMI.

Compilacion y ejecucion del programa

```
Actividades Terminal 24 de mar 00:27 • equipo0@Tarea-5-3-0:~$ java ClienteRMI 8
ingresa las ip
ingresa la ip del nodo :1
20.106.163.149
ingresa la ip del nodo :2
20.124.230.143
ingresa la ip del nodo :3
20.185.36.30
ingresa la ip del nodo :4
20.25.10.94

equipo1@Tarea-5-3-1:~$ rmiregistry
equipo2@Tarea-5-3-2:~$ rmiregistry
equipo3@Tarea-5-3-3:~$ rmiregistry
equipo4@Tarea-5-3-4:~$ rmiregistry
```

```
Actividades Terminal 24 de mar 00:28 • equipo0@Tarea-5-3-0:~$ 
MATRIZ C
840.000000 784.000000 728.000000 672.000000 616.000000 560.0000
00 504.000000 448.000000
924.000000 860.000000 796.000000 732.000000 668.000000 604.0000
00 540.000000 476.000000
1008.000000 936.000000 864.000000 792.000000 720.000000 648.0000
00 576.000000 504.000000
1092.000000 1012.000000 932.000000 852.000000 772.000000 692.0000
00 612.000000 532.000000
1176.000000 1088.000000 1000.000000 912.000000 824.000000 736.0000
00 648.000000 560.000000
1260.000000 1164.000000 1068.000000 972.000000 876.000000 780.0000
00 684.000000 588.000000
1344.000000 1240.000000 1136.000000 1032.000000 928.000000 824.0000
00 720.000000 616.000000
1428.000000 1316.000000 1204.000000 1092.000000 980.000000 868.0000
00 756.000000 644.000000
checksum cuando n =8es igual a :53760.0
equipo0@Tarea-5-3-0:~$ 

equipo1@Tarea-5-3-1:~$ rmiregistry
equipo2@Tarea-5-3-2:~$ rmiregistry
equipo3@Tarea-5-3-3:~$ rmiregistry
equipo4@Tarea-5-3-4:~$ rmiregistry
```

Imágenes 16 y 17. Capturas de la prueba del código cuando N=8.

Actividades Terminal 24 de mar 00:43

```
equipo0@Tarea-5-3-0:~$ java ClienteRMI 4000
ingresa las ip
ingresa la ip del nodo :1
20.106.163.149
ingresa la ip del nodo :2
20.124.230.143
ingresa la ip del nodo :3
20.185.36.30
ingresa la ip del nodo :4
20.25.10.94
```

equipo1@Tarea-5-3-1:~\$ rmiregistry

equipo1@Tarea-5-3-1:~\$ java ServidorRMI

equipo3@Tarea-5-3-3:~\$ rmiregistry

equipo3@Tarea-5-3-3:~\$ java ServidorRMI

equipo4@Tarea-5-3-4:~\$ java ServidorRMI

equipo2@Tarea-5-3-2:~\$ rmiregistry

equipo2@Tarea-5-3-2:~\$ java ServidorRMI

equipo4@Tarea-5-3-4:~\$ rmiregistry

Actividades Terminal 24 de mar 00:44

```
equipo0@Tarea-5-3-0:~$ java ClienteRMI 4000
ingresa las ip
ingresa la ip del nodo :1
20.106.163.149
ingresa la ip del nodo :2
20.124.230.143
ingresa la ip del nodo :3
20.185.36.30
ingresa la ip del nodo :4
20.25.10.94
checksum cuando n =4000es igual a :2.04723206400044774E18
equipo0@Tarea-5-3-0:~$
```

equipo1@Tarea-5-3-1:~\$ rmiregistry

equipo1@Tarea-5-3-1:~\$ java ServidorRMI

equipo3@Tarea-5-3-3:~\$ rmiregistry

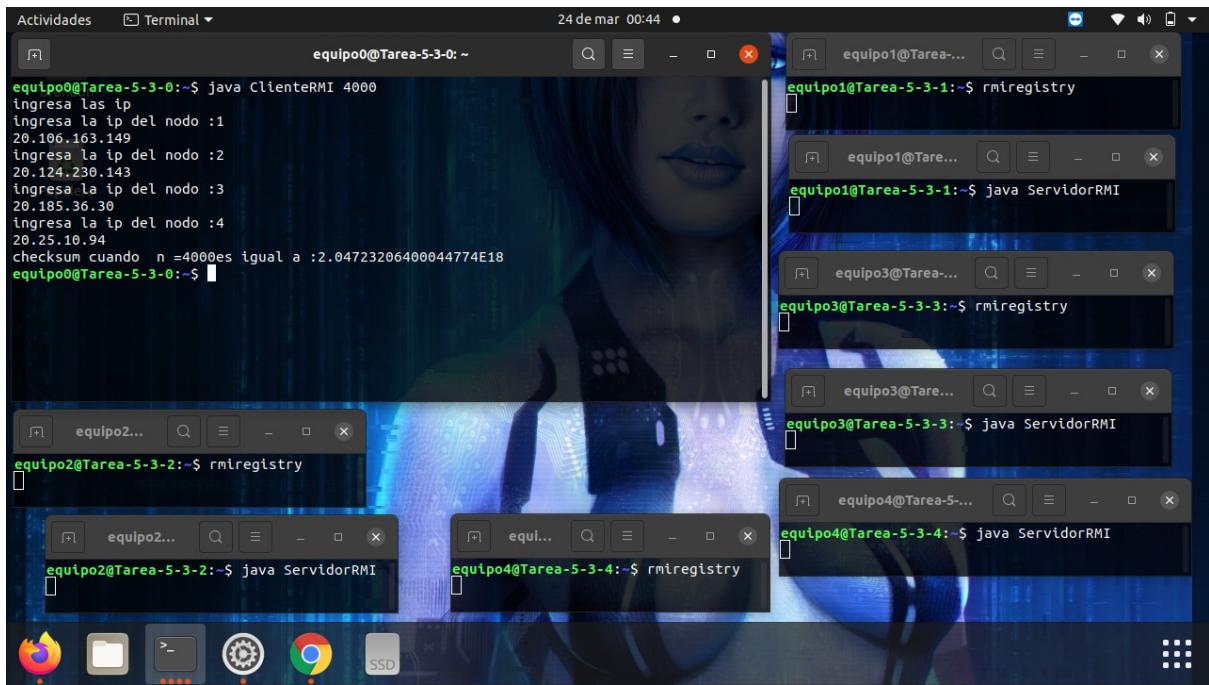
equipo3@Tarea-5-3-3:~\$ java ServidorRMI

equipo4@Tarea-5-3-4:~\$ java ServidorRMI

equipo2@Tarea-5-3-2:~\$ rmiregistry

equipo2@Tarea-5-3-2:~\$ java ServidorRMI

equipo4@Tarea-5-3-4:~\$ rmiregistry



Imágenes 18 y 18. Capturas de la prueba del código cuando N=4000