Instituto Politécnico Nacional
Escuela Superior de Cómputo

Cálculo de PI

Tarea 1

Alumno: Francisco Nicolas Sánchez García
Asignatura Desarrollo de Sistemas Distribuidos
Profesor: Carlos Pineda Guerrero
Grupo 4CV12

ÍNDICE

INTRODUCCIÓN

Hay diferentes formas de calcular el número irracional PI, específicamente antes de la era computacional, se utilizó en gran medida la serie de Gregory-Leibniz como una aproximación con gran cantidad de decimales encontrados:

$$\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots)$$

Gracias a esta serie, podemos desarrollar un algoritmo capaz de trabajar de manera distribuida en cuatro nodos que se van a encargar de realizar las operaciones (clientes) y un nodo central (servidor) de recibir los resultados de los clientes y mostrar el resultado final.

DESARROLLO

Se realizó la clase Pi.java siguiendo las indicaciones de la tarea, posteriormente, se pasa a compilar y ejecutar las 5 ventanas diferentes.

Sin embargo, se sustituyó el tipo de dato de *float* a *double* ya que usando el tipo de dato *float* había una pérdida considerable de precisión en el cálculo de la serie.

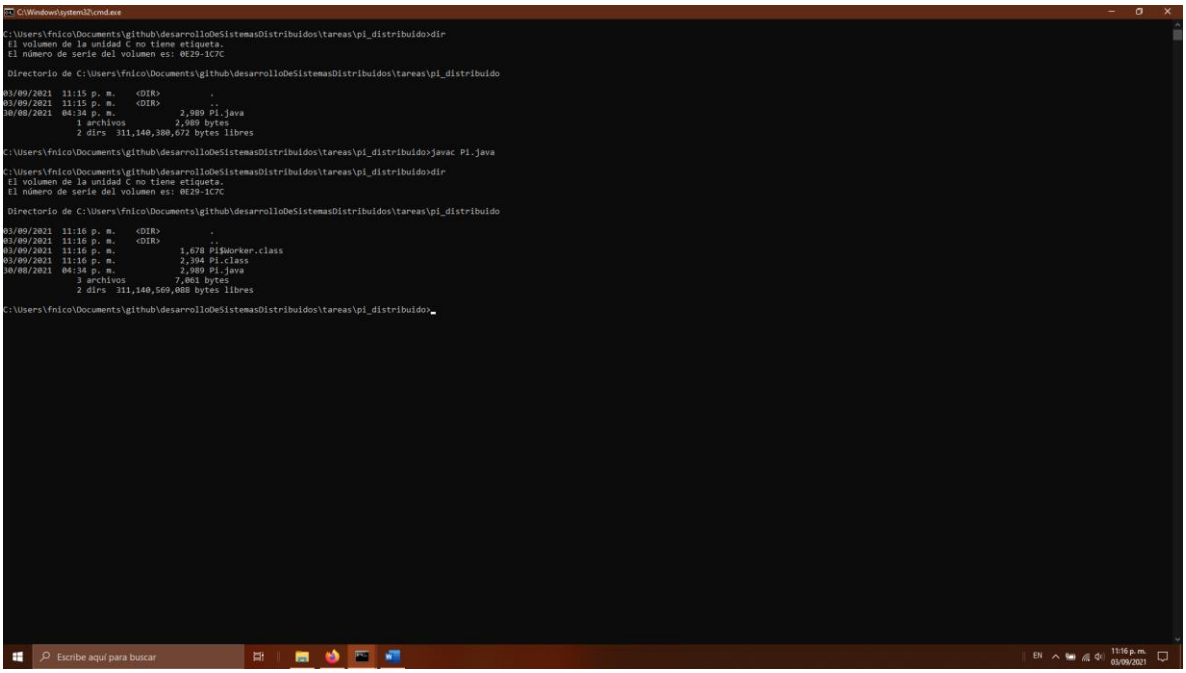
Código fuente de Pi.java

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

import jdk.jshell.spi.ExecutionControl.ExecutionControlException;

class Pi {
    static Object obj = new Object();
    static double pi = 0;

    static class Worker extends Thread {
        Socket conexion;

        Worker(Socket conexion) {
            this.conexion = conexion;
        }
```

```
        public void run() {
            // algoritmo 1
            try {
                DataOutputStream out = new
DataOutputStream(conexion.getOutputStream());
                DataInputStream in = new
DataInputStream(conexion.getInputStream());

                double suma = 0;
                // recibir suma del cliente
                suma = in.readDouble();

                synchronized (obj) {
                    pi += suma;
                }

                out.close();
                in.close();
                conexion.close();
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.out.println("Uso:");
            System.out.println("java PI <nodo>");
            System.exit(0);
        }

        int nodo = Integer.valueOf(args[0]);
        if (nodo == 0) {
            // servidor - algoritmo 2
            ServerSocket servidor;
            servidor = new ServerSocket(20000);
            Worker[] v = new Worker[4];

            int i = 0;
            while (i != 4) {
                Socket conexion;
                conexion = servidor.accept();
                v[i] = new Worker(conexion);
                v[i].start();
                i++;
            }

            i = 0;
            while (i != 4) {
                v[i].join();
                i++;
            }
            System.out.println("PI:" + pi);
            servidor.close();
        } else {
            // cliente - algoritmo 3
```

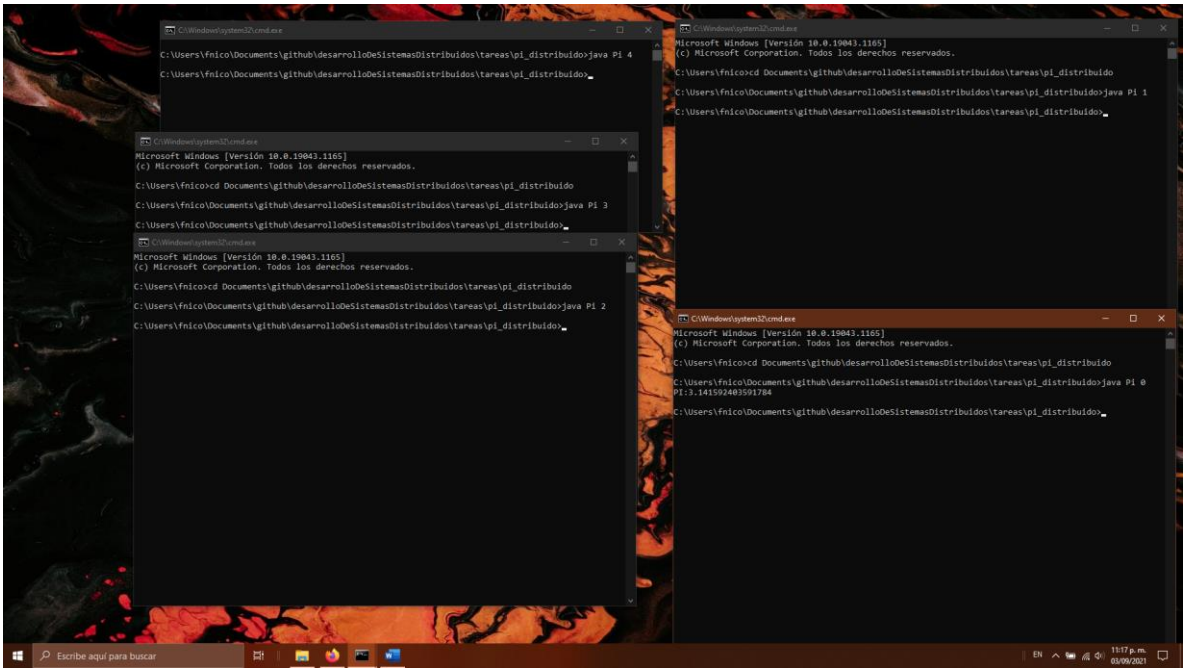
```
            // conexion con reintentos
            Socket conexion = null;
            while (true) {
                try {
                    conexion = new Socket("localhost", 20000);
                    break;
                } catch (Exception e) {
                    Thread.sleep(100);
                }
            }

            DataOutputStream out = new
DataOutputStream(conexion.getOutputStream());
            DataInputStream in = new
DataInputStream(conexion.getInputStream());

            double suma = 0;
            int i = 0;
            while (i != 1000000) {
                suma = (4.0 / (8 * i + 2 * (nodo - 2) + 3)) + suma;
                i++;
            }
            suma = nodo % 2 == 0 ? -suma : suma;
            out.writeDouble(suma);
            out.close();
            in.close();
            conexion.close();
        }
    }
}
```

Compilación del programa:



Para la ejecución del servidor, se pasa como parámetro el número 0. Y para los cuatro clientes números del 1 al 4.

A continuación, se muestra la ejecución de las cinco ventanas.



Con un resultado PI = 3.1415923457121266 como se ve en la primera consola de arriba.

CONCLUSIONES

Si bien, el valor calculado de Pi se acerca al usado comúnmente, aún dista de ser un valor que se aproxime a Pi. Por ejemplo, WolframAlpha tiene como resultado de Pi el valor de:

3.14159265358979323846264338327950288419716939937510582097494459 23...

Encontrando una precisión de solo seis dígitos decimales después de realizar un millón de operaciones en cada uno de los cuatro clientes.

Sin embargo, la verdadera importancia recae en una de las ventajas de este tipo de algoritmos. Buscamos dividir el cálculo de muchas operaciones, dando paso a poder ejecutarlo en una gran cantidad de clientes, llegando a la escalabilidad horizontal permitiéndonos no sobrecargar solo un dispositivo (en este caso el servidor).