Cálculo de la Integral usando la arquitectura cliente-servidor

Students:
Anthony Daniel, Bautista León
Katheryn Ximena, Peralta Haro
Renzo Renato, Quispe Amao
Universidad Nacional de Ingeniería, Facultad de Ciencias,
e-mail: abautistal@uni.pe, katheryn.peralta.h@uni.pe, rrquispea@uni.pe

Curso: CC4P1 Programación Concurrente y Distribuida
Examen Parcial

Abstract

El objetivo de este trabajo es desarrollar un canal de comunicación en la que un nodo maestro le envíe una tarea a varios nodos, donde estos realicen una tarea en paralelo y distribuido, calculando una integral definida mediante el método de integración por rectángulos. Se trabajó con dos lenguajes de programación, Java (Cliente) y Python (Servidor).

Keywords: Thread, JVM, socket, TCP.

Contents

1 Introducción

La comunicación entre entidades abstractas (PCs) se logra utilizando diferentes técnicas, una de ellas es utilizando sockets, que consiste en configurar una red cliente servidor para establecer el flujo de información entre transmisor y receptor.

Es posible utilizar un protocolo orientado a conexión como TCP (Transfer Control Protocol) o uno no orientado a conexión como UDP (User Datagram Protocol), la diferencia entre los dos es la fiabilidad que se puede garantizar con un protocolo que garantice la conexión con el extremo receptor, es decir que sea orientado a conexión.

Para que dos aplicaciones puedan intercambiar información entre sí se necesita:

- Un protocolo de comunicación común a nivel de red y a nivel de transporte.
- Una dirección del protocolo de red que identifique a cada uno de los computadores.
- Un número de puerto que identifi que la aplicación dentro del computador.

Los socket se utilizan para poder enviar órdenes a un servidor que está atendiendo nuestras peticiones. Por lo tanto, un socket quedaría definido por una dirección IP (La del equipo que actúa como servidor), un protocolo y un número de puerto (El utilizado para acceder a un servicio determinado).

2 Marco Teórico

2.1 Sockets

Un socket, es un método para la comunicación entre un programa del cliente y un programa del servidor en una red. Un socket se define como el punto final en una conexión. Los sockets se crean y se utilizan con un sistema de peticiones o de llamadas de función a veces llamados interfaz de programación de aplicación de sockets (API, Application Programming Interface).

Los sockets están basados en una arquitectura cliente/servidor

- En esta arquitectura uno de los programas debe estar siempre arrancado y pendiente de que alguien establezca conexión con él. Este programa se denomina servidor.
- El otro programa lo arranca el usuario cuando lo necesita y es el programa que da el primer paso en el establecimiento de la comunicación. Este programa se llama cliente.

2.2 Servidor

El servidor, está continuamente a la escucha y a la espera de que alguien se quiera conectar a él.
Si hacemos una comparación con un teléfono, un servidor es una empresa 24 horas, 365 días al año, pendiente de recibir llamadas de sus clientes.

2.3 Cliente

El cliente, en un momento dado decide conectarse a un servidor y hacerle alguna petición. Cuando el cliente no necesita al servidor, cierra la conexión.
En la comparativa del teléfono, el cliente es el que llama por teléfono a la empresa cuando necesita algo de ella. Por ejemplo, un servidor de páginas web está siempre en marcha y a la escucha. El navegador es el cliente. Cuando arrancamos el navegador y ponemos la dirección del servidor web, el navegador establece la comunicación y le pide al servidor la página web que queremos ver. El servidor la envía y el navegador la muestra.

La comunicación entre procesos consiste en la transmisión de un mensaje entre un conector de un proceso y un conector de otro proceso. Para los procesos receptores de mensajes, su conector debe estar asociado a un

puerto local y a una de las direcciones Internet de la computadora donde se ejecuta. Los mensajes enviados a una dirección de Internet y a un número de puerto concretos, sólo pueden ser recibidos por el proceso cuyo conector esté asociado con esa dirección y con ese puerto. Los procesos pueden utilizar un mismo conector tanto para enviar como para recibir mensajes.

Cada computadora permite un gran número de puertos posibles, que pueden ser usados por los procesos locales para recibir mensajes. Cada proceso puede utilizar varios puertos para recibir mensajes, pero un proceso no puede compartir puertos con otros procesos de la misma computadora. Cualquier cantidad de procesos puede enviar mensajes a un mismo puerto. Cada conector se asocia con un protocolo concreto, que puede ser UDP o TCP

2.4 Sockets Flujo (TCP)

Son un servicio orientado a la conexión, donde los datos se transfieren sin encuadrarlos en registros o bloques. Si se rompe la conexión entre los procesos, éstos serán informados de tal suceso para que tomen las medidas oportunas.

El protocolo de comunicaciones con flujos es un protocolo orientado a la conexión, ya que para establecer una comunicación utilizando el protocolo TCP, hay que establecer en primer lugar una conexión entre un par de sockets. Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

El protocolo TCP (Transmission Control Protocol) funciona en el nivel de trasporte, basándose en el protocolo de red IP (Internet Protocol).IP proporciona comunicaciones no fiables y no basadas en conexión, muy dependientes de saturación en la red, caídas de nodos, etc. Por el contrario, TCP está orientado a conexión y proporciona comunicaciones fiables basadas en mecanismos de red que gestionan el control de flujo de paquetes y de congestión en los nodos.

3 Metodología

3.1 Sockets en Java

En Java, las comunicaciones TCP se realizan utilizando la clásica abstracción de socket. Los sockets nos permiten establecer y programar comunicaciones sin tener que conocer los niveles inferiores sobre los que se asientan.
Para identificar el destino de los paquetes de datos, los sockets utilizan los conceptos de dirección y puerto. Los valores numéricos de puertos 1 – 1023 se reservan a servicios de interés general, montados a menudo sobre protocolos de uso extendido:

Puerto	Servicio
80	Para web con HTTP
25	Para correo saliente con SMTP
110	Para correo entrante con POP3
119	Para el servicio de noticias NNTP

Los valores de puertos entre 1024 y 49151 se usan para servicios específicos de uso no general, el resto (a partir de 49152) se emplean para designar servicios de uso esporádico.

3.2 Establecimiento de comunicaciones

Java proporciona dos clases de abstracción de comunicaciones TCP: una para los procesos cliente (socket) y otra para los procesos servidor (ServerSocket).
La Figura 3.1 que es el esquema básico de establecimiento de comunicaciones TCP.

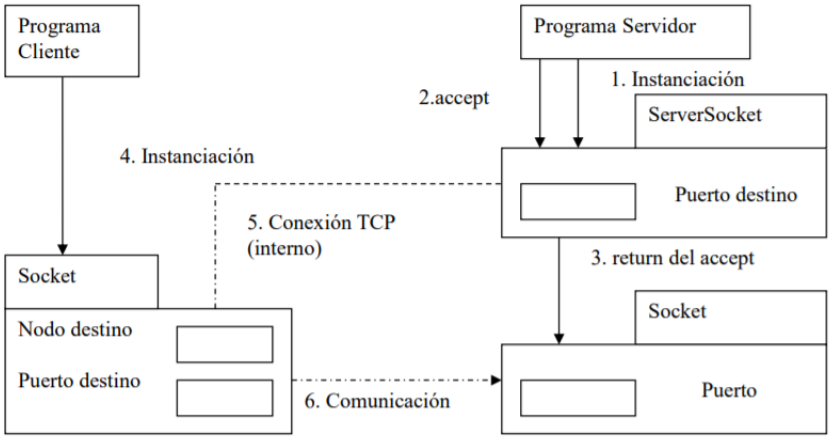

Figure 1: Comunicación TCP entre un cliente y un servidor.

3.3 Transmisión de datos

TCP es un protocolo especialmente útil cuando se desea transmitir un flujo de datos en lugar de pequeñas cantidades aisladas de información. Debido a está característica, los sockets de Java están diseñados para transmitir y recibir datos a través de los Streams definidos en el paquete java.io. La clase Socket contiene dos métodos importantes que se emplean en el proceso de transmisión de flujos de datos:

- InputStream getInputStream()
- OutputStream getOutputStream()

Estas clases son abstractas, por lo que no podemos emplear directamente todos sus métodos. En general se usan otras clases más especializadas que nos permiten trabajar con flujos de datos como:
DataOutputStream, DataInputStream,
FileOutputStream, FileInputStream, etc.

Ejemplo:

Programa Servidor:

```
import java.io.* ;

import java.net.* ;

class Servidor {

  static final int PUERTO=5000;

  public Servidor( ) {

    try {

      ServerSocket skServidor = new ServerSocket( PUERTO );

      System.out.println("Escucho el puerto " + PUERTO );

      for ( int numCli = 0; numCli < 3; numCli++; ) {

        Socket skCliente = skServidor.accept(); // Crea objeto

        System.out.println("Sirvo al cliente " + numCli);
```

```
        OutputStream aux = skCliente.getOutputStream();
        DataOutputStream flujo= new DataOutputStream( aux );
        flujo.writeUTF( "Hola cliente " + numCli );
        skCliente.close();
      }
      System.out.println("Demasiados clientes por hoy");
    } catch( Exception e ) {
      System.out.println( e.getMessage() );
    }
  }
  public static void main( String[] arg ) {
    new Servidor();
  }
}
```

Programa Cliente:

```
import java.io.*;
import java.net.*;
class Cliente {
  static final String HOST = "localhost";
  static final int PUERTO=5000;
  public Cliente( ) {
    try{
      Socket skCliente = new Socket( HOST , Puerto );
      InputStream aux = skCliente.getInputStream();
      DataInputStream flujo = new DataInputStream( aux );
      System.out.println( flujo.readUTF() );
      skCliente.close();
    } catch( Exception e ) {
      System.out.println( e.getMessage() );
```

```
    }
  }
  public static void main( String[] arg ) {
    new Cliente();
  }
}
```

4 Desarrollo de la Investigación

Para la investigación se utilizó computadoras con un promedio de 12 hilos cada una. Cada cliente ejecutaba un total de 5 hilos, 4 hilos para el cálculo de una parte de la integral y una para monitorear el ingreso de caracteres por el teclado. Los socket utilizaron el puerto 8000 para comunicarse entre sí. El servidor se ejecuta y espera a que los clientes se conecten, luego desde este se envia el string siguiente "envio n a b funcion" para calcular la integral de la siguiente función en los intervalos $[a, b]$.
Ejemplo : envio 1000000 0 13 $4 * x^2 - 2 * x^1$

- Cliente
  - Hilo
  - Cliente50
  - TCPCliente50
- Servidor
  - Servidor

4.1 Diagrama de Clases

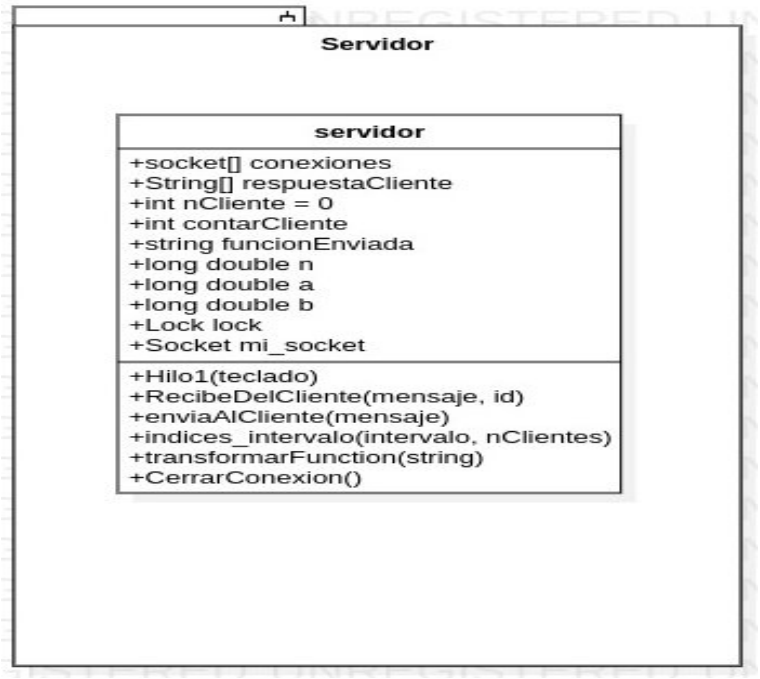

Figure 2: Servidor

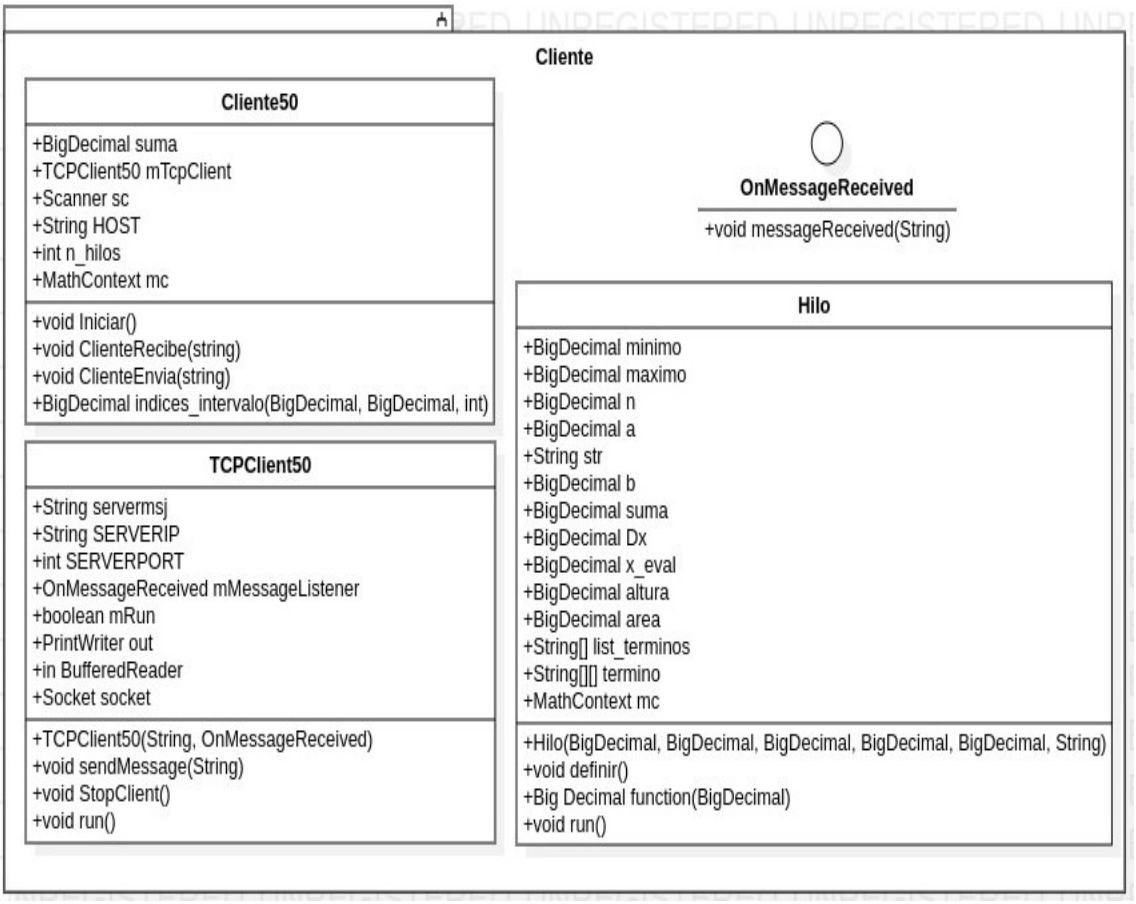

Figure 3: Cliente

5 Resultados

5.1 Cliente vs Tiempo

Para el primer experimento se utilizó un n constante, $n = 2100000$, el cual es el número de particiones del intervalo de la integral y aumentamos la cantidad de clientes en cada solicitud de cálculo de la integral siguiente.

$$f(x) = \int_{x=0}^{x=13} x^3 - 4x^2 + 4x^1 - 10 \cdot dx \tag{1}$$

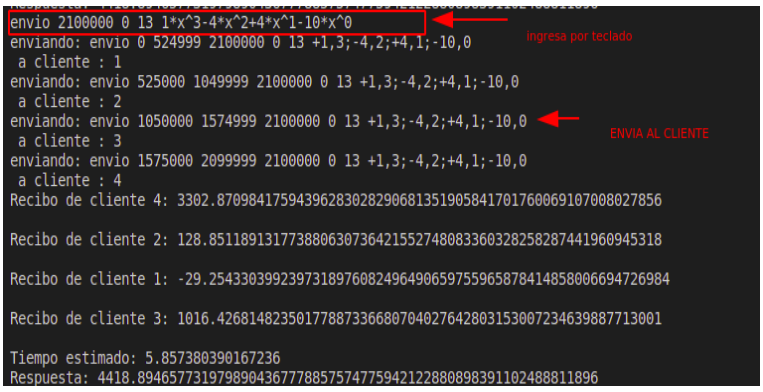

Figure 4: Servidor con 4 clientes

```
CLINTE50 El mensaje::envio 1050000 1574999 2100000 0 13 +1,3;-4,2;+4,1;-10,0
la funcion :+1,3;-4,2;+4,1;-10,0
 a :0
 b:13
n:2100000
minimo:1050000maximo:1574999
prefijos:[Ljava.math.BigDecimal;@142c192a
Hilo 0:prefijo inicio: 1050000 prefijo final :1181249
hilo 0 : Thread-25
Hilo 1:prefijo inicio: 1181249 prefijo final :1312498
hilo 1 : Thread-26
Hilo 2:prefijo inicio: 1312498 prefijo final :1443747
hilo 2 : Thread-27
Hilo 3:prefijo inicio: 1443747 prefijo final :1574996
hilo 3 : Thread-28
hilo 0  suma : 127.693965958048070421745086923658352229780801209372637963831
hilo 1  suma : 197.833595481784332864326881546269301371342187668718280989115
hilo 2  suma : 288.522823997938047689217552100205161429651225569592916564486
hilo 3  suma : 402.376429385731327898077286470143613000755857898715041612578
suma en el cliente :1016.426814823501778873366807040276428031530072346398877130 01
```

Figure 5: Cliente 3 recibe un tramo del intervalo total

número de clientes	Tiempo(s)
1	12,936
2	9,429
3	6,145
4	5,81
5	4,669
6	5,47
7	5,599

Figure 6: Tabla de cliente vs tiempo

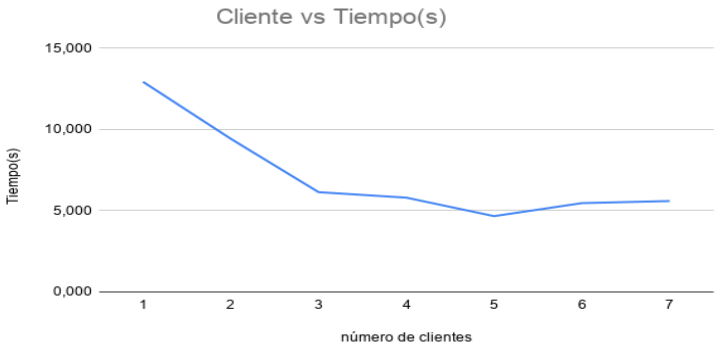

Figure 7: Cliente vs tiempo

5.2 n vs tiempo

Se desarrolló otro experimento manteniendo el número de clientes constantes(5 clientes) y aumentando el número de particiones del intervalo; es decir aumentando ”n”.

```
envio 10000000 0 13 1*x^3-4*x^2+4*x^1-10*x^0
enviando: envio 0 1999999 10000000 0 13 +1,3;-4,2;+4,1;-10,0
 a cliente : 1
enviando: envio 2000000 3999999 10000000 0 13 +1,3;-4,2;+4,1;-10,0
 a cliente : 2
enviando: envio 4000000 5999999 10000000 0 13 +1,3;-4,2;+4,1;-10,0
 a cliente : 3
enviando: envio 6000000 7999999 10000000 0 13 +1,3;-4,2;+4,1;-10,0
 a cliente : 4
enviando: envio 8000000 9999999 10000000 0 13 +1,3;-4,2;+4,1;-10,0
 a cliente : 5
Recibo de cliente 5: 2881.766580659732217314317000000

Recibo de cliente 4: 1200.825979635455139114317000000

Recibo de cliente 1: -24.490268550976095485683000000

Recibo de cliente 2: 21.883155030100982714317000000

Recibo de cliente 3: 338.926704425578060914317000000

Tiempo estimado: 7.39870023727417
Respuesta: 4418.912151199890304571585000000
```

Figure 8: Servidor con 5 clientes

```
TCP ClientC: Conectando...
TCP ClientC: Sent.
TCP ClientC: Done.
entrada in:
Cliente bandera 01
entro a clienteRecibe
CLINTE50 El mensaje::envio 0 19999 100000 0 13 +1,3;-4,2;+4,1;-10,0
la funcion :+1,3;-4,2;+4,1;-10,0
 a :0
 b:13
n:100000
minimo:0maximo:19999
prefijos:[Ljava.math.BigDecimal;@7bcf2a93
Hilo 0:prefijo inicio: 0 prefijo final :4999
hilo 0 : Thread-1
Hilo 1:prefijo inicio: 4999 prefijo final :9998
hilo 1 : Thread-2
Hilo 2:prefijo inicio: 9998 prefijo final :14997
hilo 2 : Thread-3
Hilo 3:prefijo inicio: 14997 prefijo final :19996
hilo 3 : Thread-4
hilo 0  suma : -5.97661711032993750000
hilo 1  suma : -5.85866146788184665000
hilo 2  suma : -6.33123460428985825000
hilo 3  suma : -6.32394151353804060000
suma en el cliente :-24.490454696039683000000
```

Figure 9: Cliente recibe n = 10^5

```
entro a clienteRecibe
CLINTE50 El mensaje::envio 0 49999 250000 0 13 +1,3;-4,2;+4,1;-10,0
la funcion :+1,3;-4,2;+4,1;-10,0
 a :0
 b:13
n:250000
minimo:0maximo:49999
prefijos:[Ljava.math.BigDecimal;@4d73e506
Hilo 0:prefijo inicio: 0 prefijo final :12499
hilo 0 : Thread-5
Hilo 1:prefijo inicio: 12499 prefijo final :24998
hilo 1 : Thread-6
Hilo 2:prefijo inicio: 24998 prefijo final :37497
hilo 2 : Thread-7
Hilo 3:prefijo inicio: 37497 prefijo final :49996
hilo 3 : Thread-8
hilo 0  suma : -5.976570905302790000000000
hilo 1  suma : -5.858725518510684185600000
hilo 2  suma : -6.331357918434320928000000
hilo 3  suma : -6.323687633510328598400000
suma en el cliente :-24.490341975758123712000000
```

Figure 10: Cliente recibe n= $2,5 * 10^5$

```
TCP ClientC: Conectando...
TCP ClientC: Sent.
TCP ClientC: Done.
entrada in:
Cliente bandera 01
entro a clienteRecibe
CLINTE50 El mensaje::envio 200000 299999 500000 0 13 +1,3;-4,2;+4,1;-10,0
la funcion :+1,3;-4,2;+4,1;-10,0
 a :0
 b:13
n:500000
minimo:200000maximo:299999
prefijos:[Ljava.math.BigDecimal;@678c037b
Hilo 0:prefijo inicio: 200000 prefijo final :224999
hilo 0 : Thread-1
Hilo 1:prefijo inicio: 224999 prefijo final :249998
hilo 1 : Thread-2
Hilo 2:prefijo inicio: 249998 prefijo final :274997
hilo 2 : Thread-3
Hilo 3:prefijo inicio: 274997 prefijo final :299996
hilo 3 : Thread-4
hilo 0  suma : 38.410874869629242500000000
hilo 1  suma : 63.792830163036583226800000
hilo 2  suma : 97.151939233599371134000000
hilo 3  suma : 139.559111061958517675200000
suma en el cliente :338.914755328223714536000000
```

Figure 11: Cliente recibe n= $5 * 10^5$

```
entro a clienteRecibe
CLINTE50 El mensaje::envio 600000 799999 1000000 0 13 +1,3;-4,2;+4,1;-10,0
la funcion :+1,3;-4,2;+4,1;-10,0
 a :0
 b:13
n:1000000
minimo:600000maximo:799999
prefijos:[Ljava.math.BigDecimal;@6dd0f5d6
Hilo 0:prefijo inicio: 600000 prefijo final :649999
hilo 0 : Thread-21
Hilo 1:prefijo inicio: 649999 prefijo final :699998
hilo 1 : Thread-22
Hilo 2:prefijo inicio: 699998 prefijo final :749997
hilo 2 : Thread-23
Hilo 3:prefijo inicio: 749997 prefijo final :799996
hilo 3 : Thread-24
hilo 0  suma : 192.0951054268938656250000000
hilo 1  suma : 255.8145165735843650908500000
hilo 2  suma : 331.7954113650898992042500000
hilo 3  suma : 421.1087630404457043969000000
suma en el cliente :1200.8137964060138343170000000
```

Figure 12: Cliente recibe n= 10^6

```
CLINTE50 El mensaje::envio 1200000 1799999 3000000 0 13 +1,3;-4,2;+4,1;-10,0
la funcion :+1,3;-4,2;+4,1;-10,0
 a :0
 b:13
n:3000000
minimo:1200000maximo:1799999
prefijos:[Ljava.math.BigDecimal;@968a8dd
Hilo 0:prefijo inicio: 1200000 prefijo final :1349999
hilo 0 : Thread-33
Hilo 1:prefijo inicio: 1349999 prefijo final :1499998
hilo 1 : Thread-34
Hilo 2:prefijo inicio: 1499998 prefijo final :1649997
hilo 2 : Thread-35
Hilo 3:prefijo inicio: 1649997 prefijo final :1799996
hilo 3 : Thread-36
hilo 0  suma : 38.4112373913380808101851851851851851851851851851851764022
hilo 1  suma : 63.7942898342799521792907407407407407407407407407407289880
hilo 2  suma : 97.1550814459275327946018518518518518518518518518518367939
hilo 3  suma : 139.564628305673627338774074074074074074074074074074055023
suma en el cliente :338.925236977219193122851851851851851851851851851851851797207
```

Figure 13: Cliente recibe n= $3 * 10^6$

```
CLINTE50 El mensaje::envio 2400000 3599999 6000000 0 13 +1,3;-4,2;+4,1;-10,0
la funcion :+1,3;-4,2;+4,1;-10,0
 a :0
 b:13
n:6000000
minimo:2400000maximo:3599999
prefijos:[Ljava.math.BigDecimal;@2f972041
Hilo 0:prefijo inicio: 2400000 prefijo final :2699999
hilo 0 : Thread-17
Hilo 1:prefijo inicio: 2699999 prefijo final :2999998
hilo 1 : Thread-18
Hilo 2:prefijo inicio: 2999998 prefijo final :3299997
hilo 2 : Thread-19
Hilo 3:prefijo inicio: 3299997 prefijo final :3599996
hilo 3 : Thread-20
hilo 0  suma : 38.41127364357931186921296296296296296296296296296296629805596
hilo 1  suma : 63.79443580246061306465671296296296296296296296296296629864443
hilo 2  suma : 97.15539567057042496448726851851851851851851851851851585486531
hilo 3  suma : 139.565180037569448779036574074074074074074074074074074119655
suma en el cliente :338.926285154179798677393518518518518518518518518518518635312
```

Figure 14: Cliente recibe n= $6 * 10^6$

```
CLINTE50 El mensaje::envio 6000000 7999999 10000000 0 13 +1,3;-4,2;+4,1;-10,0
la funcion :+1,3;-4,2;+4,1;-10,0
 a :0
 b:13
n:10000000
minimo:6000000maximo:7999999
prefijos:[Ljava.math.BigDecimal;@968a8dd
Hilo 0:prefijo inicio: 6000000 prefijo final :6499999
hilo 0 : Thread-29
Hilo 1:prefijo inicio: 6499999 prefijo final :6999998
hilo 1 : Thread-30
Hilo 2:prefijo inicio: 6999998 prefijo final :7499997
hilo 2 : Thread-31
Hilo 3:prefijo inicio: 7499997 prefijo final :7999996
hilo 3 : Thread-32
hilo 0  suma : 192.0956269489058136562500000000
hilo 1  suma : 255.8163977506302186894658500000
hilo 2  suma : 331.7991225412790241848292500000
hilo 3  suma : 421.1148323946400825837719000000
suma en el cliente :1200.825979635455139114317000000
```

Figure 15: Cliente recibe n= $10 * 10^6$

```
CLINTE50 El mensaje::envio 8400000 12599999 21000000 0 13 +1,3;-4,2;+4,1;-10,0
la funcion :+1,3;-4,2;+4,1;-10,0
 a :0
 b:13
n:21000000
minimo:8400000maximo:12599999
prefijos:[Ljava.math.BigDecimal;@11cba7e6
Hilo 0:prefijo inicio: 8400000 prefijo final :9449999
hilo 0 : Thread-37
Hilo 1:prefijo inicio: 9449999 prefijo final :10499998
hilo 1 : Thread-38
Hilo 2:prefijo inicio: 10499998 prefijo final :11549997
hilo 2 : Thread-39
Hilo 3:prefijo inicio: 11549997 prefijo final :12599996
hilo 3 : Thread-40
hilo 0  suma : 38.411299538045164914493575207860922146636432350718065047 7999
hilo 1  suma : 63.794540065564386781944806176438829500053989849908217313 8917
hilo 2  suma : 97.155620117123521091318891048482885217579095130115538354 2787
hilo 3  suma : 139.565574132618339372302872260015117157974300831443688681345
suma en el cliente :338.92703385335141216006014469279775402224381816218550939731 6
```

Figure 16: Cliente recibe n= $13 * 10^6$

```
CLINTE50 El mensaje::envio 30000000 39999999 50000000 0 13 +1,3;-4,2;+4,1;-10,0
la funcion :+1,3;-4,2;+4,1;-10,0
 a :0
 b:13
n:50000000
minimo:30000000maximo:39999999
prefijos:[Ljava.math.BigDecimal;@2f972041
Hilo 0:prefijo inicio: 30000000 prefijo final :32499999
hilo 0 : Thread-45
Hilo 1:prefijo inicio: 32499999 prefijo final :34999998
hilo 1 : Thread-46
Hilo 2:prefijo inicio: 34999998 prefijo final :37499997
hilo 2 : Thread-47
Hilo 3:prefijo inicio: 37499997 prefijo final :39999996
hilo 3 : Thread-48
hilo 0  suma : 192.0956733064472325462500000000000
hilo 1  suma : 255.8165649667842087477157268000000
hilo 2  suma : 331.7994524248961609680786340000000
hilo 3  suma : 421.1153718955386033052701752000000
suma en el cliente :1200.827062593666205567314536000000
```

Figure 17: Cliente recibe n= $50 * 10^6$

```
CLINTE50 El mensaje::envio 45200000 67799999 113000000 0 13 +1,3;-4,2;+4,1;-10,0
la funcion :+1,3;-4,2;+4,1;-10,0
 a :0
 b:13
n:113000000
minimo:45200000maximo:67799999
prefijos:[Ljava.math.BigDecimal;@1c662ae7
Hilo 0:prefijo inicio: 45200000 prefijo final :50849999
hilo 0 : Thread-53
Hilo 1:prefijo inicio: 50849999 prefijo final :56499998
hilo 1 : Thread-54
Hilo 2:prefijo inicio: 56499998 prefijo final :62149997
hilo 2 : Thread-55
Hilo 3:prefijo inicio: 62149997 prefijo final :67799996
hilo 3 : Thread-56
hilo 0  suma : 38.411307970934568177666614456887775080272535045814080677865 7
hilo 1  suma : 63.794574020295440355888251794826657758661914190687207350284 3
hilo 2  suma : 97.155693211290512658425517725797475495478887266381452910420 3
hilo 3  suma : 139.565702475227407862702186573261986129294052174202315949778
suma en el cliente :338.92727767774792905468257055077389446370738867708505688834 8
```

Figure 18: Cliente recibe n= $113 * 10^6$

n(10^5)	Tiempo(s)
1	1,358
2.5	1,480
5	2,089
10	1,167
30	6,671
60	11,559
100	6,226
210	38,928
500	26,683
690	123,99
1130	201,100

Figure 19: Tabla de n vs tiempo

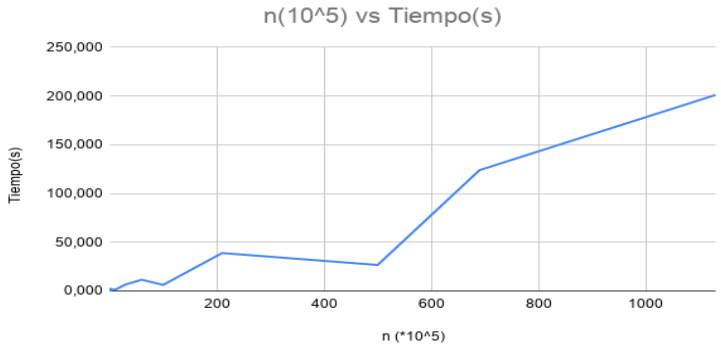

Figure 20: n vs tiempo

6 Conclusiones

- En el primer experimento se evaluó el desempeño de la arquitectura modelo servidor aumentando el número de clientes en el cálculo de una integral definida y se evidenció que a mayor número de clientes

el tiempo decrementaba y era eficiente con 6 clientes.

- Para el segundo experimento se evidenció que el aumento de las particiones aumentaba el tiempo de cálculo pero el respuesta de la integral era más precisa.

7 Anexo Código

`https://github.com/renzoqamao/CC4P1-ProgramacionConcurrente-Distribuida`

8 Anexo Documentación

- DONAHOO, Michael. TCP/IP Sockets in C: Practical guide for programmers. Estados Unidos: Morgan Kaufmann. 2009
- `http://www.mitecnologico.com/Main/ComunicacionClienteServidorSockets`
