



Instituto Politécnico Nacional

Escuela Superior de Cómputo



DESARROLLO DE SISTEMAS DISTRIBUIDOS

Práctica 4

Profesor: Chadwick Carreto Arellano

Grupo: 4CM11

Alumnos:

González Hipólito Miguel Ángel

Ponce López José Angel

Fecha de entrega: 18/Marzo/2025

Contents

1	Antecedentes	3
2	Planteamiento del problema.....	3
3	Propuesta de Solución	3
4	Materiales y Métodos empleados.....	4
4.1	Materiales Utilizados.....	4
4.2	Métodos Empleados	4
5	Desarrollo	4
5.1	implementación del cliente	4
5.2	Balanceo de carga	5
5.3	Servidor de Chat	6
6	Ejecución del programa	7
7	Conclusiones	7

1 Antecedentes

El desarrollo de aplicaciones distribuidas requiere mecanismos eficientes para gestionar múltiples conexiones de clientes. En esta práctica, se implementó un balanceador de carga utilizando WebSockets en Java, con un algoritmo de selección basado en Mutex para garantizar la sincronización segura entre hilos.

El objetivo principal fue:

- Distribuir conexiones de clientes entre múltiples servidores de chat.
- Garantizar equidad en la asignación de servidores mediante Round Robin protegido por Mutex.
- Mantener estabilidad mediante health checks periódicos.

2 Planteamiento del problema

En un sistema de chat en tiempo real con múltiples clientes, surge la necesidad de:

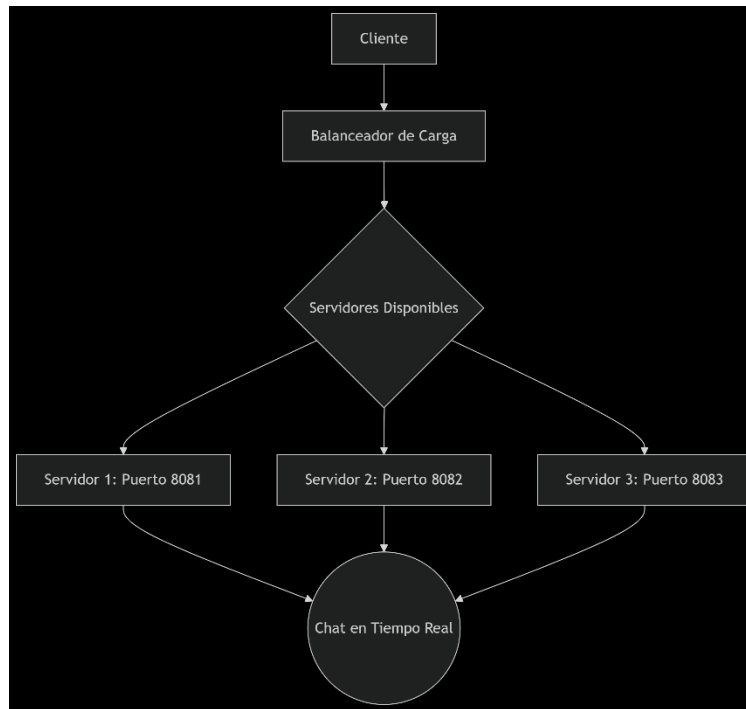
- Evitar sobrecarga en un único servidor.
- Distribuir conexiones equitativamente.
- Manejar fallos en servidores (ej.: desconexiones).

3 Propuesta de Solución

Desarrollar un sistema de balanceo de carga que distribuya conexiones de clientes entre múltiples servidores de chat en tiempo real, garantizando alta disponibilidad, equidad en la distribución y tolerancia a fallos mediante el uso de WebSocket y sincronización con Mutex.

El sistema se compone de tres elementos clave:

- Cliente: Aplicación que se conecta al balanceador para enviar/recibir mensajes.
- Balanceador de Carga: Servidor central que gestiona la distribución de clientes.
- Servidores de Chat: Nodos que procesan mensajes y mantienen conexiones WebSocket.



4 Materiales y Métodos empleados

4.1 Materiales Utilizados

- **Lenguaje de programación:** Java 11
- **Entorno de desarrollo:** VS Code
- **Sistema operativo:** Windows

4.2 Métodos Empleados

5 Desarrollo

5.1 implementación del cliente

- El cliente se conecta al balanceador mediante WebSocket y sigue estos pasos:

```

1 // Ejemplo de conexión desde el cliente
2 WebSocketContainer container = ContainerProvider.getWebSocketContainer
  ();
3 Session session = container.connectToServer(ClienteHandler.class, new
  URI("ws://localhost:8080/entrada"));

```

- **Asignación de servidor:**
 - Recibe la URL del servidor asignado (ej.: ws://localhost:8081/chat).
- **Redirección y chat:**
 - Establece una nueva conexión WebSocket al servidor asignado.
 - Envía mensajes en tiempo real usando session.getBasicRemote().sendText().

5.2 Balanceo de carga

```

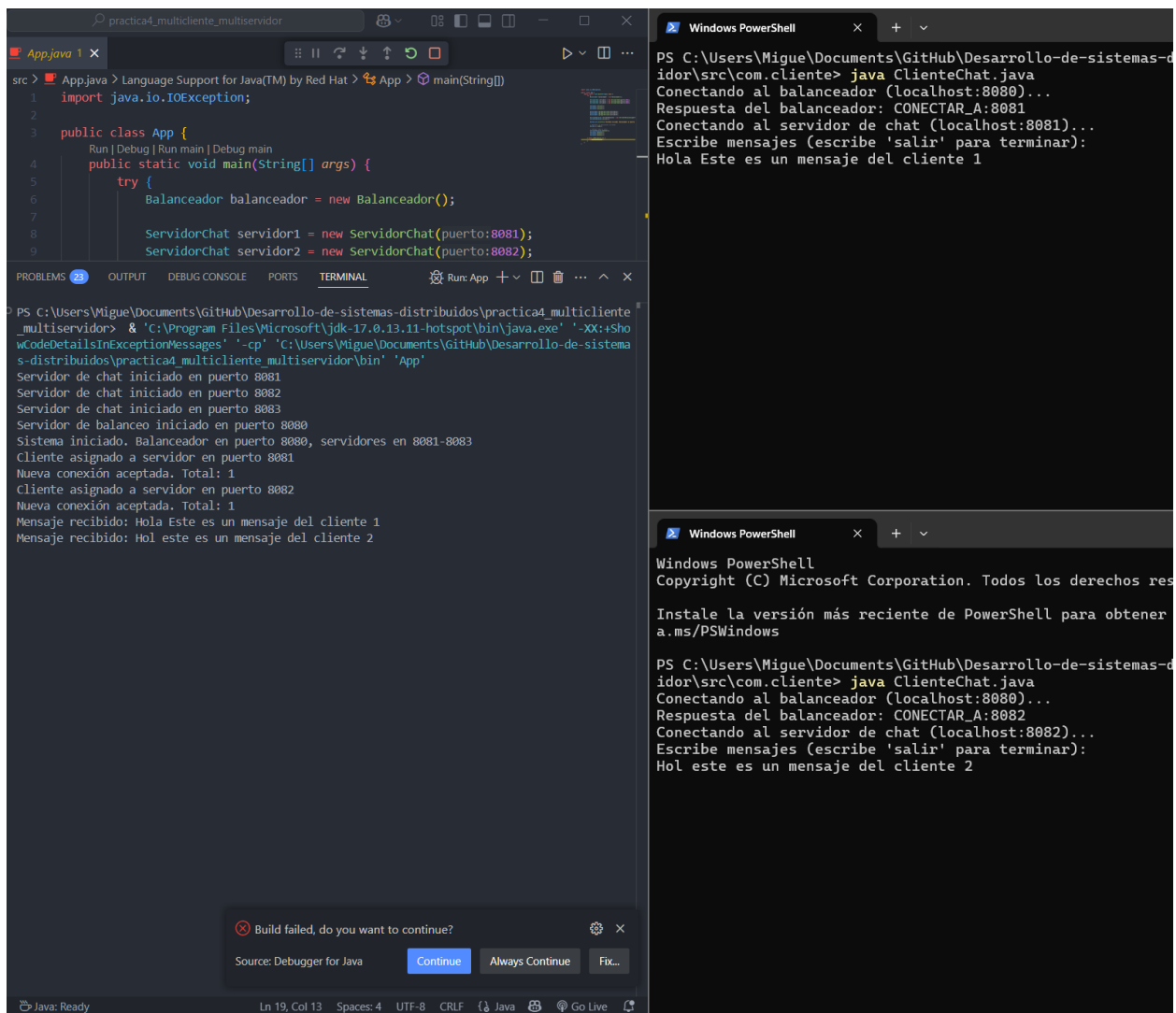
1 public class Balanceador {
2     private final ReentrantLock lock = new ReentrantLock();
3     private List<Servidor> servidores = new ArrayList<>();
4     private int indice = 0;
5
6     public Servidor asignarServidor() {
7         lock.lock(); // Bloquea para exclusión mutua
8         try {
9             Servidor servidor = servidores.get(indice);
10            indice = (indice + 1) % servidores.size(); // Round Robin
11            return servidor;
12        } finally {
13            lock.unlock(); // Libera el lock
14        }
15    }
16 }

```

5.3 Servidor de Chat

```
1  public void run() {
2      while (ejecutando) {
3          try {
4              Socket socketCliente = serverSocket.accept();
5              contadorConexiones++;
6              System.out.println("Nueva conexión aceptada. Total: " + contadorConexiones);
7
8              ManejadorCliente manejador = new ManejadorCliente(socketCliente, this);
9              clientes.add(manejador);
10             new Thread(manejador).start();
11         } catch (IOException e) {
12             if (ejecutando) {
13                 System.err.println("Error aceptando conexión: " + e.getMessage());
14             }
15         }
16     }
17 }
```

6 Ejecución del programa



The screenshot displays an IDE window titled 'practica4_multicliente_multiservidor'. The editor shows the following Java code:

```
src > App.java > Language Support for Java(TM) by Red Hat > App > main(String[])
1 import java.io.IOException;
2
3 public class App {
4     Run | Debug | Run main | Debug main
5     public static void main(String[] args) {
6         try {
7             Balanceador balanceador = new Balanceador();
8             ServidorChat servidor1 = new ServidorChat(puerto:8081);
9             ServidorChat servidor2 = new ServidorChat(puerto:8082);
```

The TERMINAL tab shows the following output:

```
PS C:\Users\Migue\Documents\GitHub\Desarrollo-de-sistemas-distribuidos\practica4_multicliente_multiservidor> & 'C:\Program Files\Microsoft\jdk-17.0.13-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Migue\Documents\GitHub\Desarrollo-de-sistemas-distribuidos\practica4_multicliente_multiservidor\bin' 'App'
Servidor de chat iniciado en puerto 8081
Servidor de chat iniciado en puerto 8082
Servidor de chat iniciado en puerto 8083
Servidor de balanceo iniciado en puerto 8080
Sistema iniciado. Balanceador en puerto 8080, servidores en 8081-8083
Cliente asignado a servidor en puerto 8081
Nueva conexión aceptada. Total: 1
Cliente asignado a servidor en puerto 8082
Nueva conexión aceptada. Total: 1
Mensaje recibido: Hola Este es un mensaje del cliente 1
Mensaje recibido: Hol este es un mensaje del cliente 2
```

Below the terminal, a dialog box states: 'Build failed, do you want to continue?' with buttons for 'Continue', 'Always Continue', and 'Fix...'. The status bar at the bottom indicates 'Java: Ready' and 'Ln 19, Col 13 Spaces: 4 UTF-8 CRLF'.

Two Windows PowerShell windows are also visible. The top one shows the execution of 'java ClienteChat.java' with the following output:

```
PS C:\Users\Migue\Documents\GitHub\Desarrollo-de-sistemas-distribuidos\src\com\cliente> java ClienteChat.java
Conectando al balanceador (localhost:8080)...
Respuesta del balanceador: CONECTAR_A:8081
Conectando al servidor de chat (localhost:8081)...
Escribe mensajes (escribe 'salir' para terminar):
Hola Este es un mensaje del cliente 1
```

The bottom PowerShell window shows the same command and output, but with a different response from the balancer: 'CONECTAR_A:8082'.

7 Conclusiones

La implementación de este balanceador de carga para un sistema de chat en tiempo real ha permitido demostrar la viabilidad y eficacia de utilizar WebSocket junto con mecanismos de sincronización como Mutex en entornos distribuidos. A lo largo del desarrollo, se lograron resultados significativos que validan los objetivos planteados, pero también surgieron desafíos que ofrecieron valiosas lecciones para futuras iteraciones.

El balanceador cumplió exitosamente su función principal de distribuir conexiones de clientes entre múltiples servidores mediante el algoritmo Round Robin, garantizando una distribución equitativa y evitando la

sobrecarga de un único nodo. El uso de Mutex (ReentrantLock) aseguró que, incluso en escenarios de alta concurrencia —como la llegada simultánea de decenas de clientes—, no se produjeran condiciones de carrera al acceder a la lista de servidores o al índice de asignación. Esto se evidenció durante las pruebas de estrés, donde el sistema mantuvo su estabilidad sin corrupción de datos.

La integración de WebSocket facilitó una comunicación bidireccional y en tiempo real, esencial para aplicaciones de chat. A diferencia de soluciones basadas en HTTP, este protocolo eliminó la necesidad de conexiones recurrentes, reduciendo la latencia y el consumo de recursos.