Comparación de programas (Sockets y Threads)

Jose Eduardo Valeriano, Francisco Guzmán, Tomas Ramírez, Santiago Pineda

Pontificia Universidad Javeriana

***Resumen*— Este informe presenta el desarrollo y análisis de dos conjuntos de programas escritos en Java, orientados a la comprensión de conceptos fundamentales de comunicación en red y programación concurrente. El primer conjunto implementa comunicación entre cliente y servidor mediante sockets utilizando los protocolos TCP y UDP, mientras que el segundo conjunto simula la atención de clientes en una tienda mediante hilos, comparando la ejecución secuencial con la ejecución concurrente a través de las interfaces Thread y Runnable. A través de diversos experimentos, se evidenció que TCP ofrece una comunicación confiable, aunque más lenta, mientras que UDP permite una transmisión más rápida, pero sin garantías de entrega. Por su parte, el uso de hilos permitió reducir significativamente los tiempos de procesamiento al ejecutar tareas en paralelo. Los resultados obtenidos demuestran la relevancia de aplicar correctamente estos conceptos para diseñar sistemas eficientes, escalables y tolerantes a fallos.**

***Abstract*--This report presents the development and analysis of two sets of programs written in Java, aimed at understanding fundamental concepts of network communication and concurrent programming. The first set implements client-server communication through sockets using the TCP and UDP protocols, while the second set simulates customer service in a store using threads, comparing sequential execution with concurrent execution through the Thread and Runnable interfaces. Through various experiments, it was shown that TCP offers reliable but slower communication, while UDP provides faster transmission without delivery guarantees. On the other hand, using threads significantly reduced processing times by executing tasks in parallel. The results demonstrate the importance of properly applying these concepts to design efficient, scalable, and fault-tolerant systems.**

I. Introducción

La creciente demanda de aplicaciones distribuidas y de alto rendimiento ha impulsado el estudio y la implementación de técnicas avanzadas de comunicación en red y programación concurrente. En este contexto, el lenguaje Java ofrece herramientas integradas que permiten abordar ambos conceptos de manera didáctica y eficiente, como el uso de sockets para la transmisión de datos y de hilos (threads) para la ejecución concurrente de tareas.

El presente trabajo tiene como propósito analizar y comparar el comportamiento de dos conjuntos de programas. El primero implementa comunicación cliente-servidor mediante sockets utilizando los protocolos TCP y UDP, evaluando sus diferencias en cuanto a velocidad, fiabilidad y control de conexión. El segundo conjunto aborda la concurrencia mediante la simulación de un proceso de atención a clientes, comparando la ejecución secuencial con la ejecución concurrente a través de las interfaces Thread y Runnable.

Este tipo de ejercicios resulta fundamental en el ámbito académico porque permiten comprender no solo el funcionamiento técnico de estas herramientas, sino también su impacto en el diseño y rendimiento de sistemas distribuidos. Además, constituyen una base sólida para el desarrollo de aplicaciones escalables, tolerantes a fallos y capaces de aprovechar eficientemente los recursos de hardware disponibles.

II. Objetivos

A. Objetivo General

Analizar y comparar el funcionamiento de programas en Java que utilizan comunicación mediante sockets (TCP y UDP) y concurrencia con hilos, evaluando su impacto en fiabilidad, diseño y tiempos de ejecución.

B. Objetivos específicos

- Describir el rol de los programas cliente y servidor en TCP y UDP.
- Comparar características de los protocolos TCP y UDP en cuanto a conexión, fiabilidad y sobrecarga.
- Evaluar diferencias de rendimiento entre ejecución secuencial y concurrente (Thread vs Runnable).
- Medir tiempos de ejecución obtenidos en pruebas experimentales.
- Documentar evidencias mediante capturas y análisis de resultados.

III. Descripción de los .zip

A. Proyecto Sockets

Implementa comunicación cliente-servidor en Java mediante la API de sockets, utilizando los protocolos TCP y UDP. Está compuesto por cuatro archivos: sockettcpcli.java y sockettcpser.java (cliente y servidor TCP), y socketudpcli.java y socketudpser.java (cliente y servidor UDP).

Los programas TCP emplean ServerSocket y Socket para establecer una conexión confiable y persistente, usando flujos de entrada/salida para el intercambio ordenado de datos. Los programas UDP, en cambio, usan DatagramSocket y DatagramPacket para enviar y recibir mensajes sin conexión, lo que reduce la latencia pero sin garantizar la entrega ni el orden.

Este conjunto permite comparar el funcionamiento de ambos protocolos, evidenciando las diferencias en establecimiento de conexión, fiabilidad y rendimiento.

B. *Proyecto Thereads*

Es un escenario de atención de clientes para analizar la ejecución secuencial y concurrente en Java mediante el uso de hilos. Está compuesto por seis archivos: Cliente.java, Cajera.java, CajeraThread.java, Main.java, MainThread.java y MainRunnable.java.

La versión secuencial, implementada en Main.java junto con Cajera.java, procesa los clientes uno por uno en un único hilo, acumulando el tiempo total de atención. Las versiones concurrentes (MainThread.java con CajeraThread.java, y MainRunnable.java con Runnable) crean múltiples hilos para atender a los clientes de forma paralela, reduciendo significativamente el tiempo de ejecución total.

Este conjunto permite comparar de forma práctica el impacto del uso de hilos (Thread y Runnable) en el rendimiento, evidenciando las ventajas de la concurrencia frente a la ejecución secuencial.

IV. Comparaciones

A. *Socket*

Implementan comunicación en red con Transmission Control Protocol (TCP) y User Datagram Protocol (UDP).

Archivo	Función principal	Tipo de programa	Aspectos técnicos clave
sockettcpser.java	Servidor TCP	Servidor	Usa ServerSocket para aceptar conexiones y Socket para flujos de entrada/salidas confiables.
sockettcpcli.java	Cliente TCP	Cliente	Usa Socket para conectarse al servidor, enviar y recibir datos de forma ordenada.
socketudpser.java	Servidor UDP	Servidor	Usa DatagramSocket para recibir y responder paquetes sin conexión establecida.
socketudpcli.java	Cliente UDP	Cliente	Usa DatagramPacket y DatagramSocket para enviar mensajes sin establecer conexión.

Fig. 1. tabla TCP y UDP

todos los archivos manejan comunicación en red. Los clientes envían mensajes y los servidores los reciben. La diferencia principal es el protocolo de transporte: TCP es confiable y orientado a conexión; UDP es más rápido, pero sin fiabilidad garantizada.

B. *Thereads*

Compara ejecución secuencial y concurrente usando hilos (Thread y Runnable). Las tablas muestran claramente estas diferencias en sus archivos y funciones.

Archivo	Función principal	Tipo de programa	Aspectos técnicos clave
Cliente.java	Define datos de un cliente (nombre, lista de compras, tiempos).	Clase de datos	Solo contiene atributos y métodos get, set.
Cajera.java	Procesa a un cliente de forma secuencial.	Lógica secuencial	Simula el tiempo de atención con Thread.sleep().
CajeraThread.java	Procesa a un cliente en un hilo independiente.	Subclase de Thread	Ejecuta la misma lógica de Cajera pero en paralelo.
Main.java	Ejecuta varias cajeras de forma secuencial.	Programa principal	Usa new Cajera() sin hilos.
MainThread.java	Ejecuta varias cajeras en paralelo con Thread.	Programa principal	Crea múltiples CajeraThread y hace start () en cada una.
MainRunnable.java	Ejecuta varias cajeras en paralelo con Runnable.	Programa principal	Implementa Runnable y pasa los objetos a instancias Thread.

Fig. 2. tabla Thereads - Runnable

Todos los archivos trabajan en simulación de procesos concurrentes locales, comparando ejecución secuencial (un solo flujo) vs paralela (varios hilos al mismo tiempo) usando Thread y Runnable.

V. Evidencia de ejecución

En esta sección se presentan las capturas de pantalla obtenidas durante la ejecución de los programas de ambos proyectos. Las imágenes muestran el funcionamiento correcto de los clientes y servidores en red, así como la diferencia de tiempos entre la ejecución secuencial y concurrente con hilos, validando el comportamiento esperado de cada implementación.

A. *Ejecución de programas de concurrencia*

```
~$ java threadsJarroba.Main
Picked up _JAVA_OPTIONS: -Xms64m
La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg
Procesado el producto 1 ->Tiempo: 2seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 3 ->Tiempo: 5seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 5 ->Tiempo: 12seg
Procesado el producto 6 ->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 15seg
Procesado el producto 1 ->Tiempo: 16seg
Procesado el producto 2 ->Tiempo: 19seg
Procesado el producto 3 ->Tiempo: 24seg
Procesado el producto 4 ->Tiempo: 25seg
Procesado el producto 5 ->Tiempo: 26seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 26seg
```

Fig. 3. Ejecución del programa Main (secuencial)

La consola muestra cómo una única cajera atiende a todos los clientes de forma consecutiva. Cada cliente inicia su atención solo cuando el anterior ha terminado. El tiempo total de procesamiento corresponde a la suma de los tiempos individuales.

```
~$ java threadsJarroba.MainThread
Picked up _JAVA_OPTIONS: -Xms64m
La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 0seg
Procesado el producto 1 del cliente Cliente 2->Tiempo: 1seg
Procesado el producto 1 del cliente Cliente 1->Tiempo: 2seg
Procesado el producto 2 del cliente Cliente 1->Tiempo: 4seg
Procesado el producto 2 del cliente Cliente 2->Tiempo: 4seg
Procesado el producto 3 del cliente Cliente 1->Tiempo: 5seg
Procesado el producto 3 del cliente Cliente 2->Tiempo: 9seg
Procesado el producto 4 del cliente Cliente 1->Tiempo: 10seg
Procesado el producto 4 del cliente Cliente 2->Tiempo: 10seg
Procesado el producto 5 del cliente Cliente 2->Tiempo: 11seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 11seg
Procesado el producto 5 del cliente Cliente 1->Tiempo: 12seg
Procesado el producto 6 del cliente Cliente 1->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
```

Fig. 4. Ejecución del programa MainTheread

Cada cajera se ejecuta en un hilo independiente creado a partir de la clase CajeraThread. Los mensajes en consola aparecen intercalados, indicando que los hilos están trabajando en paralelo. El tiempo total es mucho menor que en la versión secuencial.

```
~$ java threadsJarroba.MainRunnable
Picked up _JAVA_OPTIONS: -Xms64m
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 0seg
La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg
Procesado el producto 1 ->Tiempo: 1seg
Procesado el producto 1 ->Tiempo: 2seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 3 ->Tiempo: 5seg
Procesado el producto 3 ->Tiempo: 9seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 5 ->Tiempo: 11seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 11seg
Procesado el producto 5 ->Tiempo: 12seg
Procesado el producto 6 ->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
```

Fig. 5. Ejecución del programa MainRunnable

Similar a MainThread, pero cada hilo se crea a partir de un objeto que implementa la interfaz Runnable. La salida es concurrente, demostrando que ambas formas de crear hilos en Java son equivalentes en comportamiento y rendimiento.

B. Ejecución de programas con Sockets

```
ramir@TomasRamirez:~/socket Mailbox$ java socketudpcli 172.25.231.136
Prueba de sockets UDP (cliente)
Creando socket... ok
Capturando direccion de host... ok
Introduce mensajes a enviar:
Buenas tardes clase!
Introduccion a sistemas distribuidos.
fin
ramir@TomasRamirez:~/socket Mailbox$
```

```
ramir@TomasRamirez:~/socket Mailbox$ java socketudpser
Prueba de sockets UDP (servidor)
Creando socket... ok
Recibiendo mensajes...
Buenas tardes clase!
Introduccion a sistemas distribuidos.
fin
ramir@TomasRamirez:~/socket Mailbox$
```

Fig. 6. Comunicación UDP entre cliente servidor

Se observa que el servidor inicia primero y queda en espera (accept ()). El cliente se conecta y envía un mensaje, el cual llega de forma ordenada y es confirmado por el servidor. El canal permanece abierto hasta que el cliente o el servidor cierran la conexión.

```
ramir@TomasRamirez:~/socket Mailbox$ java sockettcpser
Prueba de sockets TCP (servidor)
Buenas tardes clase!
Introduccion a sistemas distribuidos
fin
null
ramir@TomasRamirez:~/socket Mailbox$
```

```
ramir@TomasRamirez:~/socket Mailbox$ java sockettcpcli 172.25.231.136
Prueba de sockets TCP (cliente)
Capturando direccion de host... ok
Creando socket... ok
Introduce mensajes a enviar:
Buenas tardes clase!
Introduccion a sistemas distribuidos
fin
ramir@TomasRamirez:~/socket Mailbox$
```

Fig. 7. Comunicación TCP entre cliente servidor

Aquí el cliente envía un datagrama sin establecer conexión previa. El servidor recibe el mensaje de inmediato y responde. No existe canal persistente; cada mensaje es independiente, lo que permite baja latencia, pero sin garantías de entrega.

VI. Conclusiones

1. TCP ofrece comunicación confiable y ordenada, aunque con mayor sobrecarga.
2. UDP brinda rapidez, pero sin garantías, útil para transmisión en tiempo real.
3. La concurrencia con hilos reduce drásticamente los tiempos frente a la ejecución secuencial.
4. La diferencia entre Thread y Runnable radica en el diseño, no en el rendimiento.
5. La práctica demuestra la relevancia de estos conceptos para sistemas distribuidos y concurrentes.

VII. Referencias

[1] A. S. Tanenbaum, Redes de Computadoras.
[2] IEEE, Template for Conference Papers.