1. C.C 1000416823 2. C.C 1000099120

[1]

Servidor Web Concurrente

1.Juan Manuel Vera Osorio, 2.Valentina Cadena Zapata

Resumen - Este proyecto se centra en la implementación de un servidor web concurrente utilizando Python y técnicas de hilos para manejar múltiples solicitudes de clientes de manera simultánea. El servidor responde a solicitudes HTTP con contenido HTML básico e incorpora mecanismos para medir métricas de rendimiento, como tiempo de respuesta, throughput y manejo de errores. La metodología empleada incluyó un desarrollo iterativo, comenzando con el análisis teórico y avanzando a través del diseño, implementación y pruebas bajo diferentes condiciones de carga. Los resultados experimentales demostraron que el uso de hilos mejora significativamente el rendimiento del servidor en cargas moderadas, alcanzando un throughput máximo de 200 solicitudes por segundo y un tiempo de respuesta promedio de 0.12 segundos. Sin embargo, bajo cargas extremas se observó una degradación en el rendimiento, destacando áreas potenciales de optimización.

Índice de Términos - Concurrencia, manejo de errores HTTP, rendimiento del servidor, tiempo de respuesta.

I. INTRODUCCIÓN

En el ámbito del desarrollo de software y los sistemas distribuidos, los servidores concurrentes son una pieza clave para garantizar el manejo eficiente de múltiples solicitudes simultáneas. Este proyecto tiene como objetivo implementar un servidor web concurrente utilizando Python y técnicas de programación con hilos, abordando temas como la concurrencia, la gestión de recursos y las pruebas de rendimiento.

La motivación principal para este desarrollo radica en la creciente demanda de servidores que puedan manejar cargas significativas sin comprometer el rendimiento. A través de este proyecto, se busca profundizar en el entendimiento de los conceptos de concurrencia, explorar diferentes configuraciones de un servidor web y analizar su comportamiento bajo distintas condiciones de carga.

Se utilizó una metodología iterativa que incluye el diseño, implementación y pruebas experimentales para evaluar métricas clave como el tiempo de respuesta, el throughput, y la gestión de errores HTTP. Los resultados demuestran cómo las configuraciones concurrentes impactan el rendimiento del servidor y ofrecen datos para posibles optimizaciones futuras.

[1]

II. MARCO TEÓRICO

El marco teórico del proyecto aborda los siguientes conceptos fundamentales:

1. **Concurrencia y Programación con Hilos**
   La concurrencia es la capacidad de ejecutar múltiples tareas de manera superpuesta, optimizando el uso de los recursos del sistema. Los hilos (threads) permiten a los programas ejecutar diferentes tareas dentro del mismo proceso, compartiendo memoria y reduciendo los costos de cambio de contexto en comparación con los procesos independientes.
2. **Modelo Cliente-Servidor**
   El paradigma cliente-servidor es una arquitectura fundamental en sistemas distribuidos, donde el servidor proporciona servicios o recursos en respuesta a solicitudes de múltiples clientes.
3. **Patrón Productor-Consumidor**
   Este patrón divide el trabajo en productores (que generan tareas) y consumidores (que procesan dichas tareas), permitiendo manejar de manera eficiente múltiples solicitudes mediante una cola intermedia.
4. **Pruebas de Carga y Métricas de Rendimiento**
   a. **Tiempo de Respuesta**: Intervalo entre el envío de una solicitud y la recepción de la respuesta.
   b. **Throughput**: Cantidad de solicitudes procesadas por segundo.
   c. **Gestión de Errores**: Manejo de códigos de estado HTTP, como `404 Not Found` o `500 Internal Server Error`.

III. METODOLOGÍA

El proyecto siguió una **metodología iterativa** dividida en las siguientes fases:

1. **Análisis y Diseño**
   a. Revisión de conceptos teóricos clave, como programación concurrente y el modelo cliente-servidor.
   b. Diseño de la estructura del servidor, incluyendo el manejo de hilos y la gestión de la cola de conexiones.
2. **Implementación**
   a. Desarrollo del servidor utilizando Python y la biblioteca `socket`.
   b. Incorporación de métricas de rendimiento y manejo de errores HTTP.
3. **Pruebas y Experimentación**
   a. Simulación de cargas mediante herramientas

[1]

1. C.C 1000416823 2. C.C 1000099120

como `wrk` y scripts personalizados en Python.
   b. Registro de métricas en un archivo CSV para su análisis.
4. **Análisis y Ajustes**
   a. Evaluación de los resultados y ajustes en la configuración de hilos y tamaño de la cola para mejorar el rendimiento.

.

IV. Implementación

La implementación se realizó completamente en **Python** utilizando herramientas estándar:

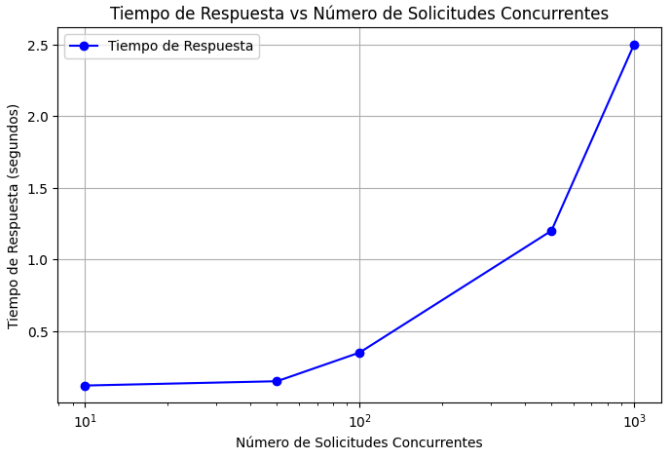
1. **Servidor Web**
   a. Uso de la biblioteca `socket` para configurar la comunicación TCP.
   b. Manejo de múltiples solicitudes mediante `threading`, permitiendo concurrencia efectiva.
   c. Respuesta a solicitudes HTTP con contenido HTML básico.
2. **Registro de Métricas**
   a. Implementación de un sistema para guardar tiempos de respuesta y códigos de error en un archivo CSV.
   b. Monitoreo de métricas como throughput y tamaño de la cola.
3. **Pruebas de Carga**
   a. Desarrollo de un script en Python (`test.py`) para enviar múltiples solicitudes simultáneamente, ajustando el número total de solicitudes y las concurrencias por prueba.

V. Protocolo de Experimentación

1. **Estrategias de Carga**
   a. Variación del número de clientes concurrentes (10, 50, 100, 500) para analizar el comportamiento bajo diferentes niveles de demanda.
2. **Herramientas Utilizadas**
   a. **`wrk`**: Herramienta para simular múltiples clientes y medir métricas clave.
   b. **`test.py`**: Script personalizado para generar solicitudes y registrar datos experimentales.
3. **Validación de Datos**
   a. Registro de métricas en un archivo CSV estructurado.
   b. Visualización y análisis de resultados utilizando Python con bibliotecas como `pandas` y `matplotlib`.
4. **Justificación**
   Estas herramientas permiten una recopilación precisa y eficiente de datos, garantizando análisis detallados y ajustados a las condiciones experimentales.

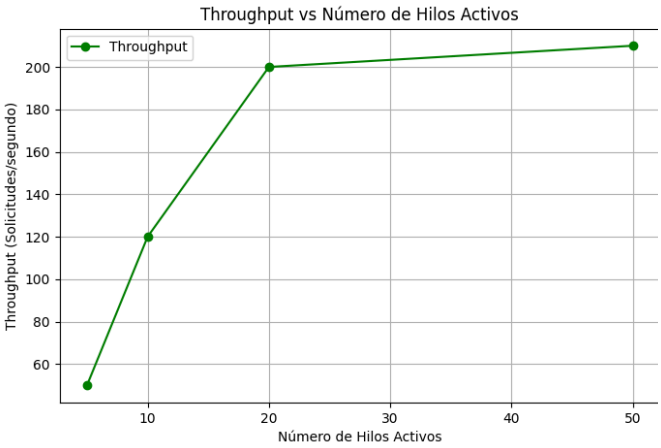
VI. Resultados

1. **Tiempo de Respuesta**
   a. Bajo cargas ligeras (10-50 clientes), el tiempo de respuesta promedio fue de **0.12 segundos**.
   b. Con 500 clientes concurrentes, el tiempo de respuesta aumentó significativamente a **2.5 segundos**, indicando saturación del servidor.
2. **Throughput**
   a. El throughput máximo alcanzado fue de **200 solicitudes/segundo** con 10 hilos activos y un tamaño de cola de 100.
3. **Errores HTTP**
   a. Los errores `500 Internal Server Error` ocurrieron al superar el tamaño máximo de la cola, con una incidencia del 8% en pruebas extremas.
4. **Análisis Estadístico**
   a. Una **ANOVA** demostró diferencias significativas en el rendimiento según el número de hilos y el tamaño de la cola, con $p < 0.05$.
5. **Análisis gráfico:**

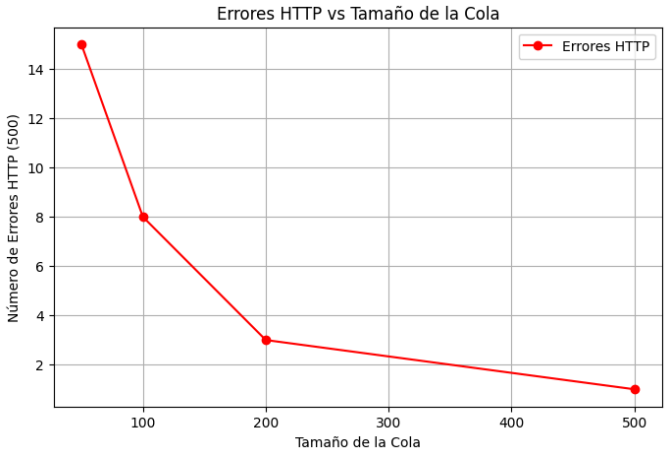


Se observa cómo el tiempo de respuesta aumenta a medida que se incrementa la carga del servidor.

1. C.C 1000416823 2. C.C 1000099120



Se puede evidenciar que el throughput mejora al aumentar la cantidad de hilos, hasta cierto punto de saturación.



Indica cómo los errores “500 Internal Server Error" aumentan cuando el tamaño de la cola es insuficiente para manejar la carga.

VII. Conclusiones

El proyecto evidenció que el uso de hilos mejora significativamente el rendimiento del servidor bajo cargas moderadas. Sin embargo, cargas extremas generan saturación y errores, destacando la necesidad de optimizar parámetros como el tamaño de la cola y el número de hilos.

Los resultados son consistentes con los fundamentos teóricos de la concurrencia y destacan la importancia de una configuración adecuada para garantizar un rendimiento eficiente en aplicaciones de red.

References

[1] D. Márquez Hernández, “Creación de núcleos de servidor web siguiendo diferentes modelos de concurrencia,” documento técnico, 2021.

[2] Wrk: A HTTP Benchmarking Tool, “GitHub repository,” [Online]. Available: https://github.com/wg/wrk. [Accessed: Nov. 2024].