Proyecto Final

Curso de Sistemas Operativos y Laboratorio

Título del proyecto

Desarrollo y prueba de rendimiento para un servidor web concurrente

Miembros del equipo

- Valentina Cadena Zapata
- Juan Manuel Vera Osorio

Resumen

Este proyecto tiene como objetivo diseñar e implementar un servidor web concurrente utilizando Python y técnicas de concurrencia mediante hilos. El servidor será capaz de gestionar múltiples solicitudes simultáneas, demostrando cómo los sistemas operativos administran recursos y procesos en aplicaciones concurrentes. La implementación se realizará utilizando sockets TCP para la comunicación entre clientes y servidor, y se evaluará el desempeño bajo diferentes condiciones de carga mediante herramientas de benchmarking. Además, se analizarán métricas como el tiempo de respuesta y la tasa de errores. El proyecto proporcionará una comprensión práctica del manejo de procesos, hilos y comunicación en red, ofreciendo una base sólida para futuros desarrollos en entornos distribuidos y aplicaciones de red robustas.

Introducción

El desafío principal que aborda este experimento es **la medición de la concurrencia en servidores web** y cómo esta afecta el rendimiento y la capacidad de respuesta del sistema. En la actualidad, los servidores web deben manejar grandes volúmenes de solicitudes simultáneas de usuarios de manera eficiente. Sin una gestión adecuada de la concurrencia, un servidor puede enfrentar tiempos de respuesta lentos, errores de servicio o incluso fallos catastróficos bajo cargas pesadas, lo que puede impactar la disponibilidad del servicio y la experiencia del usuario.

Además, los servidores modernos están equipados con un gran número de cores o unidades de procesamiento capaces de ejecutar múltiples tareas de forma simultánea. Sin embargo, aprovechar al máximo esta capacidad desde el punto de vista del software no es una tarea sencilla. Existen diversas limitaciones del sistema, como los controladores de memoria y los buses I/O, que deben ser gestionadas de manera eficiente para que la concurrencia no se convierta en un cuello de botella. Los sistemas operativos y las librerías de desarrollo proporcionan diferentes modelos de abstracción para gestionar la concurrencia, cada uno con sus ventajas y

desventajas. Parte del desafío consiste en identificar qué modelo es el más adecuado según el escenario.

En el contexto tecnológico actual, la capacidad de los servidores web para manejar concurrencia de manera eficiente es esencial para la escalabilidad y el rendimiento de aplicaciones web modernas. La creciente demanda por aplicaciones que deben estar disponibles 24/7, combinada con el uso de arquitecturas distribuidas y microservicios, ha hecho que la capacidad para gestionar solicitudes concurrentes de forma eficiente sea crítica.

Además, los servidores actuales cuentan con múltiples cores, y si bien esta capacidad física permite la ejecución simultánea de varias tareas, explotarla eficientemente mediante software requiere modelos avanzados de concurrencia. Estos modelos, proporcionados por los sistemas operativos y librerías de desarrollo, ofrecen diversas formas de aprovechar los recursos de procesamiento. No obstante, cada enfoque tiene sus limitaciones, y la correcta elección del modelo puede marcar la diferencia entre un servidor optimizado o uno que colapsa bajo una carga pesada. Optimizar el uso de estos cores mediante técnicas de concurrencia es crucial para que los servidores web sean capaces de manejar la demanda moderna sin comprometer el rendimiento.

En un mundo en el que los servicios web deben responder rápidamente y gestionar una gran cantidad de tráfico, la capacidad de un servidor web para manejar la concurrencia define la escalabilidad, eficiencia operativa y la calidad del servicio en el contexto de las arquitecturas modernas en la nube y en entornos altamente distribuidos.

Antecedentes

En el trabajo titulado "Creación de núcleos de servidor web siguiendo diferentes modelos de concurrencia", David Márquez Hernández explora los beneficios y desafíos de tres modelos para implementar servidores web: procesos, hilos y eventos. Este trabajo concluye que cada modelo presenta ventajas específicas según el entorno, pero los hilos ofrecen un equilibrio entre rendimiento y eficiencia. Además, resalta la importancia de gestionar correctamente las secciones críticas y evitar condiciones de carrera para mantener la integridad del sistema [3].

Otro referente es el proyecto "Servidor web concurrente y distribuido" de la Universidad de Costa Rica. Aquí se toma como punto de partida un servidor serial, transformándolo en concurrente mediante un patrón productor-consumidor. El enfoque es dividir las responsabilidades en dos tipos de hilos: un hilo principal que acepta conexiones y las coloca en cola, y varios manejadores que consumen dichas conexiones de forma concurrente. Esta estructura mejora significativamente la capacidad del servidor para atender múltiples clientes simultáneamente[1].

Marco teórico

La concurrencia se refiere a la capacidad de gestionar múltiples tareas al mismo tiempo mediante técnicas de intercalado de procesos o hilos en una CPU, mientras que el paralelismo implica la ejecución simultánea en múltiples núcleos. Ambos conceptos son esenciales para mejorar la eficiencia en servidores web.

Existen distintos modelos de concurrencia que son implementados en la industria según los requerimientos [3]:

- **Procesos:** Cada cliente se maneja en un proceso independiente, lo que proporciona aislamiento pero conlleva un mayor consumo de recursos.
- **Hilos:** Los hilos permiten compartir memoria dentro de un proceso, reduciendo los costos de cambio de contexto, pero requieren mecanismos de sincronización para evitar problemas de concurrencia.
- **Eventos:** Este modelo minimiza los cambios de contexto gestionando todas las operaciones en un único bucle de eventos, siendo más eficiente para ciertas aplicaciones, aunque difícil de programar

Adicionalmente se pueden definir algunos conceptos mencionados en los antecedentes relacionados a la gestión de servidores y que vale la pena clarificar.

Uno es el patrón productor-consumidor, en este un hilo principal acepta conexiones y las pone en una cola, mientras los hilos consumidores procesan dichas conexiones. Esta estrategia se usa para evitar que el hilo principal se bloquee, asegurando así que el servidor pueda aceptar nuevas solicitudes continuamente [1].

También se menciona el término sincronización, el cual se relaciona con el manejo de recursos y hace referencia a la correcta gestión de recursos compartidos es vital para evitar problemas como *deadlocks* o condiciones de carrera. Esto se logra mediante mecanismos de exclusión mutua como semáforos y bloqueos[1].

Objetivos (principal y específicos)

Objetivo general: Desarrollar e implementar un servidor web concurrente que gestione múltiples solicitudes simultáneamente.

Objetivos específicos:

- Crear un servidor web básico con capacidad para concurrencia.
- Implementar concurrencia mediante hilos, procesos o eventos.
- Medir la capacidad de carga del servidor mediante pruebas con herramientas de benchmarking.

- Analizar el desempeño del servidor en términos de tiempo de respuesta, solicitudes por tiempo, tamaño de las colas y manejo de errores http.

Metodología

Herramientas principales:

- Pruebas de carga: Apache JMeter, Locust, httperf-py
- Monitoreo del servidor: Prometheus-client.
- Análisis de logs: Logging (módulo de Python), ELK Stack (con Python-Elastic).

Actividades:

- Definición de parámetros: Establecer objetivos, solicitudes a simular y métricas clave (tiempo de respuesta, solicitudes por tiempo, tamaño de las colas y manejo de errores HTTP).
- Configuración del entorno: Preparar el servidor, herramientas de carga y monitoreo.
- Pruebas preliminares: Ejecutar pruebas de baja carga para validar configuraciones.
- Pruebas de concurrencia: Escalar solicitudes concurrentes y monitorear métricas clave a diferentes niveles de carga.
- Análisis de resultados: Identificar puntos de saturación y generar gráficos de desempeño.
- Evaluación y optimización: Ajustar configuraciones si es necesario y repetir pruebas.
- Documentación: Informar resultados y recomendaciones de optimización.

Antes de cada etapa, se introducirá un prólogo teórico para contextualizar los conceptos clave sobre concurrencia, rendimiento y optimización en servidores web. Estos apartados teóricos servirán como base para la posterior implementación de las pruebas y análisis. A pesar de iniciar con una perspectiva conceptual, el objetivo final es la aplicación práctica, culminando con la implementación y optimización del código y la configuración de los servidores web creados durante este trabajo. Cada teoría será aplicada para abordar y resolver el problema de concurrencia planteado.

Diseño de Experimentos:

Diseño Experimental

1. Variables del Experimento

- **Variables dependientes (métricas a medir)**:

    - **Tiempo de respuesta**: Medido en milisegundos desde la solicitud hasta la respuesta.
    - **Throughput**: Número de solicitudes por segundo.
    - **Tamaño de las colas**: Número de solicitudes en espera en el servidor.
    - **Errores HTTP**: Frecuencia y tipo de errores HTTP (404, 500, etc.).

- **Variables independientes (factores del experimento)**:
    - **Número de solicitudes concurrentes**: Cantidad de clientes que hacen solicitudes simultáneamente (niveles: 10, 50, 100, 500, 1000).
    - **Tamaño máximo de la cola**: Configuración del límite en la cola del servidor (niveles: 50, 100, 200).
    - **Número de hilos**: Configuraciones diferentes para el manejo de concurrencia (niveles: 5, 10, 20).

Se plantean 4 grupos de hipótesis nulas y alternativas para cubrir cada variable dependiente:

Hipótesis:

- Hipótesis 1 (H1): El tiempo de respuesta aumentará significativamente con el número de solicitudes concurrentes.
- Hipótesis nula (H0): No hay un aumento significativo en el tiempo de respuesta con el incremento de solicitudes concurrentes.

- Hipótesis 2 (H2): El throughput del servidor alcanzará un valor máximo y luego disminuirá al aumentar la carga concurrente debido a la saturación del sistema.
- Hipótesis nula (H0): El throughput del servidor no muestra cambios significativos con diferentes cargas.

- Hipótesis 3 (H3): Un tamaño de cola mayor permitirá manejar más solicitudes sin aumentar significativamente el tiempo de respuesta bajo carga alta.
- Hipótesis nula (H0): Cambiar el tamaño de la cola no afecta el tiempo de respuesta bajo cargas altas.

- Hipótesis 4 (H4): El porcentaje de errores HTTP aumentará cuando el servidor se sature, especialmente para códigos 500 (error interno del servidor).
- Hipótesis nula (H0): El porcentaje de errores HTTP no cambia significativamente con diferentes niveles de carga.

2. Diseño del Experimento

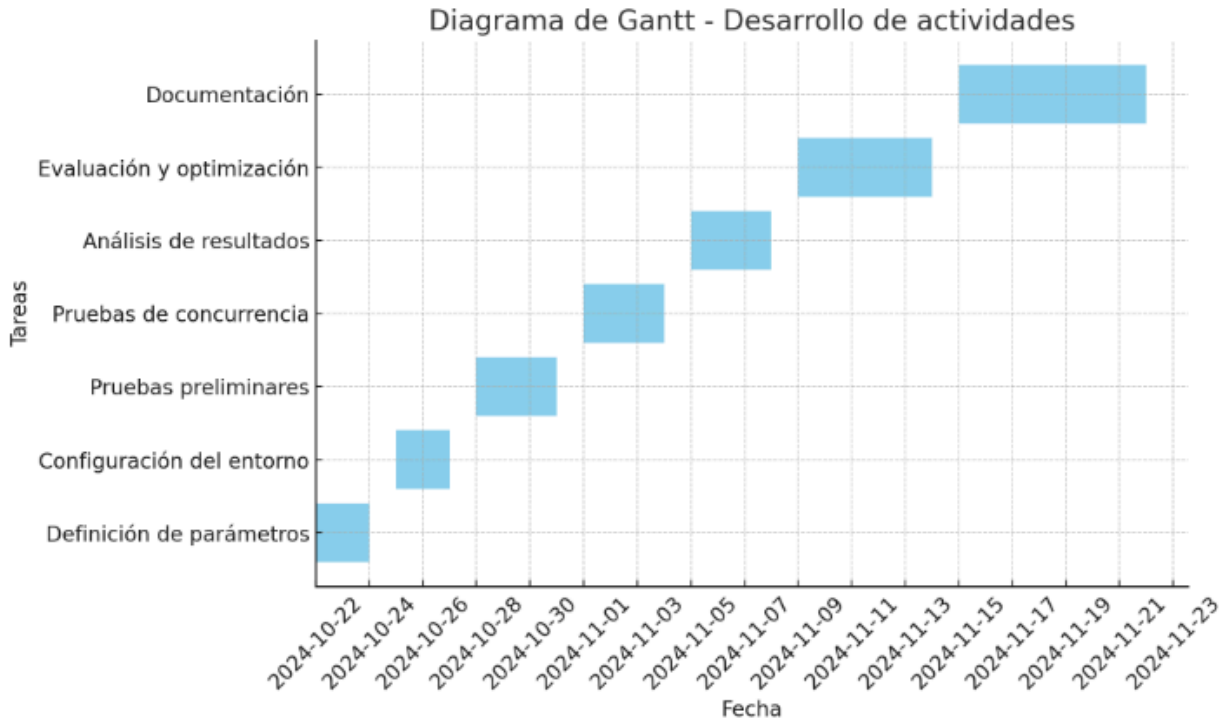
- Utilizaremos un **diseño factorial completo** en el que combinaremos todos los niveles de las variables independientes para evaluar sus efectos conjuntos y separados en las variables dependientes.
- **Niveles de los factores**:
    - **Número de solicitudes concurrentes**: 10, 50, 100, 500, 1000.
    - **Tamaño máximo de la cola**: 50, 100, 200.
    - **Número de hilos**: 5, 10, 20.

Esto genera un diseño con 45 combinaciones posibles de configuraciones.

3. Procedimiento

1. **Configuración del entorno de prueba**:
    - Implementar el servidor web en un entorno controlado.
    - Usar herramientas de prueba como `ab`, `wrk` o scripts en Python para generar la carga.
2. **Realización de las pruebas**:
    - Para cada combinación de configuración, ejecutar múltiples pruebas (al menos 5 repeticiones) y promediar los resultados.
    - Registrar el **tiempo de respuesta**, **throughput**, **tamaño de la cola**, y **número de errores HTTP**.
3. **Control de otras variables**:
    - Asegurarse de que las pruebas se realicen en un entorno estable y sin otros procesos que afecten el rendimiento.

Cronograma



Referencias

[1] Proy1: Servidor web concurrente y distribuido. (n.d.). Ucr.ac.cr. Retrieved October 19, 2024, from https://jeisson.ecci.ucr.ac.cr/concurrente/2021b/proyectos/webserv/

[2]Sistemas Operativos UdeA. (n.d.). Youtube. Retrieved October 19, 2024, from https://www.youtube.com/channel/UCZuMEsHy5jdZtdtJtDWoACQ

[3](N.d.). Retrieved October 19, 2024, from https://openaccess.uoc.edu/bitstream/10609/59647/7/dmarquezhTFG0117mem%C3%B2ria.pdf