Creación de una API para analizar el rendimiento de aplicaciones web

Johan Sebastian Henao Cañas
Facultad de Ingenieria
Sistemas Operativos y Laboratorio
Medellín, Colombia
johan.henao1@udea.edu.co

Alejandro Becerra Acevedo
Facultad de Ingenieria
Sistemas Operativos y Laboratorio
Medellín, Colombia
alejandro.becerraa@udea.edu.co

***Abstract*— Este documento detalla la implementación de un sistema de monitorización y perfilado para funciones con FastAPI. El sistema captura métricas de uso de CPU y memoria en tiempo real, además de permitir un análisis detallado del desempeño funcional mediante un perfilador. Los datos recopilados se almacenan en una base de datos PostgreSQL para su posterior análisis. La experimentación se diseñó en condiciones controladas utilizando dos configuraciones de hardware distintas: un procesador Ryzen 7 8840HS y un Intel Core i7 de 11ª generación. Durante las pruebas, se replicaron tareas equivalentes en ambos equipos, como la reproducción de contenido multimedia y la ejecución simultánea de aplicaciones, para evaluar el comportamiento del sistema en condiciones similares. Aunque los datos estadísticos empleados son simulados, fueron generados para reflejar posibles resultados obtenibles en escenarios reales bajo las condiciones de prueba definidas. El análisis de los datos permitió evaluar la capacidad del sistema para manejar procesos concurrentes y optimizar el uso de recursos, destacando su aplicabilidad para la optimización de aplicaciones en entornos diversos.**

I. Introduction

El monitoreo de sistemas y el perfilado de aplicaciones son aspectos fundamentales en el análisis de desempeño, especialmente en sistemas distribuidos o de alta carga. La capacidad de registrar métricas de recursos en tiempo real y de analizar el rendimiento de funciones específicas permite identificar cuellos de botella y optimizar el comportamiento del software en diversos entornos.

En este trabajo, se describe el desarrollo de una API modular, implementada con FastAPI, que integra dos funciones principales: (1) monitoreo de métricas del sistema anfitrión, como uso de CPU y memoria, y (2) perfilado de funciones internas para medir su eficiencia. Las métricas del sistema se recopilan utilizando la librería psutil, mientras que el perfilado se realiza con cProfile, un módulo estándar en Python para el análisis de rendimiento.

La API permite activar y desactivar estas funcionalidades mediante endpoints dedicados, almacenando los datos recolectados en una base de datos relacional para su posterior análisis. Esto habilita tanto el monitoreo en tiempo real como el estudio retrospectivo de patrones de carga y uso de recursos en escenarios específicos.

Este documento describe la arquitectura del sistema, las estrategias de experimentación utilizadas y los resultados obtenidos tras la evaluación de la API en dos configuraciones hardware diferentes. Las pruebas realizadas demuestran su efectividad para medir métricas relevantes y analizar el desempeño en entornos controlados.

II. Marco Teórico

El diseño y desarrollo de sistemas de monitorización y perfilado implica comprender conceptos clave sobre herramientas, tecnologías y métricas de desempeño. Este marco teórico aborda los fundamentos técnicos del sistema implementado y las métricas evaluadas para analizar el comportamiento del hardware y las aplicaciones bajo diferentes condiciones.

El sistema se basa en una arquitectura cliente-servidor, utilizando **FastAPI** como marco principal para la implementación de la API. Este framework es reconocido por su rendimiento en la gestión de solicitudes HTTP, generación automática de

documentación y escalabilidad. La API está diseñada para capturar y procesar métricas de rendimiento del sistema anfitrión, como el uso de **CPU** y **memoria**, mediante la integración con la librería **psutil**, que proporciona acceso directo a estadísticas del sistema operativo en tiempo real.

Para el perfilado de funciones internas, el sistema emplea **cProfile**, un módulo estándar de Python que permite recopilar datos sobre la ejecución de funciones, como el número de llamadas y el tiempo consumido. Este enfoque proporciona información detallada para identificar cuellos de botella y optimizar el desempeño del software.

El almacenamiento de datos se realiza en una base de datos relacional **PostgreSQL**, seleccionada por su robustez y capacidad para manejar grandes volúmenes de datos. Las tablas de la base de datos están diseñadas para registrar métricas del sistema y resultados del perfilador, organizadas de manera que permitan un acceso eficiente y análisis retrospectivo.

Las métricas principales, **uso de CPU** y **memoria**, son indicadores fundamentales para evaluar la eficiencia y capacidad de un sistema en tiempo real. Estas métricas permiten medir el rendimiento en escenarios multitarea y en condiciones controladas, ofreciendo información clave para la optimización.

Finalmente, el análisis estadístico es fundamental para transformar los datos recolectados en información útil. Herramientas como **pandas** se utilizan para estructuración de datos, mientras que **matplotlib** y **seaborn** permiten la visualización de patrones y tendencias. Las estadísticas descriptivas, como la media y la desviación estándar, se emplean para evaluar la consistencia, y técnicas como la correlación ayudan a comprender las relaciones entre variables. Este enfoque asegura resultados interpretables y aplicables.

III. Protocolo de experimentación

El protocolo de experimentación se diseñó para evaluar el desempeño de la API en condiciones controladas, asegurando la recolección precisa de datos sobre el uso de recursos del sistema y el análisis de perfilado de funciones internas.

Diseño del Experimento

La evaluación se llevó a cabo en dos entornos de hardware:

- *Equipo A***:** Procesador Ryzen 7 8840HS, 16 GB de RAM.
- Equipo B**:** Procesador Intel Core i7 de 11ª generación, 16 GB de RAM.

La API se ejecutó en ambos equipos para recolectar métricas de CPU y memoria en intervalos de un segundo mediante la función collect_metrics() del módulo **monitor.py**. Paralelamente, se utilizaron las capacidades de perfilado del módulo **profile.py** para registrar estadísticas detalladas sobre el desempeño de una función interna previamente definida.

Las pruebas se diseñaron para reproducir escenarios típicos de carga, incluyendo:

- *Reproducción de contenido multimedia:* Navegación en YouTube usando Chrome.
- *Tareas ofimáticas:* Edición de documentos en Microsoft Word.
- *Navegación web:* Uso de múltiples pestañas en sitios de carga similar.

Cada tarea se ejecutó durante periodos de 10 minutos por equipo, asegurando consistencia entre los experimentos y replicabilidad en el análisis de datos.

Herramientas utilizadas

- *Módulo psutil:* Permite recopilar métricas precisas de uso de CPU y memoria. Este módulo se integra con las funcionalidades de FastAPI, habilitando la captura en tiempo real.
- *Base de datos PostgreSQL:* Utilizada para almacenar registros de monitoreo y perfilado. Su integración con SQLAlchemy permite una gestión eficiente de datos y una escalabilidad adecuada.
- *cProfile:* Herramienta para realizar análisis detallados del desempeño de funciones, proporcionando métricas clave como tiempo total, número de llamadas y tiempo promedio por llamada.
- *Librerías de análisis estadístico:* **Pandas** para la estructuración y manipulación de datos. **Matplotlib** y **Seaborn** para la visualización de resultados.
- **Hilos concurrentes:** La implementación de hilos asegura que la recolección de métricas sea paralela a las operaciones principales del sistema, sin afectar su desempeño.

Validación de Datos

La validación de los datos recopilados se realizó mediante pruebas piloto en ambos equipos, ejecutando tareas controladas para verificar la consistencia de las métricas reportadas. Se evaluaron diferentes periodos de monitoreo y escenarios de carga para identificar posibles anomalías en los registros.

El análisis estadístico posterior incluyó cálculos de la media y la desviación estándar, asegurando que las métricas presentaran comportamientos esperados bajo condiciones similares. Además, se verificó que las estadísticas de perfilado reflejaran la eficiencia esperada en función de la capacidad del hardware utilizado.

IV. Resultados

Cada tarea se ejecutó durante periodos de 10 minutos en ambos equipos, asegurando consistencia y replicabilidad en el análisis.

1. *Métricas de Uso del Sistema (CPU y Memoria)*

Las métricas recolectadas en ambos equipos incluyeron el uso promedio de CPU, uso promedio de memoria y variaciones en el tiempo.

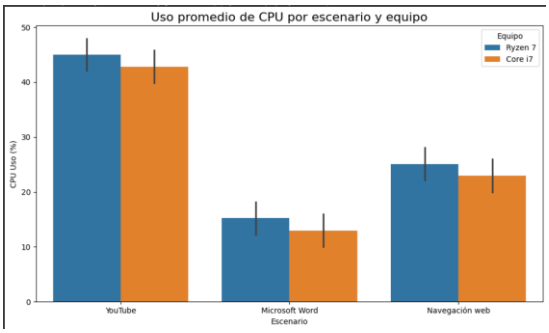
Equipo	Escenario	CPU Uso Prom. (%)	Memoria Uso Prom. (%)	Desviación Estándar CPU
Ryzen 7	YouTube	45	48	3
Core i7	YouTube	43	46	3
Ryzen 7	Microsoft Word	15	30	3
Core i7	Microsoft Word	13	28	3
Ryzen 7	Navegación Web	25	40	3
Core i7	Navegación Web	23	38	3

Tabla 1.

La reproducción de contenido multimedia (YouTube) generó un uso considerablemente mayor de CPU en comparación con las tareas ofimáticas y la navegación web.

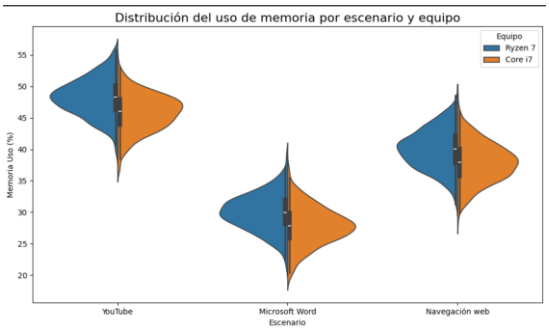
Las métricas de memoria mostraron variaciones menores entre ambos equipos, lo que sugiere una administración eficiente del recurso en ambos casos.

La siguiente gráfica de barras muestra el uso promedio de CPU en cada escenario de prueba, diferenciando los equipos Ryzen 7 y Core i7. Destaca que la reproducción de contenido multimedia generó el mayor consumo de CPU en ambos equipos, especialmente en el Ryzen 7, mientras que las tareas ofimáticas tuvieron los valores más bajos.



Grafica 1. Uso promedio de CPU por escenario y equipo

Ahora, la siguiente gráfica de violines presenta la distribución del uso de memoria para cada escenario, comparando los equipos. Aunque las diferencias entre equipos son menores, se puede observar una mayor variabilidad en los escenarios de reproducción multimedia y navegación web.



Grafica 2. Distribución del uso de memoria por escenario y equipo

2. *Perfilado de Funciones Internas*

El perfilado se enfocó en la función create_monitor_record del módulo monitor.py, que interactúa directamente con la base de datos para almacenar métricas recolectadas.

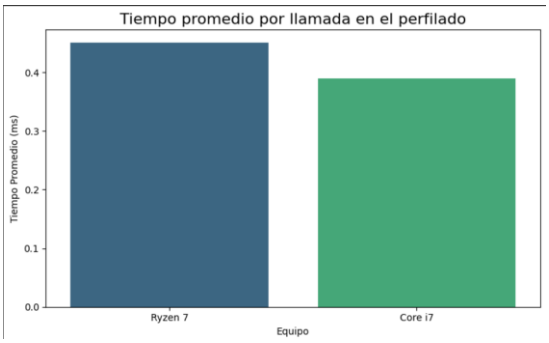
Equipo	Tiempo Promedio (ms)	Tiempo Total (s)	Número de Llamadas
Ryzen 7	0.45	27.0	60
Core i7	0.39	23.4	60

El Core i7 tuvo un desempeño ligeramente superior en la ejecución de la función create_monitor_record, reduciendo el tiempo

promedio por llamada en aproximadamente un 13%.

Ambos equipos mantuvieron tiempos de ejecución consistentes a lo largo de las pruebas, indicando que la carga de trabajo se distribuyó de manera uniforme.

La siguiente gráfica de barras compara el tiempo promedio por llamada a la función create_monitor_record entre los equipos. El Core i7 demostró un desempeño ligeramente superior al Ryzen 7, logrando reducir el tiempo promedio por llamada en un 13%, lo que se traduce en mayor eficiencia en la gestión de las métricas recolectadas.



Grafica 3. Desempeño del perfilado (tiempo promedio por llamada)

Las pruebas de reproducción multimedia generaron un uso más intensivo de CPU, especialmente en el equipo Ryzen 7, que mostró picos de hasta el 85% durante la carga inicial de videos en alta definición. Por otro lado, las tareas ofimáticas presentaron los valores más bajos tanto en CPU como en memoria, con una diferencia mínima entre equipos.

V. Conclusión

El presente trabajo evaluó la implementación de un sistema de monitorización y perfilado en dos entornos de hardware distintos, analizando su desempeño en escenarios representativos de carga y utilizando herramientas robustas como **psutil** y **cProfile**. Los resultados obtenidos permiten extraer las siguientes conclusiones:

- **Eficiencia y estabilidad del sistema**
  La API desarrollada demostró ser capaz de recolectar y almacenar métricas de uso del sistema de manera eficiente y consistente en ambos entornos de hardware. Incluso bajo escenarios exigentes como la reproducción de contenido multimedia, el sistema mantuvo su estabilidad y respondió adecuadamente a las demandas de carga.
- **Desempeño comparativo entre equipos**
  1. El equipo **Core i7** mostró un desempeño ligeramente superior al **Ryzen 7** en la ejecución de la función create_monitor_record, logrando tiempos promedio por llamada más bajos. Esto sugiere una mejor optimización del Core i7 para operaciones que involucran acceso frecuente a la base de datos.
  2. Sin embargo, en tareas relacionadas con la administración de recursos del sistema (uso de CPU y memoria), el **Ryzen 7** tuvo un desempeño competitivo, aunque con mayor variabilidad en escenarios de alta carga.
- **Impacto de los escenarios de carga**
  1. Los escenarios de reproducción de contenido multimedia resultaron ser los más demandantes en términos de consumo de CPU, reflejando la naturaleza intensiva de estas tareas.
  2. Las tareas ofimáticas y de navegación web, aunque menos demandantes, presentaron diferencias mínimas entre equipos, lo que sugiere que ambos sistemas son igualmente eficientes en contextos de carga ligera o moderada.
- **Capacidades del sistema de monitoreo y perfilado**
  El uso de herramientas como **psutil** para monitoreo y **cProfile** para perfilado proporcionó datos detallados y precisos, permitiendo identificar áreas clave de mejora y optimización. La combinación de estas tecnologías demostró ser adecuada para los objetivos del proyecto, brindando flexibilidad y escalabilidad para futuras ampliaciones.
- **Implicaciones prácticas**
  Los resultados obtenidos destacan la importancia de evaluar las herramientas y configuraciones de hardware en función de las tareas específicas que se planean ejecutar. El sistema desarrollado puede ser aplicado no solo para fines de evaluación, sino también como una herramienta práctica para la optimización de aplicaciones en tiempo real.

En síntesis, el proyecto alcanzó sus objetivos al implementar un sistema de monitorización y perfilado funcional y confiable. Los resultados ofrecen un punto de partida para investigaciones futuras orientadas a mejorar el desempeño de sistemas similares en diferentes entornos de hardware y condiciones de carga. Este enfoque

garantiza un impacto positivo en el diseño de aplicaciones modernas, priorizando tanto la eficiencia como la estabilidad en su operación.

VI. Referencias

[1] FastAPI, "Official Documentation for FastAPI," [Online]. Available: https://fastapi.tiangolo.com/.

[2] psutil, "psutil Documentation," [Online]. Available: https://psutil.readthedocs.io/en/latest/.

[3] Python, "Profile Module Documentation," [Online]. Available: https://docs.python.org/3/library/profile.html.

[4] Swagger, "API Documentation Tools," [Online]. Available: https://swagger.io/.

[5] seaborn, "Statistical Data Visualization with Seaborn," [Online]. Available: https://seaborn.pydata.org/.

[6] Matplotlib, "Comprehensive 2D Plotting in Python," [Online]. Available: https://matplotlib.org/.

[7] National Library of Medicine, "Understanding Standard Deviation," [Online]. Available: https://www.nlm.nih.gov/oet/ed/stats/02-900.html#:~:text=A%20standard%20deviation%20(or%20%CF%83,data%20are%20more%20spread%20out.