Proyecto Final

Curso de Sistemas Operativos y Laboratorio

Título del proyecto

Performance API

Miembros del equipo

- Jonathan Mazo González
- José David Henao Gallego
- Juan Esteban Aristizabal
- Sharid Samantha Madrid Ospina

Resumen

Este proyecto propone el diseño e implementación de una API de monitoreo de rendimiento orientada a aplicaciones en tiempo real. La solución recolecta métricas clave como uso de CPU, consumo de memoria RAM, actividad de dispositivos de entrada/salida (E/S) y ejecución de funciones específicas. Estas métricas se exponen mediante endpoints RESTful, compatibles con herramientas como Prometheus y Grafana para su visualización. El sistema utiliza tecnologías modernas como `psutil` y `py-spy`, permitiendo así una trazabilidad precisa del comportamiento de los procesos. Esta propuesta no solo permite detectar cuellos de botella y optimizar el rendimiento, sino que también refuerza el aprendizaje de conceptos esenciales de los sistemas operativos, como la planificación de procesos, la gestión de memoria y el análisis de desempeño.

Introducción

Hoy en día, muchas aplicaciones enfrentan problemas de rendimiento que suelen estar relacionados con un mal uso de los recursos del sistema. Detectar estos cuellos de botella no siempre es sencillo, especialmente si no se cuenta con las herramientas adecuadas que permitan observar lo que ocurre dentro de los procesos funcionales. Aunque existen algunas soluciones comerciales, muchas son difíciles de integrar o no se ajustan a los entornos de desarrollo actuales.

Este proyecto busca desarrollar una API sencilla, modular y fácil de usar que permita obtener métricas en tiempo real sobre el uso de CPU, memoria, E/S y funciones específicas dentro de una aplicación. La información recolectada se expone a través de endpoints y puede visualizarse usando herramientas como Prometheus y Grafana, lo que facilita su análisis.

El objetivo es ofrecer una solución útil tanto para pruebas como para ambientes en producción, aprovechando tecnologías modernas y conectando los conceptos teóricos del curso —como la planificación de procesos o la administración de memoria— con una aplicación práctica real.

Antecedentes o marco teórico

Para desarrollar una API que monitoree el rendimiento de aplicaciones en tiempo real, es fundamental comprender ciertos aspectos clave de los sistemas operativos y las herramientas modernas de observabilidad.

Uno de los pilares es la **gestión de procesos**, que abarca cómo el sistema operativo maneja la creación, ejecución y finalización de procesos. Entender los estados por los que pasa un proceso (nuevo, listo, en ejecución, bloqueado y terminado) y cómo se planifica su ejecución es esencial para interpretar las métricas de uso de CPU y detectar posibles cuellos de botella [1].

La **administración de memoria** es otro aspecto crucial. Cada proceso requiere memoria para ejecutarse, y el sistema operativo debe asignar eficientemente. Comprender cómo se gestiona la memoria, incluyendo conceptos como paginación y swapping, permite analizar el consumo de RAM y optimizar el rendimiento de las aplicaciones [2].

El manejo de **dispositivos de entrada/salida (E/S)** también impacta significativamente en el rendimiento. Las operaciones de lectura y escritura en discos o redes pueden convertirse en puntos críticos si no se gestionan adecuadamente. Monitorear estas operaciones ayuda a identificar y resolver problemas relacionados con la E/S [3].

Para obtener información detallada sobre estos aspectos, existen herramientas como:

- **psutil**, una biblioteca en Python que permite acceder a información sobre procesos y utilización de recursos del sistema, como CPU, memoria, discos y red. Es útil para monitoreo y gestión de procesos en tiempo real [4].

- **py-spy**, un profiler de muestreo para programas en Python que permite visualizar en qué partes del código se está gastando más tiempo, sin necesidad de modificar el código fuente ni reiniciar la aplicación [5].

- **cProfile**, una herramienta incorporada en Python que proporciona estadísticas detalladas sobre la ejecución de programas, ayudando a identificar funciones que consumen más tiempo y recursos [6].

- **perf**, una herramienta de Linux que permite analizar el rendimiento del sistema mediante contadores de eventos, proporcionando información sobre el uso de CPU, memoria y otros recursos a nivel de kernel [7].

Para visualizar y analizar las métricas recolectadas, se pueden integrar herramientas como:

- **Prometheus**, un sistema de monitoreo y alerta de código abierto que recopila y almacena métricas como series temporales, permitiendo consultas flexibles y eficientes [8].

- **Grafana**, una plataforma de análisis y visualización que se integra con Prometheus para crear dashboards interactivos y personalizados, facilitando la interpretación de las métricas del sistema [9].

Este proyecto se alinea directamente con los temas abordados en el curso de Sistemas Operativos, tanto en la teoría como en el laboratorio. Al implementar una API que recolecta y expone métricas del sistema, se aplican conceptos como la planificación de procesos, gestión de memoria y análisis de rendimiento. Además, se fortalece la comprensión práctica de herramientas y técnicas modernas utilizadas en la industria para el monitoreo y optimización de sistemas.

Objetivos (principal y específicos)

Objetivo general

Desarrollar una API capaz de analizar y exponer métricas clave relacionadas con el rendimiento de una aplicación, incluyendo el uso de CPU, memoria RAM, operaciones de

entrada/salida (E/S) y el consumo de recursos por funciones específicas, con el fin de facilitar su monitoreo, trazabilidad y optimización.

Objetivos específicos

- Implementar mecanismos de profiling utilizando herramientas como psutil, py-spy y cProfile, que permitan obtener métricas detalladas del sistema operativo.
- Desarrollar endpoints REST mediante FastAPI (o Gin) que expongan las métricas recolectadas de manera estructurada y accesible.
- Integrar Prometheus como sistema de recolección de métricas para facilitar el análisis temporal de los datos.
- Configurar Grafana para la visualización de las métricas mediante dashboards interactivos.
- Permitir la trazabilidad del uso de recursos por funciones individuales dentro de la aplicación objetivo, identificando posibles cuellos de botella.
- Validar el rendimiento y estabilidad de la API mediante pruebas en distintos escenarios de carga y condiciones controladas.

Metodología

El desarrollo de este proyecto se apoyará en herramientas y tecnologías modernas que permiten implementar soluciones de monitoreo eficientes y adaptables. La elección de cada componente responde tanto a su aplicabilidad técnica como a su facilidad de integración en entornos reales.

🔧 **Herramientas propuestas**

- **Lenguajes de programación:**
  Se contempla el uso de Python con el framework **FastAPI**, debido a su simplicidad, rapidez de desarrollo y compatibilidad con bibliotecas de monitoreo como `psutil`. Como alternativa o para futuras versiones, también se consideran Go (con Gin), Node.js o Rust.

- **Herramientas de profiling y monitoreo:**
  Para obtener métricas detalladas del sistema se emplearán herramientas como `psutil`, `py-spy`, `perf`, `cProfile` y, eventualmente, **OpenTelemetry**, si se busca una solución más robusta y estandarizada.

- **Visualización de métricas:**
  Las métricas recolectadas se expondrán a través de **Prometheus**, que actuará como recolector, y se visualizarán en tiempo real mediante **Grafana**, una herramienta flexible y ampliamente utilizada para dashboards personalizados.

- **Sistema operativo base:**
  Todo el desarrollo y pruebas se realizarán en sistemas basados en **Linux**, por ser el entorno más compatible con herramientas de observabilidad a bajo nivel.

🛠️ Etapas de desarrollo

Para cumplir con los objetivos planteados, se seguirá una secuencia lógica de actividades:

1. **Investigación y análisis de herramientas:**
   Se revisará la documentación y funcionamiento de las herramientas de profiling y monitoreo que mejor se adapten al proyecto.

2. **Diseño de la arquitectura de la solución:**
   Se definirá la estructura general de la API, los módulos de recolección de métricas, los endpoints necesarios y cómo se integrarán con Prometheus y Grafana.

3. **Implementación de módulos de recolección:**
   Se desarrollarán componentes que recojan datos sobre uso de CPU, memoria RAM, operaciones E/S y ejecución de funciones específicas.

4. **Desarrollo de la API REST:**
   Usando FastAPI (u otra tecnología elegida), se crearán los endpoints necesarios para exponer las métricas de forma estructurada.

5. **Integración con Prometheus:**
   Se configurará Prometheus para consumir las métricas desde los endpoints `/metrics`, siguiendo el formato estándar.

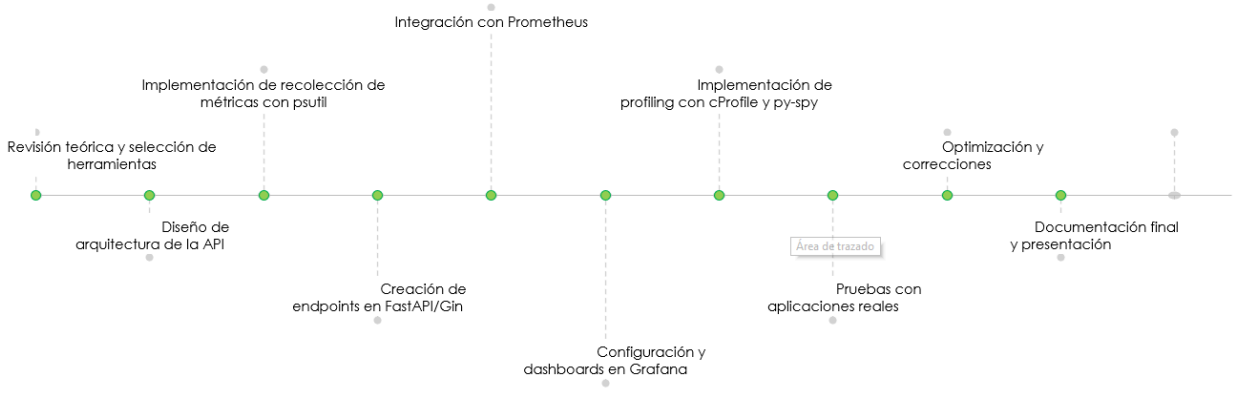
6. **Configuración de Grafana:**
   Se crearán dashboards personalizados en Grafana para visualizar en tiempo real el rendimiento de las aplicaciones monitoreadas.

7. **Pruebas y validación:**
   Se evaluará el rendimiento de la API en diferentes escenarios y se ajustará según los resultados obtenidos.

8. **Documentación final del sistema:**
   Se documentará detalladamente la arquitectura, el uso de la API, la instalación de herramientas y las pruebas realizadas.

-

Cronograma



Referencias

[1] IBM. “What is a process?” IBM Documentation. https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-what-is-process

[2] Red Hat. “Understanding Linux Memory Management.” https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/ch-memory

[3] GeeksforGeeks. “I/O Management in Operating System.” https://www.geeksforgeeks.org/i-o-management-in-operating-system/

[4] Rodola, G. “psutil – process and system utilities.” https://psutil.readthedocs.io/

[5] Ben Frederickson. “py-spy: Sampling profiler for Python programs.” https://github.com/benfred/py-spy

[6] Python Software Foundation. “The Python Profilers.” https://docs.python.org/3/library/profile.html

[7] Brendan Gregg. “perf: Linux Profiling with Performance Counters.” https://perf.wiki.kernel.org/index.php/Main_Page

[8] Prometheus Authors. “Introduction to Prometheus.” https://prometheus.io/docs/introduction/overview/

[9] Grafana Labs. “Prometheus Data Source Configuration.” https://grafana.com/docs/grafana/latest/datasources/prometheus/