[1]

Hilos y Procesos Múltiples para Acelerar Procesos de Búsqueda.

Daniel Luján Agudelo, Luis Guillermo Sánchez Cubides, José David Gómez Muñetón, Juan Pablo Arango Gaviria.

I. Introducción

La necesidad de gestionar y procesar grandes volúmenes de datos de manera eficiente es crucial en campos como la ciencia de datos y la inteligencia artificial. Este proyecto investiga el rendimiento de tres algoritmos de búsqueda: búsqueda secuencial, búsqueda paralela con *multithreading* y búsqueda paralela con *multiprocessing*. Estas técnicas serán implementadas y comparadas en términos de tiempo de ejecución y uso de recursos (CPU y memoria) al buscar en grandes conjuntos de datos almacenados en archivos.

La motivación de este proyecto surge de las limitaciones de la búsqueda secuencial al manejar grandes volúmenes de datos, mientras que el *multithreading* y el *multiprocessing* ofrecen posibilidades de mejorar el rendimiento mediante el paralelismo. El objetivo es identificar el enfoque más eficiente según el tamaño del conjunto de datos y la carga de trabajo.

II. Marco Teórico

La ejecución eficiente de programas es fundamental en sistemas modernos, especialmente cuando se manejan grandes conjuntos de datos. Este proyecto se centra en dos técnicas clave de computación paralela: *multithreading* y *multiprocessing*, que buscan reducir el tiempo de procesamiento dividiendo las tareas en unidades concurrentes.

- *Multiprocessing*: Técnica que permite ejecutar múltiples procesos en paralelo, cada uno con su propio espacio de memoria. Es ideal para sistemas multi-core y evita problemas como el *Global Interpreter Lock* (GIL) en Python.
- *Multithreading*: Permite ejecutar múltiples hilos dentro de un solo proceso, compartiendo el mismo espacio de memoria. Aunque facilita la comunicación entre hilos, introduce desafíos como condiciones de carrera y bloqueos (*deadlocks*), que requieren mecanismos de sincronización como semáforos.

III. Metodología

El proyecto sigue un enfoque estructurado que incluye el diseño e implementación de dos algoritmos de búsqueda: secuencial paralela con hilos y secuencial paralela con procesos. Se usarán herramientas del ecosistema de Python, como las bibliotecas ***threading*** y ***multiprocessing***.

Pasos principales:

- Implementación de las dos técnicas de búsqueda.
- Configuración del entorno experimental para garantizar condiciones consistentes de hardware y software.
- Ejecución de pruebas en datasets de diferentes tamaños (36 MB, 18 MB, 9 MB, 6 MB, 4 MB, 2 MB, 1 MB) y análisis del rendimiento.

IV. Implementación

Cada método de búsqueda será desarrollado en Python, asegurando consistencia en los algoritmos para que las diferencias observadas sean únicamente atribuibles a la técnica empleada.

- **Búsqueda con *Multithreading***: Se implementa multithreading para realizar la búsqueda de un valor en múltiples archivos. Para cada archivo se crea un hilo independiente y este recorre línea por línea para comparar el contenido con el valor buscado.

  Si un hilo encuentra el valor, todos los demás detienen su ejecución, evitando trabajo innecesario y mejorando el tiempo total de la búsqueda.

- **Búsqueda con *Multiprocessing***: En este método en lugar de usar hilos, se crean procesos independientes, donde cada proceso es el encargado de buscar en un archivo en específico. Cada proceso se ejecuta de forma aislada, trabajando en su propio espacio de memoria y evitando cualquier interferencia entre los mismos.

  La implementación utiliza un pool de procesos y a través del método map, el pool asigna a cada proceso un archivo y un valor a buscar, ejecutando la búsqueda en paralelo y retornando una lista de resultados.

[1]

V. Protocolo Experimental

El protocolo busca comparar los tiempos de ejecución de las tres técnicas en condiciones controladas.

- **Factores de Control**:
  - **Tamaño de Archivos**: Se evaluarán conjuntos de datos de diferentes tamaños (36 MB, 18 MB, 9 MB, 6 MB, 4 MB, 2 MB, 1 MB).
  - **Número de Archivos**: Se realizarán pruebas con 2, 4, 8, 12, 16, 32, 48 archivos.
  - **Especificaciones de *Hardware***: Todas las pruebas se ejecutarán en la misma máquina, configurando el número de hilos y procesos para coincidir con los núcleos disponibles.
- **Medición**:
  - Cada técnica será ejecutada al menos 10 veces para minimizar la variabilidad de los resultados.
  - Se registrarán métricas como tiempo de ejecución, uso de CPU y memoria, y sobrecarga en la creación/sincronización de procesos o hilos.

VI. Resultados

Los datos recopilados serán analizados estadísticamente. Se plantean las siguientes hipótesis:

- **H0 (Hipótesis Nula)**: No hay diferencias significativas en los tiempos de ejecución entre las técnicas de búsqueda.
- **H1 (Hipótesis Alternativa)**: Existen diferencias significativas en los tiempos de ejecución entre las técnicas.

Se empleará un análisis ANOVA para determinar diferencias significativas entre los tiempos medios.

	SCE	DF	F	PR(>F)
Técnica de paralelismo	202.358518	1	447.732571	2.096616e-73
Residual	252.195307	558	----	----

Tabla 1, anova

Abordamos nuestro experimento con una confianza del 95% es decir una significancia del 0.05, como podemos ver el valor p asociado a nuestro experimento es mucho menor que el nivel de significancia y esto indica que debemos rechazar la hipótesis nula, esto quiere decir que si existe una diferencia significativa entre varias técnicas de paralelismo.

Visualización de procesos e hilos:

Se decidió añadir la visualización de cómo se comportan los hilos y procesos mientras la ejecución de la aplicación.

Se utilizó el entorno de desarrollo PyCharm, que cuenta con capacidades avanzadas de depuración y monitoreo para la visualización de los hilos. Este entorno permite visualizar en tiempo real los hilos activos en la aplicación, mostrando detalles como su estado (activo, bloqueado o terminado) y las operaciones que ejecutan en cada momento.

El administrador de tareas del sistema operativo se empleó para monitorear el comportamiento de los procesos creados. Esta herramienta nos permitió observar el número de procesos en ejecución y su distribución entre los núcleos del procesador además de identificar el consumo de CPU y memoria por cada proceso.

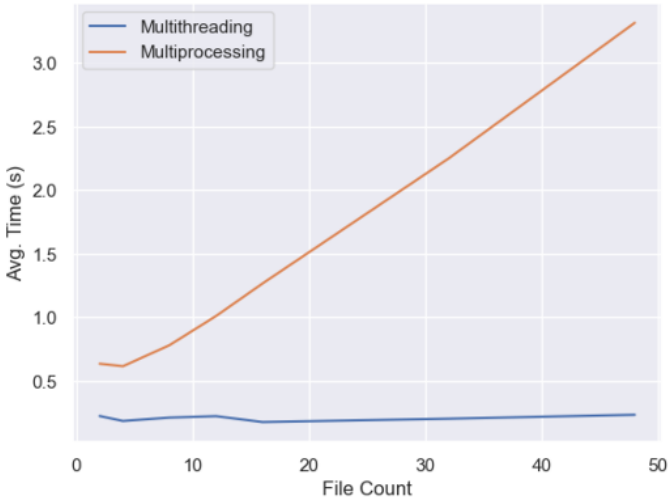

Figura 1. Tiempo búsqueda promedio

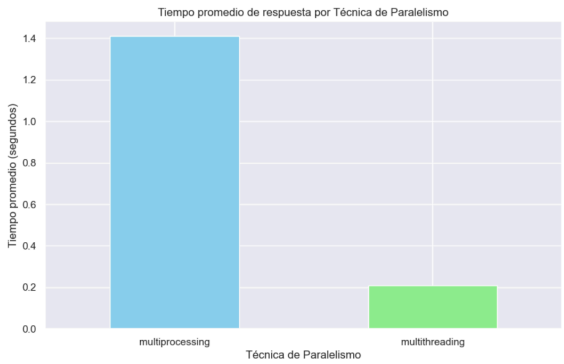

Figura 2. Tiempo búsqueda promedio

En estas gráficas podemos evidenciar que los hilos mantienen un tiempo de respuesta casi constante durante toda la ejecución, mientras que los múltiples procesos aumentan el tiempo de respuesta mientras más procesos se usen para la búsqueda.

VII. Conclusiones

- El resultado obtenido en las gráficas permite evidenciar que el procesamiento bajo el modelo de *multiprocessing* tiende a incrementar a medida que se aumenta el número de archivos. Esto se debe principalmente a la complejidad que le implica los múltiples cambios de contexto, que a pesar de que los sistemas operativos modernos han mejorado el rendimiento de estos procesos, sigue siendo menos eficiente que el cambio de contexto que se presenta en el procesamiento bajo *multithreading,* esto debido a que en este caso se comparte el direccionamiento de memoria lo que resulta en menos esfuerzo. Debido a que cuando se tienen más archivos se deben realizar con más frecuencia estos cambios de contexto esto incorpora más tiempo en el que el sistema operativo está encargado de gestionar el procesamiento paralelo, y que a su vez aumenta el tiempo final obtenido.

- Los resultados obtenidos demuestran que el mejor método en este caso fue el multithreading, y esto se puede deber principalmente a que la tarea que se está realizando es la lectura de archivos, una operación I/O-bound (limitada por el tiempo de espera para la entrada y salida de datos). El multithreading es más eficiente en este tipo de tareas porque los hilos comparten el mismo espacio de memoria y permiten al programa aprovechar el tiempo de inactividad mientras esperan que se complete la lectura del archivo. Esto reduce significativamente el tiempo total de ejecución al realizar múltiples lecturas de forma concurrente, sin incurrir en altos costos de creación y administración.

- El multiprocessing dio peores resultados en esta implementación y esto se puede deber a varias razones. En primer lugar cada proceso se ejecuta en su propio espacio de memoria lo que implica un mayor uso de recursos, en este caso, el tiempo extra necesario para crear y sincronizar los procesos puede introducir una sobrecarga que impacte negativamente al rendimiento general de la búsqueda. Además, dado que la tarea es I/O-bound y no intensiva en CPU, el multiprocessing no puede aprovechar al máximo su capacidad de distribuir la carga de trabajo entre varios núcleos de la CPU.`z

References

[1] TechTarget, “What is multiprocessing?” [Online]. Available: https://www.techtarget.com/searchdatacenter/definition/multiprocessing.

[2] Diego Coder, “Entendiendo los procesos, hilos y multihilos,” Medium, [Online]. Available: https://medium.com/@diego.coder/entendiendo-los-procesos-hilos-y-multihilos-9423f6e40ca7.

[3] SuperFastPython, “Python time.time() vs time.perf_counter,” [Online]. Available: https://superfastpython.com/time-time-vs-time-perf_counter/.