Santiago González-Granada
Grandes Volúmenes de Datos
EAFIT

Pregunta

¿Cuál de los métodos de concurrencia es el más rápido a la hora de descargar toda la lista de pokemones que están el GitHub?

Objetivo

Encontrar el método más rápido de concurrencia para la descarga de los nombres y figuras de los pokemones compartidos en el GitHub

Introducción

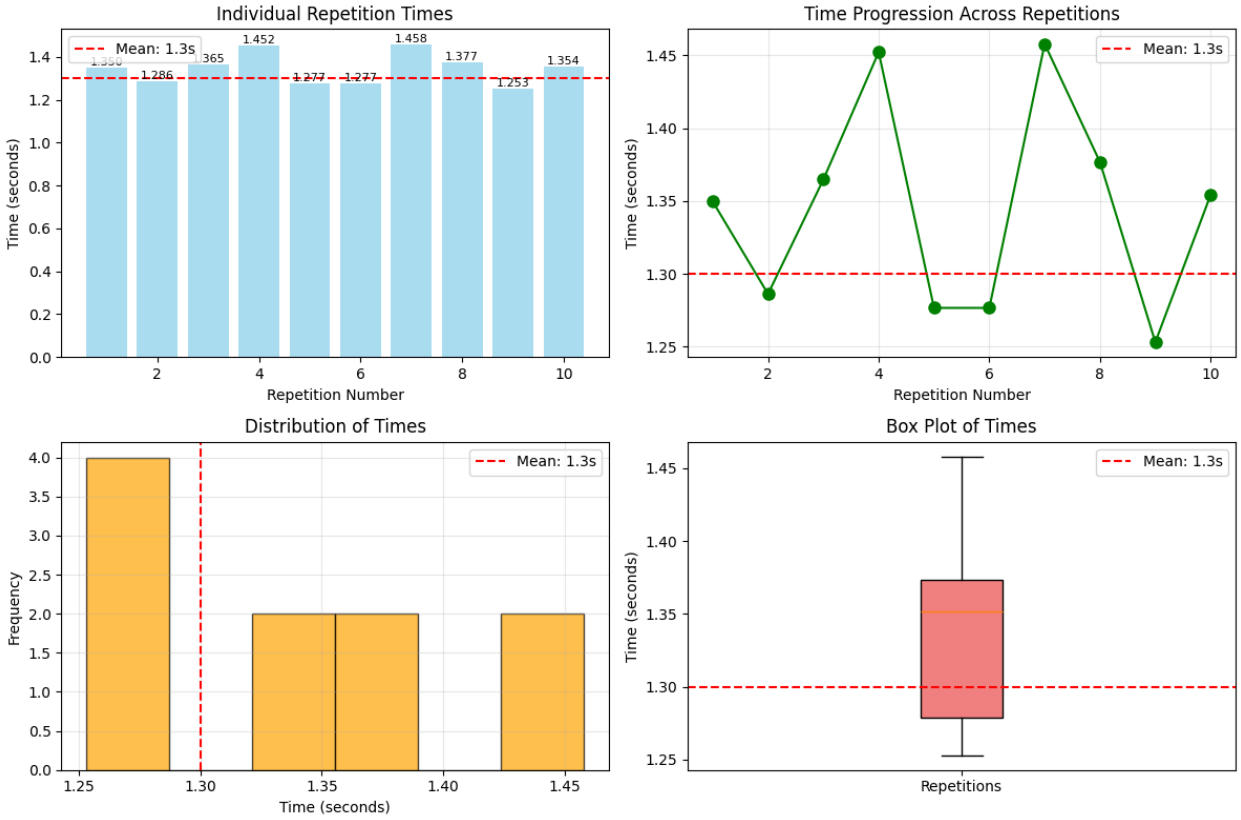
En este esta actividad se realizaron mediciones de los distintos tipos de concurrencia para la descarga de los nombres y figuras de 151 pokemones compartidos en un GitHub. Además, de la creación de un dataloader para cada uno de los métodos de concurrencia (*asyncio, multiprocessing y threading*).

Desarrollo

En la primera parte de la actividad, se corrieron distintos métodos de descarga de información con el fin de conocer cuál de tales métodos era el más rápido a la hora de procesar los nombres y los Sprite de los Pokemones. Como primera instancia, se tuvo que descargar los repositorios compartidos previamente en clase por parte del profesor y luego se corrieron dichos métodos, arrojando los siguientes resultados. Los resultados a continuación mostrados, se hacen para los siguientes 4 métodos, con cada uno siendo corrido aproximadamente 10 veces para tener una imagen más amplia de los distintos métodos utilizados. Se muestra distribución de tiempo de ejecución, además de medidas estadísticas básicas.

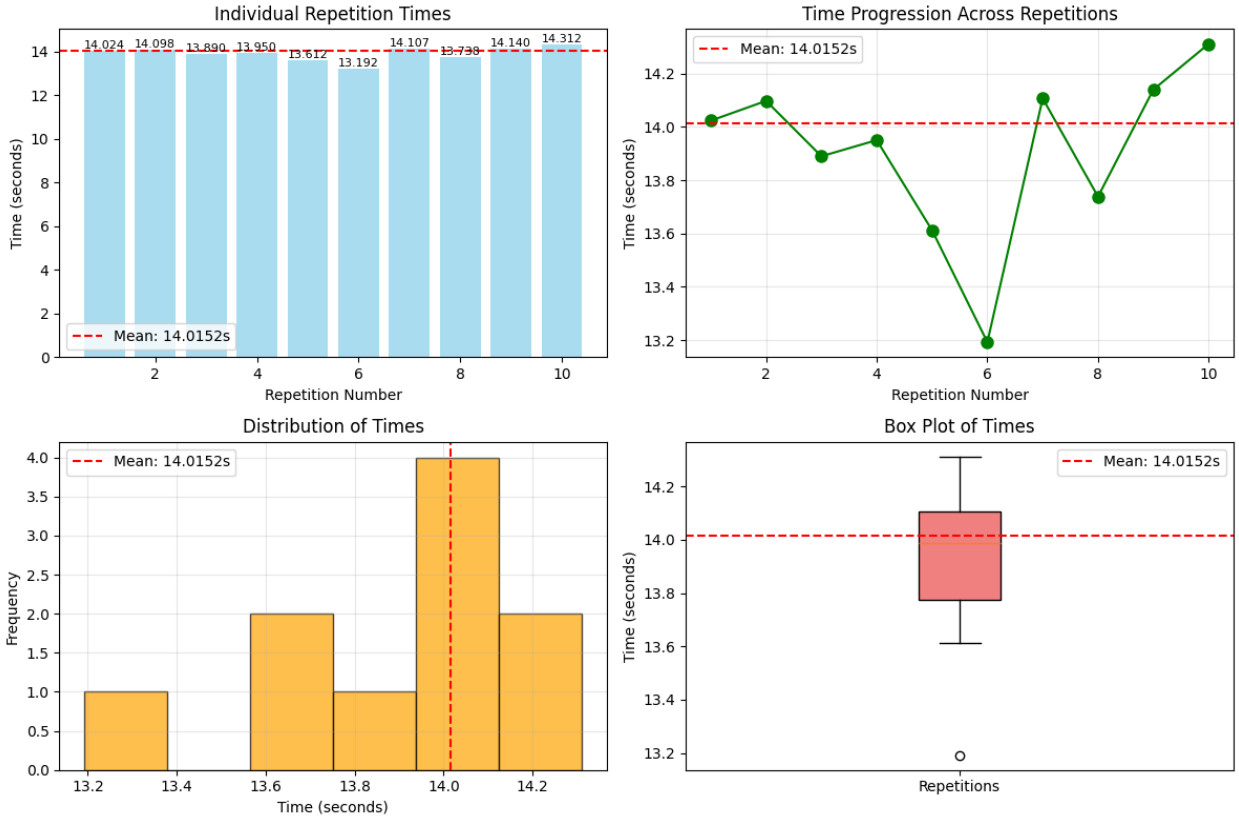
- **Asyncio**

En este caso, el método asíncrono logró un tiempo medio de ejecución de aproximadamente 1,3 segundos, lo que demuestra una gran eficiencia en comparación con otros enfoques. Este valor, obtenido tras varias iteraciones, confirma su fiabilidad en condiciones normales. Como ilustra el gráfico, los tiempos se mantuvieron estables con fluctuaciones mínimas, lo que subraya la consistencia y la escalabilidad de asíncrono para cargas de trabajo más grandes.



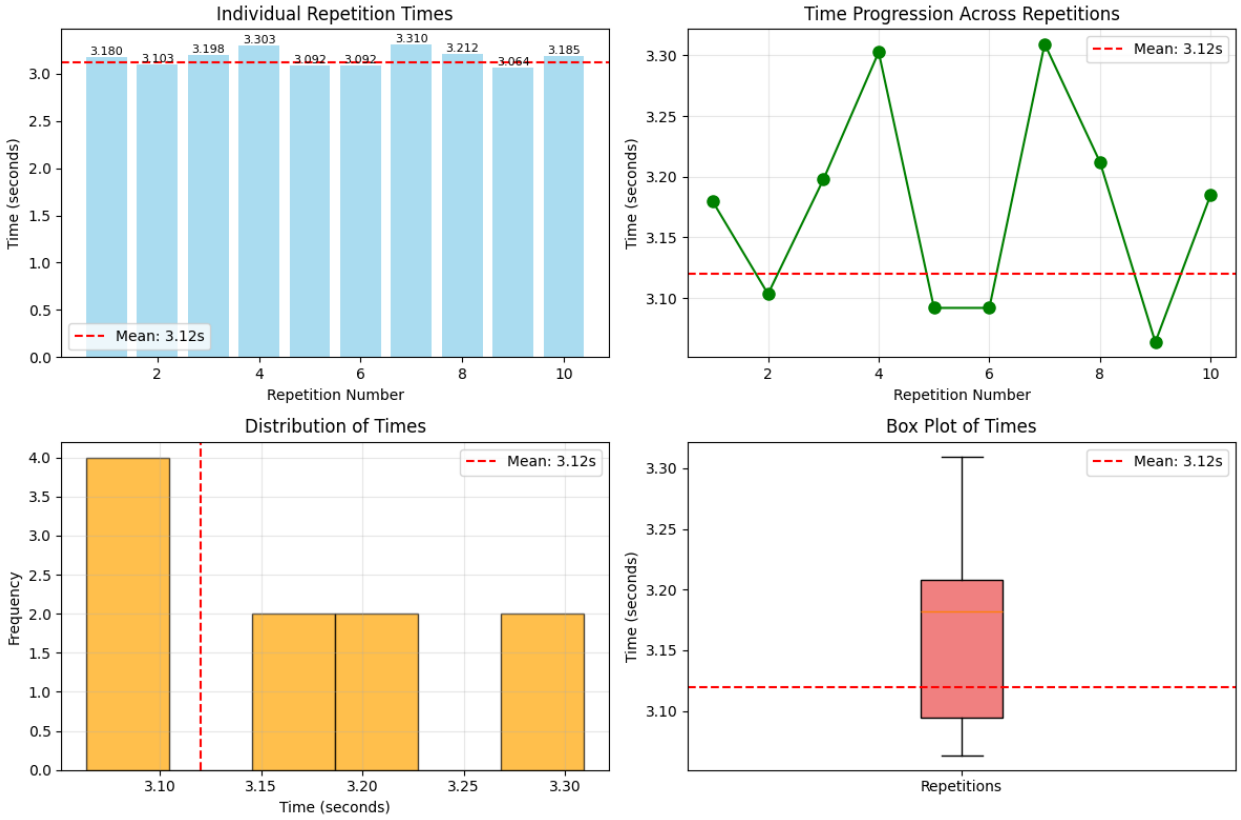
- **Threading**

En este caso, el método de *threading* alcanzó un tiempo medio de ejecución de 14,02 segundos, con una desviación estándar de aproximadamente 0,32 segundos en las 10 ejecuciones realizadas. Aunque los resultados fueron bastante consistentes, como indica la baja variabilidad, este método resultó ser significativamente más lento en comparación con alternativas como asyncio. El mayor tiempo promedio sugiere que el subprocesamiento es menos eficiente para manejar esta carga de trabajo en particular, lo que lo hace menos adecuado para escenarios que exigen una ejecución más rápida o un procesamiento de datos a gran escala.



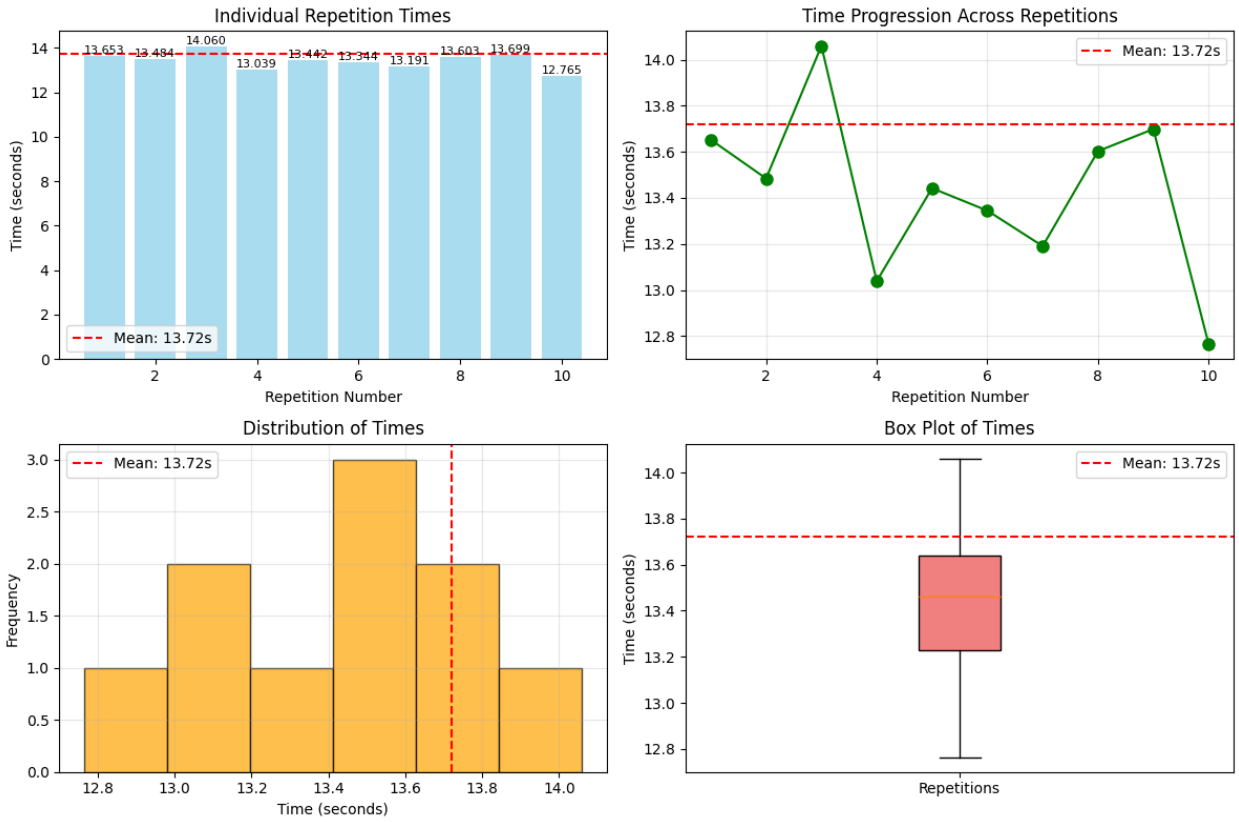
- **Multiprocessing**

En el caso del método de *multiprocessing*, el tiempo medio de ejecución fue de aproximadamente 3,12 segundos, como se ilustra en el gráfico siguiente. Este resultado muestra un rendimiento considerablemente más rápido que el del subprocesamiento, aunque ligeramente más lento que el de asyncio. En general, el multiprocesamiento demostró ser un enfoque eficaz para paralelizar tareas, ya que ofrece un buen equilibrio entre velocidad y consistencia en todas las ejecuciones.



- **Sequential**

Por último, la ejecución de *Sequential* produjo un tiempo de ejecución promedio de 13,72 segundos, con una desviación estándar de aproximadamente 0,51 segundos, como se muestra en el gráfico siguiente. Aunque los resultados fueron consistentes, este método fue considerablemente más lento que el multiprocesamiento y el asíncrono, lo que pone de relieve sus limitaciones cuando se aplica a tareas que requieren una mayor eficiencia o que implican conjuntos de datos más grandes.



Conclusiones Parciales

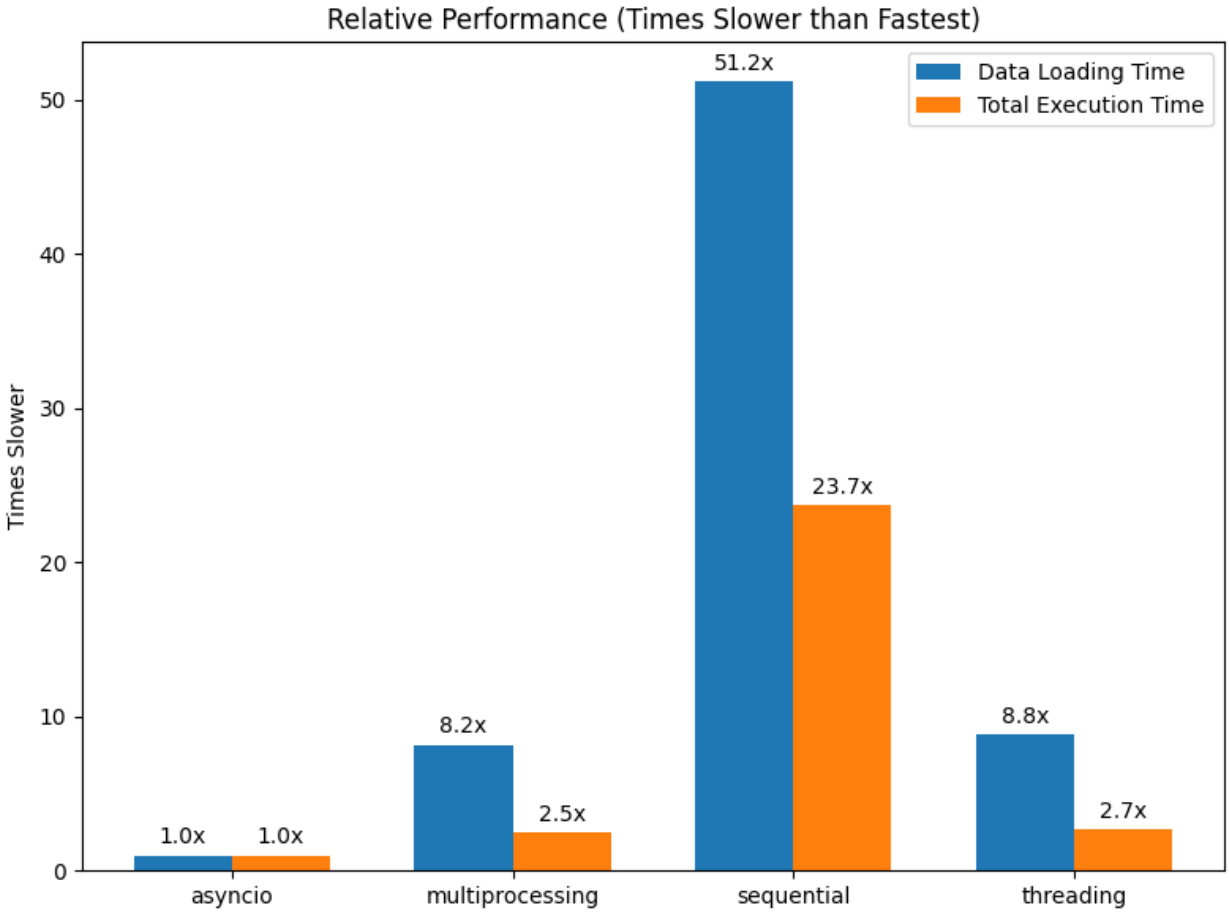
El orden de rendimiento, de más rápido a más lento, fue asyncio- multiprocessing- threading y Sequential. Si bien asyncio logró los tiempos de ejecución más cortos, el multiprocesamiento también ofreció resultados competitivos, superando claramente tanto a los threading como a la Sequential, que fueron los más lentos y mostraron tiempos de ejecución muy similares.

Según estos resultados, es más conveniente utilizar el método asyncio, ya que permite gestionar de forma eficiente las tareas simultáneas, reduce el tiempo total de ejecución y mantiene un rendimiento constante en múltiples ejecuciones. Esto lo hace especialmente adecuado para conjuntos de datos más grandes o situaciones en las que la optimización del tiempo es fundamental.

DATALOADER

Para este caso, además de concurrencia, se implementó la actividad utilizando para los mismos métodos un dataloader con el fin de optimizar la disponibilidad de imágenes con los más bajos requisitos de memoria posible. A continuación, se muestra una tabla con los resultados. Cabe aclarar, que para este caso no hubo más de una corrida para ilustrar la siguiente tabla.

Method	Data_Loading_Time	Total_Execution_Time
asyncio	1.77	7.08
multiprocessing	14.45	17.69
sequential	90.54	167.59
threading	15.65	18.78



En forma de qué tantas veces es más lento cada uno de los distintos métodos en contra , podría asimilar la diferencia es muy alta entre las diferencias entre modelos conociendo que asyncio es el más rápido, luego multiprocessing y threading compiten por el segundo puesto donde tienen resultados estadísticamente similares, y por último Sequential siendo el más lento de todos.

CONCLUSIONES

En ambas pruebas —tanto en concurrencia directa como en la implementación del dataloader— el método asyncio se consolidó como el más rápido y eficiente. No solo redujo de manera significativa los tiempos de ejecución frente a los demás, sino que además mostró consistencia en los resultados, con una baja variabilidad entre corridas. Esto lo hace altamente recomendable para escenarios donde se requiere escalabilidad y optimización de tiempo.

El método de multiprocessing obtuvo un rendimiento intermedio. Si bien fue más lento que asyncio, alcanzó resultados competitivos que lo ubican como una alternativa válida en contextos donde el paralelismo a nivel de procesos pueda aprovechar mejor los recursos del hardware disponible. En particular, resulta útil cuando las tareas a ejecutar son intensivas en cómputo más que en I/O.

Aunque threading mantuvo una baja desviación estándar, sus tiempos promedio fueron significativamente más altos que los de asyncio y multiprocessing. Esto evidencia que, para este tipo de tareas de descarga y procesamiento de imágenes, el subprocesamiento no aporta mejoras relevantes y, en algunos casos, puede ser ineficiente en comparación con otras técnicas de concurrencia.

La ejecución secuencial fue, de manera consistente, la más lenta de todas. Tanto en las pruebas iniciales como en el dataloader, presentó tiempos de ejecución muy superiores al resto de métodos. Esto confirma sus limitaciones y lo descarta como opción práctica cuando se buscan soluciones escalables y de rápida respuesta.

Finalmente, asyncio es la mejor opción para descargas masivas y procesamiento de datos en este caso, seguido por multiprocessing como una alternativa competitiva. Threading no aporta mejoras significativas y Sequential resulta el menos recomendable por su bajo rendimiento.