Comparación de Técnicas de Computación Paralela, Contenedores y Cloud Computing para Acelerar Procesos de Web Scraping

1st Cynthia Zhou
Facultad de Ingeniería
Universidad del Pacífico
Lima, Peru
cz.zhoul@alum.up.edu.pe

2nd Fabrizio Montalvo
Facultad de Ingeniería
Universidad del Pacífico
Lima, Peru
fh.montalvop@alum.up.edu.pe

3rd Paolo Salazar
Facultad de Ingeniería
Universidad del Pacífico
Lima, Peru
pc.salazarp@alum.up.edu.pe

I. Introducción

En la actualidad, la cantidad de datos disponibles en internet crece a un ritmo exponencial. Según estimaciones recientes, para el año 2025 se espera que el volumen total de datos digitales supere los 180 zettabytes, mientras que se calcula que existen más de 1.5 mil millones de sitios web, de los cuales millones son actualizados diariamente [1]. Este vasto ecosistema de información representa una fuente invaluable de conocimiento para empresas, investigadores y desarrolladores.

En este contexto, la recolección automatizada de datos desde sitios web, conocida como web scraping, se ha convertido en una herramienta esencial en aplicaciones que van desde el análisis de mercado hasta la investigación académica. Sin embargo, a medida que aumentan los volúmenes de datos y la complejidad de las páginas web, las estrategias tradicionales de scraping secuencial presentan limitaciones significativas en cuanto a rendimiento, escalabilidad y eficiencia en el uso de recursos. [2]

Para abordar estos desafíos, han emergido enfoques modernos que aprovechan tecnologías como: (i) la computación paralela, que distribuye las tareas de scraping entre múltiples hilos o núcleos de procesamiento para reducir los tiempos de ejecución; [3] (ii) el uso de contenedores (por ejemplo, mediante Docker), que permite implementar entornos portables, consistentes y eficientes para ejecutar scrapers de forma aislada; [4] y (iii) la computación en la nube, que brinda recursos escalables bajo demanda, ideales para adaptarse a cargas de trabajo variables sin depender de infraestructura local. [5]

Este proyecto tiene como objetivo comparar y analizar el rendimiento de estas tres técnicas —computación paralela, contenedores y computación en la nube— aplicadas al proceso de web scraping. Como caso de estudio, se utilizará la extracción de datos estructurados sobre jugadores de fútbol desde una plataforma web pública. Se evaluarán aspectos clave del rendimiento como el tiempo de ejecución, el uso de CPU y memoria, la escalabilidad del sistema y la facilidad de implementación. A través de esta comparación, se busca proporcionar una guía práctica para seleccionar la estrategia más adecuada según los requerimientos específicos de cada proyecto.

II. Estado del arte

La optimización de procesos de scraping mediante técnicas de computación de alto desempeño ha sido ampliamente explorada en los últimos años. Un ejemplo representativo de esta línea de investigación es el trabajo de Guang Sun, Huanxin Xiang y Shuanghu Li, titulado "On Multi-Thread Crawler Optimization for Scalable Text Searching" (2019). En este estudio [6], los autores proponen un optimizador multihilo para crawlers enfocados en grandes bases de datos de texto, utilizando como caso de prueba Wikipedia. El enfoque combina técnicas de búsqueda en anchura (BFS) y en profundidad (DFS) para aumentar la eficiencia de la recolección de datos. Sus experimentos demuestran que el uso de multithreading reduce drásticamente el tiempo de scraping en comparación con una implementación secuencial, bajando de aproximadamente 1000 segundos a 273 segundos en pruebas específicas.

Complementariamente, el estudio de Ryan Woodall, Douglas Kline, Ron Vetter y Minoo Modaresnezhad, titulado "A Cloud-based System for Scraping Data from Amazon Product Reviews at Scale" (2022), presenta una solución basada en arquitectura en la nube para extraer información de reseñas de productos en Amazon. [7] El sistema utiliza un servicio de scraping de terceros que comunica las tareas completadas a través de una Azure Function. Posteriormente, los datos se almacenan en Azure Data Lake y se transforman mediante SQL en una base de datos relacional. Esta arquitectura orientada a microservicios resalta por su modularidad, escalabilidad y facilidad de integración con flujos ETL, y se plantea como una alternativa robusta frente a soluciones puramente locales.

Finalmente, el trabajo de A. Prusty et al., titulado "Horizontally Scalable Web Crawler using Containerization and a Graphical User Interface" (2020), explora una arquitectura basada en Docker y Kubernetes para la contenerización de procesos de scraping. En este diseño [8], cada contenedor ejecuta una instancia aislada del scraper, coordinada desde una interfaz

gráfica que permite a los usuarios definir parámetros como profundidad del crawling y palabras clave. La escalabilidad horizontal se logra desplegando nuevos contenedores por cada solicitud del usuario, eliminando los cuellos de botella tradicionales de los sistemas monolíticos. Esta estrategia resulta útil tanto en entornos académicos como empresariales donde se requiere ejecutar múltiples tareas de extracción en paralelo sin interferencia entre sí.

En conjunto, estos trabajos ofrecen una base teórica y práctica para comparar enfoques modernos de scraping: multihilo, contenedores y computación en la nube frente a metodologías secuenciales, validando empíricamente sus beneficios en escalabilidad, eficiencia y flexibilidad operativa.

III. Marco Teórico

En la presente investigación, se hará uso de diferentes tecnologías y conceptos clave los cuales serán explicados a detalle a continuación.

1) **Web Scraping**: El web scraping es una técnica de programación mediante la cual se puede automatizar el proceso de extracción de datos de la web. Gracias a esta técnica, es posible obtener información estrcuturada de las diferentes página web con formato HTML. [9] Esta herramienta es esencial en la actualidad porque permite extraer grandes cantidades de datos de manera eficiente.
2) **Computación Paralela**: La computación paralela es una técnica que permite aprovechar todos los recursos de una computadora para poder realizar múltiples operaciones al mismo tiempo. Esto se logra mediante la distribución de tareas a los distintos núcleos y procesadores disponibles en un dispositivo. De esta manera, se puede dividir una tarea compleja y tardada en subtareas más pequeñas, siempre y cuando se puedan ejecutar de manera paralela, lo que mejora significativamente el rendimiento y el tiempo de ejecución de los algoritmos. [10]
3) **Contenedores de Docker**: Un contenedor es una unidad de software que empaqueta el código de una aplicación junto a su configuración y dependencias. Mediante el sistema de contenedores de Docker, es posible distribuir lo recursos de un dispositivo para crear múltiples contenedores capaces de realizar diferentes tareas en simultáneo. [11] A diferencia de las máquinas virtuales, los contenedores de Docker no requieren un sistema operativo por cada instancia, lo que los hace mucho más eficientes para optimizar los recursos en un proceso de paralelización.
4) **Cloud Computing**: La computación en la nube es un técnica mediante la cual se hace uso de recursos computacionales externos a través de Internet. De esta manera, los usuarios ya no se limitan a los recursos de un solo dispositivo o ubicación, sino que pueden acceder a sus aplicaciones y documentos desde cualquier lugar con una conexión a Internet [12] Plataformas como Amazon Web Services y Microsoft Azure, permite que los usuarios no tengan que invertir en hardware costoso y logren ejecutar algoritmos y aplicaciones de manera remota y eficiente.

IV. Metodología

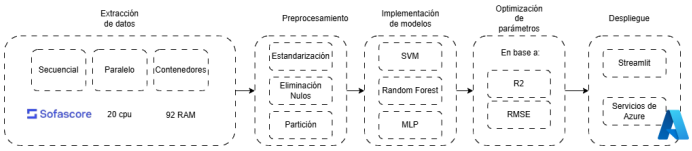

Fig. 1. Flujo de la Metodología.

A. Implementación secuencial del proceso de scraping

La primera fase del proyecto consiste en el desarrollo de una versión secuencial del scraper en Python, el cual actúa como línea base para todas las pruebas comparativas. Este script recorre un rango de identificadores numéricos únicos que corresponden a perfiles de jugadores en el sitio web Transfermarkt, accediendo a cada página mediante solicitudes HTTP. A través de la librería BeautifulSoup, se procesan las respuestas HTML para extraer información estructurada como nombre, edad, club, nacionalidad, posición y valor de mercado. Este enfoque no implementa ningún tipo de paralelismo ni optimización de infraestructura, por lo que su rendimiento representa el punto de partida frente al cual se evaluarán las demás técnicas.

B. Paralelismo local con hilos y procesos

La segunda etapa consiste en la implementación de una versión paralela del scraper utilizando las herramientas ThreadPoolExecutor y ProcessPoolExecutor de la biblioteca estándar concurrent.futures de Python. Estas implementaciones permiten ejecutar múltiples instancias concurrentes del proceso de scraping, distribuyendo los identificadores entre distintos hilos o procesos. Se realizaron pruebas experimentales con distintas configuraciones (2, 4, 8 y 16 workers) para evaluar el impacto del grado de paralelismo en el tiempo de ejecución. Esta fase permite determinar el punto de eficiencia óptimo en entornos locales y corroborar la mejora respecto al enfoque secuencial, especialmente en tareas intensivas en entrada/salida.

C. Contenerización con Docker

En la tercera fase, se desarrollará una imagen Docker que encapsula todo el entorno de ejecución del scraper, incluyendo dependencias, scripts y configuraciones. Esto permitirá ejecutar múltiples instancias en paralelo de forma aislada, asignando a cada contenedor un subconjunto específico del rango de IDs a procesar. Se experimentará con el lanzamiento de 1, 3, 5 y 10 contenedores simultáneamente en un mismo sistema, evaluando la eficiencia y escalabilidad horizontal del proceso. Esta etapa demuestra los beneficios de la contenerización en términos de portabilidad, organización del entorno y facilidad para distribuir la carga de trabajo.

D. Despliegue en Microsoft Azure

La cuarta etapa se centra en la evaluación del scraper desplegado en la nube utilizando Microsoft Azure. Se configurarán instancias virtuales en Azure Virtual Machines (VMs) con características equivalentes a los entornos locales, y se ejecutarán tanto la versión secuencial como las variantes paralelas y contenerizadas del scraper. Esta etapa permite analizar no solo el rendimiento en un entorno remoto, sino también el costo estimado de procesamiento por hora, la elasticidad de los recursos, y la facilidad de automatización del entorno de pruebas. Asimismo, se documentan los procedimientos de despliegue, configuración de red, y supervisión de cargas. El objetivo principal es validar que las soluciones diseñadas sean efectivas y replicables en infraestructuras en la nube.

E. Diseño experimental y métricas de evaluación

Para garantizar la validez comparativa de las pruebas, se utilizará un conjunto homogéneo de datos en todos los escenarios: un rango de identificadores de jugadores dentro del sitio Transfermarkt. En cada técnica se realizan al menos tres ejecuciones por configuración y se tomarán promedios de las métricas observadas. Las variables a evaluar son: tiempo total de ejecución, cantidad de registros procesados por segundo, uso promedio de CPU y RAM, facilidad de escalamiento y portabilidad. En el caso del entorno Azure, se incluye el costo estimado en base al tiempo de ejecución y tipo de instancia utilizada. Esta metodología estandarizada permite contrastar de forma objetiva los beneficios y limitaciones de cada enfoque.

V. Resultados

A. Tiempo de ejecución

Para realizar la comparativa de estrategias, se utilizó un dispositivo con las siguientes especificaciones:

- Procesador: Intel Core i9-7900X
- Sistema Operativo: Windows 11
- Memoria RAM: 16 GB
- Cantidad de procesadores: 16

Luego de ejecutar el scraping mediante la lógica secuencial y la lógica en paralelo con 32 threads se obtuvieron los siguientes resultados:

Medida	Secuencial	Paralelo
Jugadores procesados	1000	1000
Tiempo de ejecución	70 minutos	2.69 minutos
Tiempo por jugador	4.2 segundos	0.16 segundos

TABLE I
Comparación de tiempo de ejecución y tiempo por jugador entre los enfoques secuencial y paralelo

Finalmente calculamos que el speedup resultante luego de optimizar el código mediante el multithreading fue de 26.02 aproximadamente.

References

[1] Domo Inc., *Data Never Sleeps 11.0*, 14 de diciembre de 2023.

[2] R. Mitchell, *Web Scraping with Python: Collecting More Data from the Modern Web*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2018.

[3] A. Grama, A. Gupta, G. Karypis, y V. Kumar, *Introduction to Parallel Computing*, 2ª ed., Boston, MA, EE. UU.: Addison-Wesley, 2003.

[4] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux J.*, vol. 2014, no. 239, Art. no. 2, Marzo 2014. [En línea]. Disponible en: https://dl.acm.org/doi/10.5555/2600239.2600241

[5] M. Armbrust *et al.*, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, Abril 2010. [En línea]. Disponible en: https://doi.org/10.1145/1721654.1721672

[6] G. Sun, H. Xiang, y S. Li, "On Multi-Thread Crawler Optimization for Scalable Text Searching," *Journal on Big Data*, vol. 1, no. 2, pp. 89–106, 2019. [En línea]. Disponible en: https://pdfs.semanticscholar.org/1545/432272c6c4c352416f7ad29bcfb26550d840.pdf

[7] R. Woodall, D. Kline, R. Vetter, y M. Modaresnezhad, "A Cloud-based System for Scraping Data from Amazon Product Reviews at Scale," *Journal of Information Systems Applied Research*, vol. 15, no. 3, pp. 24–31, 2022. [En línea]. Disponible en: https://jisara.org/2022-15/n3/JISARv15n3.pdf#page=24

[8] A. Prusty, O. Mejia, A. Shah, P. Kancherlapalli, A. Suresh, y R. Schiebel, "Horizontally Scalable Web Crawler using Containerization and a Graphical User Interface," *International Journal of Engineering Research & Technology (IJERT)*, vol. 9, no. 5, pp. 292–297, 2020. [En línea]. Disponible en: https://pdfs.semanticscholar.org/b8dd/7f8a92065fbb93b1306b37730096303067fa.pdf

[9] M. A. Khder, "Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application," *Int. J. Advance Soft Compu. Appl.*, vol. 13, no. 3, pp. 145–160, 2021. [En línea]. Disponible en: doi.org/10.15849/IJASCA.211128.11

[10] W. P. Petersen y P. Arbenz, *Introduction to Parallel Computing*, Oxford University Press, 2004.

[11] E. Casalicchio and V. Perciballi, "Measuring Docker Performance: What a Mess!!!," in *Proceedings of the ICPE '17 Companion*, L'Aquila, Italy, 2017, pp. 81-84, [En línea]. Disponible en: doi.org/10.1145/3053600.3053605.

[12] S. P. Mirashe and N. V. Kalyankar, "Cloud Computing," *Journal of Computing*, vol. 2, no. 3, pp. 78-82, Mar. 2010. [En línea]. Disponible en: https://arxiv.org/abs/1003.4074