Universidad CENFOTEC

Maestría en Ingeniería del Software con énfasis en Arquitectura y Diseño

Trabajo Final de Graduación

Tema:

Recomendación de prácticas de desarrollo de software, en una arquitectura de microservicios, que sean ambientalmente sostenibles

Caso de uso hipotético: sistema de compra de tiquetes en línea

Estudiante:

Christian Aarón Campos Lucas

Fecha: Julio, 2023

Declaratoria de derechos de autor

Yo, Christian Aarón Campos Lucas, en calidad de titular de los derechos de autor del presente trabajo final de graduación, hago la siguiente declaratoria:

- La presente investigación ha sido realizada para optar por el grado de máster, en la carrera de ingeniería del software con énfasis en arquitectura y diseño, en la Universidad Cenfotec.
- Este trabajo es absolutamente original y auténtico y las fuentes o referencias citadas en esta investigación están debidamente identificadas y aparecen en la sección de bibliografía.
- Como autor de la investigación, autorizo a la Universidad Cenfotec para que consulte y utilice este trabajo con fines exclusivamente académicos y educativos.
- Autorizo el uso parcial o total de la investigación en trabajos futuros, reclamando el derecho de ser reconocido como autor de la obra y ser referenciado como tal.
- No se permite el uso comercial ni la reproducción o distribución con fines lucrativos sin mi consentimiento previo y por escrito, usando mi información de contacto que previamente he proveído a la Universidad Cenfotec.

Agradezco su atención y consideración.

Atentamente,

Christian Aarón Campos Lucas

Hoja de aprobación de proyecto

Universidad Cenfotec
Carrera de Postgrado
Maestría Profesional en Desarrollo de Software

TRIBUNAL EXAMINADOR

Este proyecto fue aprobado por el Tribunal Examinador de la carrera: **Maestría Profesional en Desarrollo de Software,** requisito para optar por el título de grado de **Maestría,** para el estudiante: **Campos Lucas Christian Aarón.**

DANIEL ENRIQUE FLORES CORDERO (FIRMA)
Firmado digitalmente por DANIEL ENRIQUE FLORES CORDERO (FIRMA)
Fecha: 2023.07.04 17:36:21 -06'00'

MBA Daniel Enrique Flores Cordero
Tutor

Luis C. Naranjo Z.

Dr. Luis Carlos Naranjo Zeledón
Lector 1

IGNACIO TREJOS ZELAYA (FIRMA)
Firmado digitalmente por IGNACIO TREJOS ZELAYA (FIRMA)
Fecha: 2023.07.04 18:13:22 -06'00'

M.Sc. Ignacio Trejos Zelaya
Lector 2

Es copia fiel del original firmado digitalmente, el cual debe ser guardado junto al Documento final

San José, Costa Rica, 3 de julio de 2023

Firmada digitalmente, de conformidad con la Ley de Certificados, Firmas Digitales y Documentos Electrónicos N° 8454, destacando el artículo 9°-

Tabla de Contenidos

Índice de tablas

Índice de ilustraciones

Resumen

El cambio climático, ocasionado por el efecto invernadero, corresponde a una gran amenaza para la supervivencia de la humanidad y de muchas especies. Varias investigaciones demuestran que si la temperatura del planeta asciende a 1.5 grados centígrados en el futuro cercano, se provocarían daños irreversibles en diversos ecosistemas que perjudicarían a la humanidad.

El sector de las tecnologías de la información es gran contribuyente con la problemática mencionada anteriormente, y de hecho se estima que para el 2040, si no se toman acciones, el sector podría incrementar la emisión de los gases del efecto invernadero de un 1.6% en el 2007 a un 14% en el 2040.

Desde la perspectiva del desarrollo del software se necesitan tomar acciones para contribuir con que se desarrollen aplicaciones sostenibles. El desarrollo del software verde, se enfoca en aplicar consideraciones verdes en las actividades que forman parte de la ingeniería del software.

La presente investigación determinó que entre la comunidad de ingenieros del software aún existe mucho desconocimiento, tanto en la problemática que se ocasiona por las malas prácticas de desarrollo de software, así como en posibles soluciones que se pueden implementar.

Debido a lo anterior, en este trabajo el enfoque consistió en buscar formas de identificar, por medio de mediciones del uso de los recursos del hardware, cuáles prácticas de desarrollo de software podrían repercutir negativamente en el consumo energético de una aplicación basada en microservicios.

Para lograr realizar esas mediciones, se utilizaron algunas herramientas, como Grafana, Prometheus y JMeter con el fin de realizar pruebas de carga y además, monitorear el comportamiento de los recursos del hardware ante las distintitas ejecuciones de pruebas con diferentes prácticas de desarrollo de software.

Los resultados obtenidos demostraron que existen lenguajes de programación que consumen más energía que otros, e incluso, entre versiones del mismo lenguaje también existen diferencias en el uso de los recursos de hardware.

Además, se estudió la relación que existe entre el rendimiento de una aplicación y su impacto en el hardware, y se logró identificar que, si bien en algunos escenarios a mayor rendimiento, mejor uso de los recursos del sistema, en otros escenarios, como por ejemplo, en el uso de hilos o de cache, el porcentaje del uso del hardware se incrementó.

Las investigaciones y pruebas se realizaron teniendo en mente un caso hipotético de venta de entradas en línea, para lo cual, se desarrolló un microservicio a modo de prototipo con el cual realizar las experimentaciones y mediciones.

Palabras clave: Desarrollo de software verde, consumo energético, prácticas de desarrollo de software sostenible, eficiencia energética del software.

Capítulo 1. Introducción

1.1 Generalidades

El desarrollo de la investigación se hará teniendo en mente una aplicación, que se desea implementar en el futuro, como parte de un emprendimiento del autor. Por este motivo se realizará un prototipo que sea útil para la experimentación.

Las propuestas de esta investigación, en el desarrollo de software, tienen la intención de ayudar a resolver problemas de sostenibilidad.

Al no conocer los diferentes contextos que puede tener el desarrollo de una aplicación en general, la presente investigación plantea un análisis en un caso hipotético específico: una aplicación de compra de tiquetes en línea.

Por lo tanto, las recomendaciones resultantes de esta investigación están basadas para este caso específico. De realizar un análisis en otro caso, las pruebas en los anexos del 8 al 16 deberán ser ejecutadas nuevamente.

1.2 Antecedentes del Problema

El cambio climático corresponde a una amenaza para la supervivencia de la humanidad y de muchas especies, es por ello que distintas organizaciones han levantado la voz, como un ruego desesperado de que el curso de acción actual pueda ser cambiado, con el fin de preservar el medio ambiente para las futuras generaciones y evitar daños severos que podrían ser irreversibles.

Basado en lo anterior, a lo largo de los años, se han creado organizaciones y se han implementado medidas que se enfocan en dirigir los esfuerzos de los cambios que se deben realizar (United Nations Climate Change, n.d.), con el fin de evitar que los daños ocasionados en el ambiente, producto de la actividad humana, puedan llegar a un punto irreversible.

Por ejemplo, en el año 1994, se creó la Convención Marco de las Naciones Unidas sobre el Cambio Climático (UNFCCC por sus siglas en inglés) en la cual participaron 166 países y ha sido ratificada por 198 a la fecha. Su propósito primordial es evitar que la humanidad siga interfiriendo negativamente en el ambiente. A partir de entonces, y hasta la fecha, se han tomado decisiones que buscan disminuir el impacto negativo de la interacción humana con el medio ambiente. (United Nations Climate Change, n.d.)

A pesar de que la industria de las tecnologías y comunicaciones tienen su impacto positivo en la humanidad, también tienen consecuencias negativas en el ambiente. De hecho, muchas acciones se están tratando de implementar ,con el fin de poder contribuir con la meta común de mejorar el calentamiento global, sin embargo, aún queda mucho camino por recorrer.

Asimismo, desde la perspectiva del desarrollo de software, se han realizado varias investigaciones que han demostrado los efectos negativos, que esta rama de la tecnología, genera en el medio ambiente, y se ha tratado de definir cómo mitigar ese impacto.

Tomando en cuenta lo anterior, la comunidad de ingenieros del software y grandes corporaciones a nivel mundial, han tomado conciencia con el fin de reducir el impacto

negativo generado por el software en el medio ambiente. Por eso, algunas fundaciones han sido creadas, como por ejemplo, la *Green Software Foundation* (Green Software Foundation, n.d.) y se han realizado estudios que permiten identificar las acciones que la comunidad de ingenieros del software debe implementar, con el fin de desarrollar software que sea más amigable con el ambiente, es decir, energéticamente sostenible.

Por ejemplo, entre el año 2000 y el año 2018, se habían realizado más de 500 publicaciones con respecto al desarrollo del software sostenible o software verde (Calero, et al., 2019), y con eso se logra demostrar que se han realizado amplias investigaciones, que tienen el objetivo de ayudar a la comunidad de ingenieros, a mejorar las prácticas de desarrollo de software y proponer soluciones.

En la actualidad se ha publicado amplia literatura, como resultado de las investigaciones que se han realizado y las fundaciones que se han creado. Por ejemplo, el autor Tim Frick escribió un libro llamado *Design For Sustainability* (Frick, 2016), el cual se enfoca en demostrar cómo se logra el diseño y el desarrollo de productos y servicios digitales, de tal forma que sea más sostenible y amigable con el ambiente.

Sin embargo, a pesar de la amplia información que se ha puesto a disposición del público en general y de toda la literatura que existe actualmente, aún queda mucho trabajo por realizar para despertar la conciencia de la comunidad de profesionales de tecnología y especialmente en los desarrolladores e ingenieros del software.

1.3 Definición y Descripción del Problema

1.3.1 Efecto Invernadero

El daño ocasionado al ambiente en los últimos años, debido a la expansión de la actividad industrial, ha causado que la temperatura global del planeta haya aumentado, al menos en un grado centígrado, desde la época preindustrial y, de acuerdo a recientes investigaciones, si el nivel de la temperatura global asciende a 1.5 grados centígrados en el futuro cercano, se provocarían de manera inevitable, graves peligros en muchos ecosistemas y en la humanidad en general (IPCC, 2022).

El efecto invernadero, es el proceso que mantiene la temperatura global de la tierra en los niveles adecuados para permitir la vida, ya que evita que el calor del planeta se vaya al espacio. Lo anterior se logra con una combinación de gases que incluyen el dióxido de carbono (CO2).

Sin embargo, producto de la actividad humana en distintas actividades, se han acumulado niveles más altos de los gases de efecto invernadero en la atmósfera, lo cual provoca un sobrecalentamiento en la temperatura global de la tierra (Kweku, et al., 2018).

1.3.2 Impacto negativo del área de las Tecnologías de la Información y Comunicación

El área de las Tecnologías de la Información y Comunicaciones (TIC) ha influido positivamente a casi todas las disciplinas y campos a nivel mundial, y su contribución a la humanidad ha sido trascendental, pero desafortunadamente, no se puede ignorar que

todos esos avances tecnológicos que se disfrutan hoy, han causado un impacto negativo en el medio ambiente.

Basado en algunos datos expuestos en el acuerdo de París, se puede concluir que los gases del efecto invernadero son ocasionados por varios sectores económicos, de los cuáles algunos son catalogados como grandes contribuyentes. Por ejemplo, el sector de la producción de energía, contribuye con un 29% y el sector de transporte con un 27% del total de la producción de gases del efecto invernadero.

En el caso del sector de las TIC, al ser dependiente del consumo energético para funcionar en áreas como, las telecomunicaciones, el *Cloud Computing*, los *datacenters* entre otros, también deja su huella de carbono.

De hecho se estima que para el 2040, si no se toman acciones, el sector podría incrementar la emisión de los gases del efecto invernadero de un 1.6% en el 2007 a un 14% en el 2040 (Belkhir & Elmeligi, 2018).

1.3.3 Impacto negativo del software

Una de las áreas del sector de las Tecnologías de la Información y Comunicaciones, que representa una potencial amenaza, es el de la ingeniería del software. Esto es debido a que el software, a pesar de ser un producto inmaterial, necesita de recursos materiales como por ejemplo CPU, memoria y otros componentes del hardware para funcionar.

Por ejemplo, entrenar un modelo de procesamiento de lenguaje natural podría emitir más de 626 000 libras (283.9 toneladas) de CO2 al ambiente, el equivalente a las

emisiones de CO2 que producen 5 carros en toda su vida útil incluyendo el proceso de manufactura. (Strubell, Ganesh, & McCallum, 2019)

Adicionalmente, según datos de la Universidad de Cambridge, en su sitio Cambridge Bitcoin Electricity Consumption Index, la cantidad de energía total que genera el bitcoin en un año es de 94.38 TWh, más de lo que generan países como Finlandia, Colombia, Venezuela, Suiza, Bélgica, entre otros. (University of Cambridge Judge Business School, 2023)

Ilustración 1: Demanda de energía de la red Bitcoin. Fuente: CBECI

Ante la evidencia presentada anteriormente, la comunidad de ingenieros del software tiene la responsabilidad de tomar conciencia, con el fin de diseñar código que sea capaz de aprovechar los recursos de hardware de una manera óptima y, de este

modo, evitar el consumo de energía excesivo e innecesario, energía que, generalmente, es producida por fuentes no renovables.

1.4 Justificación

1.4.1 Relevancia Social

Tomando como contexto la problemática descrita anteriormente, el presente trabajo pretende servir de inspiración para que, profesionales del área de la ingeniería del software, puedan ser influenciados a emular las prácticas y recomendaciones que se sugerirán en esta investigación y además, puedan aprender y conocer, que existen formas en las cuáles se puede desarrollar el software de una manera más amigable con el ambiente.

El desarrollo de software verde pretende crear código que sea altamente efectivo en su utilización de los recursos del hardware y, que durante su diseño, creación y utilización, se logre el objetivo de disminuir el consumo energético.

La concientización, que el presente trabajo pretende despertar en los profesionales de la industria del software, ayuda a que como gremio, se pueda participar en el cambio que se requiere para ayudar al medio ambiente y evitar, de esta forma, que el calentamiento global traiga consecuencias negativas a la humanidad.

1.4.2 Implicaciones prácticas

La práctica del desarrollo de software verde, además de ayudar al medio ambiente, puede tener consecuencias positivas en el rendimiento de las aplicaciones

que se desarrollen, mejorando de esta forma la experiencia de los usuarios que las utilicen.

Lo anterior es debido a que se busca que se apliquen prácticas de desarrollo de software que sean altamente efectivas por medio de la optimización del uso de los recursos del hardware y de los procesos del software.

Como consecuencia, la aplicación del desarrollo de software verde, podría sugerir un aumento en el rendimiento de las aplicaciones y posiblemente, en muchas ocasiones, una disminución en los costos operativos.

1.4.3 Valor teórico

Esta investigación ofrece brindar información teórica valiosa, ya que aún no hay suficiente documentación con respecto a prácticas de desarrollo de software sostenibles, especialmente en aplicaciones con un perfil similar al que se propone en la presente investigación.

Además, el presente trabajo, ayudará a crear las bases para las futuras investigaciones que tengan casos de uso similares al de la aplicación propuesta, y también, se brindará documentación que será provechosa para expandir y ampliar en trabajos futuros.

1.4.4 Carácter innovador

El desarrollo de software verde es una disciplina que se ha investigado desde hace varios años, sin embargo, su popularidad no ha alcanzado los niveles que se esperan para llegar a ocasionar un impacto significativo en el medio ambiente.

El trabajo actual tiene la intención de realizar comparaciones de distintas prácticas de desarrollo de software, que puedan ser útiles en arquitecturas de microservicios, en diseños basado en eventos y en diferentes enfoques que son útiles para garantizar el rendimiento ideal de la aplicación propuesta en esta investigación.

El resultado de las comparaciones mencionadas anteriormente, podrá ser explotado por otros investigadores, y de esta forma enriquecer y ampliar los estudios que se tienen actualmente.

1.5 Pregunta de investigación

¿Cuáles son las prácticas de desarrollo de software que se deben implementar en una aplicación, que usa la arquitectura de microservicios, para que sea energéticamente sostenible?

1.6 Objetivos

Se ha usado la taxonomía original de Bloom de 1956 para plantear los objetivos de esta investigación, debido a su robustez y uso como estándar de facto en el sistema educativo costarricense.

1.6.1 Objetivo General

Recomendar prácticas de desarrollo de software, en una arquitectura de microservicios, que sean ambientalmente sostenibles en el caso hipotético: sistema de compra de tiquetes en línea.

1.6.2 Objetivos Específicos.

1. Seleccionar los recursos del sistema que serán medidos, mediante el análisis de la documentación referente al consumo energético, para el establecimiento de métricas verdes.
2. Evaluar distintas prácticas de desarrollo de software, mediante la comparación de los resultados obtenidos usando las métricas verdes, para el establecimiento de recomendaciones sostenibles.

1.7 Alcances y Limitaciones

1.7.1 Alcances

El presente trabajo, buscar sugerir prácticas de desarrollo de software que puedan ser útiles en el prototipo de la aplicación hipotética, por lo tanto, la validez de las pruebas y los resultados se circunscriben en el caso de uso sugerido en esta investigación.

Además, según el ciclo de vida del software, las etapas que serán objeto de investigación corresponden a las fases de desarrollo y pruebas solamente. Por lo que, el

levantado de requerimientos, diseño, mantenimiento o despliegue, estarán fuera del alcance.

Asimismo, no se realizarán pruebas en todas las prácticas de desarrollo de software, ni tampoco en cada una de las tecnologías, lenguajes de programación, paradigmas o servicios que existen en el mercado, sino que se hará enfoque en las que apliquen para el diseño del prototipo de la aplicación hipotética, tomando en cuenta los requerimientos de la misma.

Cabe destacar que, tampoco se pretende realizar las pruebas en múltiples dispositivos de hardware, por ejemplo, diferentes modelos de CPU o sistemas operativos, entre otros, ya que la complejidad del diseño de las mismas podría elevarse y requerir más tiempo del que se tiene disponible para realizar las propuestas.

Finalmente, no está dentro del alcance de la investigación actual, validar los resultados obtenidos en otras aplicaciones similares, debido a que, podrían existir diferencias importantes en los requerimientos funcionales y no funcionales, en las tecnologías utilizadas y en la arquitectura de software diseñada, que deberán ser validados en trabajos futuros.

1.7.2 Limitaciones

Una limitación importante en el desarrollo del presente proyecto, es la disponibilidad limitada de diversos recursos de hardware que faciliten la realización de las pruebas y, de esta forma, evaluar distintos comportamientos en otros dispositivos. La

limitante incluye, carencia de diversos modelos, marcas y fabricantes de: computadoras, memoria RAM, CPU, entre otros. Lo anterior se pretende mitigar por medio de la documentación de la metodología utilizada con el fin de que pueda ser aplicada en otros ambientes.

Cabe resaltar que, existe una limitación importante en cuanto a las mediciones del consumo energético de los procesos específicos de una computadora. Lo anterior es debido a la carencia de herramientas confiables y especializadas para la medición exacta del consumo energético de uno o varios procesos de una aplicación. Por lo tanto, es posible que la medición del consumo energético, se base en estimaciones o mediciones generales del comportamiento del hardware ante la aplicación de diversas prácticas de desarrollo de software.

Es importante destacar que, esta investigación presenta una limitación en cuanto a la aplicación de todas las prácticas de desarrollo de software en el prototipo propuesto. Lo anterior se debe a, la falta de conocimiento exhaustivo en todas las prácticas, herramientas y lenguajes de programación que existen en el campo. Debido a esto, es posible que algunas prácticas, herramientas o lenguajes no sean incluidos en este estudio, sino que se dejarán para trabajos futuros, sin embargo, se incluirán en este trabajo prácticas y herramientas relevantes y ampliamente aceptadas.

Finalmente, la limitación del tiempo disponible para la experimentación y la evaluación de las distintas prácticas de desarrollo de software, y su impacto en el consumo energético, podría influir en la profundidad y amplitud de los casos de uso, variables y distintos escenarios, que podrían presentarse en el prototipo de la aplicación hipotética. Sin embargo, a pesar de esta limitante, se buscará demostrar una

metodología que pueda ser replicada en otras prácticas de desarrollo de software, con distintas variables.

1.8 Viabilidad

1.8.1 Punto de Vista Técnico

La investigación se realizará haciendo uso de la documentación disponible, relacionada al desarrollo del software verde, con el propósito de seleccionar herramientas de software idóneas que faciliten una medición adecuada de las métricas o parámetros necesarios.

El prototipo de la aplicación que servirá como modelo para esta investigación, será desarrollado conforme se vaya decidiendo cuáles componentes del software y de la arquitectura demuestren un mejor resultado en cuanto al uso eficiente de los recursos de hardware y por ende a una mayor sostenibilidad.

1.8.2 Punto de Vista Operativo.

Debido a que, la investigación gira en torno a una aplicación hipotética, no se tienen restricciones que puedan impedir el progreso de la investigación que se pretende realizar.

Adicionalmente, no existen dependencias con personal humano o con equipamiento específico de algún patrocinador, que sugieran bloqueos durante el transcurso de la investigación.

1.8.3 Punto de Vista Económico.

Para evitar costos en el desarrollo de la investigación, se utilizará documentación de acceso público para indagar sobre la información requerida, además, las pruebas se realizarán haciendo uso de herramientas gratuitas, o en su efecto, herramientas pagas en su versión gratuita.

Finalmente, dado que trabajará en un prototipo y no en aplicación completa, no se incurrirán en gastos operativos.

Capítulo 2. Estado de la cuestión

2.1 Estado de la Cuestión

Se han realizado numerosos estudios e investigaciones, que pretenden demostrar cuales son las prácticas de desarrollo de software amigables con el ambiente, por lo cual, con el fin de aprovechar esos recursos de una mejor manera, se seguirá el plan descrito en los siguientes apartados.

2.1.1 Planificación de la revisión

En esta etapa se plantean algunas interrogantes, la primera con la intención de justificar la problemática descrita, por lo tanto, la investigación se realizará con el enfoque de encontrar respuestas a la siguiente pregunta: ¿Cuál es el impacto negativo que el software genera en el medio ambiente?

Una vez justificada la problemática para la comprensión de los lectores, se presenta la siguiente pregunta, la cual, tiene el objetivo de obtener información que ayude a resolver la problemática anteriormente descrita: ¿Cuáles son las prácticas de desarrollo de software que se deben implementar en una aplicación de microservicios para que sea amigable con el ambiente?

Ante esta interrogante se busca entender, lo mejor posible, cómo se pueden desarrollar aplicaciones que sean energéticamente sostenibles y qué aspectos son los que pueden ayudar a determinar si una aplicación es “verde” o no. Para ello se planea utilizar las siguientes palabras clave: “desarrollo de software verde”, “software verde”,

“desarrollo de software sostenible”, “eficiencia energética en software”, “consumo energético en el software” y similares.

Con el fin de validar si las prácticas de desarrollo de software encontradas son eficientes, se plantea la siguiente interrogante: ¿Cuáles son las métricas verdes que se pueden definir con el fin de medir la validez y el impacto de las prácticas de software verdes aplicadas? Las palabras clave que serán útiles para responder esta interrogante son “métricas verdes de software”, “medición de energía en el software”, “métricas sostenibles en software” y similares.

Por tal motivo, se realizará una investigación que, en primer lugar, permita responder las interrogantes planteadas anteriormente, y después, identificar hallazgos adicionales en otras investigaciones relacionadas con el tema.

Finalmente, se clasificará la información obtenida para su posterior análisis, con el fin de determinar si podrá ser usada para cumplir con los objetivos de la presente investigación.

2.1.2 Ejecución de la revisión

Con el fin de determinar cuál de las investigaciones recopiladas, ayudará a alcanzar el cumplimiento de los objetivos de esta investigación y, además, generará una mayor confianza en el lector, se filtrarán los resultados tomando en cuenta los siguientes parámetros:

- **H-index**
    - Significado: Es un índice que permite identificar la relevancia del aporte de un investigador o una revista, basado en el número de citas que recibe y no en la cantidad de investigaciones que aporte o en las revistas que lo hace.
    - Parámetro de medición: Mayor a 20

- **SJR (SCImago Journal Rank)**
    - Significado: El SJC, es el resultado del número de citaciones de peso recibidas en un año dividido entre la cantidad de documentos publicados en los últimos tres años.
    - Parámetro de medición: Mayor a 0.4

- **Citaciones por documento**
    - Significado: Es el promedio de citaciones que ha recibido un documento en un período determinado, el cual es contrastado con las citaciones de los dos años anteriores.
    - Parámetro de medición: Mayor a 2

- **Cuartiles:**
    - Significado**:** Los cuartiles son clasificaciones que se hacen que van de Q1-Q4, en la cual los del cuartil Q1 son los mejores 25% en algún ranking, el Q2 entre el 25%-50%, y así sucesivamente.
    - Parámetro de medición: Q1 o Q2 en los dos últimos años.

- **Año de publicación**
    - Significado: Fecha en la que el documento investigativo fue publicado.
    - Parámetro de medición: Después del 2019.

La anterior información ayudará a demostrar la validez y confiabilidad de la información revisada durante la investigación y así poder tener una selección de fuentes de manera más efectiva.

Con respecto a publicaciones de universidades, se cambiarán los parámetros por los siguientes:

- **Ranking de universidad**
    - Significado: Existen plataformas que permiten identificar el ranking de las universidades, por lo que se tomará en cuenta esa información con el fin de obtener el ranking que tenía la universidad al momento de realizar la publicación.
    - Parámetro de medición: El valor aceptado para la investigación actual es que la universidad se encuentre entre las mejores 200 universidades del mundo

- **Puntaje general**
    - Significado: Existen plataformas que permiten identificar el puntaje actual de la universidad dada, bajo ciertos criterios como por ejemplo: reputación académica, red internacional de investigación, entre otros.
    - Parámetro de medición: El valor aceptado para la investigación actual, es que la universidad tenga un puntaje general por encima de 80 puntos.

- **Año de publicación**
    - Significado: Fecha en la que el documento investigativo fue publicado.
    - Parámetro de medición: Después del 2018.

Finalmente, con respecto a los libros, el parámetro para determinar si será utilizado en la investigación se describe a continuación:

- **Año de publicación**
    - Significado: Fecha en la que el libro fue publicado.
    - Parámetro de medición: Después del 2013.

Cabe resaltar que, a pesar de que se espera poder aplicar todos los filtros basados en los parámetros anteriores a toda la literatura encontrada, cabe la posibilidad de que existan referencias valiosas para el presente trabajo que no cumplan con alguno, por lo tanto, bajo un análisis profundo se decidirá si se incluye basado en el valor que puede ofrecer a la presente investigación.

2.1.3 Resumen de los resultados

2.1.3.1 Pregunta uno

Con respecto a la primera pregunta: ¿Cuál es el impacto negativo que el software genera en el medio ambiente?, se realizó un análisis de literatura resultando en las siguientes referencias seleccionadas:

Artículo	H-Index	SJR	Citaciones	Cuartiles	Año
5Ws of Green and Sustainable Software	45	1	4.5	Q1	2019
Assessing ICT global emissions footprint: Trends to 2040 & recommendations.	232	1.9	10.9	Q1	2018
Global warming: Review on driving forces and mitigation.	68	0.4	2.7	Q2	2019
Energy and policy considerations for Deep learning in NLP.	-	-	-	-	2019

Tabla 1: Impacto negativo del medio ambiente. Fuente: Elaboración propia.

Con el fin de responder la primera pregunta planteada, y con la información obtenida de los artículos anteriores, se puede demostrar que la sostenibilidad está ganando amplia importancia en la tecnología y se han empezado a tomar acciones para solventar esta problemática. Sin embargo, el software ha quedado rezagado en sus esfuerzos con respecto al hardware. Además del esfuerzo que se necesita realizar en

disminuir el consumo energético del software, se debe empezar concientizando a la población, ya que existen varias partes del mundo, en la cual, no se está trabajando o investigando activamente sobre este tema. (Calero, et al., 2019)

Adicionalmente, a pesar de que la tecnología beneficia en todos los aspectos de nuestra vida, e incluso, contribuye en los esfuerzos de sostenibilidad, el otro lado de la moneda indica que, con el fin de soportar todos los nuevos dispositivos o aplicaciones que hacen nuestra vida más fácil, existe un gasto energético que contribuye negativamente en los gases de efecto invernadero. De hecho, se han realizado estimaciones que demuestran que la demanda energética, generada por los dispositivos relacionados a la tecnología de información y comunicaciones (TIC), crece exponencialmente. Por ejemplo, en un plazo de 12 años creció un 156%, y se estima que ese patrón continúe.

Otras estimaciones indican que el área de TIC, para el 2040, podría ser el responsable de más del 50% del consumo global de electricidad y, en el peor de los escenarios el responsable de más del 100% del total de consumo de electricidad. (Belkhir & Elmeligi, 2018)

Además, a pesar de que en la actualidad se desarrollan aplicaciones haciendo uso de proveedores de la nube, como por ejemplo, AWS, GCP y Azure, se podría pensar que se delega la responsabilidad del consumo energético en la empresa que brinda el servicio de la nube. Sin embargo, existen estudios que indican que esos proveedores consumen energía de fuentes no renovables, los cuales afectan negativamente al medio ambiente, por ejemplo, el porcentaje de energía que Amazon AWS genera proveniente del carbón, es seis veces más en relación a China. (Strubell, Ganesh, & McCallum, 2019)

2.1.3.2 Pregunta dos

Con el fin de responder la siguiente pregunta, ¿Cuáles son las prácticas de desarrollo de software que se deben implementar en una aplicación de microservicios para que sea amigable con el ambiente?, se realizó una investigación, después de la cuál, al aplicarse los filtros descritos anteriormente, se logró obtenerse la siguiente base de conocimiento:

Artículo	H-Index	SJR	Citaciones	Cuartiles	Año
Software development lifecycle for energy efficiency: technique and tools	172	5.09	16.062	Q1	2019
Ranking programming languages by energy efficiency	65	0.5	1.5	Q3	2021
Green and sustainable software engineering – a systematic mapping study.	-	-	-	-	2018

Tabla 2: Prácticas de desarrollo de software amigables con el ambiente. Fuente: Elaboración propia.

Con el fin de responder la segunda pregunta, los artículos anteriores pudieron sugerir algunas prácticas de desarrollo de software que podría ser energéticamente amigable con el ambiente.

Por ejemplo, algunas de las prácticas que podrían contribuir con la reducción del consumo energético de las aplicaciones podrían incluir:

- Operaciones masivas.
- Coordinación de hardware, por ejemplo, reduciendo el acceso a la memoria.
- Programación concurrente.
- Estructura de datos eficientes.
- Entre otros.

Adicionalmente, se sugiere que se pueda hacer una selección del lenguaje de programación de una forma que se pueda hacer uso de lenguajes que sean energéticamente eficientes. (Georgiou, Rizou, & Spinellis, 2019), (Pereira, et al., 2021)

2.1.3.3 Pregunta tres

Finalmente, la pregunta ¿Cuáles son las métricas verdes que se pueden definir, con el fin de medir la validez y el impacto de las prácticas de software aplicadas? arrojó las siguientes investigaciones que resultaron útiles:

Artículo	H-Index	SJR	Citaciones	Cuartiles	Año
Green algorithms: quantifying the carbon footprint of computation	128	3.9	15.6	Q1	2021

Estimation of energy consumption in machine learning. Journal of Parallel and Distributed Computing	92	1.2	5.2	Q1	2019
A systemtic literatura review on green software metrics.	-	-	-	-	2014

Tabla 3: Métricas verdes en el software. Fuente: Elaboración propia.

La base de conocimiento de la tabla anterior ofrece información que podría ser utilizada para responder la tercera pregunta. De hecho, de la literatura encontrada, se pudo determinar que existen trabajos que permiten identificar métricas relacionadas al consumo energético, y una vez identificadas, clasificarlas para determinar el propósito, el tipo de resultados de medición y el ambiente en el cual son usadas.

Adicionalmente, se logra identificar que entre los recursos de hardware más medidos por su relevancia en el consumo energético se encuentran el CPU, la memoria RAM y dispositivos de almacenamiento. La utilización de métricas con el fin de medir los recursos mencionados anteriormente, se han ido incrementando en los últimos años. (Lago, Gu, & Bozzelli, 2014)

Finalmente, existen estudios que sugieren diferentes mecanismos para estimar el consumo energético en aplicaciones y también la huella de carbono derivada de la misma. (García- Martin, Rodrigues, Riley, & Grahn, 2019)

2.1.3.4 Consideraciones adicionales

Se considera importante recalcar que en las tablas anteriores no se incluyen las investigaciones que no lograron pasar los filtros, que se han descrito con anterioridad, y que, a pesar de que se examinaron minuciosamente con el fin de determinar si ofrecían valor a la investigación actual, no lograron demostrar el valor suficiente o no se encontró respaldado de su contenido en otras investigaciones.

Adicionalmente, es importante mencionar que no se incluyen en esta sección las investigaciones que ayudaron a ampliar los conceptos y las tecnologías sobre las cuales se genere cierte grado de discusión en el trabajo actual, por lo que, se invita al lector a revisar la totalidad de los trabajos investigados en la sección de bibliografía.

2.2 Marco Conceptual

Con el fin de brindar la oportunidad al lector de tener un entendimiento más preciso de los conceptos que se mencionan a lo largo de la presente investigación, se han desarrollado explicaciones de los términos más importantes, los cuales han sido clasificados en distintos grupos y se presentan a continuación:

2.2.1 Conceptos relacionados a la sostenibilidad

Aplicaciones de software sostenible

Las aplicaciones de software sostenible o software verde, son aquellas que han aplicado los principios de sostenibilidad y que, principalmente se enfocan en reducir el consumo energético, con el fin de generar un impacto menor en los dispositivos del hardware donde se ejecuta el software.

Aplicaciones verdes son la meta de la ingeniería del software verde, la cual se enfoca en aplicar consideraciones verdes en las actividades que forman parte de la ingeniería del software. (Calero, et al., 2019)

Calentamiento global

El calentamiento global es una de las mayores consecuencias de la actividad humana en el medio ambiente, debido a que, diferentes prácticas de producción o consumo de energía no renovable, han causado una concentración incrementada de

gases de efecto invernadero, tales como, CO2, CH4, N2O (Dióxido de carbono, metano y óxido nitroso), en la atmósfera y que son el principal causante del calentamiento del planeta (Al-Ghussain, 2019).

Desarrollo de software verde

Anteriormente se mencionó cual es la meta de la ingeniería del software verde, sin embargo, la disciplina del desarrollo de software verde, tiene como objetivo crear software fiable y duradero, que satisfaga las necesidades de los clientes, pero al mismo tiempo, que ayude a reducir el impacto ambiental del mismo, por medio de la concientización del consumo energético que éste produce.

Adicionalmente, se puede decir que consiste en prácticas que permiten ligar la ingeniería del software con los principios de sostenibilidad. (Mourão, Karita, & do Carmo Machado, 2018)

Efecto invernadero

El efecto invernadero es el fenómeno que hace alusión a un proceso natural del planeta, que ayudan a mantener la temperatura de la tierra a niveles que permitan preservar la vida tal y como la conocemos actualmente. Básicamente, existen ciertos elementos químicos compuestos que están presentes en la atmósfera y debido a ellos se evita que el calor de la tierra se escape al espacio.

Sin embargo, el aumento desmedido de estos elementos compuestos en la atmósfera, como, por ejemplo, CO2, CH4, N2O (Dióxido de carbono, metano y óxido nitroso), hace que más calor del necesario se mantenga en la atmósfera, ocasionando que, la temperatura global del planeta aumente perjudicando ecosistemas y diferentes formas de vida. (Kweku, et al., 2018)

Métricas verdes

De acuerdo al diccionario en inglés de Oxford, métricas se refiere a un conjunto de números o estadísticas que son usadas para medir algo.

Por lo tanto, basado en la definición anterior el presente trabajo hace referencia a este concepto como el conjunto de indicadores o estadísticas que derivan de los diferentes componentes de la aplicación descrita, como por ejemplo, el hardware, que sean el resultado de la aplicación de prácticas verdes.

Tal y como se indica en el primer objetivo de esta investigación, se analizarán las mediciones resultantes de las prácticas de desarrollo verde, para tener la facultad de realizar un análisis estadístico que permita y facilite la toma de decisiones.

Prácticas de desarrollo de software sostenible

Son aquellas prácticas o metodologías de desarrollo de software que permiten crear aplicaciones que hagan un uso eficiente de los recursos del hardware y de esa forma, ayudar a reducir la huella de carbono emitida por el software en sus distintas

etapas del ciclo de vida. Las prácticas de desarrollo sostenible, son el objeto de estudio de la ingeniería de software verde.

2.2.2 Conceptos relacionados a la arquitectura y diseño de aplicaciones

Arquitectura de capas

Este tipo de arquitectura en el diseño del software, conocido en inglés como *Layered Architecture*, es en realidad el patrón de arquitectura más popular. La idea principal dentro de la arquitectura de capas es que los componentes del software están organizados en capas horizontales y cada una de ella realiza una tarea específica.

Por lo general, en la mayoría de aplicaciones se usan tres capas principales, las cuales son: presentación, lógica del negocio y persistencia de datos. Incluso, en las aplicaciones desarrolladas con Spring Framework, es usual ver las tres capas identificadas por distintas anotaciones que representan cada capa. (Richards, 2015)

Este tipo de patrón de arquitectura permite que existan más capas, pero para efectos de esta investigación, las mencionadas anteriormente son las que se utilizarán en el prototipo que se desarrollará.

Arquitectura hexagonal

Esta arquitectura es también conocida como puertos y adaptadores y tiene como idea principal, poder encapsular la lógica del negocio con el fin de que no interactúe con

los componentes encargados de la entrada o salida de datos, sino que cualquier comunicación con el mundo exterior se maneja por medio de puertos que interactúan con los adaptadores.

En su libro *Clean Architecture*, Robert C. Martin, describe tres componentes principales en la arquitectura hexagonal, el núcleo de la aplicación, es decir, donde se maneja la lógica de negocio, los puertos de entrada y salida, los cuales son interfaces que comunican la lógica de negocio con el mundo exterior y los adaptadores con los cuales diferentes tecnologías de entrada y salida se comunicarán con los puertos para alcanzar a la lógica de negocio. (Martin, 2018)

Arquitectura de microservicios

La arquitectura de microservicios fue propuesta con el fin de alcanzar un diseño que permita independencia entre los distintos componentes de una aplicación. La idea principal del microservicio es que, cada servicio independiente, se encargue de realizar una tarea específica, pero realizándola de la mejor forma. Existen algunas características que han sido aceptadas con respecto a la arquitectura de microservicios las cuales algunas se mencionan a continuación

- Gobernanza descentralizada, es decir, se prefiere que cada microservicio sea gobernado de forma autónoma sin dependencias de otros microservicios del sistema.

- Manejo de datos descentralizado, cada microservicio debe, preferiblemente, tener bases de datos separadas con el fin de que cada componente sea responsable de los datos que maneja.
- Diseño para fallas, esto se refiere a que un fallo en un microservicio no debe afectar el sistema en general, sino que los demás componentes deben ser capaces de continuar su funcionamiento normal.
- Componentes en torno a los servicios, lo cual se refiere que un cambio en un componente no debe requerir cambios en otros servicios.

(Vural & Koyuncu, 2021)

***Command Query Responsibility Segregation* (CQRS):**

CQRS es un patrón que se encarga de separar las responsabilidades de las operaciones realizadas en las bases de datos. Para ello se dividen en dos denominadas comando y consulta.

Comando (*Command*) son las transacciones que se encargan de mutar o cambiar los datos de la base de datos, por ejemplo, actualizar, crear o eliminar. Por otro lado, consulta (*query*) se encarga de todas las operaciones que son solo de lectura de datos.

Al separar esas responsabilidades de esta forma se permite la utilización de otros patrones como *Event Sourcing* y además se permite que las aplicaciones puedan mejorar el rendimiento ya que debido a la separación de responsabilidades se pueden optimizar las operaciones de escritura y lectura. (Zhong, Li, & Wang, 2019).

Domain Driven Design

Domain Driven Design o DDD por sus siglas, se refiere a la práctica de basar el diseño de los microservicios, tomando como enfoque principal, las diversas áreas de dominio de conocimiento de un negocio o sistema, de tal forma que cada decisión de diseño se realice entorno a las características específicas del negocio.

Con este enfoque, se toma la opinión de expertos del dominio que representan al negocio y se crea un lenguaje ubicuo, el cual es el lenguaje común que se utiliza entre todos los involucrados del sistema.

Existen ciertos conceptos o bloques de construcción que permiten desarrollar una arquitectura enfocada en el dominio, los cuales se describen a continuación:

- Dominio: Un área de interés sobre la que un experto tiene control.
- Límite del dominio: Representa los conceptos de un modelo, sus relaciones y sus reglas y puede incluir conceptos de múltiples dominios.
- Entidad: Son objetos que pueden ser creados con identidad propia.
- Objeto de valor: Un objeto que no tiene una identidad y solo son útiles para realizar ciertos cálculos o para el aprovechamiento de algún atributo.
- Evento del dominio: Representa un evento, del cual el experto del dominio se interesa.
- *Aggregate*: Es una combinación entre ciertos objetos donde hay una relación que no puede dejar de existir.

- Servicio: Se crea cuando una operación no pertenece solamente a una entidad o *aggregate*, sino que involucra más, haciendo necesario que la funcionalidad se implemente mejor como un servicio.
- Repositorio: Es la parte que se encarga de llevar a cabo las operaciones de persistencia de datos.

(Vural & Koyuncu, 2021)

Event Driven Design

Es un diseño en el cual, los sistemas que lo implementan, reaccionan ante eventos que ocurren en su entorno, por lo tanto, una arquitectura basada en eventos, está integrada por componentes o servicios que envían y reciben eventos o mensajes de forma asíncrona, que posteriormente son procesados de forma asíncrona.

2.2.3 Conceptos generales relacionados a la tecnología

Bases de datos no relaciones

En contraste con las bases de datos relacionales, las no relacionales, o bien, no SQL, ayudan a cumplir los requerimientos emergentes de los sistemas actuales con respecto a la flexibilidad, disponibilidad y fiabilidad con que se deben manipular los datos.

Este tipo de base de datos son ideales cuando se necesita manejar de forma sencilla y eficiente altos volúmenes de datos no estructurados, heterogéneos y

dinámicos. Los modelos de datos de las bases de datos no relacionales son: *key-value*, orientado a columnas, documentos y gráficos.

Las principales ventajas de este tipo de bases de datos se mencionan a continuación:

- Eficiente ejecución de operaciones de lectura y escritura.
- Baja latencia.
- Fácil de expandir.
- Bajo costo en términos de administración y operación.

(Sicari, Rizzardi, & Coen-Porisini, 2022)

Bases de datos relaciones

Una base de datos relacional, es una colección de información que organiza y almacena datos de forma estructurada en tablas con un identificador único.

Las tablas son compuestas de columnas, con diferentes tipos de valores, que le dan un significado semántico a la información que se está almacenando.

Adicionalmente, las filas representan una instancia del objeto que representa la tabla.

Las tablas pueden relacionarse entre sí por medio del uso de llaves primarias y llaves foráneas, las cuales son los identificadores usados para representar enlaces entre tablas y facilitan unir las mismas.

Este tipo de bases de datos son útiles cuando la información y los datos deben ser representados de forma estructurada y cuando se requiere un modelo que provea una forma intuitiva de representar los datos y sus relaciones.

Una de las mayores ventajas es que estas bases de datos cumplen con los principios ACID (*Atomicity, Consistency, Isolation, Durability*), en español, atomicidad, consistencia, aislamiento y durabilidad, lo cual es beneficioso en términos de rendimiento, y en asegurar la validez de los datos independientemente de errores o posibles fallas. (Google Cloud., n.d.)

Batches

Un *batch* en el desarrollo de software, es un conjunto de tareas, los cuales, son ejecutados en una sola sesión o transacción, en lugar de que se efectúen en procesos individuales. Se utiliza para completar trabajos repetitivos o de alto volumen y son bastante útiles cuando es ineficiente, o requiere muchos recursos del sistema, ejecutar transacciones individuales. (Amazon Web Services, Inc., n.d.)

Ciclo de vida del software

Se refiere al proceso que debe de ocurrir para la creación de productos de software, el cual se divide en diferentes fases que son útiles para asegurarse que el resultado final sea el esperado y garantizar una buena calidad del producto por medio de una planificación desde el inicio del proyecto.

Existen diferentes fases y cada organización decide cuales etapas incluir en sus procesos y cuales no, pero por lo general consiste de las siguientes etapas:

1. Planeamiento
2. Análisis
3. Diseño
4. Implementación
5. Pruebas
6. Despliegue

(Amazon Web Services, Inc, n.d.)

Hilos

En el desarrollo del software, los hilos son subprocesos con distintas responsabilidades de ejecución que coexisten dentro de un proceso. Son útiles para alcanzar la concurrencia y el paralelismo en aplicaciones complejas.

Cuando son utilizados de forma correcta, los hilos pueden reducir los costos de desarrollo y mantenimiento y mejorar el rendimiento de aplicaciones complejas. (Goetz, et al., 2006).

Capítulo 3. Marco Metodológico

3.1 Tipo de Investigación

Basado en el propósito de esta investigación y en los objetivos que se buscan satisfacer, se ha determinado que la investigación será de tipo **evaluativa**, ya que no se busca crear conocimiento nuevo ni tampoco satisfacer las necesidades de un cliente en específico, sino que más bien, se pretende utilizar la información resultante de la comparación de una arquitectura de microservicios de una aplicación en específico, aplicando diferentes parámetros que permitan encontrar hallazgos y, basado en eso, emitir un criterio y recomendaciones que ayuden a cumplir los objetivos propuestos en la presente investigación.

3.2 Alcance Investigativo

El alcance investigativo que se utilizará en esta investigación son los estudios de alcance **descriptivo**, ya que se pretende describir las diferencias de resultados, al aplicar distintas prácticas de desarrollo de software, en un prototipo de una aplicación, utilizando la arquitectura de microservicios.

Para ello, se planea identificar cuáles son las prácticas de desarrollo de software, que ayudarán en la creación de una aplicación que sea amigable con el ambiente.

Además de la investigación de alcance descriptivo, la misma se podrá transformar en una de alcance **explicativo**, porque no solo se planea identificar y describir las distintas prácticas de desarrollo de software, sino también, se busca explicar e interpretar el resultado de las pruebas que se realicen.

3.3 Enfoque

El enfoque que se realizará en esta investigación es el **mixto**, por motivo de que se busca realizar mediciones del tipo **cuantitativo,** mediante las métricas que serán utilizadas en la implementación de las pruebas en las prácticas de desarrollo de software seleccionadas.

Además del enfoque cuantitativo, se tomará en cuenta el enfoque **cualitativo**, ya que se pretende realizar interpretaciones de los hallazgos del enfoque cuantitativo. Esto permitirá que, el enfoque cuantitativo, tenga significado en el contexto de los objetivos que se buscan satisfacer en la investigación actual.

El paradigma naturalista será de gran importancia en el enfoque cualitativo, ya que se buscará comprender los resultados obtenidos y a partir de allí, construir conocimiento en base a los hallazgos encontrados.

3.4 Diseño

La presente investigación se dividirá en cuatro principales actividades, que se mencionan a continuación:

- La recolección de información: Tal y como se ha explicado en secciones anteriores, la información será obtenida mediante la búsqueda de estudios, cuyos resultados, permitan responder las preguntas planteadas, ligadas al cumplimiento de cada objetivo. Adicionalmente, se buscará indagar, por medio de entrevistas a ingenieros del software, con el fin de obtener más información. En esta etapa se determinarán las prácticas de desarrollo de software que se pretenden evaluar.

- El desarrollo del prototipo de la aplicación: La base para la experimentación será la creación de un prototipo, el cual será un microservicio, que servirá de modelo de la aplicación hipotética, y sobre el cual se ejecutarán las pruebas posteriores.
- La ejecución de las pruebas: Esta actividad será la ejecución de las pruebas con las herramientas que se determinen para tal objetivo. Las pruebas se realizarán en el prototipo desarrollado.
- El análisis de los resultados obtenidos: La etapa final, necesaria para la generación de las recomendaciones, será el análisis de los resultados de las pruebas realizadas.

El cumplimiento de las fases anteriores, pretende ser suficiente para lograr el cumplimiento de los objetivos de la presente investigación.

3.5 Población y Muestreo

La población que se utilizará durante el desarrollo de esta investigación, será la del perfil de ingeniero del software, ingeniero en sistemas o carreras a fin.

Además, el muestreo fue **probabilístico** ya que se buscará obtener opiniones de las personas con las siguientes características:

- Personas que sean graduados en ingeniería del software
- Experiencia con arquitecturas de microservicios

- Experiencia con instrumentación para mediciones de rendimiento por medio variables dadas
- Personas con más de tres años de experiencia en desarrollo de software.

3.6 Instrumentos de Recolección de Datos

Los instrumentos que se utilizarán para la recolección de datos serán principalmente las siguientes:

- Encuestas
- Entrevistas

Las entrevistas tienen el propósito de identificar el nivel de conocimiento que las personas tiene en cuanto al desarrollo de software verde y la opinión de forma generalizada sobre la importancia de su implementación.

3.7 Técnicas de Análisis de Información

Para analizar la información recopilada durante la investigación, tanto en las entrevistas, como en publicaciones sobre el tema, se podrá hacer uso de herramientas como diagramas circulares.

Adicionalmente, se podrá utilizar el método estadístico para identificar patrones y tendencias en los datos resultantes de las pruebas que se realicen en las pruebas de las prácticas de desarrollo de software.

3.8 Estrategia de Desarrollo de la Propuesta

La estrategia que se planea utilizar, con el fin de desarrollar la propuesta, está basada en una aplicación hipotética de compra de tiquetes en línea, cuyos requerimientos simplificados se pueden encontrar en el anexo 2.

La razón por la que se determinó utilizar esa aplicación hipotética en específico, está basado en una encuesta que se realizó a 85 personas, en la cual, se logró obtener la percepción de la comunidad, acerca de las aplicaciones existentes de compras de tiquetes en línea. Los resultados de la encuesta se pueden consultar en el anexo 1.

A pesar de que la aplicación no será desarrollada de forma completa, las decisiones de diseño se tomarán teniendo en mente las características de esta aplicación hipotética.

Capítulo 4. Proceso de desarrollo

4.1 Selección de métricas verdes según el objetivo uno de la investigación

De acuerdo con las investigaciones disponibles, que fueron analizadas y que ayudaron con el cumplimiento del primer objetivo de esta investigación, se ha definido que las mediciones o métricas, se realizarán entorno al impacto energético que generan los siguientes componentes de hardware: memoria RAM y CPU.

Para revisar información de la documentación consultada véase el anexo 3 y para encontrar la justificación de la selección de dichos componentes se pueden revisar el anexo 4.

Realizar estimaciones del impacto energético depende de múltiples variables que podrían hacer la medición exacta muy compleja, por lo tanto, la metodología utilizada no fue hacer mediciones sobre todo el conjunto de posibilidades que se podrían presentar, tales como sistema operativo, versiones, fabricantes de dispositivos, modelos, entre otros, sino realizar mediciones que permitan identificar cuáles prácticas de software incrementan el porcentaje de uso del hardware, y por ende, aumento en el consumo energético.

4.1.1 Estrategia de medición

La estrategia utilizada para estimar el impacto energético de los componentes de hardware seleccionados, fue haciendo comparaciones entre los distintos componentes o prácticas de desarrollo de software que se podrían implementar en un microservicio,

por ejemplo, lenguajes de programación, diversos diseños de arquitectura, técnicas para mejorar el rendimiento, entre otros.

Cabe destacar que no existe una métrica ideal que se podría derivar de cada uno de los componentes de hardware, ya que las mediciones dependen de diversos factores en el código y la arquitectura de la aplicación, así como en el hardware seleccionado.

4.1.2 Herramientas de medición

Para realizar las mediciones y comparaciones del uso de los componentes de hardware, se han utilizado las siguientes herramientas:

- Spring Boot Actuator
- Prometheus
- Grafana
- Apache JMeter

La descripción de las herramientas anteriores y la justificación de selección pueden ser encontradas en el anexo 5.

Haciendo uso de las herramientas anteriores se creó un gráfico de monitoreo con las siguientes métricas:

- CPU del Proceso: Esta métrica (*process_cpu_usage*) es utilizada para monitorear el comportamiento del CPU de un proceso específico de la máquina virtual de Java (JVM) y ayuda a entender su comportamiento a través del tiempo. Esta métrica representa el tiempo del CPU que el proceso

ha usado desde que inició. El valor es expresado como una fracción del núcleo del CPU, así que en una máquina con un núcleo, el intervalo iría entre 0 a 1.

- CPU del Sistema: Contrario al CPU del proceso, esta métrica (*system_cpu_usage*) permite capturar el comportamiento del uso general del CPU del sistema el cual incluye todos los procesos y tareas del sistema.
- Memoria: Con el fin de realizar mediciones de la memoria RAM, se decidió utilizar dos métricas *jvm_memory_used_bytes* y *jvm_memory_max_bytes*. Esas dos se utilizaron para obtener el porcentaje de utilización de la memoria RAM que fue asignada a la máquina virtual de Java.

4.2 Evaluación de prácticas de desarrollo de software según el objetivo dos de la investigación

El desarrollo de software verde consiste en prácticas que permiten ligar la ingeniería del software con los principios de sostenibilidad (Mourão, Karita, & do Carmo Machado, 2018).

Para definir cuáles prácticas de desarrollo de software cumplen con la aseveración anterior, en primer lugar se hizo una selección de las prácticas que aplicarían para el prototipo de la aplicación hipotética, se realizaron pruebas en cada una de esas prácticas seleccionadas y se realizaron comparaciones de los resultados para determinar el impacto del hardware.

Las prácticas seleccionadas se pensaron en concordancia con preguntas que normalmente se podrían plantear ante la creación de un microservicio, por ejemplo:

- ¿Cuál lenguaje de programación se debe utilizar?
- ¿Cuántos microservicios se debe utilizar en la aplicación y cuántas instancias?
- ¿Cuáles prácticas pueden ayudar a que la aplicación tenga un buen diseño pensado en rendimiento?
- ¿Cuál diseño de arquitectura podría satisfacer las necesidades de los requerimientos funcionales y no funcionales?
- Entre otras preguntas.

4.2.1 Especificaciones del hardware utilizado

El hardware utilizado durante la ejecución de las distintas pruebas se menciona a continuación:

4.2.1.1 Opción 1

- Laptop MSI modelo Prestige 15Evo A11M
- Procesador Intel Core i7-1185G7
- Memoria RAM 16GB

4.2.1.2 Opción 2

- Laptop MacbookPro
- Procesador Apple M1 Max
- Memoria RAM 32GB

4.2.2 Especificaciones del software utilizado

El software a nivel de sistema que se utilizó para la realización de las pruebas se describe a continuación:

- Sistema operativo Microsoft Windows 11 Home para la opción de hardware 1.
- Sistema operativo macOS Ventura 13.0.1 para la opción de hardware 2.
- IntelliJ IDEA 2022.1.4 (Community Edition)
- Java 17 – Spring Boot versión 3.0.4 y Java 11 – Spring Boot versión 2.7.9
- Postman
- Herramientas de pruebas descritas en el anexo 5.

Adicionalmente, para el prototipo se creó un microservicio con las características que se encuentran en el anexo 6.

4.2.3 Prácticas seleccionadas para evaluación

La estrategia que se utilizó para identificar el impacto de las prácticas de desarrollo de software seleccionadas fue el monitoreo de los componentes de hardware, y la repercusión energética de los mismos. La justificación de la selección de las prácticas se puede encontrar en el anexo 7.

4.2.3.1 Comparación de versiones de Java

4.2.3.1.1 Descripción

Con el fin de identificar el consumo de recursos de hardware de las versiones 8, 11 y 17 de Java, se decidió utilizar algoritmos, algunos de los cuales fueron utilizados en una investigación que tenía el propósito de identificar los lenguajes de programación más amigables con el ambiente. (Pereira, et al., 2021)

Los algoritmos seleccionados son los siguientes:

1. Binary Tree: Este algoritmo consiste en un método para estructurar información de forma jerárquica. Un árbol binario tiene nodos, que podrían interpretarse como ramas, y cada nodo puede tener hasta dos nodos, uno izquierdo y otro derecho. Las operaciones básicas que se pueden realizar en este algoritmo son: inserción, búsqueda y eliminación de datos contenidos en nodos específicos.

2. Frannkuch-redux: Este algoritmo consiste en la búsqueda de todas las permutaciones de un conjunto de elementos dados, con el fin de representarlo en su forma canónica, lo cual quiere decir que, se busca la optimización de la permutación para hacer una representación única y estandarizada de aquellas que son equivalentes.

3. Quick Sort: Este es uno de los algoritmos de ordenamiento más conocido y ampliamente utilizado, el cual se basa en la premisa del “divide y vencerás” para ordenar las listas de elementos dados.

4.2.3.1.2 Metodología de pruebas

Con el fin de identificar patrones en el consumo energético se definió la metodología de pruebas de la siguiente forma:

- Se desarrolló una aplicación usando Spring Boot y se expusieron tres API's, uno para cada uno de los algoritmos.
- Se midieron cada uno de los algoritmos de forma separada, llamando al endpoint que correspondía y se obtuvieron evidencias de ejecución.
- Por cada algoritmo se realizó una prueba de carga con JMeter con un tiempo de cinco minutos y se repitió el proceso tres veces.
- La prueba se realizó en los dos sistemas de hardware descritos anteriormente.

Los resultados de las pruebas con los diferentes escenarios pueden ser consultados en el anexo 8.

4.2.3.2 Comparación de diseño de arquitectura

4.2.3.2.1 Descripción

Esta prueba tenía como propósito demostrar el impacto en los recursos del hardware, de los diseños de arquitectura hexagonal y por capas, utilizando el mismo código y las mismas funcionalidades como base. Para lograr lo anterior, se utilizó el prototipo de microservicio denominado *activity manager*. Véase el anexo 6 para consultar las características del prototipo.

4.2.3.2.2 Metodología de pruebas

Con el fin de identificar patrones en el consumo energético se definió la metodología de pruebas de la siguiente forma:

- Se desarrolló el microservicio mencionado anteriormente y se diseñó tomando en cuenta ambos tipos de arquitecturas.
- Posteriormente se realizó un *endpoint* con una simulación en la cual se realizan las siguientes acciones:
    - Crear un actividad (*Activity*).
    - Crear un lugar destinado para la actividad (*Facility*).
    - El lugar destinado se creó con dos sectores (*Sectors*).
        - El Sector A con una ocupación de 500 espacios.
        - El Sector B con una ocupación de 750 espacios.
    - Una vez creados los modelos en la base de datos, la siguiente acción es habilitar la actividad.
    - Cuando lo anterior sucede, se crean entradas (*ActivitySeating*) correspondiente al total de la capacidad de los sectores.
    - Con las entradas creadas, se compran todas las entradas.
    - Se devuelve el resultado de la simulación
- Se realizaron las mediciones utilizando un controlador paralelo en JMeter.
- El período de ejecución fue de cinco minutos y la ejecución se repitió cinco veces.
- La prueba se realizó en los dos sistemas de hardware descritos anteriormente.

Los resultados de las pruebas con los diferentes escenarios pueden ser consultados en el anexo 9.

4.2.3.3 Comparación de diferencias en rendimiento y tiempo de respuesta

4.2.3.3.1 Descripción

El propósito de esta prueba fue evaluar el impacto en el hardware en ejecuciones que, podrían llegar a tomar más tiempo que otras, en brindar una respuesta. Para esta prueba, se tomó como base, el código alguno de los diseños de arquitectura evaluados anteriormente.

4.2.3.3.2 Metodología de pruebas

Dado que se utilizó el mismo código en ambas instancias, se realizaron dos validaciones específicas, una simulando más tiquetes que crear y la otra un reporte en el cuál hubiera más datos que obtener en la base de datos.

Para una de las pruebas de tiempo de respuesta se realizó el siguiente enfoque:

- Se tomó como base la simulación de la prueba anterior, sin embargo, la diferencia se da en el momento de realizar la petición, ya que en uno de los llamados se crea una actividad en una facilidad con un 50% más de capacidad que la otra.
- Lo anterior se realiza con el fin de obtener un tiempo de respuesta mayor, sin realizar cambio alguno en el código.
- Se decidió utilizar la arquitectura de capas.
- Se realizaron las mediciones utilizando un controlador paralelo en JMeter.

- El período de ejecución fue de cinco minutos y la prueba se repitió tres veces.
- La prueba se realizó en los dos sistemas de hardware descritos anteriormente.

Adicionalmente, con el fin de medir el impacto de los tiempos de respuesta causados por la cantidad de información en las bases de datos se realizó el siguiente enfoque:

- Se realizó una prueba de reportes en la cual se obtiene toda la información de actividades, facilidades y sectores, así como los tiquetes disponibles de la base de datos.
- El escenario específico consiste en tener una base de datos con más información que otra con el fin de que el tiempo de respuesta varíe.
- Se realizaron las mediciones utilizando un controlador paralelo en JMeter.
- El período de ejecución fue de cinco minutos y las pruebas se repitieron tres veces.
- La prueba se realizó en los dos sistemas de hardware descritos anteriormente.

Los resultados de las pruebas con los diferentes escenarios pueden ser consultados en el anexo 10.

4.2.3.4 Comparación de impacto en el hardware en aplicaciones creadas con contenedores.

4.2.3.4.1 Descripción

La intención de esta prueba no era determinar si es buena idea o no utilizar los contenedores, sino más bien, identificar el impacto energético al hacer uso de contenedores, con el fin de justificar la razón por la que se debe pensar correctamente la cantidad de microservicios a utilizar en nuestra aplicación y la cantidad de instancias requeridos por cada microservicio.

4.2.3.4.2 Metodología de pruebas

Nuevamente, se realizó el escenario de simulación descrito en pruebas anteriores, sin embargo la diferencia se da en que una de las aplicaciones está iniciada con Docker, mientras que la otra está siendo ejecutada directamente en la computadora.

- Esta prueba fue realizada utilizando dos aplicaciones iguales, la única diferencia es que una fue levantada en un contenedor usando Docker, mientras que la otra no usa un contenedor.
- Para este escenario se realizaron las mediciones utilizando un controlador paralelo en JMeter.
- El período de ejecución fue de cinco minutos y la prueba se repitió tres veces.
- La prueba se realizó en los dos sistemas de hardware descritos anteriormente.

Los resultados de las pruebas con los diferentes escenarios pueden ser consultados en el anexo 11.

4.2.3.5 Comparación de impacto en el hardware en aplicaciones haciendo uso de batches.

4.2.3.5.1 Descripción

Dado que el prototipo de la aplicación utilizado en esta investigación podría realizar una gran cantidad de operaciones, como por ejemplo, crear varios tiquetes al habilitar actividades de forma simultánea y secuencial, se decidió realizar pruebas en las cuales las transacciones puedan ser realizadas por medio de batches y no de forma lineal o uno por uno.

4.2.3.5.2 Metodología de pruebas

Al igual que en los escenarios anteriores se utilizó la simulación con las siguientes características:

- Durante la compra de los tiquetes de las actividades, se realizó una simulación de compra de forma masiva, de modo que, para marcar como vendidas las entradas en la base de datos, se realizan por medio de batches, en lugar de realizar las compras una por una.
- Para este escenario se realizaron las mediciones utilizando un controlador paralelo en JMeter.

- El período de ejecución fue de cinco minutos y las pruebas se repitieron tres veces.
- La prueba se realizó en los dos sistemas de hardware descritos anteriormente.

Los resultados de las pruebas con los diferentes escenarios pueden ser consultados en el anexo 12.

4.2.3.6 Comparación de impacto en el hardware en aplicaciones haciendo uso de hilos.

4.2.3.6.1 Descripción

Con el fin de mejorar el tiempo de respuesta de las aplicaciones es común que se piense en la utilización de hilos, por lo tanto, con el fin de mejorar los tiempos de respuesta e identificar la viabilidad energética de los hilos se planificó esta prueba.

4.2.3.6.2 Metodología de pruebas

Las pruebas se realizaron con la simulación de reportes descrita anteriormente, en la cual se distinguen las siguientes características:

- La simulación de reporte saca información de cuatro diferentes tablas.
- En el primer escenario sin hilos, la consulta a la base de datos es lineal, sin embargo con el uso de hilos la misma se realiza de forma paralela.
- En cada consulta a la base de datos se ingresa un tiempo de espera de 200 ms, con el fin de demostrar de una mejor forma la diferencia de tiempo entre un escenario y el otro.

- Para este escenario se realizaron las mediciones utilizando un controlador paralelo en JMeter.
- El período de ejecución fue de cinco minutos y las pruebas se repitieron cinco veces.
- La prueba se realizó en los dos sistemas de hardware descritos anteriormente.

Los resultados de las pruebas con los diferentes escenarios pueden ser consultados en el anexo 13.

4.2.3.7 Comparación de impacto en el hardware en aplicaciones haciendo uso de cache.

4.2.3.7.1 Descripción

Dado que el microservicio prototipo se diseñó con la premisa de que se debe realizar muchas consultas a la base de datos, y con el fin de explorar el costo energético de diseños como, por ejemplo, CQRS, en donde se separan las responsabilidades de consultas y de operaciones de creación, modificación y eliminación de datos, se pensó utilizar esta prueba para identificar el impacto del uso del cache.

4.2.3.7.2 Metodología de pruebas

Las pruebas se realizaron utilizando la base de datos relacional Redis, con el fin de almacenar consultas recurrentes, además de Spring Boot Cache.

- El enfoque que se utilizó fue de una simulación de creación de actividades, con su respectivo lugar de eventos y sus sectores.
- La simulación incluía habilitar la actividad y por ende la creación de los tiquetes de la base de datos.
- El escenario que incluía el cache habilitado, ingresaba en la base de datos redis los datos que se insertaban en la base de datos relacional para su posterior consulta.
- El escenario terminaba con una serie de consultas a la base de datos con propósitos de reporte para la cual o se consultaba la base de datos relacional o la de cache redis.
- En cada consulta a la base de datos se ingresa un tiempo de espera de 200 ms con el fin de demostrar de mayor forma la diferencia de tiempo entre un escenario y el otro.
- Para este escenario se realizaron las mediciones utilizando un controlador paralelo en JMeter.
- El período de ejecución fue de cinco minutos y la cantidad de pruebas realizadas fue de seis en Windows y tres en macOS.

Los resultados de las pruebas con los diferentes escenarios pueden ser consultados en el anexo 14.

4.2.3.8 Comparación de impacto en el hardware en aplicaciones haciendo uso de RabbitMQ y Controladores REST.

4.2.3.8.1 Descripción

El motivo de la evaluación de estas dos tecnologías, es porque se puede obtener información relevante que permita identificar el costo energético que podría haber en aplicaciones que utilicen diseños de arquitectura como *Event Driven Design*.

4.2.3.8.2 Metodología de pruebas

Las pruebas se realizaron con una implementación de RabbitMQ y otra con un controlador REST:

- El enfoque que se utilizó fue de una simulación de creación de actividades, con su respectivo lugar de eventos y sus sectores.
- La simulación incluía habilitar la actividad y por ende la creación de los tiquetes de la base de datos.
- En un escenario se realizaron peticiones por medio de un controlador REST mientras que en el otro escenario se creó un *listener* que escuchaba mensajes enviados desde JMeter a un *topic* específico.
- Para este escenario se realizaron las mediciones utilizando un controlador paralelo en JMeter.
- El período de ejecución fue de cinco minutos y la cantidad de pruebas realizadas fue de tres.

- La prueba se realizó en los dos sistemas de hardware descritos anteriormente.

Los resultados de las pruebas con los diferentes escenarios pueden ser consultados en el anexo 15.

4.3 Alternativas de diseño consideradas

Las consideraciones de diseño propuestas en esta investigación fueron aquellas referentes al desarrollo y diseño de microservicios, por ejemplo, arquitectura en capas o hexagonal, diseño basado en dominios (DDD), Command Query Responsibility Segregation (CQRS), Event Driven Design, uso de bases de datos relacionales y no relacionales, entre otros.

Sin embargo, existen muchas alternativas que pudieron ser consideradas, y que podrían ser compatibles con el diseño de microservicios que se pretende para la aplicación hipotética.

A continuación se discuten algunas de las alternativas de diseño que se consideraron para la presente investigación:

4.3.1 Desarrollo de software móvil

El desarrollo de software móvil es un campo en el que las investigaciones, con respecto al consumo energético, han sido muy amplias, especialmente porque los dispositivos en los que se ejecuta el software, tienen una fuente de energía limitada el cual es una batería.

A pesar de que no se ahondó en esta rama del desarrollo del software, se reconoce que existe mucho potencial y área de mejora, especialmente porque, según investigaciones, para el año 2022 se estimaba que el número de dispositivos móviles iba a llegar a 12.3 billones, excediendo el total de la población humana (Ometov, et al., 2020).

4.3.2 Desarrollo de software en la nube

El desarrollo de software en la nube, ha permitido que muchas empresas usen los servicios que distintos proveedores ofrecen, ya que logran trasladar la administración de los recursos de infraestructura, que usan sus aplicaciones, a los compañías que ofrecen los servicios en nube.

Si bien, lo anterior es un gran paso, porque permite que las empresas se enfoquen más en sus aplicaciones que en la infraestructura que la soporta, desde el punto de vista de consumo energético es algo en lo que se debe prestar atención.

De hecho, el uso de la nube está soportado por varios datacenters distribuidos alrededor del mundo, los cuales según estimaciones, elevarán su consumo energético de 200 TWh en el año 2016, a 2967 TWh en el año 2030 (Katal, Dahiya, & Choudhury, 2022) .

4.3.3 Justificación del uso de microservicios

A pesar de que los diseños alternativos mencionados anteriormente, son campos que necesita ser investigados a profundidad, se decidió dejarlos para trabajos futuros, debido a que, los resultados del diseño de arquitectura de microservicios, sirven de base

para esas futuras indagaciones, ya que por ejemplo, en muchas ocasiones, los servicios que usan las aplicaciones móviles son diseñados en microservicios que están alojados en algún proveedor de nube.

Debido a lo anterior, se decidió empezar con los microservicios y posteriormente continuar con otros componentes.

Capítulo 5. Análisis y discusión de resultados

5.1 Análisis de prácticas de desarrollo de software verde según el objetivo dos de la investigación

5.1.1 Comparación de versiones de Java

Tal y como se pudo observar en el anexo 8, los resultados de las versiones de Java fueron muy variadas, sin embargo, se logró ver un patrón que es necesario analizar para tomar la decisión de utilizar una versión en lugar de otra.

5.1.1.1 Binary Tree - Windows

Tabla 4 Resumen de resultados de algoritmo binary tree con Java 8 en Windows. Fuente elaboración propia

Java 8	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.299	0.855	13.3	1880
Ejecución 2	0.307	0.825	13.4	1802
Ejecución 3	0.288	0.851	17.6	1840
Promedio	**0.298**	**0.843**	**14.76**	**1840**

Tabla 5 Resumen de resultados de algoritmo binary tree con Java 11 en Windows. Fuente elaboración propia

Java 11	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.285	0.850	27.4	1712
Ejecución 2	0.271	0.845	33.8	1629
Ejecución 3	0.283	0.843	28.2	1661
Promedio	**0.279**	**0.846**	**29.8**	**1667**

Tabla 6 Resumen de resultados de algoritmo binary tree con Java 17 en Windows. Fuente elaboración propia

Java 17	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.252	0.852	22.8	1636
Ejecución 2	0.259	0.850	21.5	1539
Ejecución 3	0.261	0.852	27.4	1571
Promedio	**0.257**	**0.851**	**23.9**	**1582**

5.1.1.2 Binary Tree - MacOS

Tabla 7 Resumen de resultados de algoritmo binary tree con Java 8 en macOS. Fuente elaboración propia

Java 8	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.338	0.846	13.1	787
Ejecución 2	0.343	0.845	18.9	772
Ejecución 3	0.345	0.853	20.6	773
Promedio	**0.342**	**0.848**	**17.5**	**777**

Tabla 8 Resumen de resultados de algoritmo binary tree con Java 11 en macOS. Fuente elaboración propia

Java 11	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.268	0.849	22.6	657
Ejecución 2	0.265	0.846	20.7	642
Ejecución 3	0.268	0.853	25.3	648
Promedio	**0.267**	**0.849**	**22.8**	**647**

Tabla 9 Resumen de resultados de algoritmo binary tree con Java 17 en macOS. Fuente elaboración propia

Java 17	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.23	0.852	19.6	602
Ejecución 2	0.231	0.850	25.5	587
Ejecución 3	0.232	0.853	15.2	466
Promedio	**0.231**	**0.851**	**20.1**	**551**

5.1.1.3 Redux - Windows

Tabla 10 Resumen de resultados de algoritmo redux con Java 8 en Windows. Fuente elaboración propia

Java 8	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.299	0.953	17.9	15708
Ejecución 2	0.311	0.950	23.1	15456
Ejecución 3	0.297	0.947	16.7	15536
Promedio	**0.302**	**0.950**	**19.23**	**15566**

Tabla 11 Resumen de resultados de algoritmo redux con Java 11 en Windows. Fuente elaboración propia

Java 11	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.314	0.953	48.7	16588
Ejecución 2	0.311	0.951	30	16526
Ejecución 3	0.309	0.948	38.2	16509
Promedio	**0.302**	**0.950**	**38.9**	**16541**

Tabla 12Resumen de resultados de algoritmo redux con Java 17 en Windows. Fuente elaboración propia

Java 17	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.329	0.951	16.5	16807
Ejecución 2	0.334	0.953	25.5	16669
Ejecución 3	0.347	0.948	33.8	16706
Promedio	**0.336**	**0.950**	**25.26**	**16727**

5.1.1.4 Redux – MacOS

Tabla 13 Resumen de resultados de algoritmo redux con Java 8 en macOS. Fuente elaboración propia

Java 8	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.311	0.950	21.9	12496
Ejecución 2	0.309	0.949	27.1	12487
Ejecución 3	0.312	0.950	8.3	12471
Promedio	**0.310**	**0.949**	**19.1**	**12484**

Tabla 14 Resumen de resultados de algoritmo redux con Java 11 en macOS. Fuente elaboración propia

Java 11	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.315	0.950	3.7	12647
Ejecución 2	0.316	0.949	4.9	12636
Ejecución 3	0.313	0.950	6.1	12604
Promedio	**0.314**	**0.949**	**4.9**	**12629**

Tabla 15 Resumen de resultados de algoritmo redux con Java 17 en macOS. Fuente elaboración propia

Java 17	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.319	0.950	3.8	12615
Ejecución 2	0.318	0.499	5	12629
Ejecución 3	0.319	0.949	2.5	12632
Promedio	**0.318**	**0.777**	**3.7**	**12625**

5.1.1.5 Quick Sort - Windows

Tabla 16 Resumen de resultados de algoritmo quick sort con Java 8 en Windows. Fuente elaboración propia

Java 8	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.119	0.385	13	1317
Ejecución 2	0.116	0.378	15.4	1288
Ejecución 3	0.119	0.385	12.5	1295
Promedio	**0.118**	**0.382**	**13.6**	**1300**

Tabla 17 Resumen de resultados de algoritmo quick sort con Java 11 en Windows. Fuente elaboración propia

Java 11	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.116	0.385	3.5	1226
Ejecución 2	0.117	0.378	3.4	1208
Ejecución 3	0.118	0.385	3.5	1217
Promedio	**0.117**	**0.382**	**3.4**	**1217**

Tabla 18 Resumen de resultados de algoritmo quick sort con Java 17 en Windows. Fuente elaboración propia

Java 17	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.108	0.384	3.4	1259
Ejecución 2	0.111	0.383	3.4	1241
Ejecución 3	0.112	0.379	3.4	1251
Promedio	**0.110**	**0.382**	**3.4**	**1250**

5.1.1.6 Quick Sort - MacOs

Tabla 19 Resumen de resultados de algoritmo quick sort con Java 8 en macOS. Fuente elaboración propia

Java 8	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.143	0.543	9	391
Ejecución 2	0.143	0.544	13.4	387
Ejecución 3	0.143	0.541	12.2	386
Promedio	**0.143**	**0.542**	**11.5**	**388**

Tabla 20 Resumen de resultados de algoritmo quick sort con Java 11 en macOS. Fuente elaboración propia

Java 11	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.151	0.543	3.1	398
Ejecución 2	0.151	0.543	3.3	396
Ejecución 3	0.152	0.542	3.2	395
Promedio	**0.151**	**0.542**	**3.2**	**396**

Tabla 21 Resumen de resultados de algoritmo quick sort con Java 17 en macOS. Fuente elaboración propia

Java 17	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.135	0.542	2.5	360
Ejecución 2	0.136	0.541	2.6	359
Ejecución 3	0.136	0.546	2.7	358
Promedio	**0.135**	**0.543**	**2.6**	**359**

5.1.1.7 Resumen de resultados

5.1.1.7.1 Windows

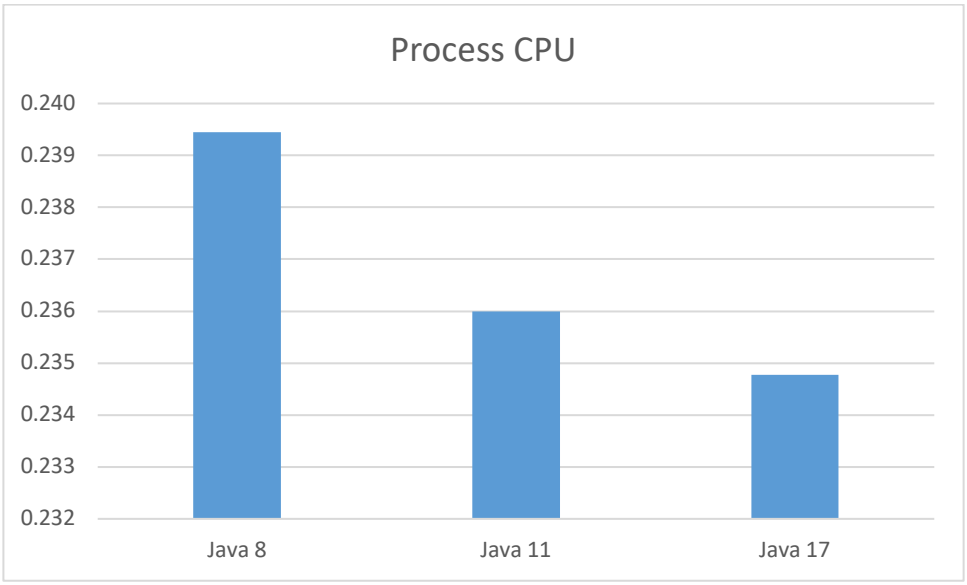

Ilustración 2 Promedio de resultados del uso de CPU del proceso en diferentes versiones de Java con Windows. Fuente: Elaboración propia

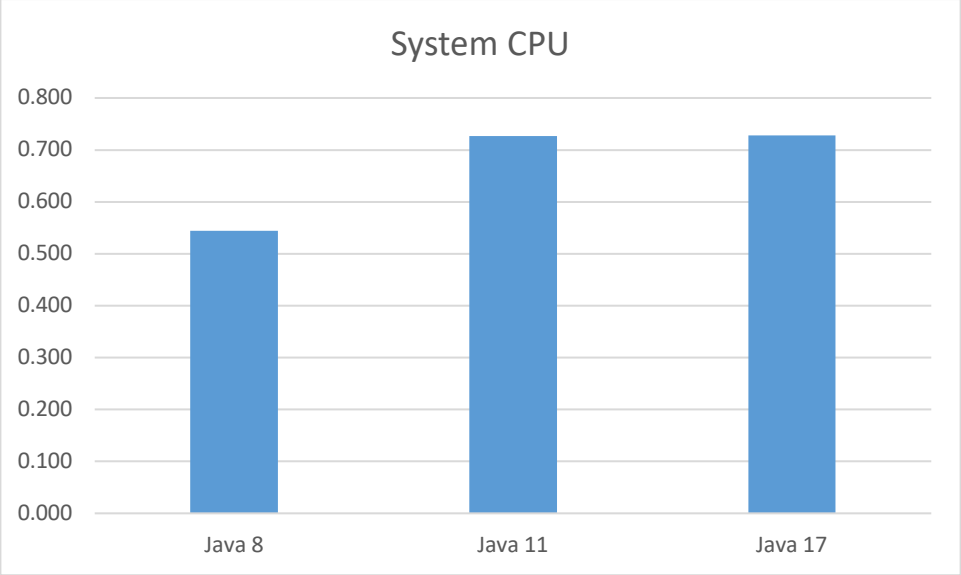

Ilustración 3 Promedio de resultados del uso de CPU del sistema en diferentes versiones de Java con Windows. Fuente: Elaboración propia

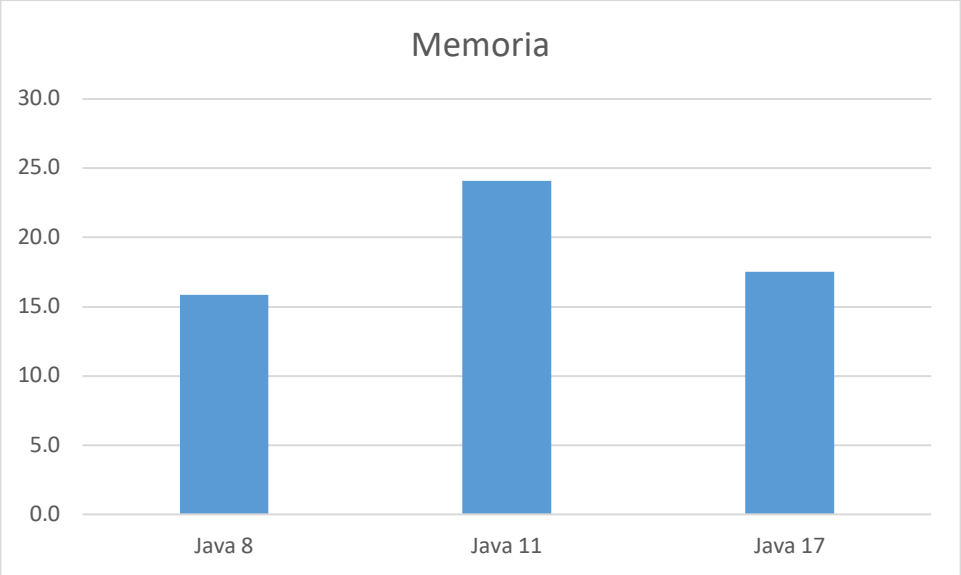

Ilustración 4 Promedio de resultados del uso de memoria en diferentes versiones de Java con Windows. Fuente: Elaboración propia

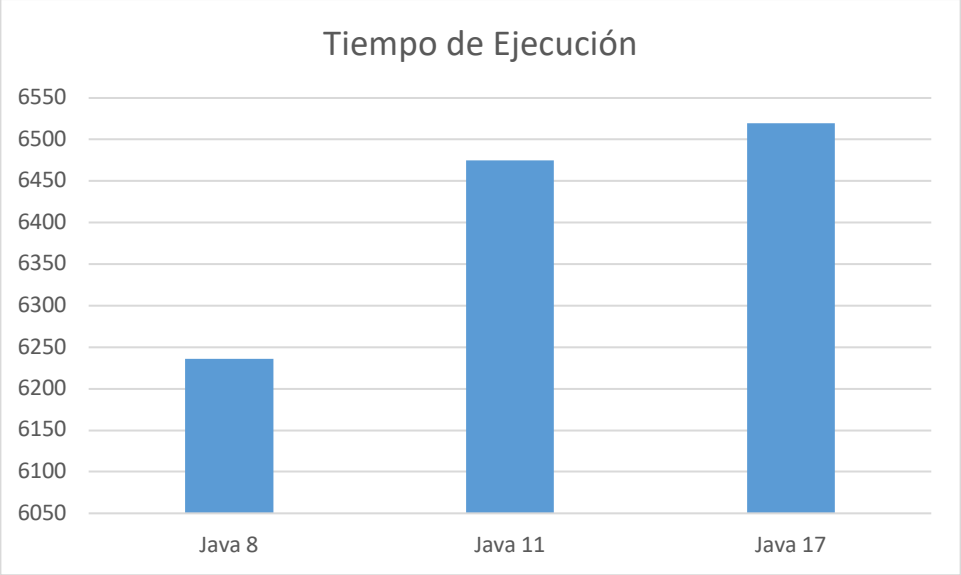

Ilustración 5 Promedio de resultados de tiempos de respuesta en diferentes versiones de Java con Windows. Fuente: Elaboración propia

5.1.1.7.2 MacOS

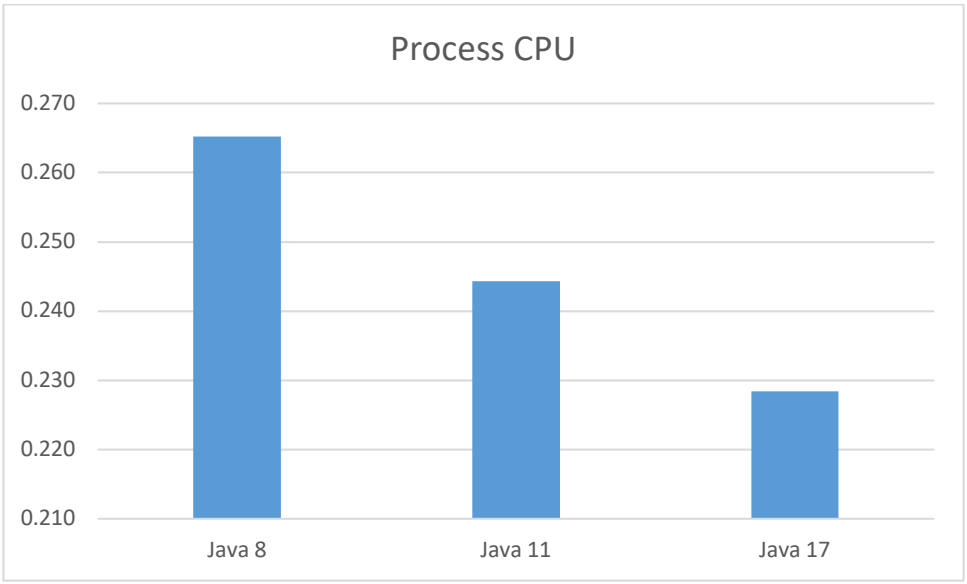

Ilustración 6 Promedio de resultados del uso de CPU del proceso en diferentes versiones de Java con macOS. Fuente: Elaboración propia

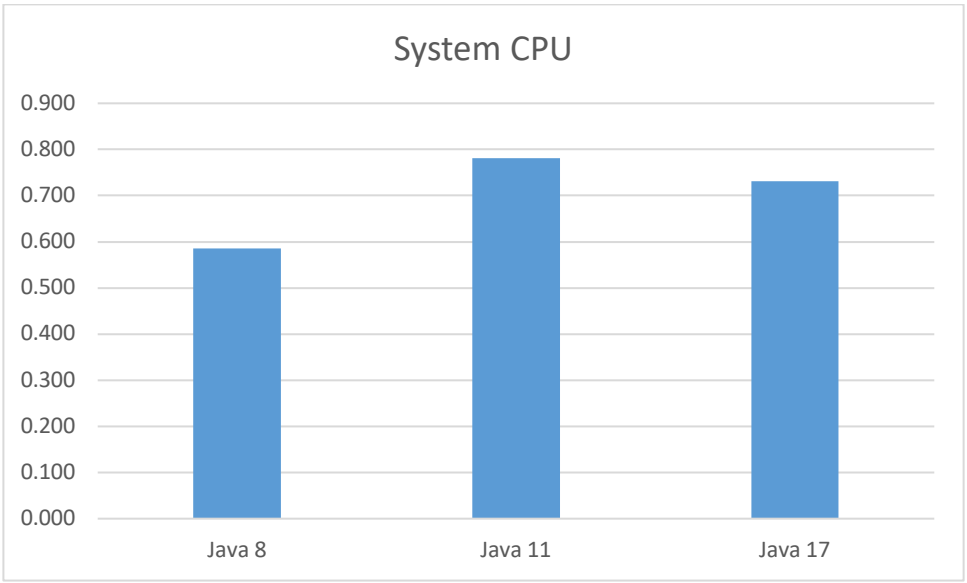

Ilustración 7 Promedio de resultados del uso de CPU del proceso en diferentes versiones de Java con macOS. Fuente: Elaboración propia

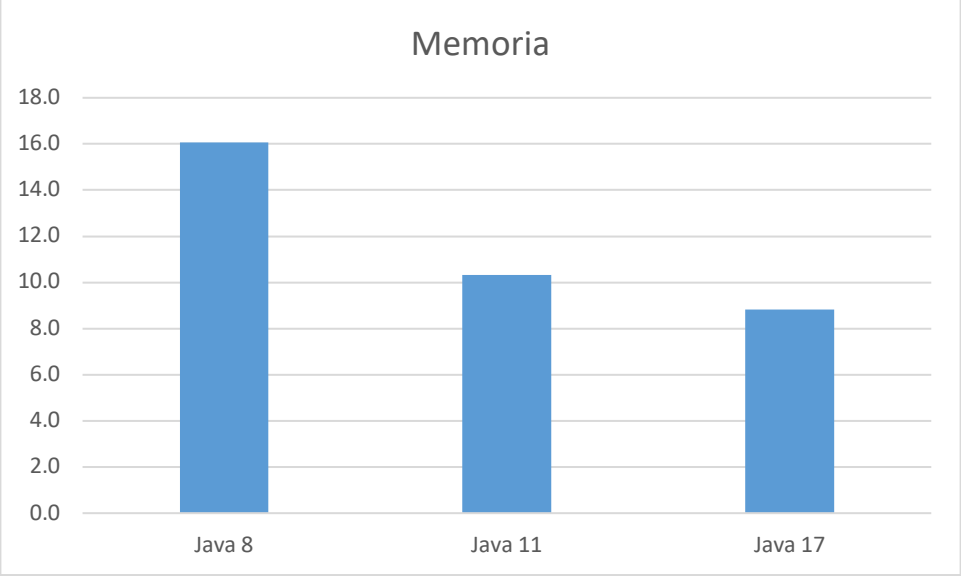

Ilustración 8 Promedio de resultados del uso de CPU del proceso en diferentes versiones de Java con macOS. Fuente: Elaboración propia

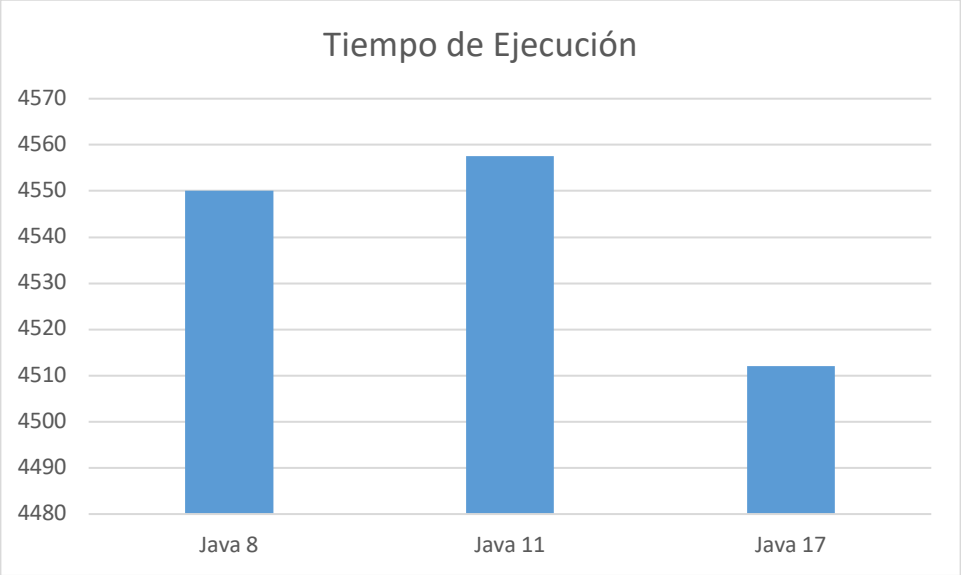

Ilustración 9 Promedio de resultados del uso de CPU del proceso en diferentes versiones de Java con macOS. Fuente: Elaboración propia

De los resultados de las ejecuciones de las pruebas de Java se puede demostrar lo siguiente:

- Java 17 demostró el mejor rendimiento en las pruebas ejecutadas en el sistema operativo MacOS, tanto en tiempo de respuesta, como porcentaje de uso del CPU del proceso y de memoria.
- En el caso de las pruebas realizadas con el sistema operativo Windows, Java 8 presentó un promedio de tiempo de ejecución mucho menor, sin embargo, pese a ello, el uso del CPU del proceso fue más alto.
- El CPU del sistema es similar en Java 11 y Java 17, sin embargo, es menor en Java 8 en ambos sistemas operativos.
- En el caso de la memoria, Java 8 presentó el porcentaje de uso más bajo, mientras que Java 11 el porcentaje de uso más alto, de acuerdo a las ejecuciones realizadas en el sistema operativo Windows. Sin embargo, en el caso del sistema operativo MacOS, Java 8 presenta el porcentaje de memoria más alto y Java 17 el más bajo.
- En el sistema operativo Windows, la diferencia en los recursos de CPU del sistema y de memoria entre Java 8 y las demás versiones, no son proporcionales al tiempo de ejecución, con lo cual se puede interpretar que para haber tenido un tiempo de respuesta menor, hace un uso intensivo de hardware.
- En ambos sistemas operativos, Java 11 no tuvo resultados sustancialmente positivos en comparación con las otras versiones.

5.1.2 Comparación de diseño de arquitectura hexagonal y en capas

5.1.2.1 Windows

Tabla 22 Resumen de resultados de diseño de arquitectura en capas en Windows. Fuente elaboración propia

En capas	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0497	0.163	4.2	6556
Ejecución 2	0.0384	0.133	4.1	6547
Ejecución 3	0.0415	0.142	4.4	6359
Ejecución 4	0.0406	0.135	4.4	7272
Ejecución 5	0.0434	0.145	4.5	6340
Promedio	**0.0427**	**0.1436**	**4.32**	**6614**

Tabla 23 Resumen de resultados de diseño de arquitectura en hexagonal en Windows. Fuente elaboración propia

Hexagonal	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0509	0.166	4.1	6557
Ejecución 2	0.0389	0.133	4.1	6547
Ejecución 3	0.0422	0.142	4.3	6358

Ejecución 4	0.0397	0.134	4.3	7272
Ejecución 5	0.0439	0.144	4.4	6343
Promedio	**0.0431**	**0.143**	**4.2**	**6615**

5.1.2.2 MacOS

Tabla 24 Resumen de resultados de diseño de arquitectura en capas en macOS. Fuente elaboración propia

En capas	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0742	0.337	2.6	2967
Ejecución 2	0.0717	0.338	2.7	2921
Ejecución 3	0.0714	0.338	3.1	2915
Ejecución 4	0.0715	0.338	3	2906
Ejecución 5	0.0715	0.337	3	2907
Promedio	**0.0720**	**0.337**	**2.8**	**2923**

Tabla 25 Resumen de resultados de diseño de arquitectura hexagonal en macOS. Fuente elaboración propia

Hexagonal	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0766	0.336	2.6	2981
Ejecución 2	0.0749	0.337	2.8	2935
Ejecución 3	0.0749	0.339	2.7	2929
Ejecución 4	0.0748	0.340	2.8	2921
Ejecución 5	0.0748	0.335	2.9	2921
Promedio	**0.0752**	**0.337**	**2.7**	**2937**

5.1.2.3 Resumen de resultados

6.1.2.3.1 Windows

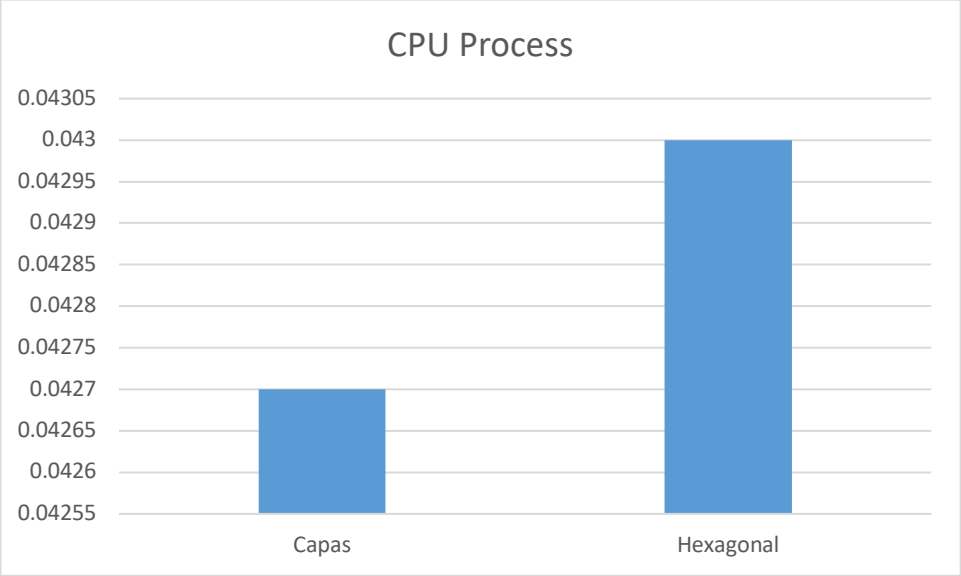

Ilustración 10 Promedio de resultados del uso de CPU del proceso entre distintos diseños de arquitectura con Windows. Fuente: Elaboración propia

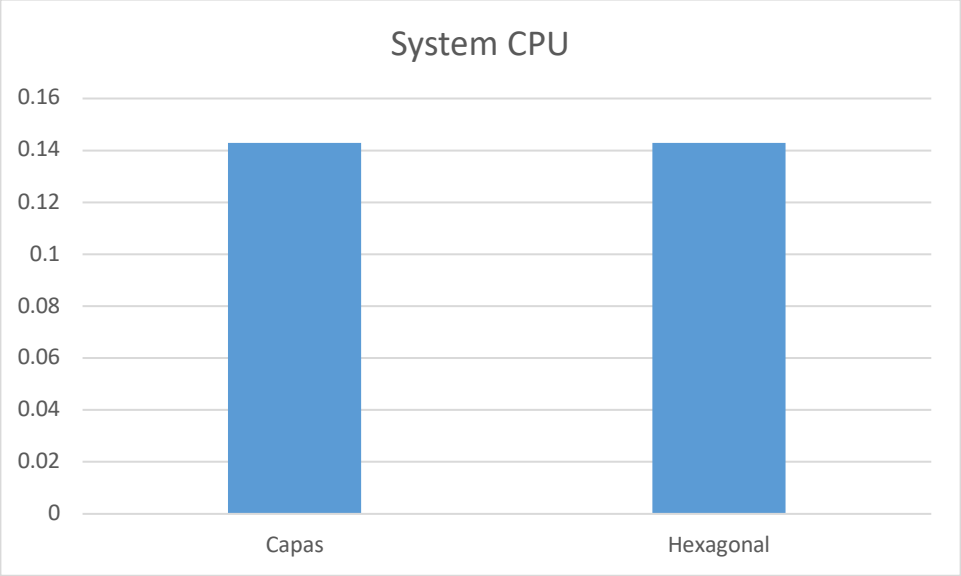

Ilustración 11 Promedio de resultados del uso de CPU del sistema entre distintos diseños de arquitectura con Windows. Fuente: Elaboración propia

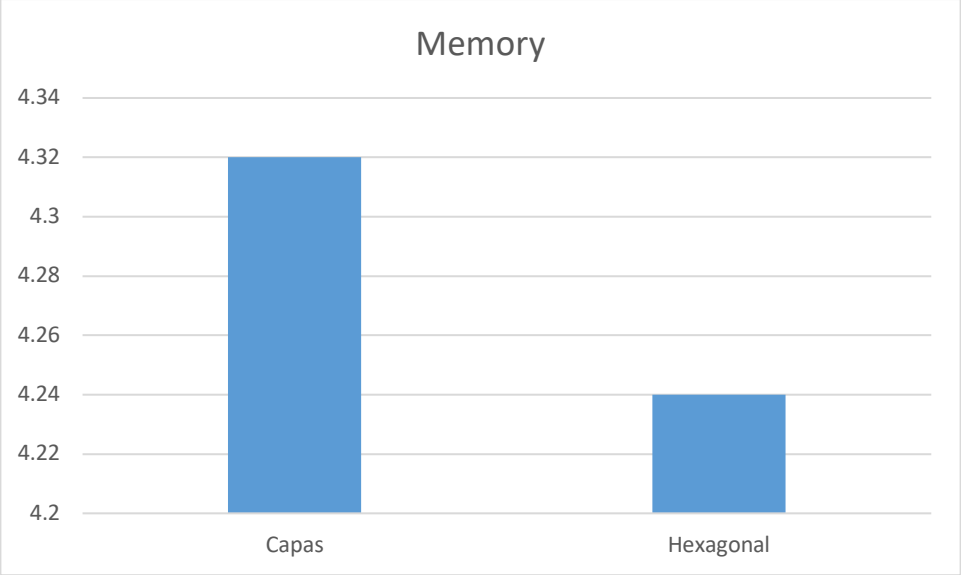

Ilustración 12 Promedio de resultados del uso de memoria entre distintos diseños de arquitectura con Windows. Fuente: Elaboración propia

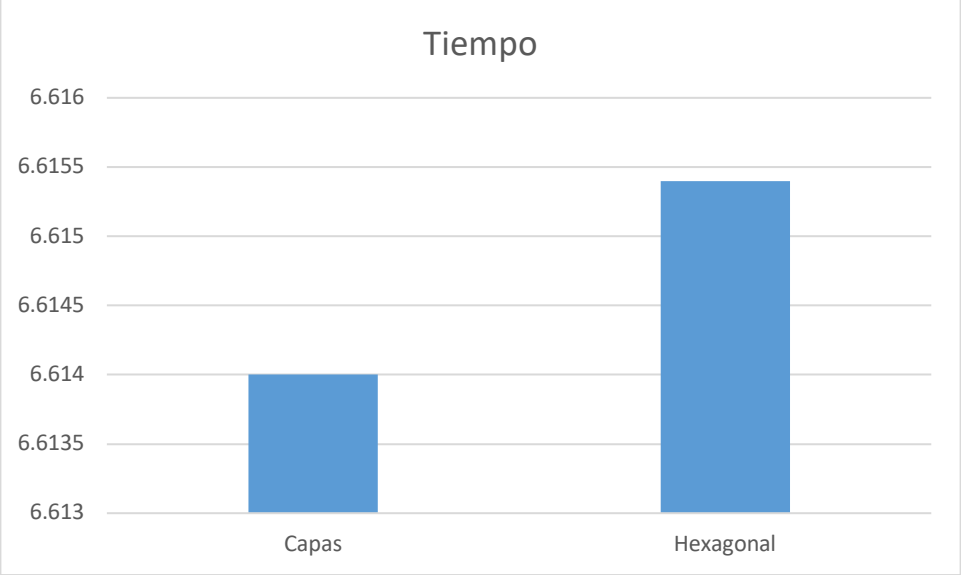

Ilustración 13 Promedio de resultados de tiempos de respuesta entre distintos diseños de arquitectura con Windows. Fuente: Elaboración propia

5.1.2.3.2 MacOS

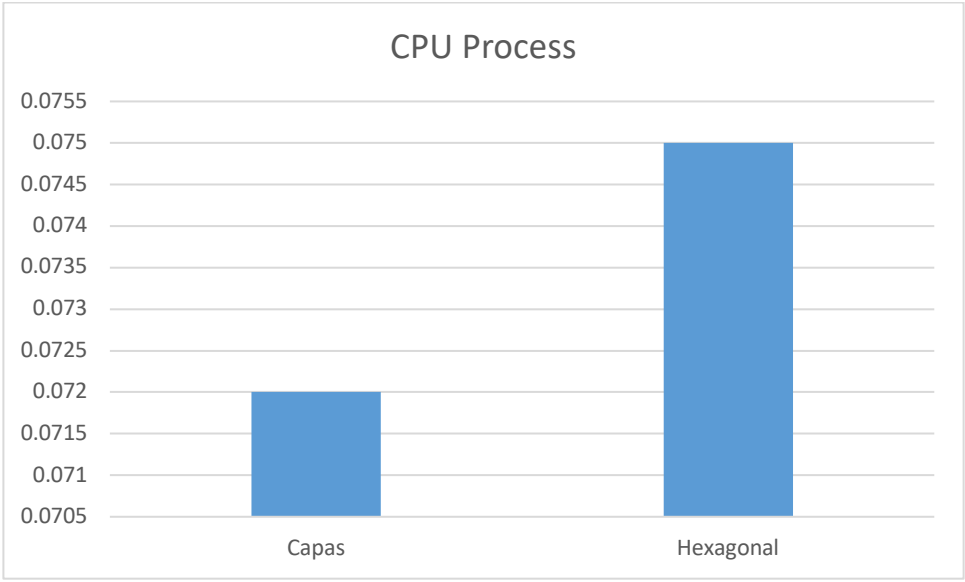

Ilustración 14 Promedio de resultados del uso de CPU del proceso entre distintos diseños de arquitectura con macOS. Fuente: Elaboración propia

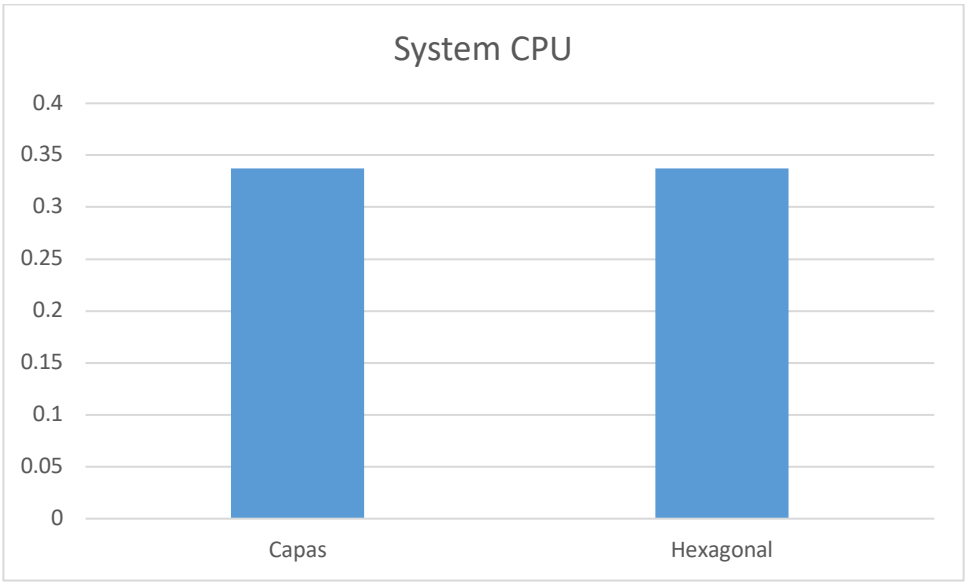

Ilustración 15 Promedio de resultados del uso de CPU del sistema entre distintos diseños de arquitectura con Windows. Fuente: Elaboración propia

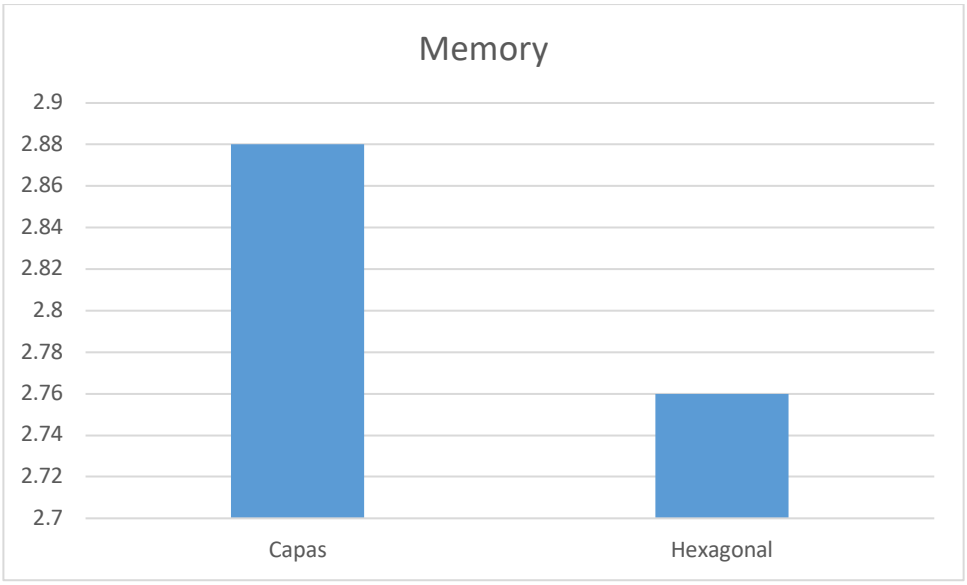

Ilustración 16 Promedio de resultados del uso de memoria entre distintos diseños de arquitectura con Windows. Fuente: Elaboración propia

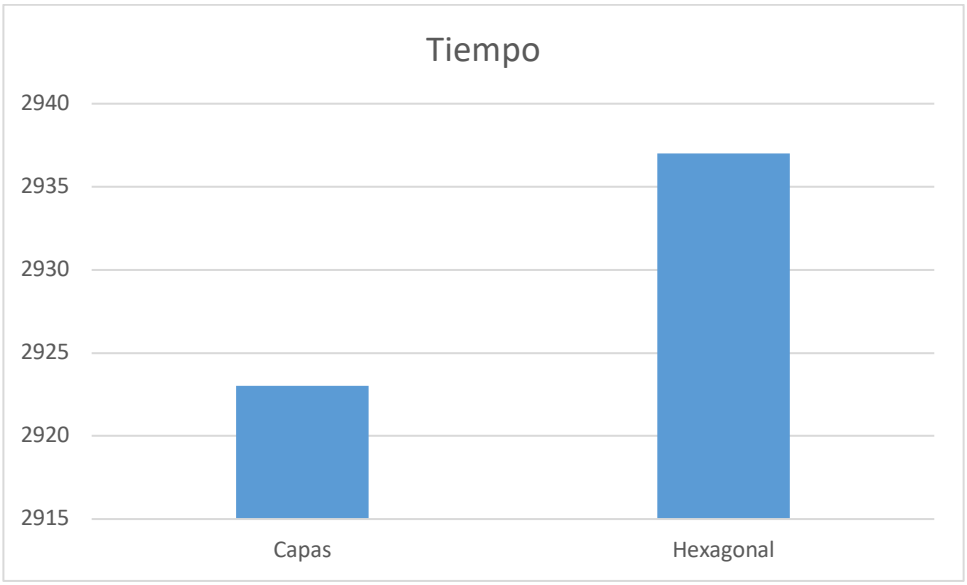

Ilustración 17 Promedio de resultados de tiempo de respuesta entre distintos diseños de arquitectura con macOS. Fuente: Elaboración propia

De los resultados de las ejecuciones de las pruebas de diseño de arquitectura se puede demostrar lo siguiente:

- El CPU del proceso es más alto en la arquitectura hexagonal que en la arquitectura en capas en ambos sistemas operativos.
- En el caso del procesamiento del sistema se mantiene similar en ambos diseños en ambos sistemas operativos.
- La memoria es ligeramente más alta en la ejecución de la arquitectura de capas en ambos sistemas operativos.
- El tiempo es muy similar en ambos, con un leve retraso en el diseño hexagonal en ambos sistemas operativos

5.1.3 Comparación de pruebas de rendimiento

5.1.3.1 Pruebas de operaciones de escritura

5.1.3.1.1 Windows

Tabla 26 Resumen de resultados de pruebas de rendimiento en operaciones de escritura con tiempos de respuesta alto en Windows. Fuente elaboración propia

Tiempo de respuesta alto	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0425	0.110	4.2	6173
Ejecución 2	0.0419	0.106	4.5	6199
Ejecución 3	0.0431	0.108	4.5	6171
Promedio	**0.0425**	**0.108**	**4.4**	**6181**

Tabla 27 Resumen de resultados de pruebas de rendimiento en operaciones de escritura con tiempos de respuesta bajo en Windows. Fuente elaboración propia

Tiempo de respuesta bajo	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0216	0.108	4	3217
Ejecución 2	0.0173	0.105	4.2	2981
Ejecución 3	0.0187	0.110	4.2	2911
Promedio	**0.0192**	**0.107**	**4.1**	**3019**

5.1.3.1.2 MacOS

Tabla 28 Resumen de resultados de pruebas de rendimiento en operaciones de escritura con tiempos de respuesta alto en macOS. Fuente elaboración propia

Tiempo de respuesta alto	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0821	0.213	3.3	2447
Ejecución 2	0.0766	0.200	3.5	2400
Ejecución 3	0.0767	0.200	3.5	2431
Promedio	**0.0784**	**0.204**	**3.4**	**2426**

Tabla 29 Resumen de resultados de pruebas de rendimiento en operaciones de escritura con tiempos de respuesta bajo en macOS. Fuente elaboración propia

Tiempo de respuesta bajo	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0379	0.209	3.2	1217
Ejecución 2	0.0334	0.200	3.3	1181
Ejecución 3	0.0334	0.198	3.4	1193
Promedio	**0.0349**	**0.202**	**3.3**	**1197**

5.1.3.1.3 Resumen de resultados

De los resultados de las ejecuciones de las pruebas de rendimiento de escritura se puede demostrar lo siguiente:

- En ejecuciones de escritura con rendimiento menor se tiene un uso del CPU mucho mayor, tanto del CPU del proceso como el del sistema en ambos sistemas operativos.
- Del mismo modo la memoria es mucho mayor, en ejecuciones con un rendimiento con más tiempo.

5.1.3.1.3.1 Windows

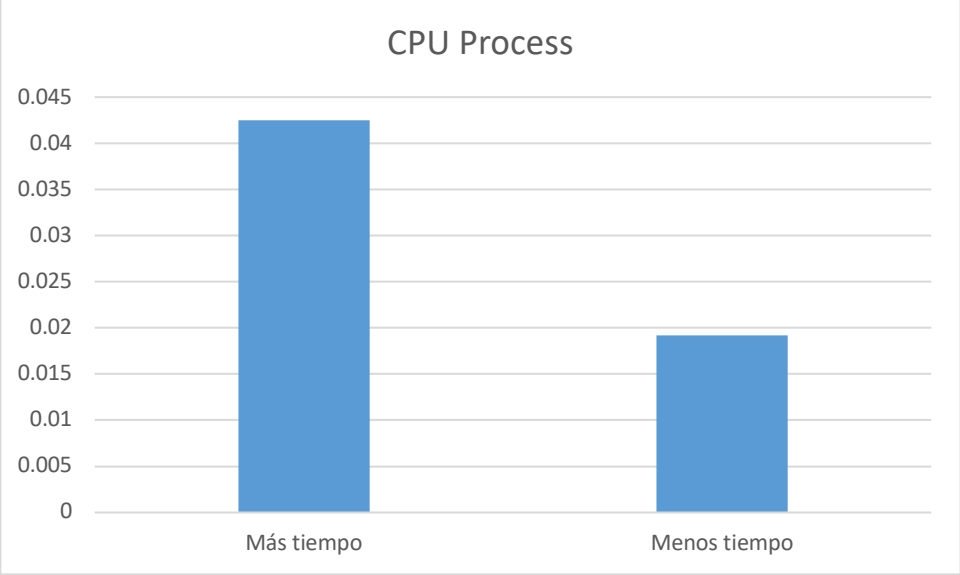

Ilustración 18 Promedio de resultados del CPU del proceso en operaciones de escritura con Windows. Fuente: Elaboración propia

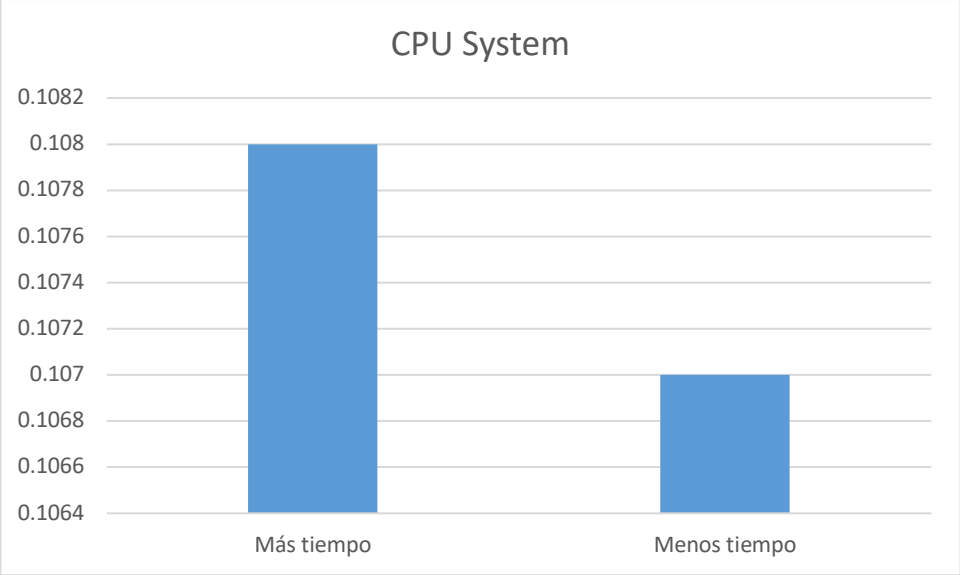

Ilustración 19 Promedio de resultados del CPU del sistema en operaciones de escritura con Windows. Fuente: Elaboración propia

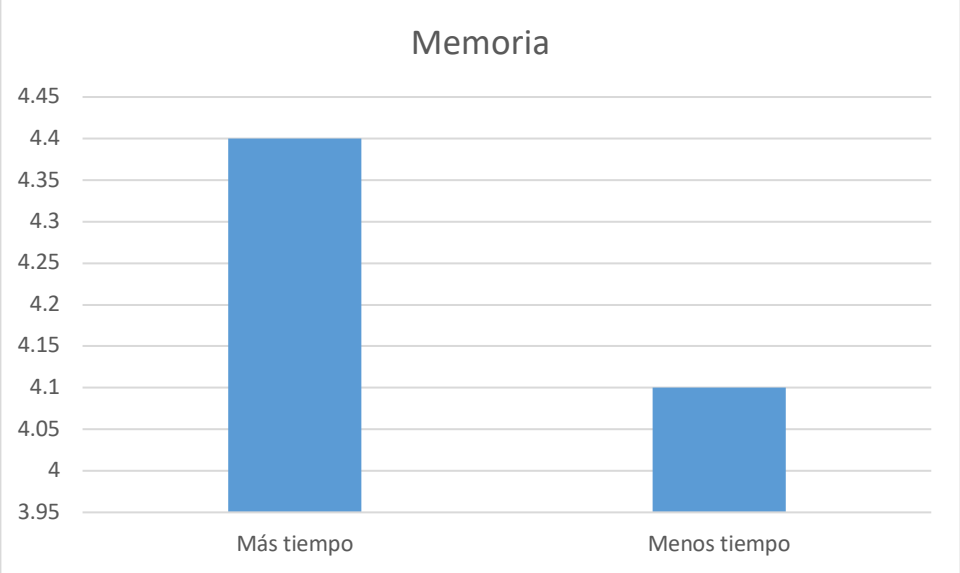

Ilustración 20 Promedio de resultados de la memoria en operaciones de escritura con Windows. Fuente: Elaboración propia

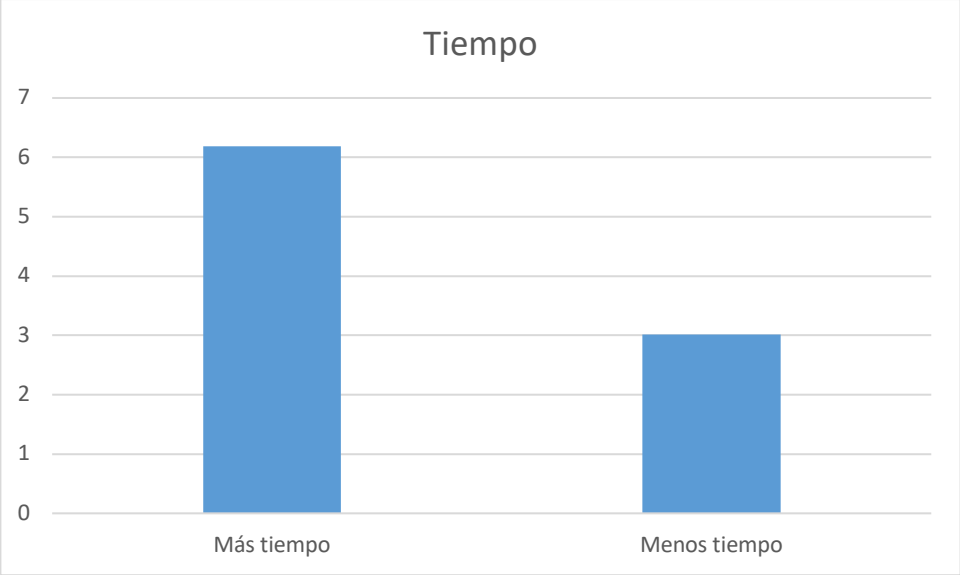

Ilustración 21 Promedio de resultados del tiempo de ejecución total en operaciones de escritura con Windows. Fuente: Elaboración propia

5.1.3.1.3.2 MacOS

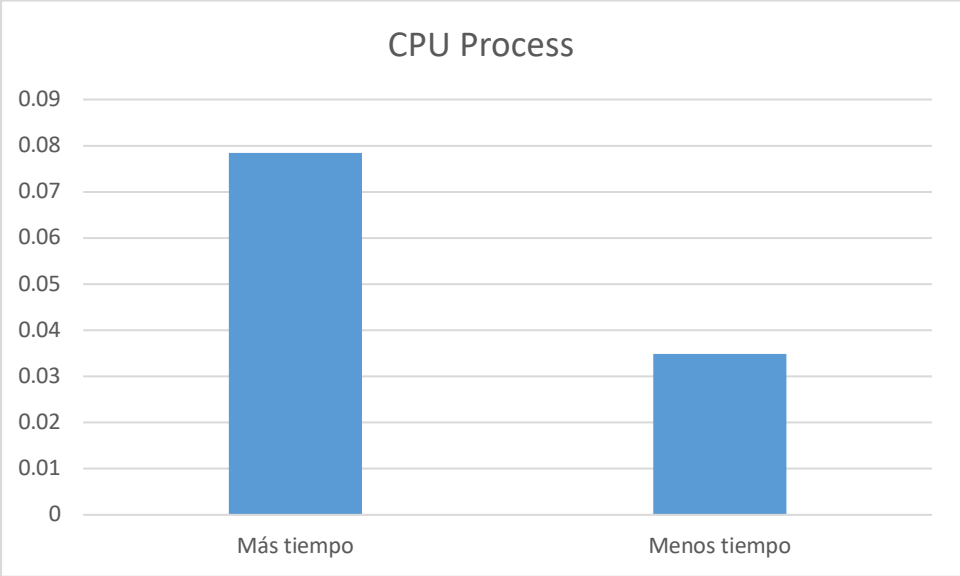

Ilustración 22 Promedio de resultados del CPU del proceso en operaciones de escritura con macOS. Fuente: Elaboración propia

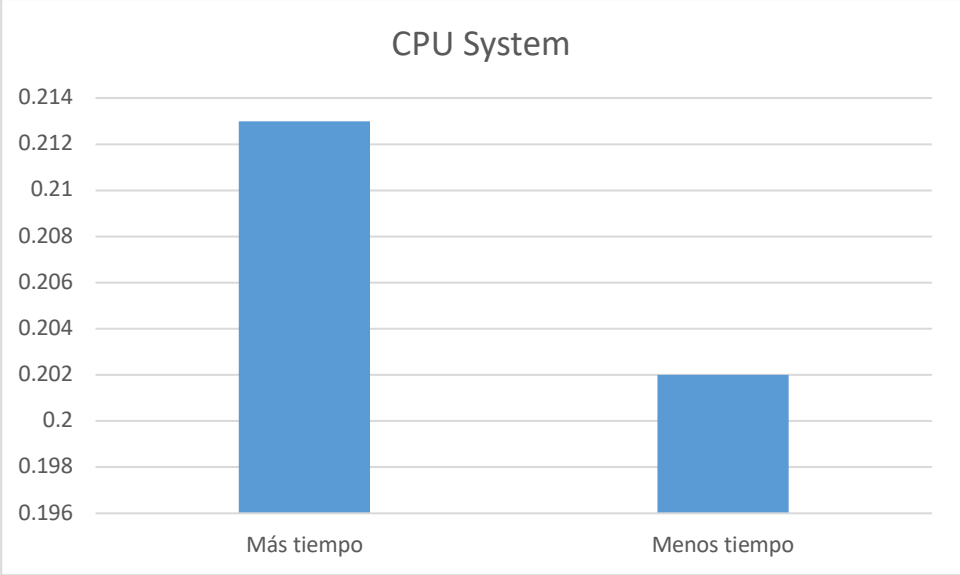

Ilustración 23 Promedio de resultados del CPU del sistema en operaciones de escritura con macOS. Fuente: Elaboración propia

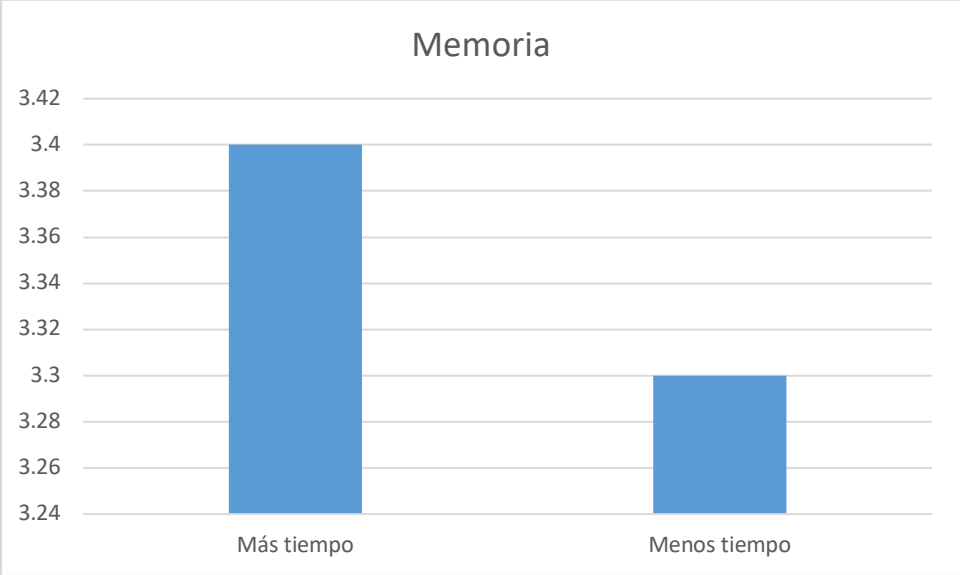

Ilustración 24 Promedio de resultados de la memoria en operaciones de escritura con macOS. Fuente: Elaboración propia

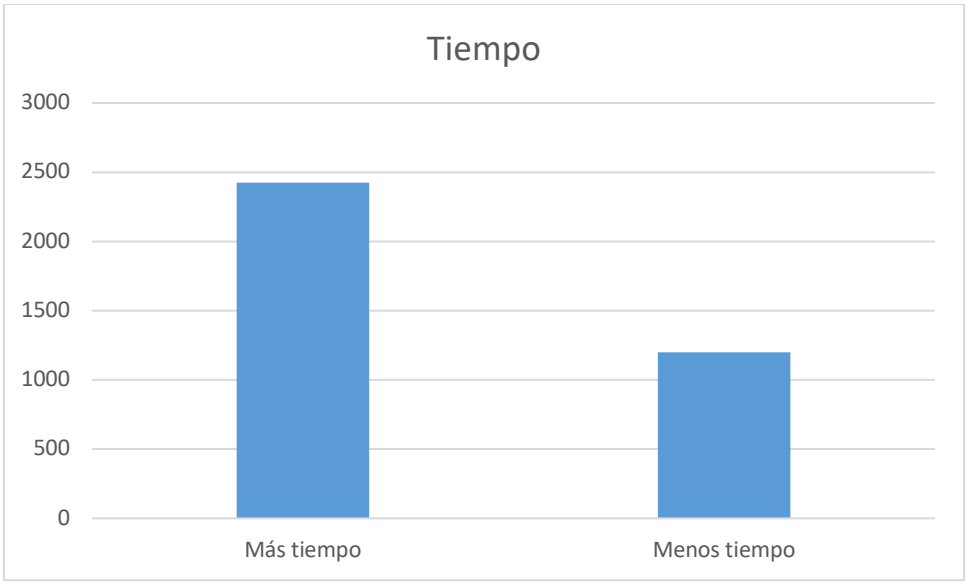

Ilustración 25 Promedio de resultados del tiempo de ejecución total en operaciones de escritura con macOS. Fuente: Elaboración propia

5.1.3.2 Pruebas de operaciones de lectura

5.1.3.2.1 Windows

Tabla 30 Resumen de resultados de pruebas de rendimiento en operaciones de lectura con tiempos de respuesta alto en Windows. Fuente elaboración propia

Tiempo de respuesta alto	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0924	0.143	11.7	2455
Ejecución 2	0.0740	0.108	11.1	2351
Ejecución 3	0.107	0.157	11.2	2349
Promedio	**0.0911**	**0.136**	**11.3**	**2385**

Tabla 31 Resumen de resultados de pruebas de rendimiento en operaciones de lectura con tiempos de respuesta bajo en Windows. Fuente elaboración propia

Tiempo de respuesta bajo	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.00874	0.123	8.3	1718
Ejecución 2	0.00584	0.0952	9.2	1693
Ejecución 3	0.00917	0.150	7.9	1693
Promedio	**0.00791**	**0.122**	**8.4**	**1701**

5.1.3.2.2 MacOS

Tabla 32 Resumen de resultados de pruebas de rendimiento en operaciones de lectura con tiempos de respuesta alto en macOS. Fuente elaboración propia

Tiempo de respuesta alto	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0724	0.168	9.8	2472
Ejecución 2	0.0658	0.143	15.4	2344
Ejecución 3	0.0720	0.142	10.1	2349
Promedio	**0.0700**	**0.151**	**11.7**	**2388**

Tabla 33 Resumen de resultados de pruebas de rendimiento en operaciones de lectura con tiempos de respuesta bajo en macOS. Fuente elaboración propia

Tiempo de respuesta bajo	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0180	0.169	13.3	2171
Ejecución 2	0.0185	0.133	10.5	1938
Ejecución 3	0.0203	0.151	9.7	1938
Promedio	**0.0189**	**0.151**	**11.16**	**2015**

5.1.3.2.3 Resumen de resultados

De los resultados de las ejecuciones de las pruebas de lectura se puede demostrar lo siguiente:

- Del mismo modo que en las operaciones de escritura, el CPU es mucho mayor tanto en el CPU del proceso como del sistema.
- El porcentaje de memoria es mayor en las ejecuciones con más tiempo de ejecución.
- Lo anterior es consistente tanto en sistema operativo Windows como MacOS

5.1.3.2.3.1 Windows

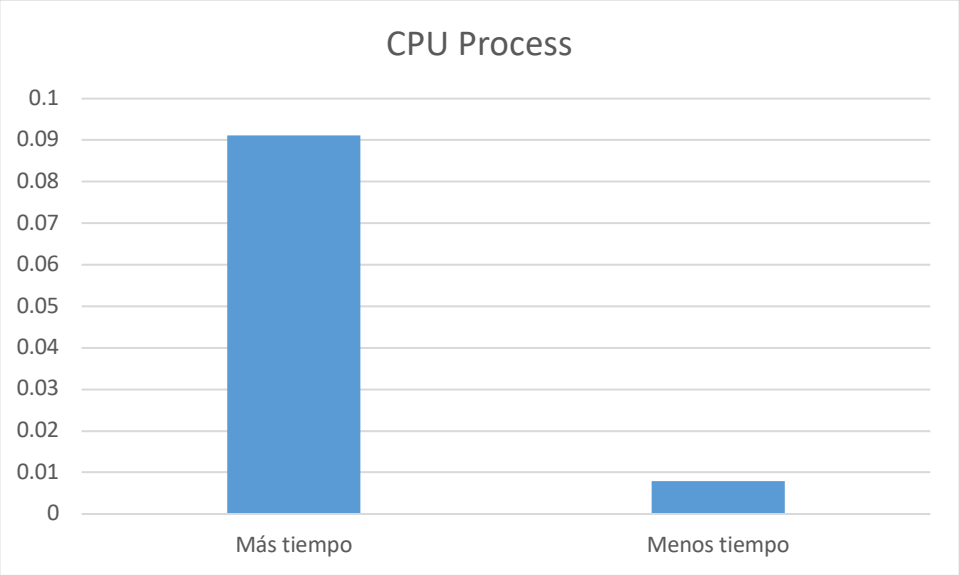

Ilustración 26 Promedio de resultados del CPU del proceso en operaciones de lectura con Windows. Fuente: Elaboración propia

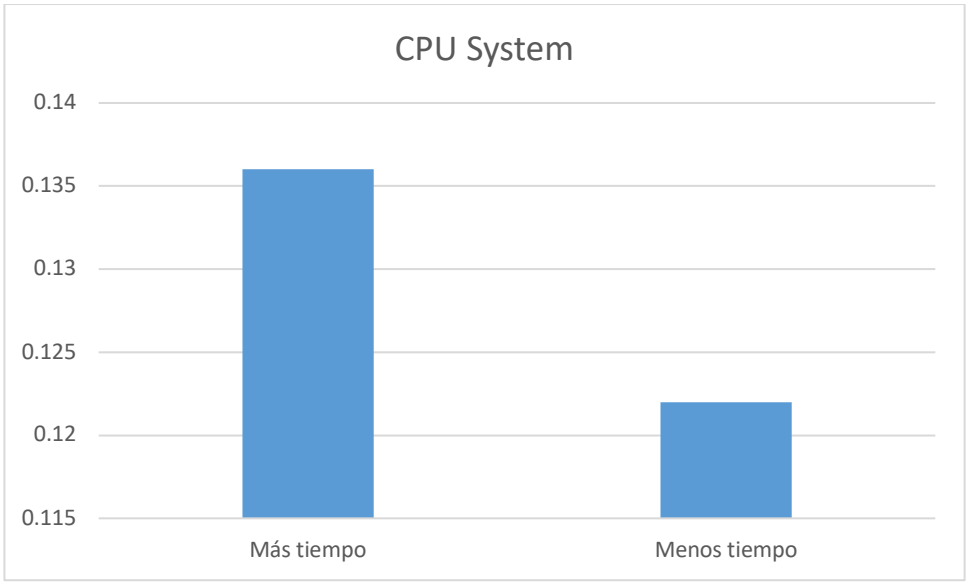

Ilustración 27 Promedio de resultados del CPU del sistema en operaciones de lectura con Windows. Fuente: Elaboración propia

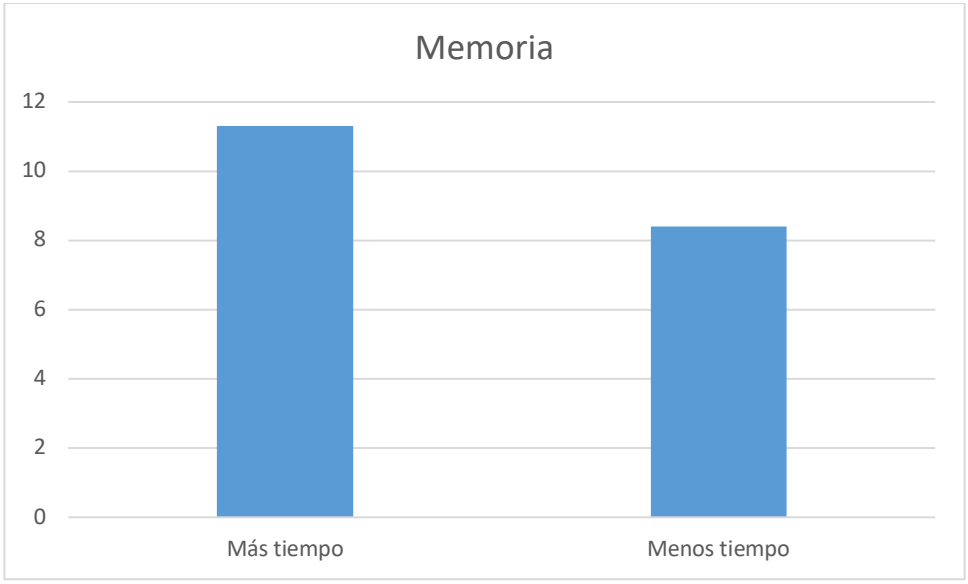

Ilustración 28 Promedio de resultados de la memoria en operaciones de lectura con Windows. Fuente: Elaboración propia

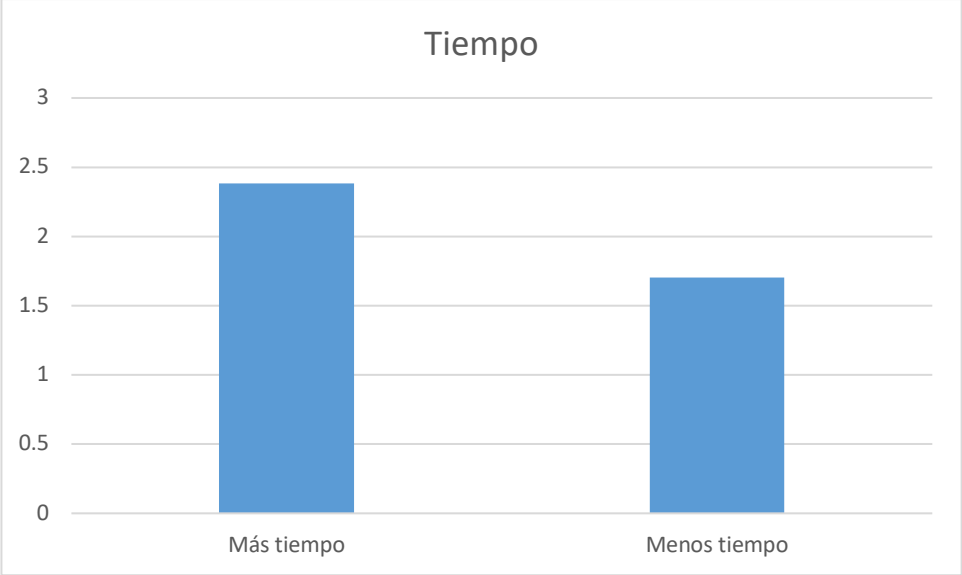

Ilustración 29 Promedio de resultados del tiempo de ejecución total en operaciones de escritura con Windows. Fuente: Elaboración propia

5.1.3.2.3.2 MacOS

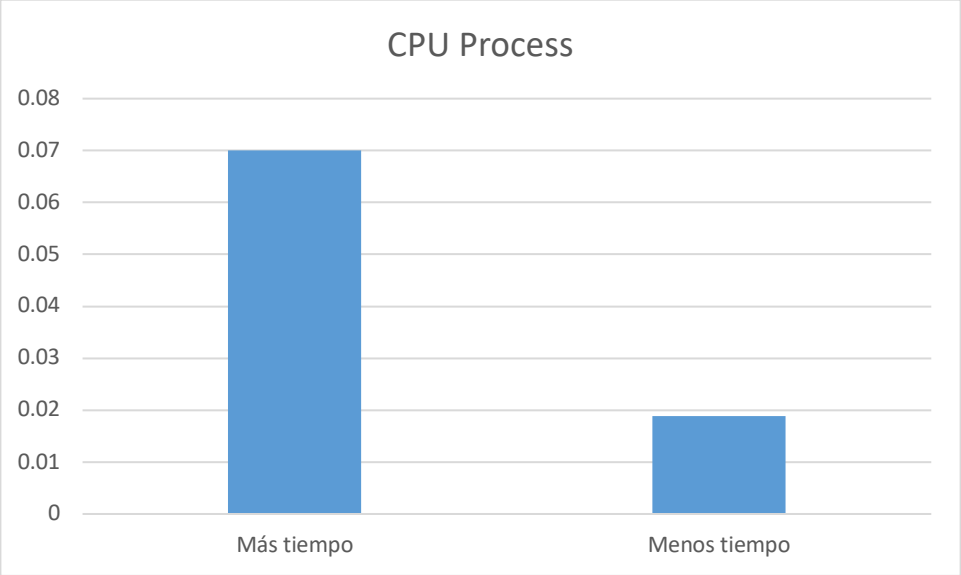

Ilustración 30 Promedio de resultados del CPU del proceso en operaciones de lectura con macOS. Fuente: Elaboración propia

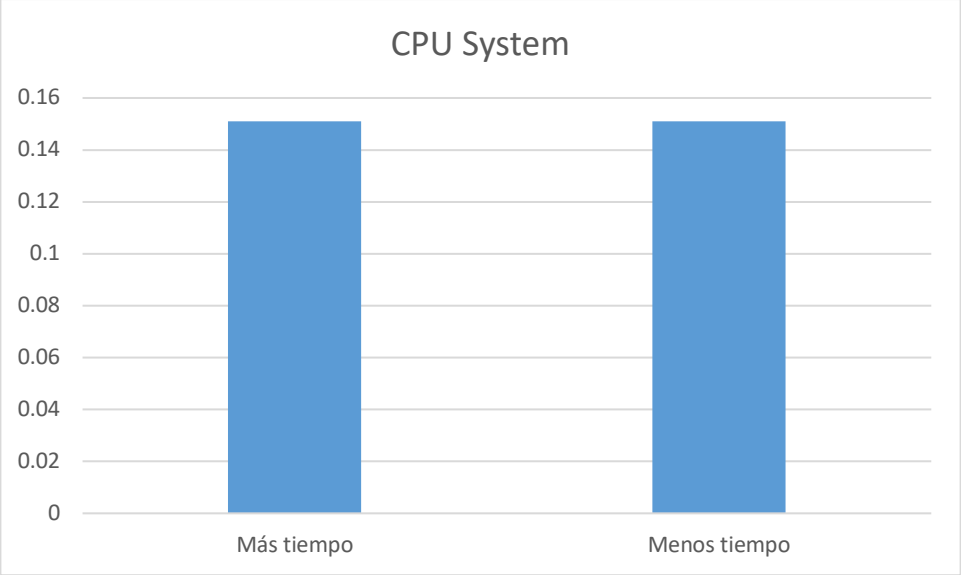

Ilustración 31 Promedio de resultados del CPU del sistema en operaciones de lectura con macOS. Fuente: Elaboración propia

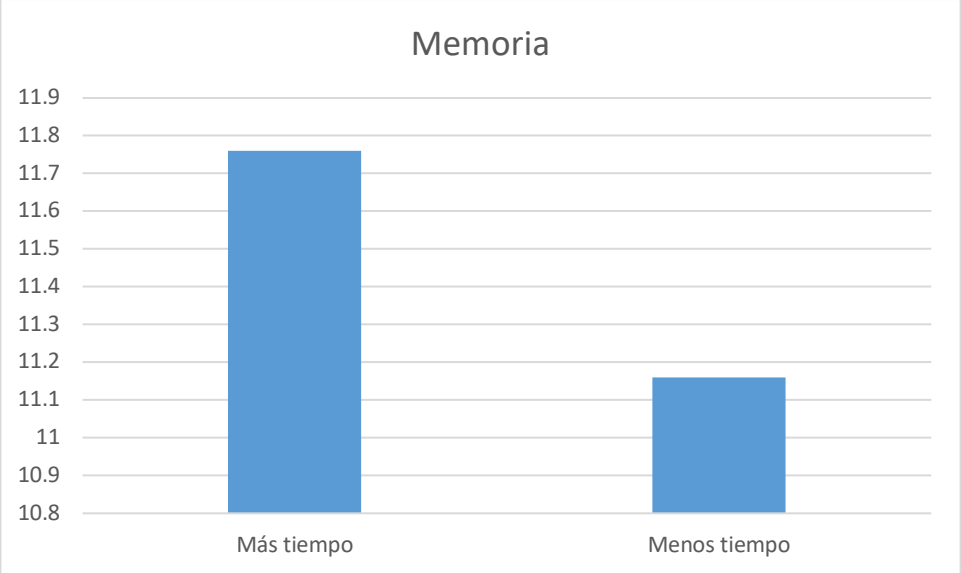

Ilustración 32 Promedio de resultados de la memoria en operaciones de lectura con macOS. Fuente: Elaboración propia

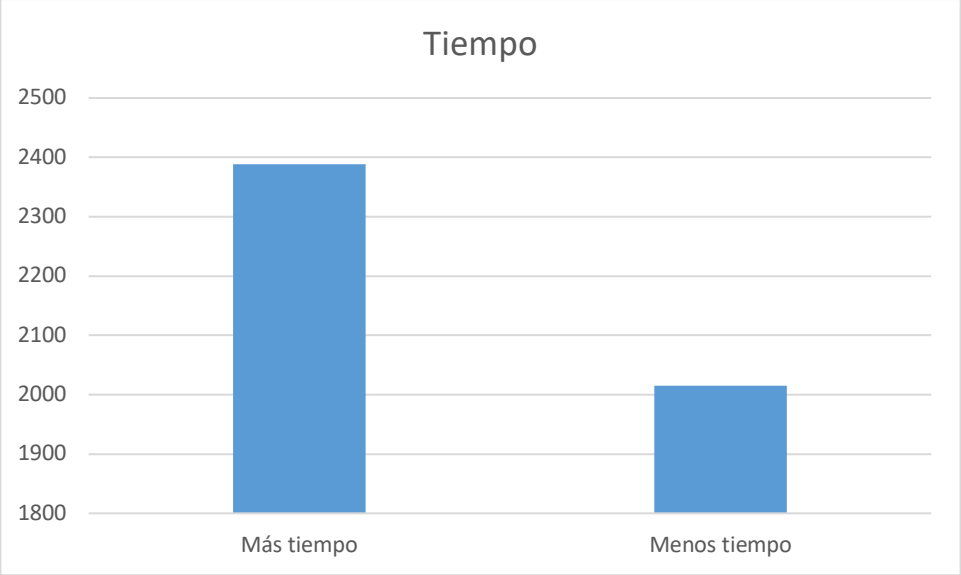

Ilustración 33 Promedio de resultados del tiempo de ejecución total en operaciones de escritura con macOS. Fuente: Elaboración propia

5.1.4 Comparación de pruebas con contenedores

5.1.4.1 Windows

Tabla 34 Resumen de resultados de pruebas con Docker en Windows. Fuente elaboración propia

Con Docker	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0170	0.0926	4.7	7948
Ejecución 2	0.0262	0.106	4.7	7317
Ejecución 3	0.0202	0.0921	4.8	7307
Promedio	**0.0216**	**0.0969**	**4.7**	**7524**

Tabla 35 Resumen de resultados de pruebas sin Docker en Windows. Fuente elaboración propia

Sin Docker	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.00649	0.0637	3.0	15246
Ejecución 2	0.00676	0.0871	2.7	14286
Ejecución 3	0.00437	0.0736	2.8	14360
Promedio	**0.00587**	**0.0736**	**2.8**	**14630**

5.1.4.2 MacOS

Tabla 36 Resumen de resultados de pruebas con Docker en macOS. Fuente elaboración propia

Con Docker	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0191	0.0769	15.4	3270
Ejecución 2	0.0142	0.0625	15.5	2863
Ejecución 3	0.0135	0.0574	17.6	2883
Promedio	**0.0156**	**0.0656**	**16.16**	**3005**

Tabla 37 Resumen de resultados de pruebas sin Docker en macOS. Fuente elaboración propia

Sin Docker	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0552	0.238	3	14817
Ejecución 2	0.0670	0.180	2.1	17343
Ejecución 3	0.0652	0.191	2.3	18108
Promedio	**0.0624**	**0.203**	**2.4**	**16756**

5.1.4.3 Resumen de resultados

De los resultados de las ejecuciones de las pruebas de diseño con contenedores se puede demostrar lo siguiente:

- Todos los recursos son más utilizados cuando se realizan las ejecuciones con contenedores Docker en sistemas operativos Windows.
- En el caso de los sistemas operativos MacOS, la memoria es consumida aproximadamente 8 veces más cuando se utiliza Docker.
- En ambos sistemas operativos, el tiempo de ejecución es mayor cuando se ejecuta sin contenedores.

5.1.4.3.1 Windows

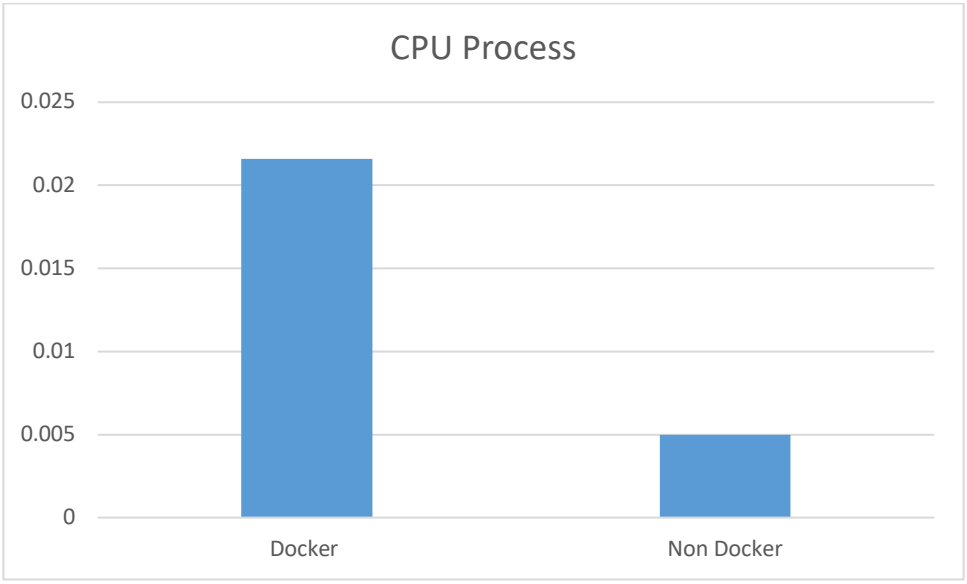

Ilustración 34 Promedio de resultados del CPU del proceso en pruebas de Docker en Windows. Fuente: Elaboración propia

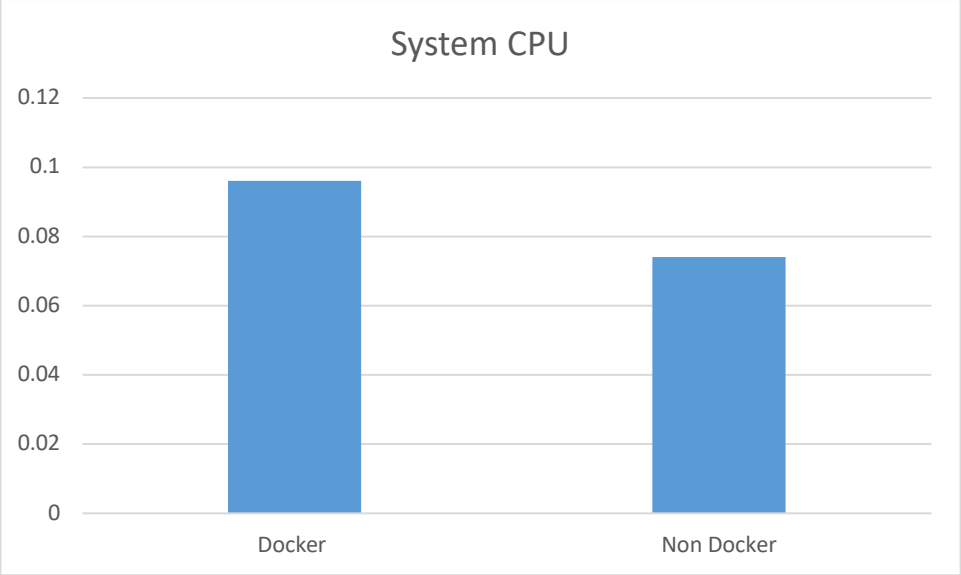

Ilustración 35 Promedio de resultados del CPU del sistema en pruebas de Docker en Windows. Fuente: Elaboración propia

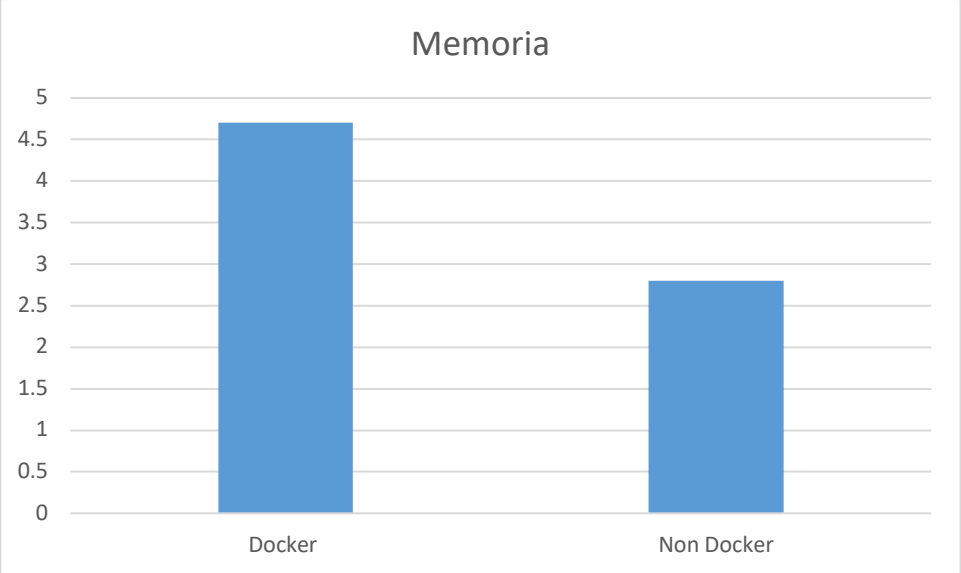

Ilustración 36 Promedio de resultados de memoria en pruebas de Docker en Windows. Fuente: Elaboración propia

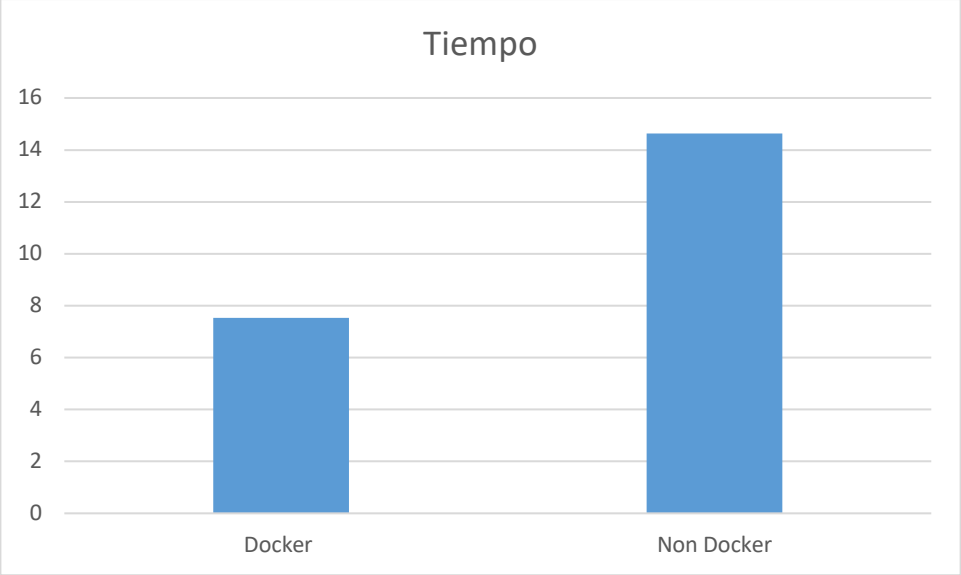

Ilustración 37 Promedio de resultados de tiempo de respuesta en pruebas de Docker en Windows. Fuente: Elaboración propia

5.1.4.3.2 MacOS

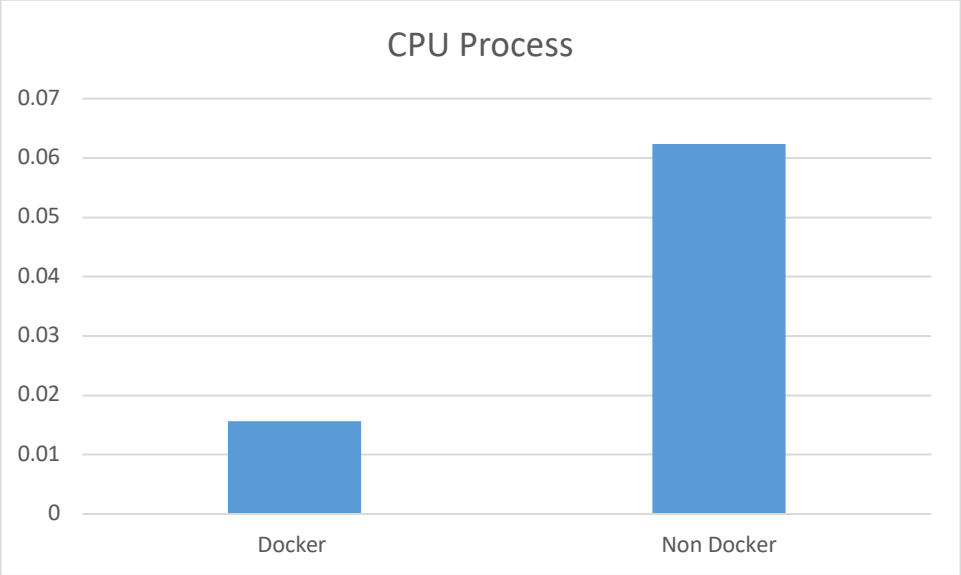

Ilustración 38 Promedio de resultados del CPU del proceso en pruebas de Docker en macOS. Fuente: Elaboración propia

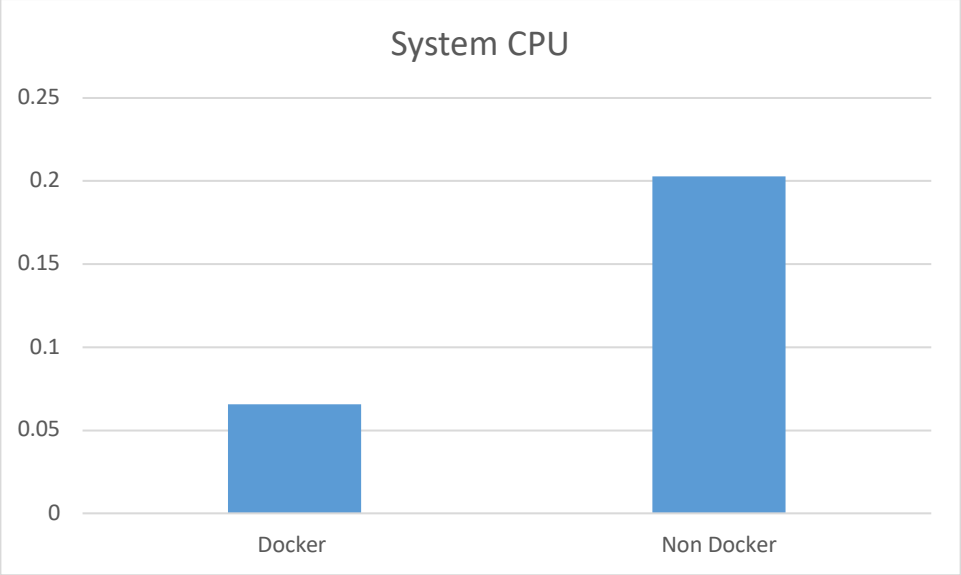

Ilustración 39 Promedio de resultados del CPU del sistema en pruebas de Docker en macOS. Fuente: Elaboración propia

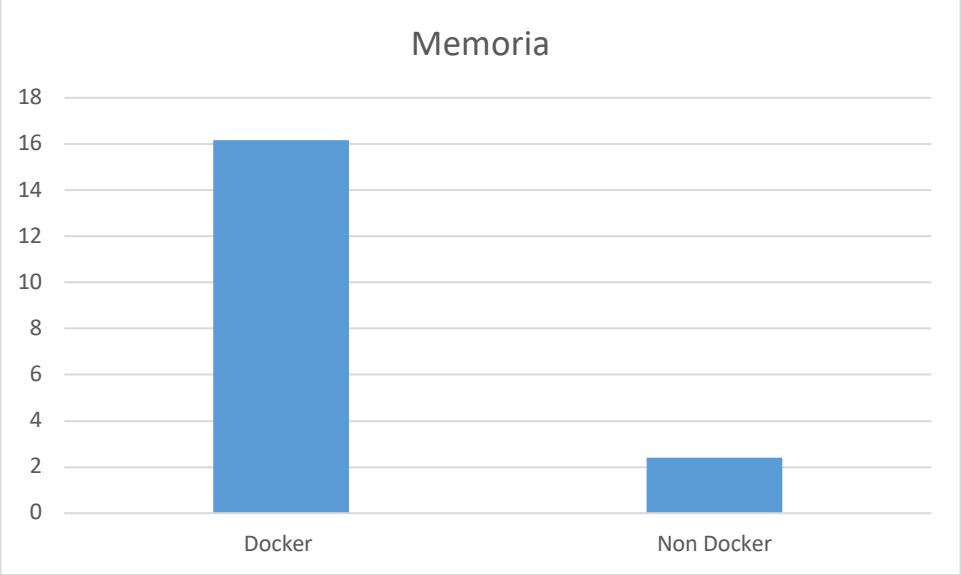

Ilustración 40 Promedio de resultados de memoria en pruebas de Docker en macOS. Fuente: Elaboración propia

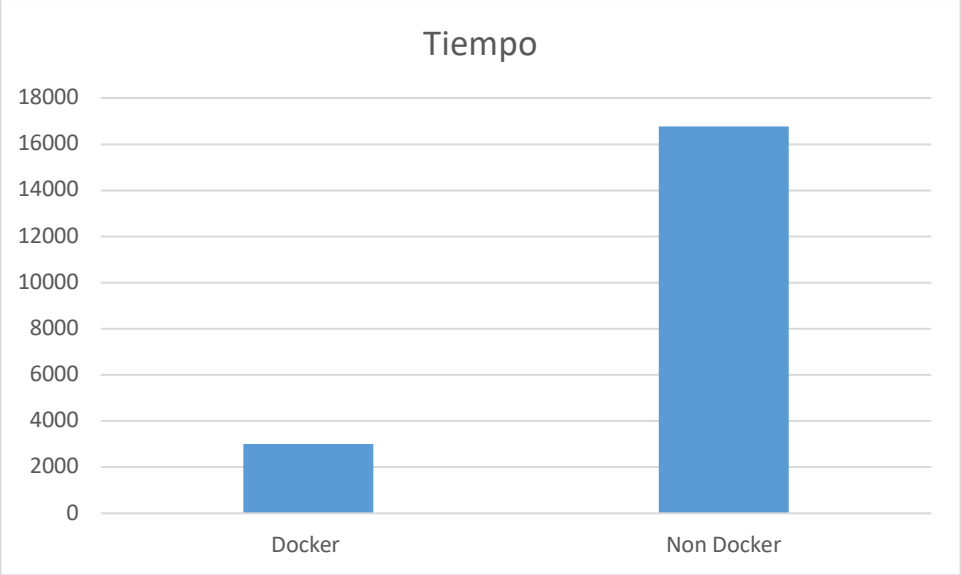

Ilustración 41 Promedio de resultados de tiempo de respuesta en pruebas de Docker en macOS. Fuente: Elaboración propia

5.1.5 Comparación de pruebas con batches

5.1.5.1 Windows

Tabla 38 Resumen de resultados de pruebas con batches en Windows. Fuente elaboración propia

Con Batches	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0227	0.122	4.2	3123
Ejecución 2	0.0127	0.0974	4.3	2944
Ejecución 3	0.0149	0.0977	4.4	2919
Promedio	**0.0167**	**0.105**	**4.3**	**2995**

Tabla 39 Resumen de resultados de pruebas sin batches en Windows. Fuente elaboración propia

Sin Batches	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0499	0.121	3.9	6493
Ejecución 2	0.0422	0.0989	4.3	6215
Ejecución 3	0.0414	0.0985	4.4	6181
Promedio	**0.0445**	**0.1061**	**4.2**	**6296**

5.1.5.2 MacOS

Tabla 40 Resumen de resultados de pruebas con batches en macOS. Fuente elaboración propia

Con Batches	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0352	0.206	2.8	1106
Ejecución 2	0.0301	0.196	2.7	1077
Ejecución 3	0.0301	0.199	2.9	1073
Promedio	**0.0318**	**0.200**	**2.8**	**1085**

Tabla 41 Resumen de resultados de pruebas sin batches en macOS. Fuente elaboración propia

Sin Batches	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0785	0.199	2.7	2373
Ejecución 2	0.0777	0.201	2.8	2337
Ejecución 3	0.0776	0.200	2.8	2327
Promedio	**0.0779**	**0.200**	**2.7**	**2345**

5.1.5.3 Resumen de resultados

De los resultados de las ejecuciones de las pruebas de uso de batches se puede demostrar lo siguiente:

- El CPU es menor al usar batches en la aplicación, tanto para el CPU del proceso como del sistema en el caso del sistema operativo Windows.
- En el caso de MacOS, el CPU del proceso es menor haciendo uso de batches, sin embargo, el CPU del sistema es igual en ambas variantes.
- La memoria es muy similar en ambas y la diferencia no es relevante en ambos sistemas operativos.
- El tiempo de ejecución del diseño con batches es menor que sin el uso de batches en ambos sistemas operativos.

5.1.5.3.1 Windows

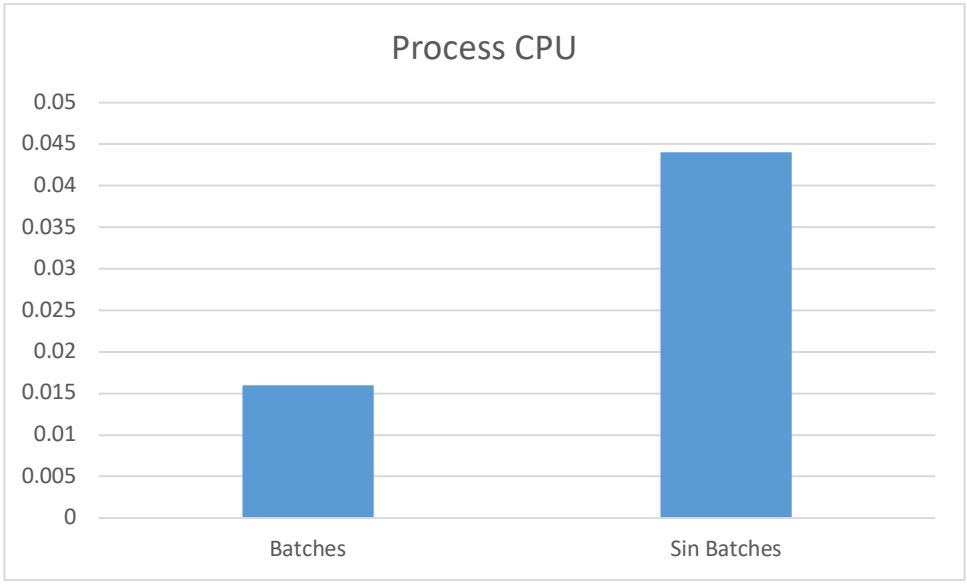

Ilustración 42 Promedio de resultados del CPU del proceso en pruebas de uso de batches en Windows. Fuente: Elaboración propia

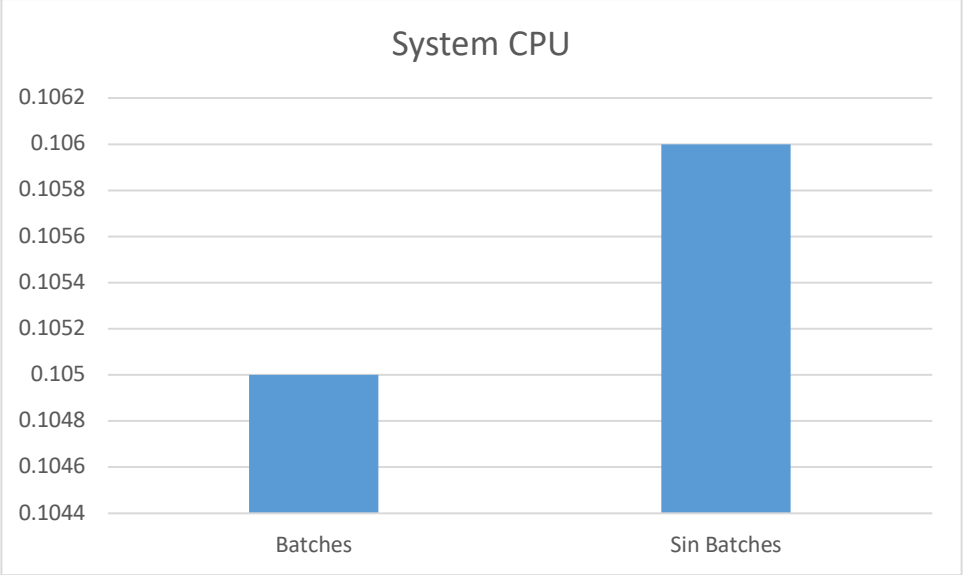

Ilustración 43 Promedio de resultados del CPU del sistema en pruebas de uso de batches en Windows. Fuente: Elaboración propia

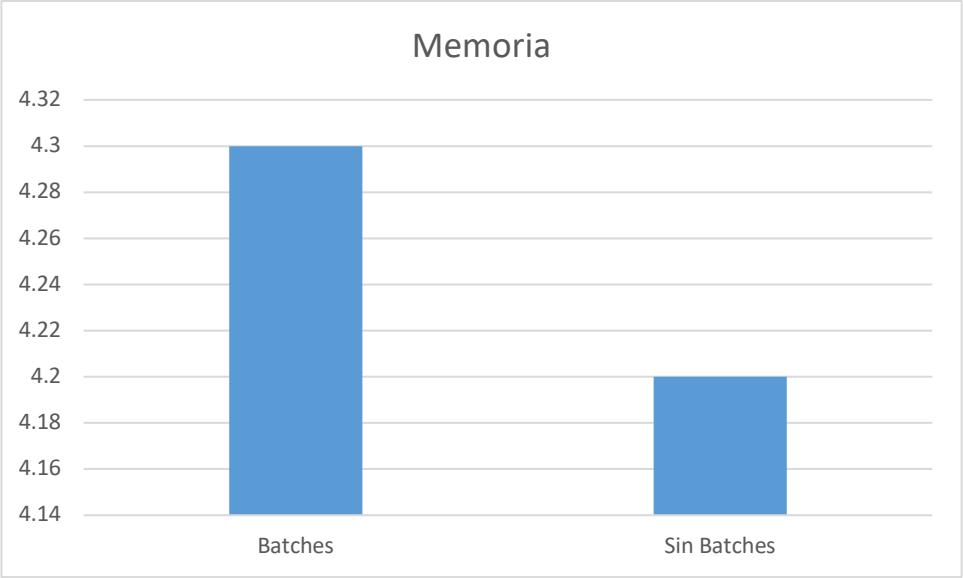

Ilustración 44 Promedio de resultados de la memoria en pruebas de uso de batches en Windows. Fuente: Elaboración propia

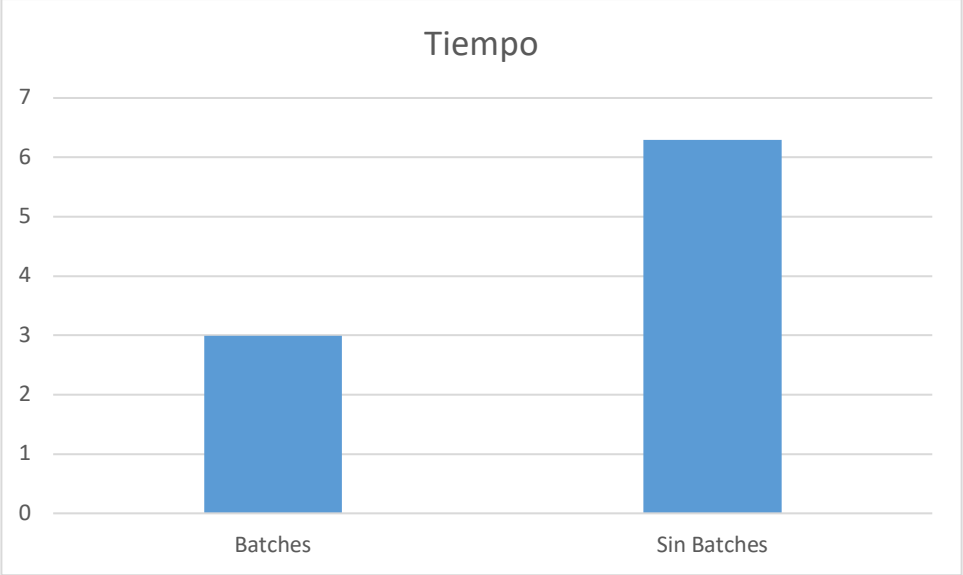

Ilustración 45 Promedio de resultados de tiempo de respuesta en pruebas de uso de batches en Windows. Fuente: Elaboración propia

5.1.5.3.2 MacOS

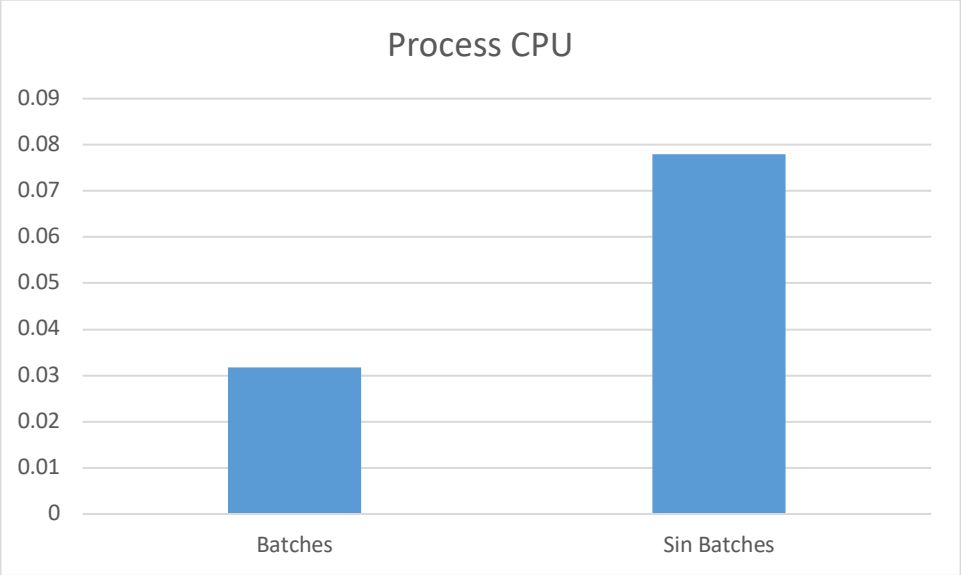

Ilustración 46 Promedio de resultados del CPU del proceso en pruebas de uso de batches en macOS. Fuente: Elaboración propia

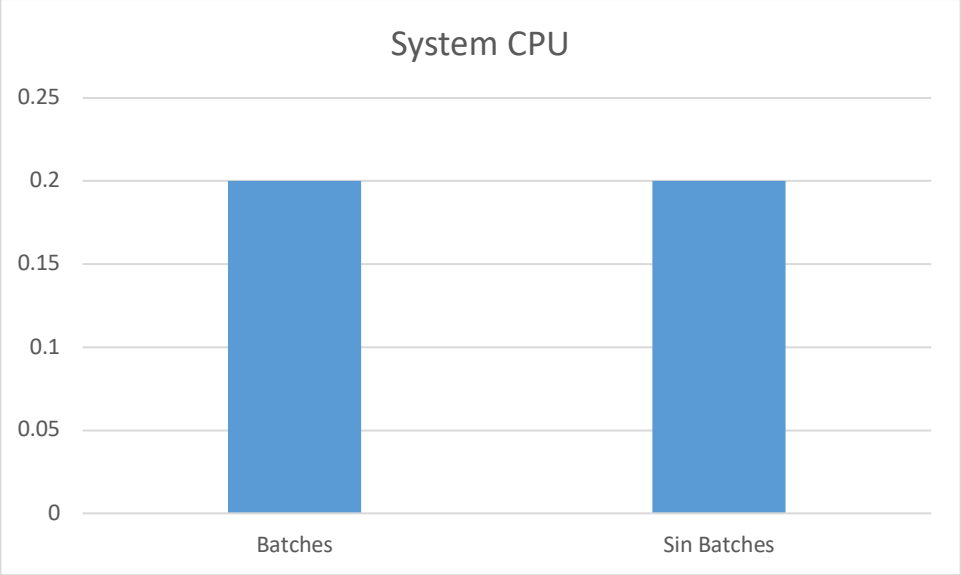

Ilustración 47 Promedio de resultados del CPU del sistema en pruebas de uso de batches en macOS. Fuente: Elaboración propia

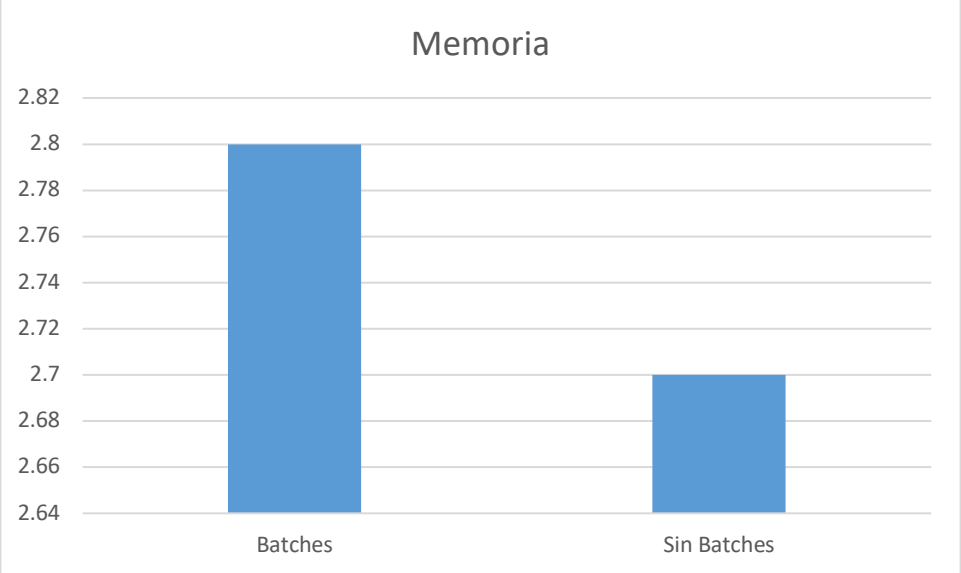

Ilustración 48 Promedio de resultados de la memoria en pruebas de uso de batches en macOS. Fuente: Elaboración propia

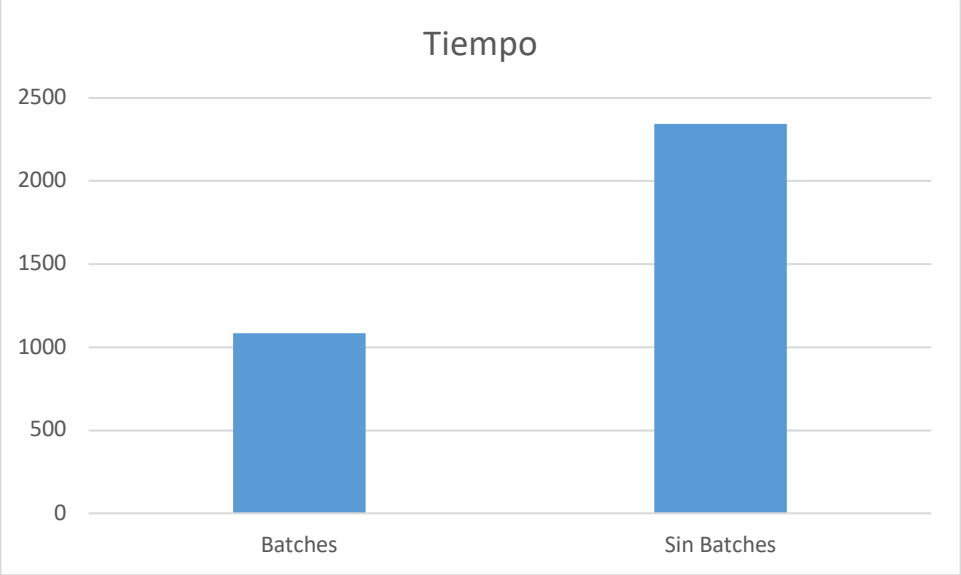

Ilustración 49 Promedio de resultados de tiempo de respuesta en pruebas de uso de batches en macOS. Fuente: Elaboración propia

5.1.6 Comparación de pruebas con hilos

5.1.6.1 Windows

Tabla 42 Resumen de resultados de pruebas con hilos en Windows. Fuente elaboración propia

Con Hilos	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0564	0.151	9.7	1167
Ejecución 2	0.0526	0.135	9.4	1145
Ejecución 3	0.0677	0.157	11	1159
Ejecución 4	0.0566	0.144	10.6	1139
Ejecución 5	0.0691	0.158	10.8	1153
Promedio	**0.0604**	**0.149**	**10.3**	**1152.6**

Tabla 43 Resumen de resultados de pruebas sin batches en Windows. Fuente elaboración propia

Sin Hilos	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0613	0.142	9.7	2656
Ejecución 2	0.0552	0.144	9.9	2661
Ejecución 3	0.0542	0.152	10	2651
Ejecución 4	0.530	0.147	10.4	2648

Ejecución 5	0.0539	0.154	9.6	2666
Promedio	**0.0555**	**0.147**	**9.9**	**2656.4**

5.1.6.2 MacOS

Tabla 44 Resumen de resultados de pruebas con hilos en macOS. Fuente elaboración propia

Con Hilos	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0675	0.217	18.2	1776
Ejecución 2	0.0637	0.218	21.1	1775
Ejecución 3	0.0848	0.244	10.1	1810
Promedio	**0.0720**	**0.226**	**16.46**	**1787**

Tabla 45 Resumen de resultados de pruebas sin hilos en macOS. Fuente elaboración propia

Sin Hilos	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0796	0.218	13.1	3290
Ejecución 2	0.0766	0.210	11.5	3199
Ejecución 3	0.0817	0.242	10.6	3305
Promedio	**0.0793**	**0.223**	**11.7**	**3264**

5.1.6.3 Resumen de resultados

De los resultados de las ejecuciones de las pruebas de uso de hilos se puede demostrar lo siguiente:

- A pesar de que el tiempo de ejecución haciendo uso de concurrencia es menor, todos los recursos del hardware, tanto del CPU como de la memoria se incrementan con respecto a las ejecuciones sin hilos en el sistema operativo Windows.
- En el caso de MacOS, el comportamiento es similar al de Windows, con la salvedad de que el CPU del proceso es menor con hilos.
- Lo anterior demuestra que no siempre un tiempo de ejecución menor significa menor consumo de recursos de hardware.

5.1.6.3.1 Windows

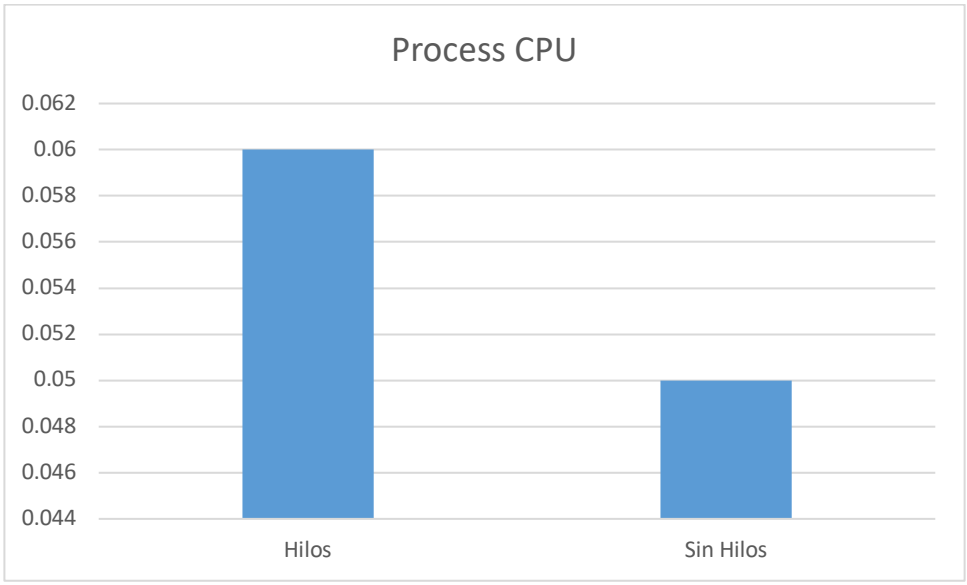

Ilustración 50 Promedio de resultados del CPU del proceso en pruebas de uso de hilos en Windows. Fuente: Elaboración propia

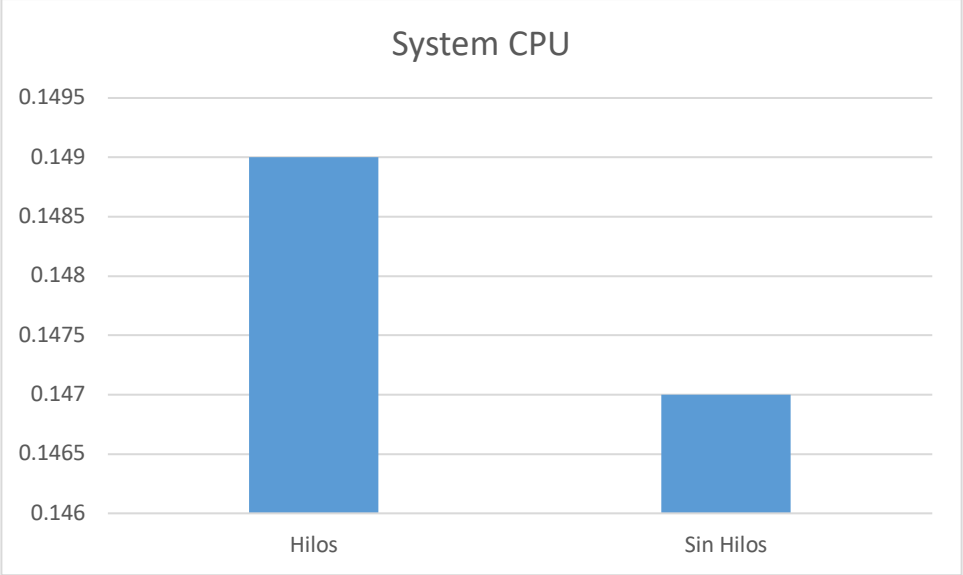

Ilustración 51 Promedio de resultados del CPU del sistema en pruebas de uso de hilos en Windows. Fuente: Elaboración propia

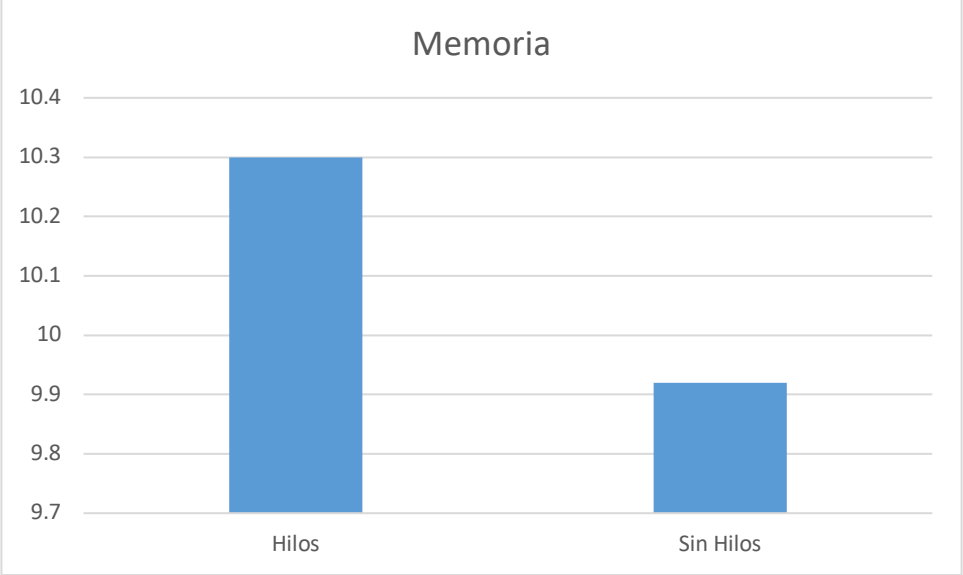

Ilustración 52 Promedio de resultados de la memoria en pruebas de uso de hilos en Windows. Fuente: Elaboración propia

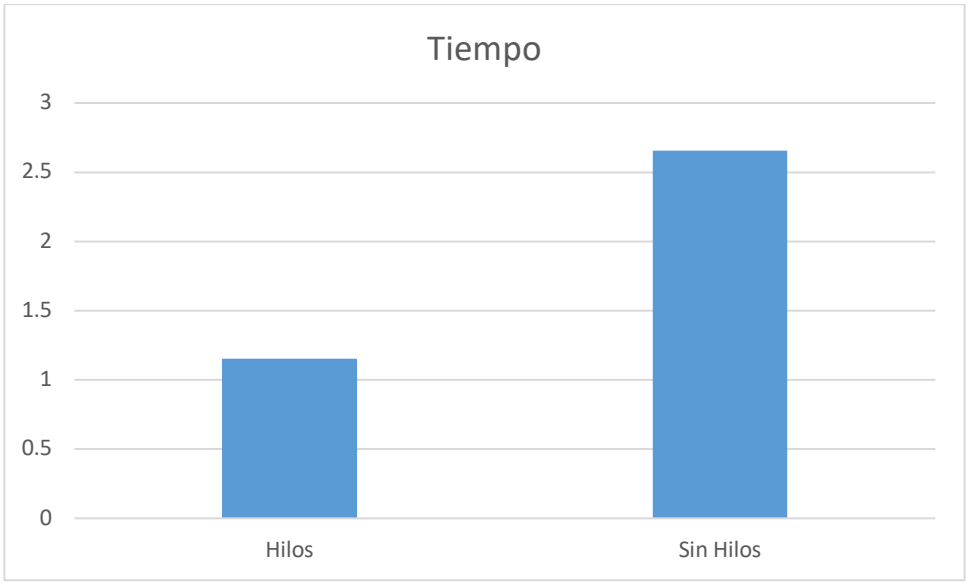

Ilustración 53 Promedio de resultados de tiempos de respuesta en pruebas de uso de hilos en Windows. Fuente: Elaboración propia

5.1.6.3.2 MacOS

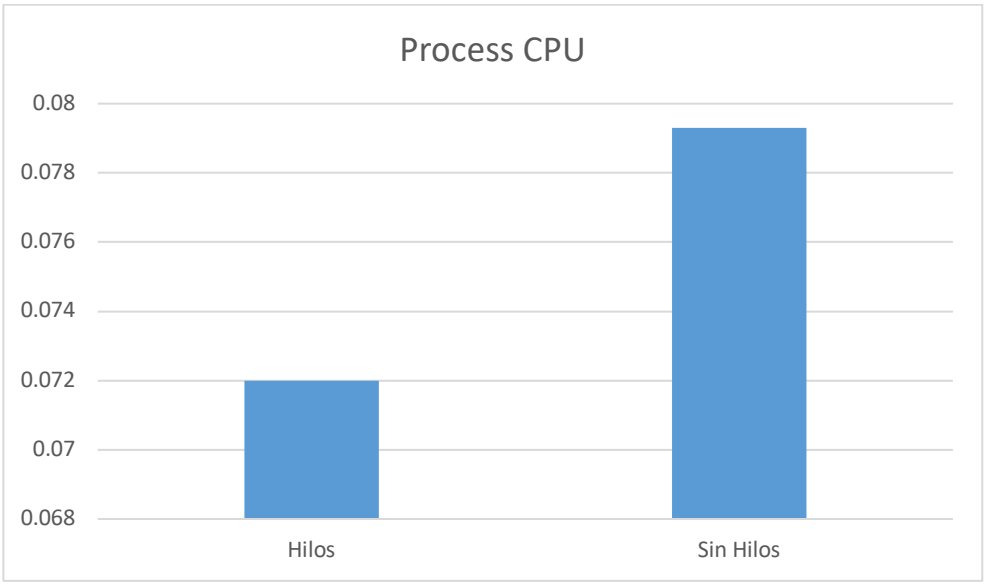

Ilustración 54 Promedio de resultados del CPU del proceso en pruebas de uso de hilos en macOS. Fuente: Elaboración propia

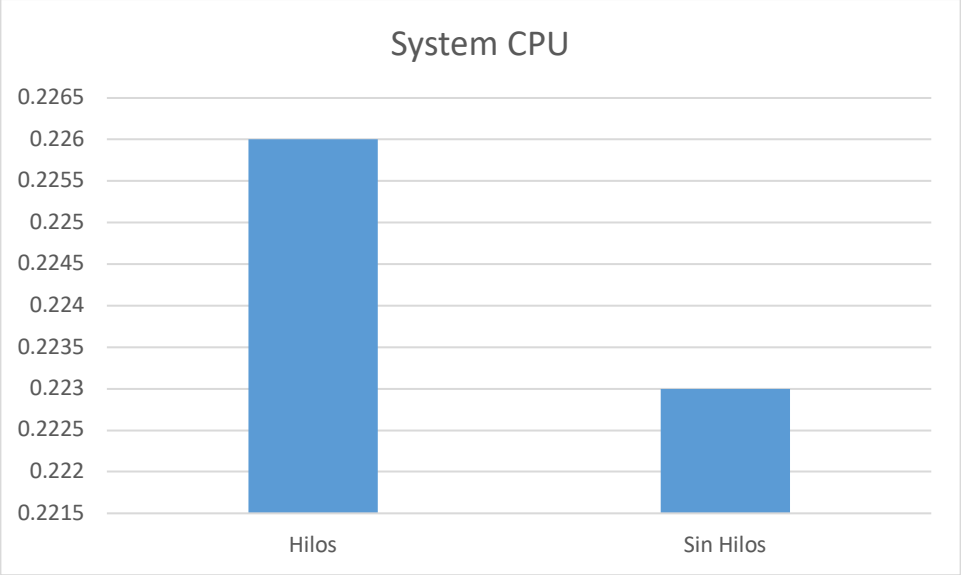

Ilustración 55 Promedio de resultados del CPU del sistema en pruebas de uso de hilos en macOS. Fuente: Elaboración propia

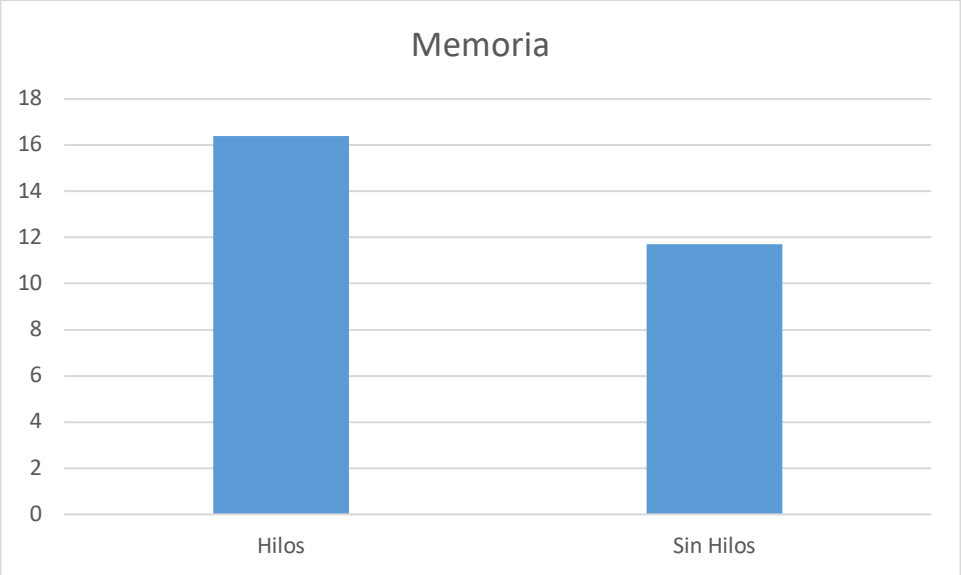

Ilustración 56 Promedio de resultados de la memoria en pruebas de uso de hilos en macOS. Fuente: Elaboración propia

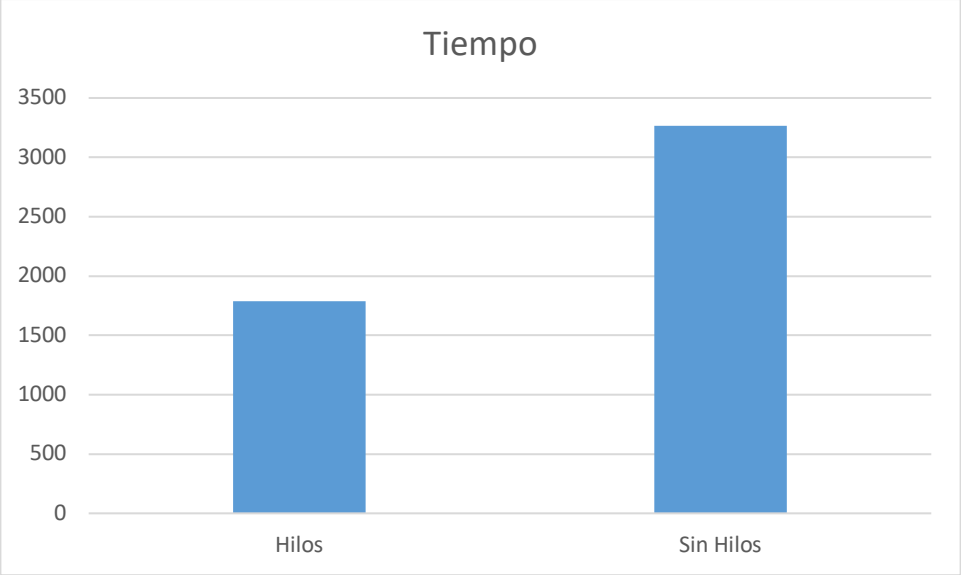

Ilustración 57 Promedio de resultados de tiempos de respuesta en pruebas de uso de hilos en macOS. Fuente: Elaboración propia

5.1.7 Comparación de pruebas con cache

5.1.7.1 Pruebas de operaciones de lectura de consultas ligeras

5.1.7.1.1 Windows

Tabla 46 Resumen de resultados de pruebas de consultas ligeras con cache en Windows. Fuente elaboración propia

Con Cache	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0556	0.162	5.7	3020
Ejecución 2	0.0318	0.114	6.7	4078
Ejecución 3	0.0376	0.132	7	3415
Promedio	**0.0416**	**0.136**	**6.4**	**3504**

Tabla 47 Resumen de resultados de pruebas consultas ligeras sin cache en Windows. Fuente elaboración propia

Sin Cache	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0424	0.164	4.4	3003
Ejecución 2	0.0302	0.115	4.6	4064
Ejecución 3	0.0354	0.132	4.8	3403
Promedio	**0.0360**	**0.137**	**4.6**	**3460**

5.1.7.1.2 Resumen de resultados

De los resultados de las ejecuciones de las pruebas de cache con operaciones más costosas se puede demostrar lo siguiente:

- Con el uso del cache, tanto el CPU como la memoria son mayor que sin el uso del cache, sin embargo, para operaciones costosas el tiempo de ejecución lo compensa, ya que es menor que las ejecuciones con cache.

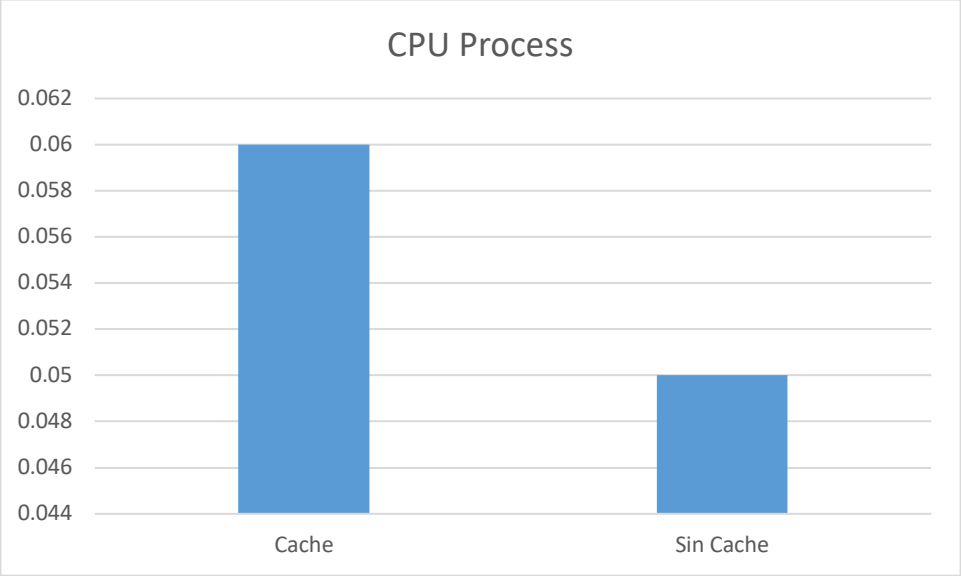

Ilustración 58 Promedio de resultados del CPU del proceso en pruebas de uso de cache con consultas ligeras en Windows. Fuente: Elaboración propia

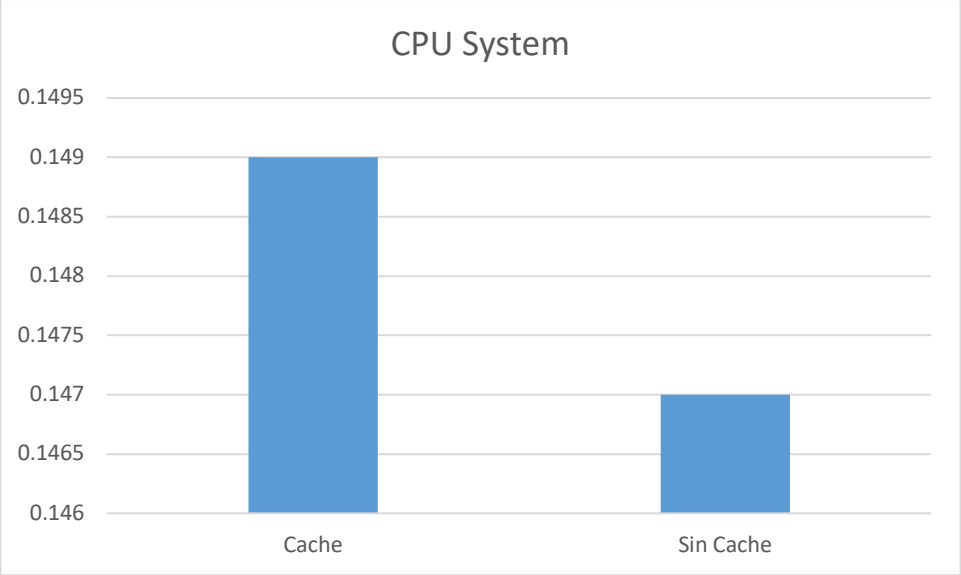

Ilustración 59 Promedio de resultados del CPU del sistema en pruebas de uso de cache con consultas ligeras en Windows. Fuente: Elaboración propia

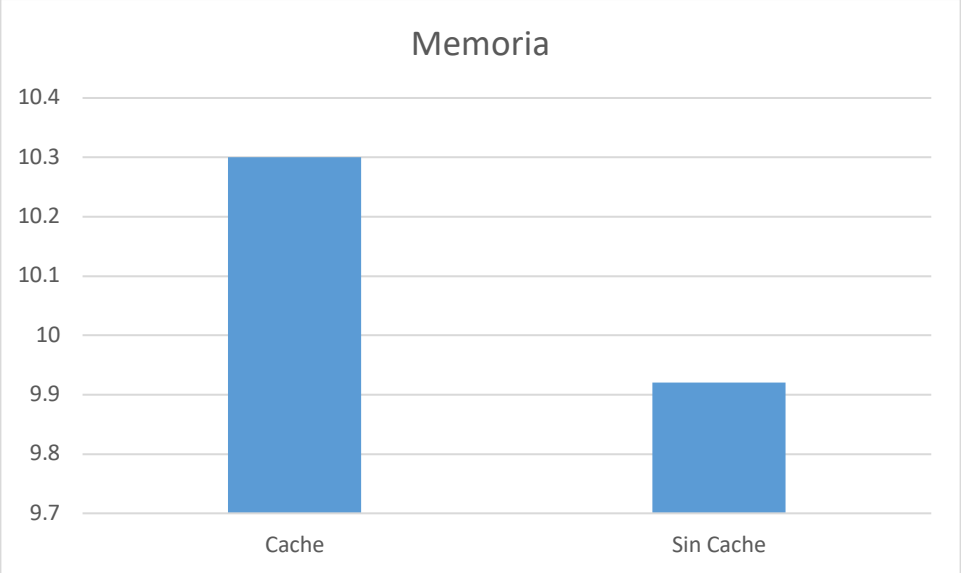

Ilustración 60 Promedio de resultados de la memoria en pruebas de uso de cache con consultas ligeras en Windows. Fuente: Elaboración propia

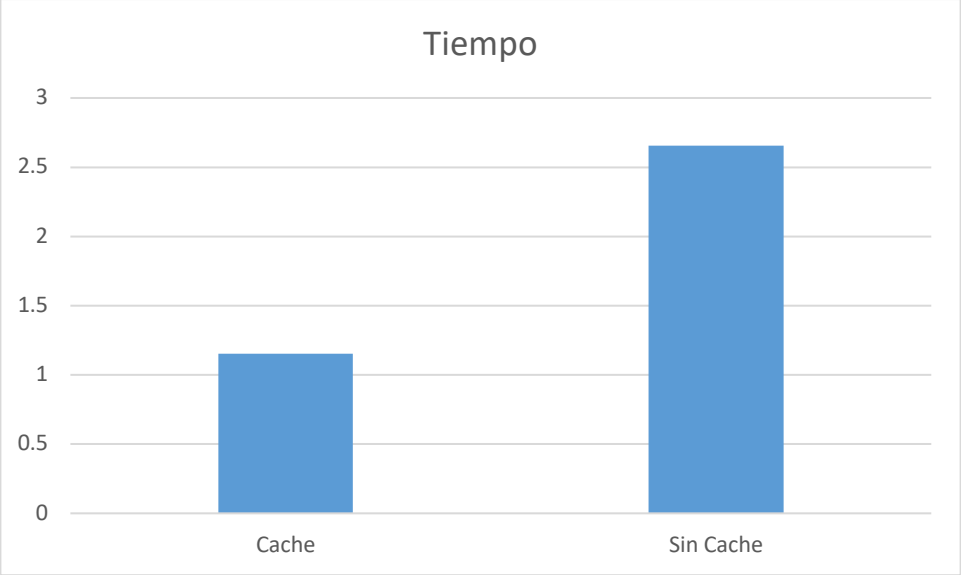

Ilustración 61 Promedio de resultados de tiempo de respuesta en pruebas de uso de cache con consultas ligeras en Windows. Fuente: Elaboración propia

5.1.7.2 Pruebas de operaciones de lectura de consultas pesadas

5.1.7.2.1 Windows

Tabla 48 Resumen de resultados de pruebas de consultas pesadas con cache en Windows. Fuente elaboración propia

Con Cache	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0350	0.131	5.1	3578
Ejecución 2	0.0318	0.113	7.5	3502
Ejecución 3	0.311	0.115	7.5	3464
Promedio	**0.1259**	**0.119**	**6.7**	**3514**

Tabla 49 Resumen de resultados de pruebas de consultas pesadas sin cache en Windows. Fuente elaboración propia

Sin Cache	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0354	0.132	4.3	4174
Ejecución 2	0.0298	0.113	4.3	4108
Ejecución 3	0.0296	0.114	4.6	4065
Promedio	0.0316	0.119	4.4	4115

5.1.7.2.2 MacOS

Tabla 50 Resumen de resultados de pruebas de consultas pesadas con cache en macOS. Fuente elaboración propia

Con Cache	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0550	0.195	3.7	1236
Ejecución 2	0.0543	0.196	3.7	1211
Ejecución 3	0.0486	0.189	4.2	1200
Promedio	**0.0526**	**0.193**	**3.8**	**1215**

Tabla 51 Resumen de resultados de pruebas de consultas pesadas sin cache en macOS. Fuente elaboración propia

Sin Cache	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0550	0.195	2.8	1857
Ejecución 2	0.0541	0.196	2.6	1821
Ejecución 3	0.0531	0.192	2.8	1826
Promedio	**0.0540**	**0.194**	**2.7**	**1834**

5.1.7.2.3 Resumen de resultados

De los resultados de las ejecuciones de las pruebas de cache con operaciones menos costosas se puede demostrar lo siguiente:

- En el caso de las ejecuciones en Windows, contrario a ejecuciones con operaciones más costosas, el uso del cache hace una mejor utilización de los componentes de CPU y memoria, aunque a un costo de tiempo un poco mayor que sin el cache, sin embargo la diferencia es casi medio segundo.
- En el caso de las ejecuciones en MacOS, todas las mediciones fueron mejores haciendo uso de cache, exceptuando la memoria que se incrementó con respecto a las ejecuciones sin cache.

5.1.7.2.3.1 Windows

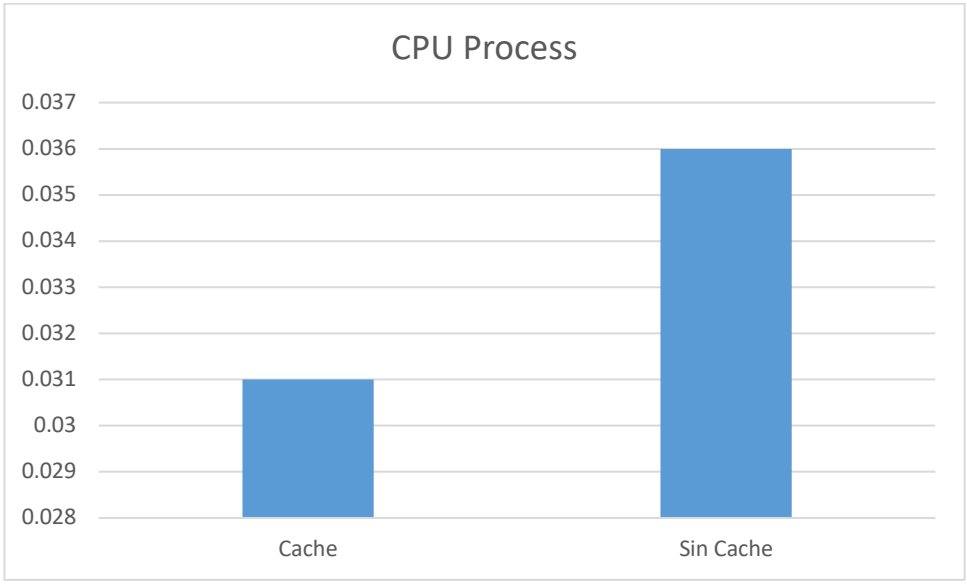

Ilustración 62 Promedio de resultados del CPU del proceso en pruebas de uso de cache con consultas pesadas en Windows. Fuente: Elaboración propia

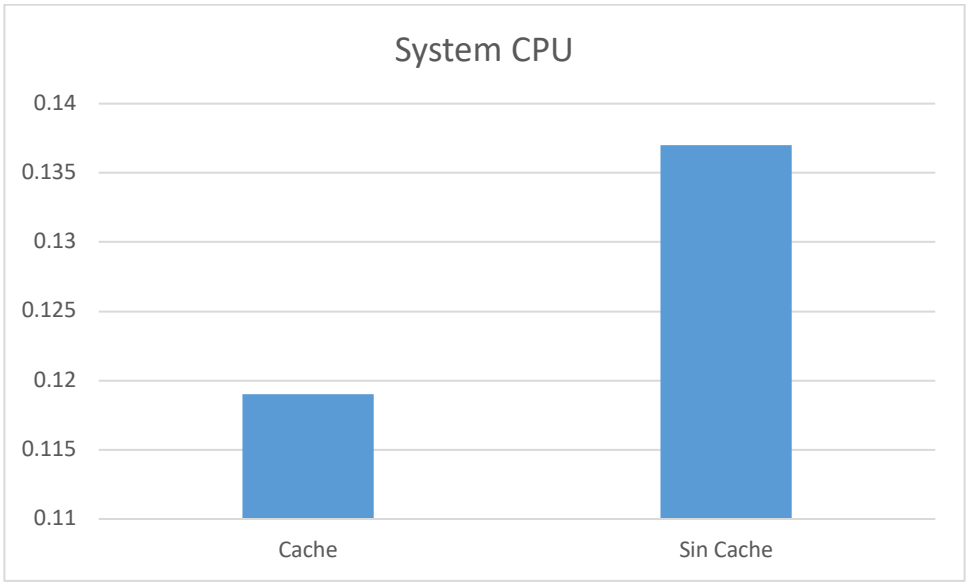

Ilustración 63 Promedio de resultados del CPU del sistema en pruebas de uso de cache con consultas pesadas en Windows. Fuente: Elaboración propia

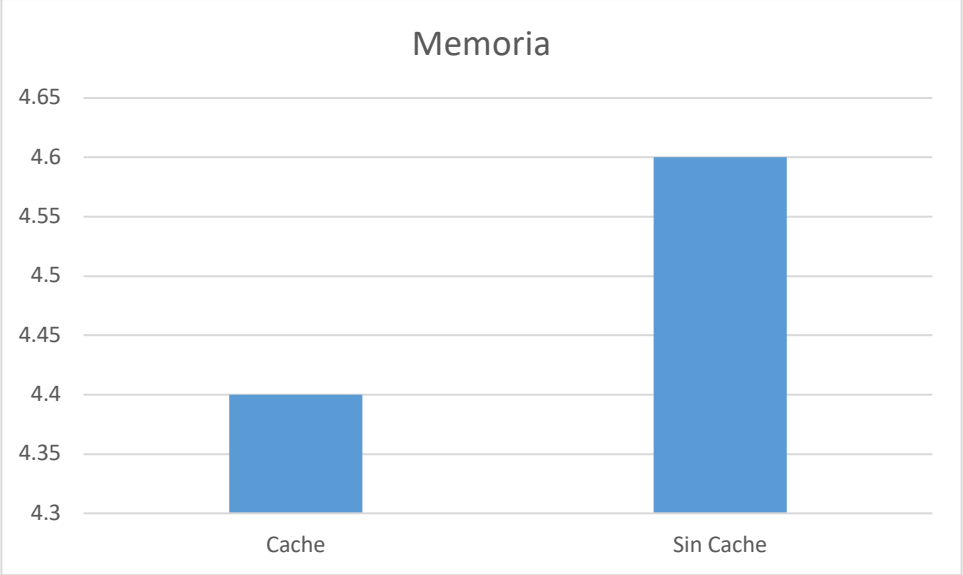

Ilustración 64 Promedio de resultados de la memoria en pruebas de uso de cache con consultas pesadas en Windows. Fuente: Elaboración propia

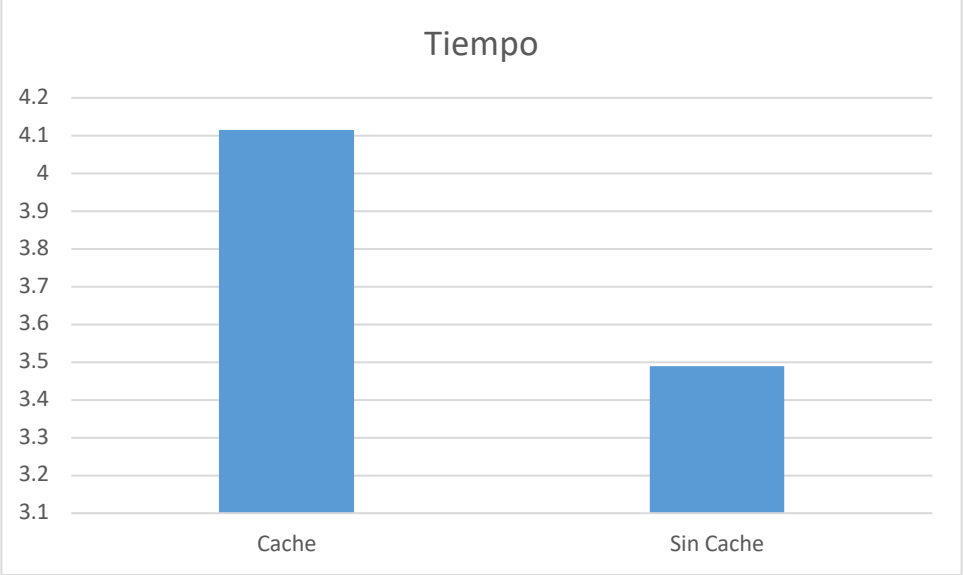

Ilustración 65 Promedio de resultados de tiempos de respuesta en pruebas de uso de cache con consultas pesadas en Windows. Fuente: Elaboración propia

5.1.7.2.3.2 MacOS

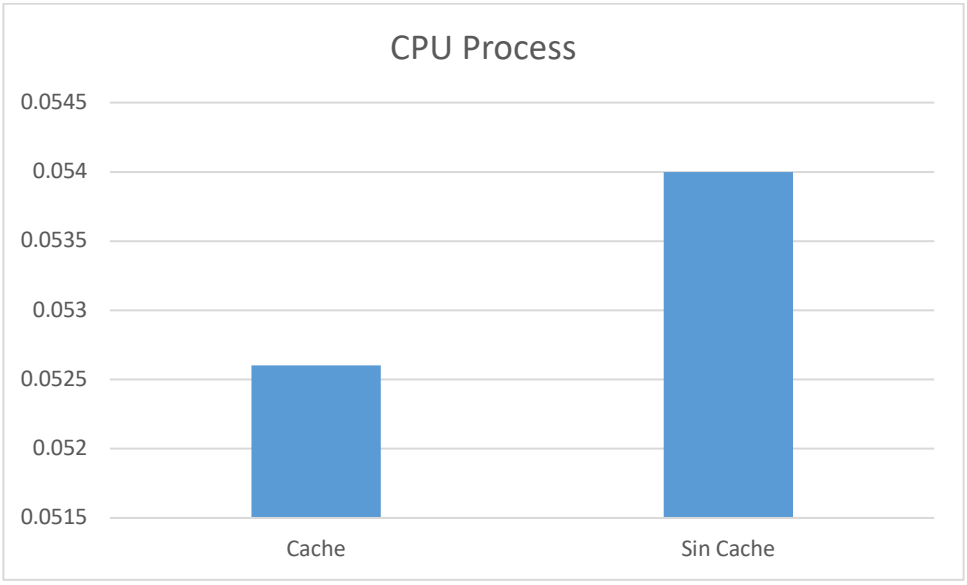

Ilustración 66 Promedio de resultados del CPU del proceso en pruebas de uso de cache con consultas pesadas en macOS. Fuente: Elaboración propia

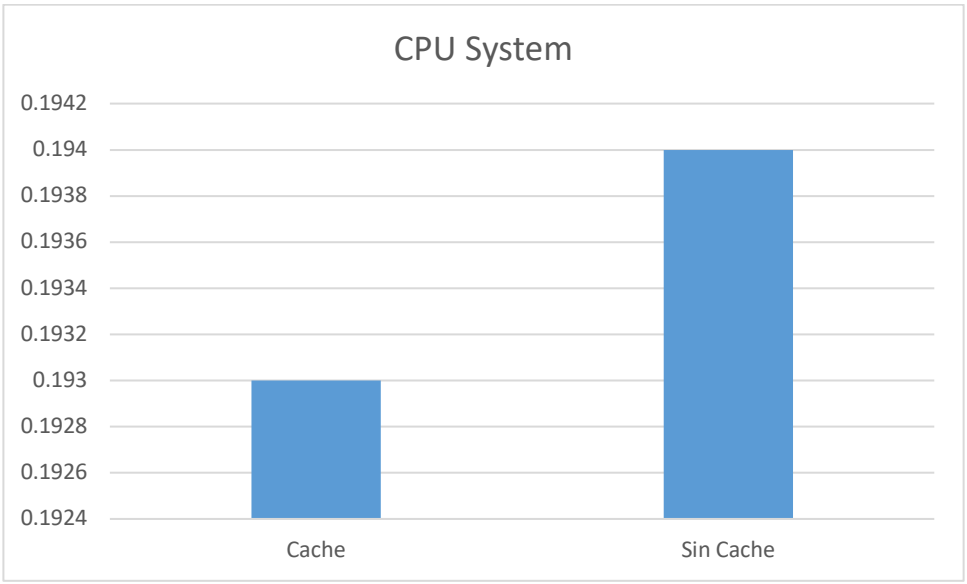

Ilustración 67 Promedio de resultados del CPU del sistema en pruebas de uso de cache con consultas pesadas en macOS. Fuente: Elaboración propia

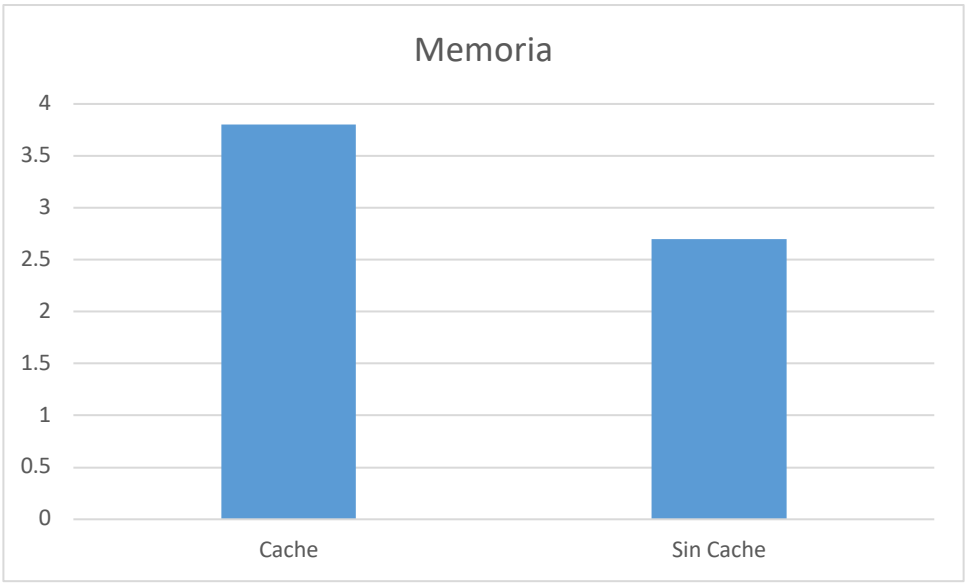

Ilustración 68 Promedio de resultados de la memoria en pruebas de uso de cache con consultas pesadas en macOS. Fuente: Elaboración propia

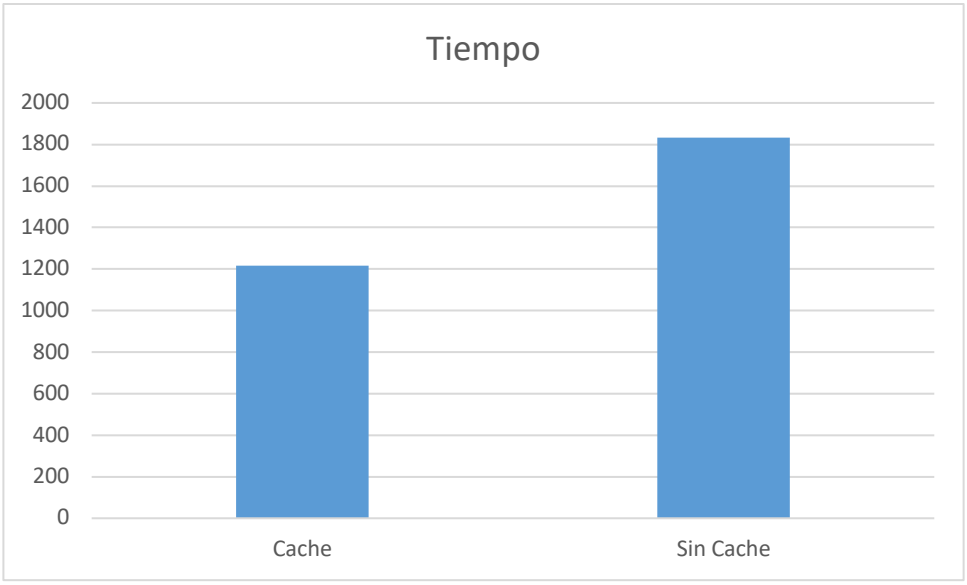

Ilustración 69 Promedio de resultados de tiempos de respuesta en pruebas de uso de cache con consultas pesadas en macOS. Fuente: Elaboración propia

5.1.8 Comparación de pruebas con REST y Mensajes con RabbitMQ

5.1.8.1 Windows

Tabla 52 Resumen de resultados de pruebas de comunicación con REST en Windows. Fuente elaboración propia

Con REST	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0379	0.189	6.5	N/A
Ejecución 2	0.0369	0.183	6.7	N/A
Ejecución 3	0.0394	0.180	6.3	N/A
Promedio	**0.0380**	**0.184**	**6.3**	**N/A**

Tabla 53 Resumen de resultados de pruebas de comunicación con RabbitMQ en Windows. Fuente elaboración propia

Con RabbitMQ	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0431	0.186	5.2	N/A
Ejecución 2	0.0453	0.184	5.2	N/A
Ejecución 3	0.0452	0.180	5.6	N/A
Promedio	**0.0445**	**0.183**	**5.9**	**N/A**

5.1.8.2 MacOS

Tabla 54 Resumen de resultados de pruebas de comunicación con REST en macOS. Fuente elaboración propia

Con REST	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0641	0.782	5.2	N/A
Ejecución 2	0.0662	0.806	5.4	N/A
Ejecución 3	0.0588	0.808	5.4	N/A
Promedio	**0.0630**	**0.798**	**5.3**	**N/A**

Tabla 55 Resumen de resultados de pruebas de comunicación con RabbitMQ en macOS. Fuente elaboración propia

Con RabbitMQ	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0742	0.783	3.4	N/A
Ejecución 2	0.0706	0.807	3.4	N/A
Ejecución 3	0.0677	0.809	3.9	N/A
Promedio	**0.070**	**0.799**	**3.5**	**N/A**

5.1.8.3 Resumen de resultados

De los resultados de las ejecuciones de las pruebas de uso de REST o RabbitMQ se puede demostrar lo siguiente:

- En ambos sistemas operativos el CPU del proceso fue menor con REST que con RabbitMQ.
- En el caso del CPU del sistema, la diferencia fue mínima en ambos sistemas operativos. En MacOS el uso del CPU fue menor en la prueba de REST, mientras que en Windows, la pruena en RabbitMQ fue menor.
- La memoria fue mayor en REST en ambos sistemas operativos.

5.1.2.8.3.1 Windows

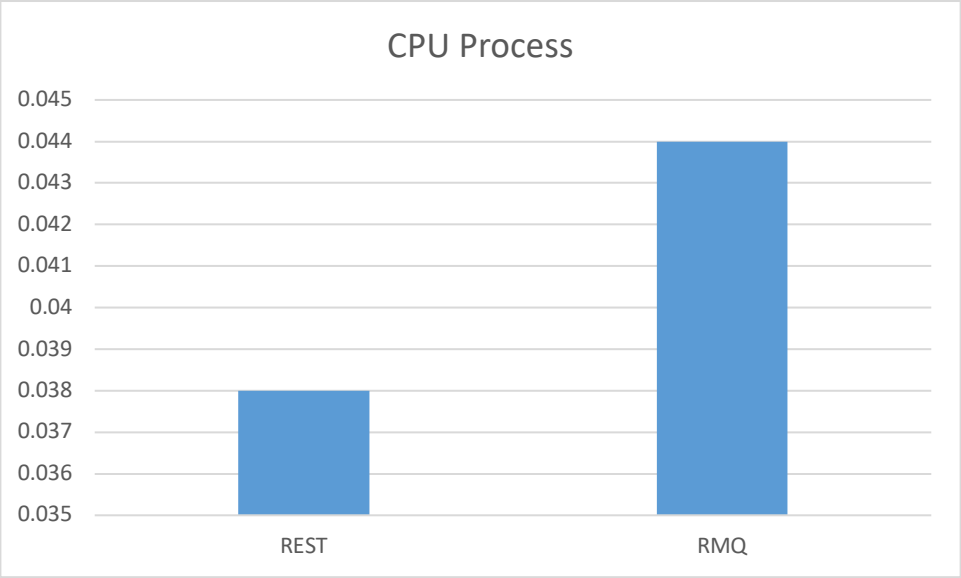

Ilustración 70 Promedio de resultados del CPU del proceso en pruebas de comunicación entre servicios en Windows. Fuente: Elaboración propia

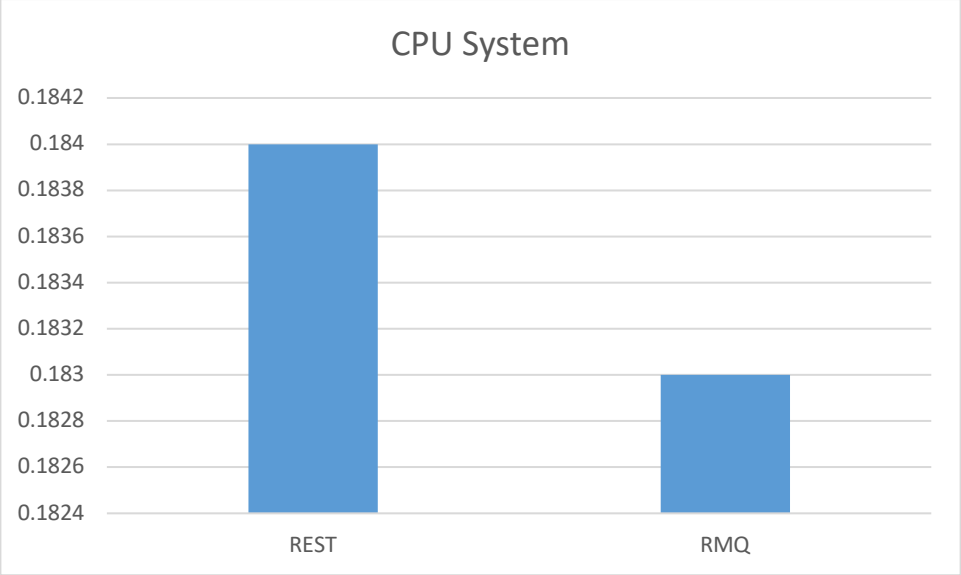

Ilustración 71 Promedio de resultados del CPU del sistema en pruebas de comunicación entre servicios en Windows. Fuente: Elaboración propia

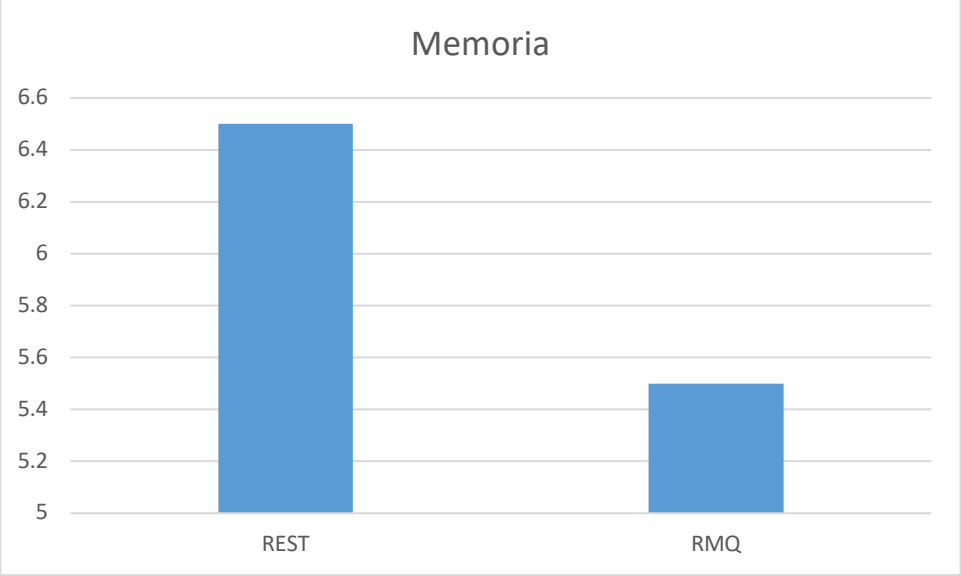

Ilustración 72 Promedio de resultados de la memoria en pruebas de comunicación entre servicios en Windows. Fuente: Elaboración propia

5.1.2.8.3.2 MacOS

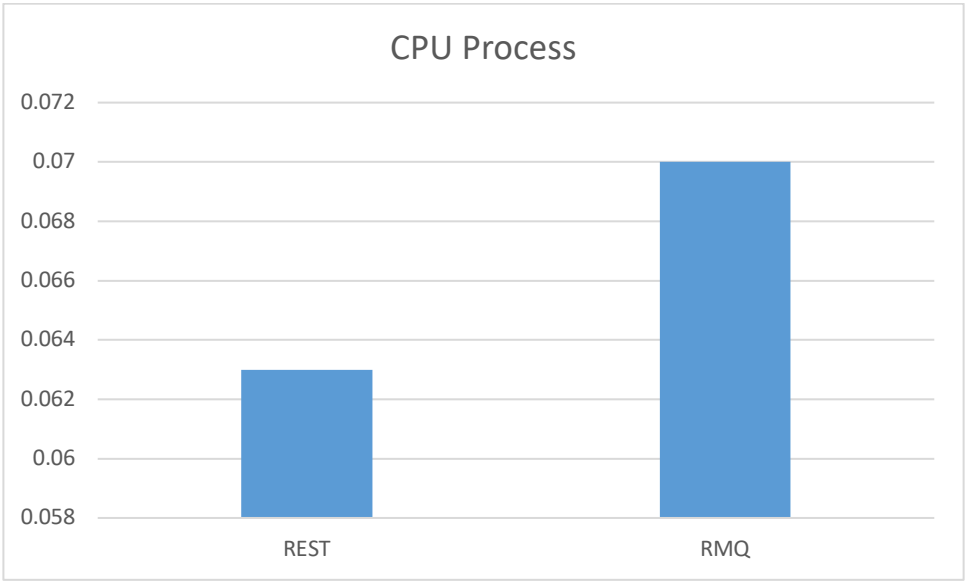

Ilustración 73 Promedio de resultados del CPU del proceso en pruebas de comunicación entre servicios en macOS. Fuente: Elaboración propia

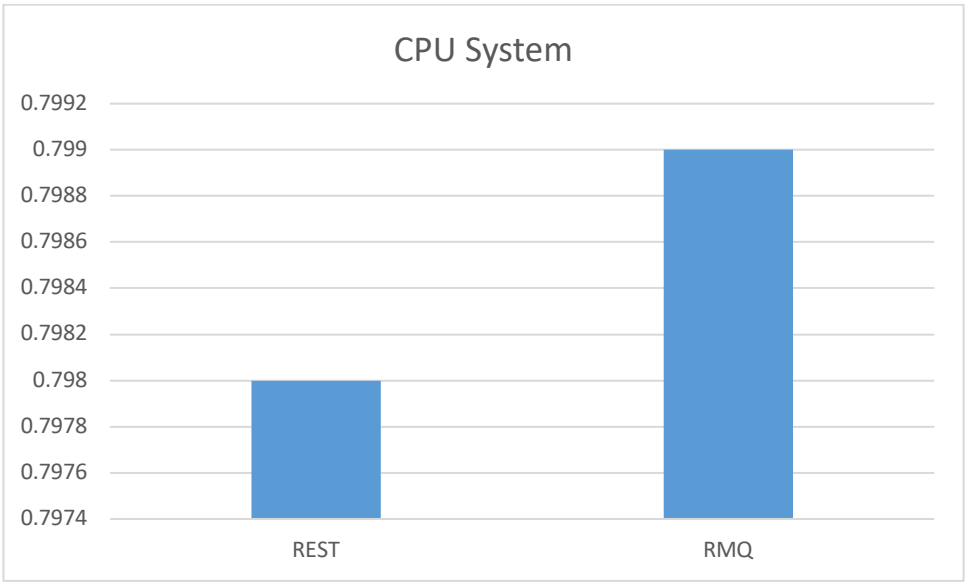

Ilustración 74 Promedio de resultados del CPU del sistema en pruebas de comunicación entre servicios en macOS. Fuente: Elaboración propia

Ilustración 75 Promedio de resultados de la memoria en pruebas de comunicación entre servicios en macOS. Fuente: Elaboración propia

5.2 Análisis de resultado de entrevista

Tal y como se indicó anteriormente en el presente documento, parte de la investigación que se efectuó, incluía realizar una entrevista a ingenieros del software, con los siguientes propósitos:

- Comprender qué tan familiarizados están los entrevistados acerca del tema de desarrollo de software verde.
- Conocer si existen prácticas de desarrollo de software sostenibles que se aplican en la actualidad entre los entrevistados.
- Conocer herramientas de medición de consumo energético que se hayan utilizado.

Las respuestas de la entrevista pueden ser encontradas en el anexo 17, sin embargo, a continuación algunas particularidades de los resultados:

- La entrevista se realizó a 20 personas que tenían más de 3 años de experiencia.
- El 90% de los entrevistados estaban de nada a poco familiarizados con el desarrollo de software verde.
- Solo un 5% de los entrevistados conocían prácticas de desarrollo de software amigables con el ambiente.
- De igual forma, solo el 5% de los entrevistados habían tenido experiencia estimando el consumo energético del software.
- El 10% conocían de herramientas disponibles para realizar mediciones del consumo energético.

De la entrevista anterior se puede aprender que en realidad el tema de desarrollo de software sostenible es un tema poco conocido, por lo que se espera, por medio de este trabajo, y similares que se pueda concientizar más a la comunidad de desarrolladores del software.

Capítulo 6. Descripción del diseño elegido

6.1 Diseño elegido

Basado en los resultados de las pruebas realizadas en el prototipo, se han seleccionado y recomendado algunas prácticas de desarrollo de software. Las decisiones se tomaron tomando en cuenta la siguiente prioridad:

1. CPU del proceso: El CPU del proceso es el indicador que demuestra el impacto del consumo en el componente de Hardware de forma individual en cada proceso, y el CPU al ser uno de los recursos que más consume energía se le da la prioridad correspondiente.
2. Tiempo de ejecución: El tiempo de ejecución puede indicar un menor tiempo en el uso de los recursos del hardware.
3. Memoria: La memoria es el tercer componente en orden de prioridad por el hecho de que en ocasiones su medición y estimación depende del *Garbage Collector* y de la liberación de memoria que se produce por medio del mismo.
4. CPU del sistema: El CPU del sistema puede incluir consumo que no depende del proceso del microservicio, sino que son más relacionados por el sistema, por lo tanto, es algo que no se puede controlar.

Las recomendaciones sugeridas se pueden ver a continuación:

- El lenguaje de programación elegido es Java, en la versión 17. La razón de la selección de esta versión, es el hecho de que, en el sistema operativo Windows, en dos de las tres pruebas realizadas demostró mejores resultados que las otras

dos versiones probadas. De hecho, a pesar de que en los promedios finales no se logra determinar la prevalencia de los resultados, en las pruebas individuales si logra demostrar que es más eficiente tanto en CPU como en memoria. En el caso del sistema operativo MacOS, los resultados fueron mayormente satisfactorios con la versión 17 de Java.

- Entre la arquitectura hexagonal y la de capas, se ha seleccionado la última, por los resultados obtenidos en las pruebas realizadas, ya que, a pesar de que la diferencia fue a penas notoria, en aplicaciones a larga escala podría verse un ahorro energético sustancial a través del tiempo.
- Con el fin de tener buenos tiempos de respuesta, se propone realizar un diseño en la base de datos en la cual las tablas puedan mantener la información necesaria y que los datos puedan ser depurados y convertidos en reportes totales, para que la cantidad de información en las tablas pueda reducirse. Adicionalmente, se recomienda todas las optimizaciones a nivel de la base de datos, como la utilización de índices, con el fin de que los resultados puedan ser obtenidos de manera más eficiente.
- También, se propone utilizar el Domain Driven Design para identificar, por medio de los dominios del negocio, la cantidad de microservicios que son requeridos para el desarrollo de la aplicación y hacer un análisis de la cantidad de instancias que cada microservicio requiere.
- Adicionalmente, se propone el uso de batches en los escenarios que lo permitan, para evitar realizar muchas consultas a la base de datos, y para mejorar el tiempo de respuesta y hacer un uso eficiente de los recursos del hardware.

- A pesar de que el uso de hilos incrementa un poco más el uso de los componentes de hardware, lo hace por menos tiempo, por lo tanto, se incentiva el uso de hilos.
- El uso del cache se recomienda en situaciones donde la información que se obtiene de la base de datos es consultada múltiples veces, o cuando la cantidad de consultas a la misma, es demasiada como para generar una sobrecarga en la base de datos.
- El uso de REST y RabbitMQ no es excluyente una de la otra, por motivo de que es posible que ambos enfoques de comunicación deban coexistir en la misma aplicación. Sin embargo, basado en los resultados se recomienda usar REST en sistemas donde el uso de la memoria no es una preocupación, por motivo de que al usar REST la memoria se incrementa. Adicionalmente, se recomienda planificar el uso de RabbitMQ en casos de uso que generen mayor ganancia energética, por ejemplo, cuando es necesario producir mensajes asíncronos.

6.2 Justificación de decisiones

Con el fin de justificar las prácticas seleccionadas que fueron propuestas en la sección anterior, se determinó que se realizaría una última prueba con el fin de evaluar la validez de las decisiones tomadas.

Para ello, se realizó una prueba final, en la cual se utilizó el prototipo desarrollado, y se crearon dos instancias del mismo, una usando las recomendaciones resultantes del análisis de los resultados, mientras que el otro contiene las pruebas opuestas seleccionadas.

El escenario de prueba que se desarrolló fue el de la simulación de creación de eventos, y simulación de reportes. Las pruebas se ejecutaron tres veces utilizando JMeter por un lapso de 5 minutos cada una.

Los resultados de las pruebas se pueden observar en el anexo 16.

6.2.1 Resumen de resultados finales

6.2.1.1 Windows

Tabla 56 Resumen de resultados de pruebas finales con recomendaciones en Windows. Fuente elaboración propia

Prueba final con recomendaciones	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0121	0.0838	7.4	4743
Ejecución 2	0.170	0.0934	7.9	4572
Ejecución 3	0.0185	0.101	7.6	4503
Promedio	**0.0158**	**0.101**	**7.6**	**4606**

Tabla 57 Resumen de resultados de pruebas finales sin recomendaciones en Windows. Fuente elaboración propia

Prueba final sin recomendaciones	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0381	0.0811	11.2	10548
Ejecución 2	0.0430	0.0959	12.6	10391
Ejecución 3	0.0425	0.101	16.3	10419
Promedio	**0.0412**	**0.0926**	**13.2**	**10452**

6.2.1.1 MacOS

Tabla 58 Resumen de resultados de pruebas finales con recomendaciones en macOS. Fuente elaboración propia

Prueba final con recomendaciones	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0185	0.142	3.3	2747
Ejecución 2	0.0189	0.141	3.7	2765
Ejecución 3	0.0244	0.146	4.2	2830
Promedio	**0.0206**	**0.143**	**3.7**	**2780**

Tabla 59 Resumen de resultados de pruebas finales sin recomendaciones en macOS. Fuente elaboración propia

Prueba final sin recomendaciones	CPU del proceso	CPU del sistema	Memoria	Tiempo
Ejecución 1	0.0560	0.139	7	6205
Ejecución 2	0.0560	0.141	7.4	6202
Ejecución 3	0.0556	0.145	6.4	6145
Promedio	**0.0558**	**0.141**	**6.9**	**6184**

6.2.2 Análisis de resultados finales

Basado en los resultados de las pruebas finales realizadas, se puede concluir lo siguiente:

- El CPU del proceso, la memoria y el tiempo de ejecución son dramáticamente menores en la instancia donde se aplicaron las recomendaciones, comparado con la instancia donde no se aplicaron.
- La única métrica que salió más alta en el caso de la instancia con las recomendaciones, es el CPU del sistema, sin embargo, la diferencia es irrelevante.

6.2.2.1 Windows

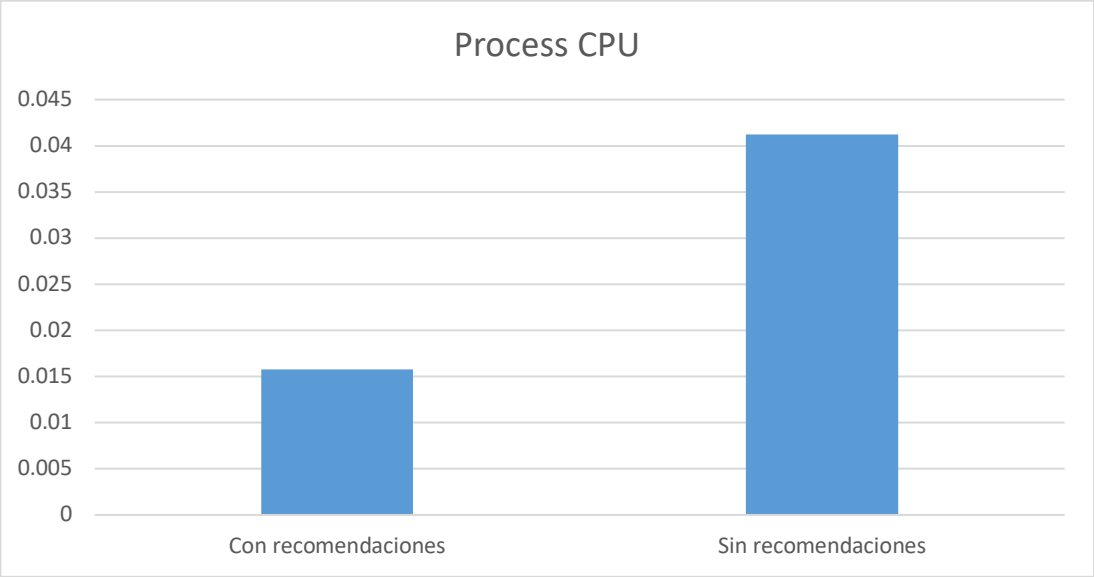

Ilustración 76 Promedio de resultados del CPU del proceso en pruebas finales en Windows. Fuente: Elaboración propia

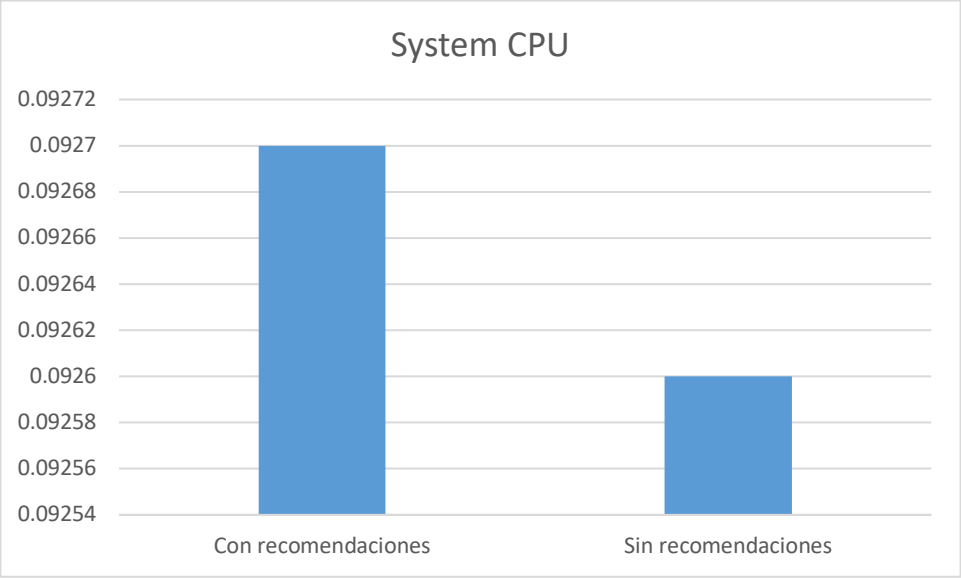

Ilustración 77 Promedio de resultados del CPU del sistema en pruebas finales en Windows. Fuente: Elaboración propia

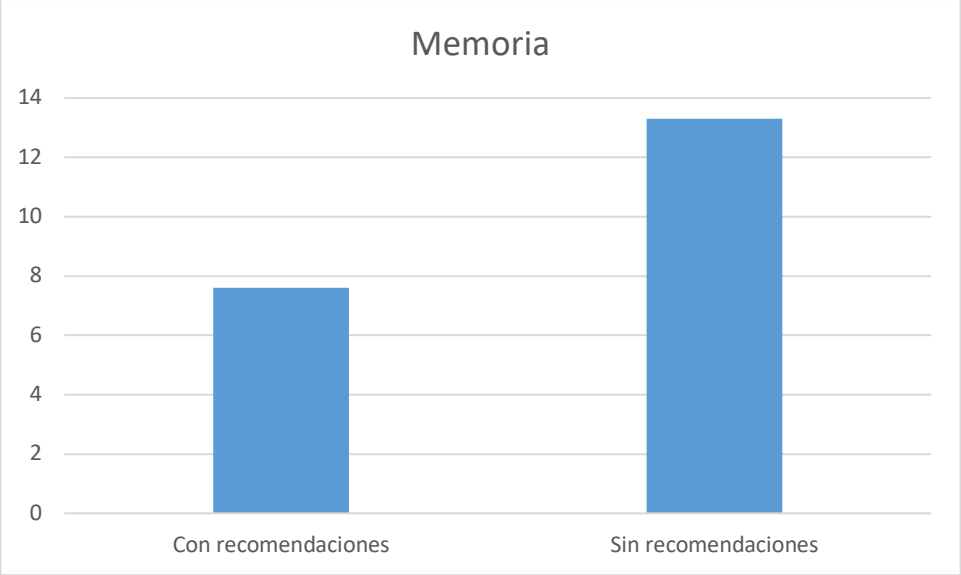

Ilustración 78 Promedio de resultados de la memoria en pruebas finales en Windows. Fuente: Elaboración propia

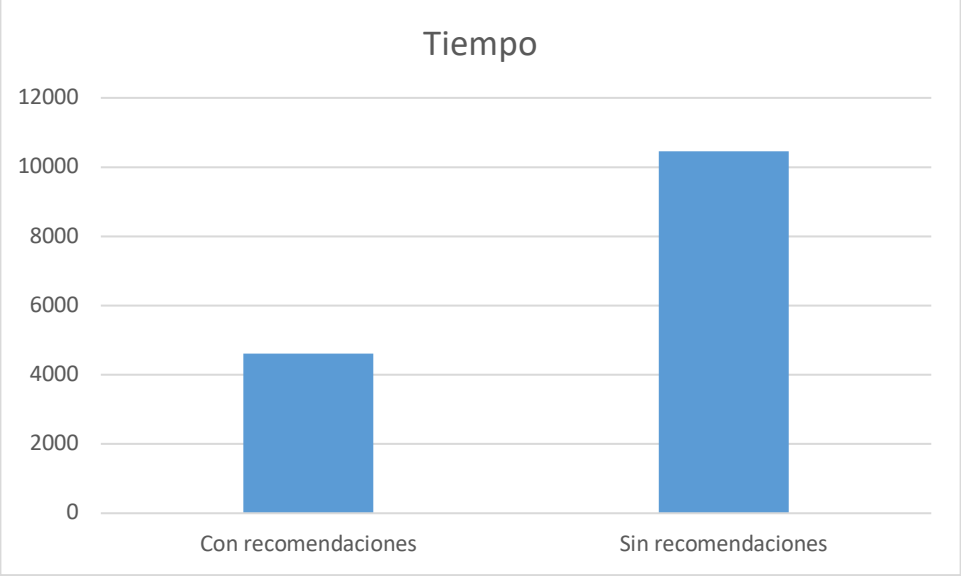

Ilustración 79 Promedio de resultados de tiempo de respuesta en pruebas finales en Windows. Fuente: Elaboración propia

6.2.2.2 MacOS

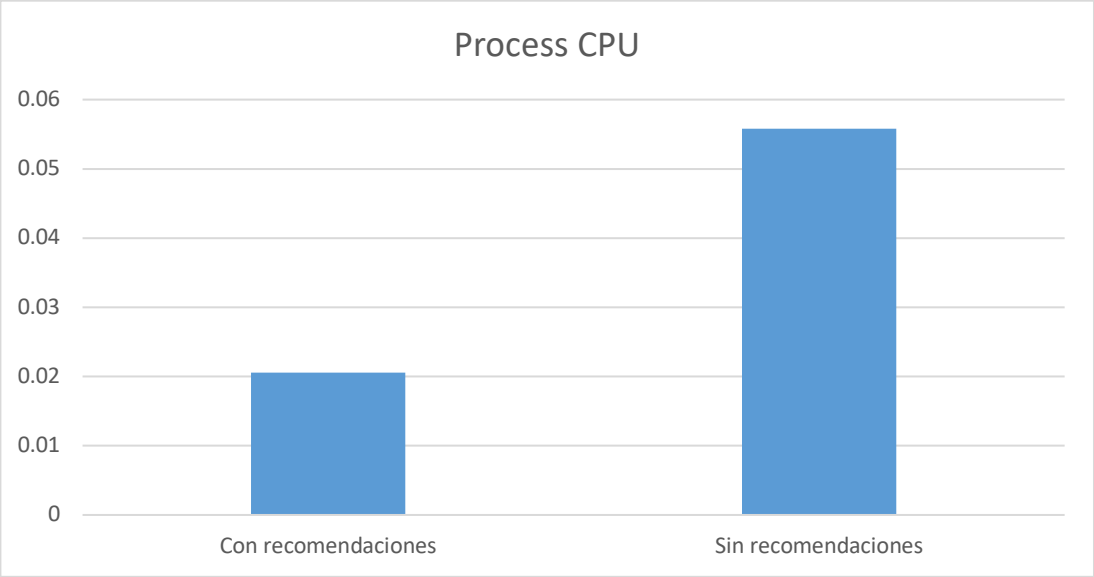

Ilustración 80 Promedio de resultados del CPU del proceso en pruebas finales en macOS. Fuente: Elaboración propia

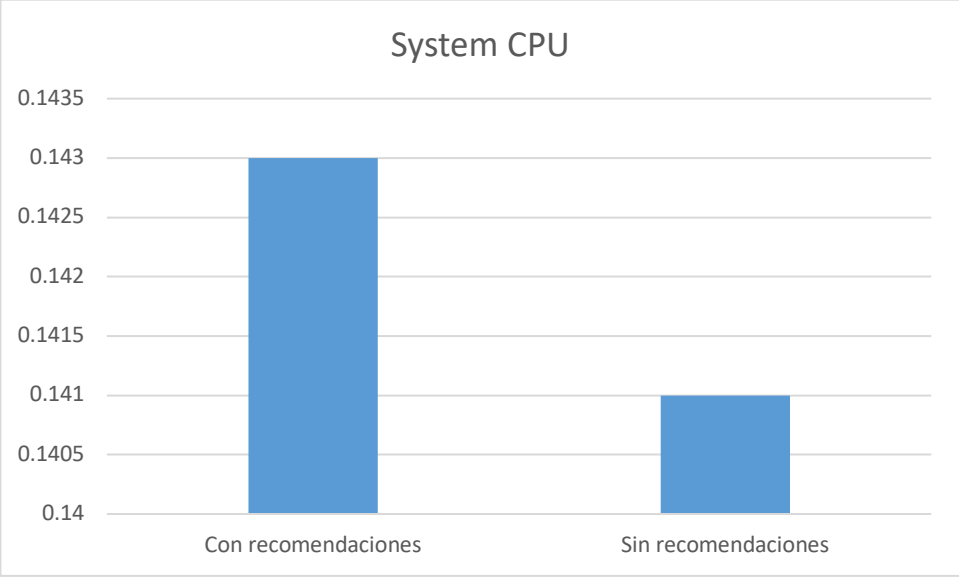

Ilustración 81 Promedio de resultados del CPU del sistema en pruebas finales en macOS. Fuente: Elaboración propia

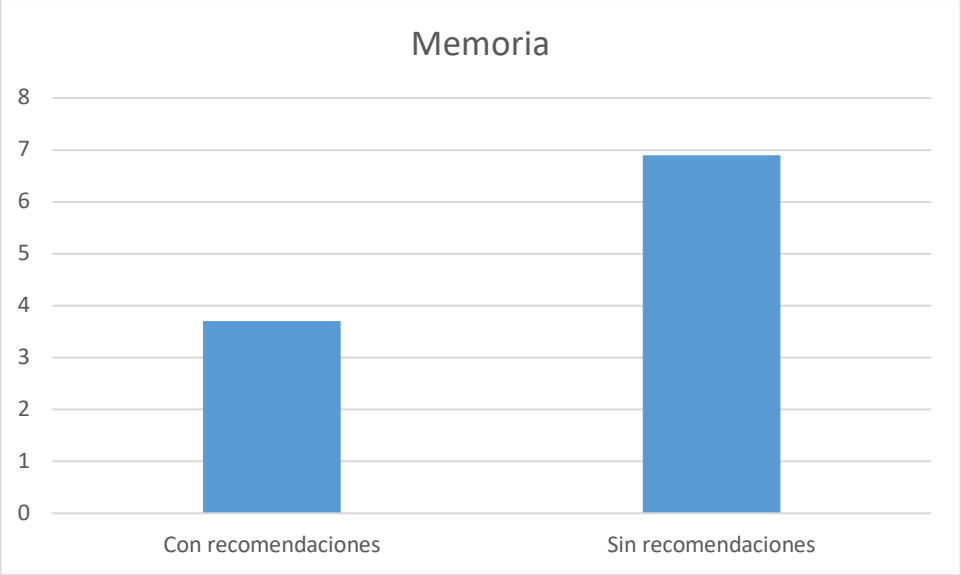

Ilustración 82 Promedio de resultados de la memoria en pruebas finales en macOS. Fuente: Elaboración propia

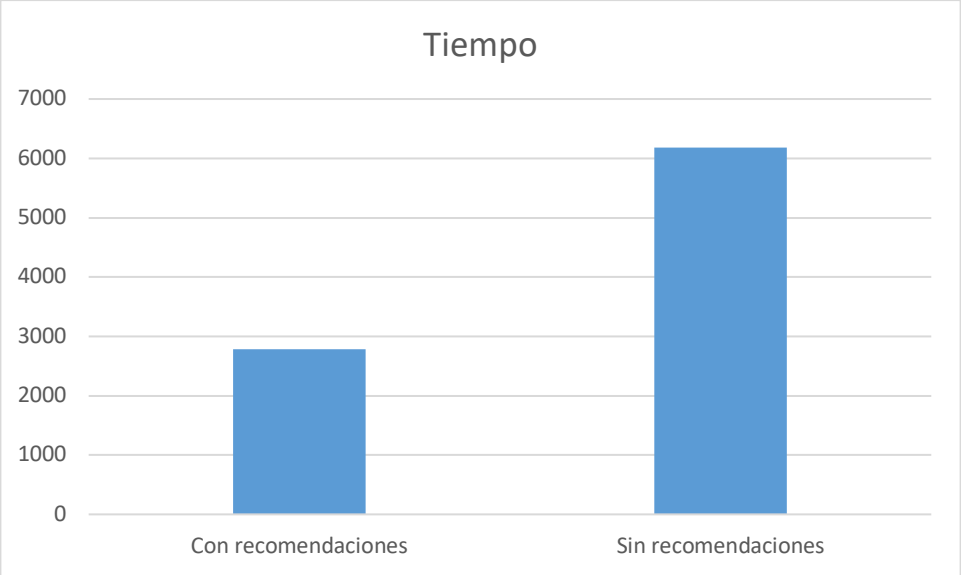

Ilustración 83 Promedio de resultados de tiempo de respuesta en pruebas finales en macOS. Fuente: Elaboración propia

Capítulo 7. Conclusiones y trabajo futuro

7.1 Conclusiones

El desarrollo del software verde es un área que requiere aún de mucha investigación, especialmente en arquitecturas modernas, y en prácticas de desarrollo que son ampliamente aceptadas y utilizadas en la mayoría de las aplicaciones de la actualidad.

Además, basado en los resultados de la presente investigación, se evidencia que aún es necesario concientizar a la comunidad de desarrolladores de software, ya que la mayoría desconoce del tema y de la problemática.

7.1.1 Conclusiones del objetivo 1

Asimismo, la medición de los componentes del hardware es muy importante, ya que, permite monitorear e identificar mejoras que puedan ser implementadas a nivel del código, para reducir el impacto energético y al mismo tiempo disminuir costos.

También, se considera que aún es necesario trabajar en definir herramientas de medición energética para los componentes del hardware, especialmente para que estas puedan demostrar el consumo energético en Joules o Watts, y que puedan ser implementados en el código que se desarrolla.

Lo anterior es debido a que existe carencia de herramientas que sean útiles en distintos fabricantes de componentes de hardware e incluso en diversos sistemas operativos.

7.1.2 Conclusiones del objetivo 2

La investigación demostró resultados satisfactorios ante la comparación de diversas prácticas del desarrollo de software, las cuales estaban enfocadas en el diseño de la arquitectura de microservicios, tomando como base el prototipo de la aplicación hipotética propuesta y en el rendimiento de la aplicación.

Entre las prácticas que demostraron buenos resultados se pueden mencionar: el uso de batches para transacciones que podrían generar uso intensivo de la base de datos, el uso de concurrencia en las aplicaciones para tener un mejor tiempo de respuesta, el uso moderado de contenedores para la utilización eficiente de recursos, el uso de cache para tener fácil acceso a datos recurrentes, entre otros.

Por otro lado, diseños como arquitectura hexagonal y prácticas que perjudiquen el rendimiento de la aplicación, tienden a tener un mayor impacto en el porcentaje de uso de los recursos de hardware.

Finalmente, a pesar de que existen prácticas que no arrojaron resultados positivos, no necesariamente se sugiere que se descarten, sino que se deben buscar alternativas que permitan compensar el costo generado.

7.2 Trabajo Futuro

A parte de las recomendaciones resultantes de la presente investigación y de las pruebas realizadas demostradas en el capítulo cinco, se considera necesario ahondar más en otras prácticas de desarrollo de software, para que de esta forma exista más

documentación, sobre las prácticas que más impacto negativo generan en el consumo energético de los componentes de hardware.

También, se sugiere aplicar la metodología de pruebas demostrada en esta investigación, en otros tipos de aplicaciones para recibir retroalimentación de aplicaciones que sean diferentes al modelo del prototipo que se diseñó.

Del mismo modo, se sugiere validar las recomendaciones resultantes de esta investigación, en distintos dispositivos de hardware, con sistemas operativos adicionales y parámetros de medición diferentes.

Además, se considera importante ampliar las investigaciones del desarrollo de software verde en las demás etapas del desarrollo de software, como por ejemplo, en la etapa de diseño, mantenimiento, pruebas entre otros.

También, con el fin de incentivar a las organizaciones adoptar medidas sostenibles en el desarrollo de software se recomienda para trabajos futuros, realizar investigaciones del impacto económico de la aplicaciones de las recomendaciones de esta investigación así como de otras sugerencias que se podrían agregar.

Finalmente, queda para trabajos futuros la evaluación y aplicación de prácticas de desarrollo de software, en distintas aplicaciones de software, como por ejemplo: desarrollo de software móvil, desarrollo de software en la nube, desarrollo de software web, inteligencia artificial, blockchain, entre otros.

Glosario

Cache: El cache es un mecanismo de almacenamiento temporal de datos que se utiliza para tener datos en una ubicación cercana y de rápido acceso con el fin de mejorar la velocidad de acceso de los datos y con esto reducir el tiempo de respuesta.

Concurrencia: Se refiere a la capacidad de un sistema de ejecutar múltiples tareas o procesos de forma simultánea.

Contenedor: Los contenedores son una tecnología que permite que las aplicaciones puedan ser empaquetadas y ejecutadas de una forma sencilla e independiente.

CPU: Se refiere a la Unidad Central de Procesamiento. En un sistema informático, es el componente de hardware que recibe las instrucciones y las ejecuta.

Garbage Collector: Es un componente de los lenguajes de programación que se encarga de liberar de forma automática la memoria asignada a objetos no utilizados de un programa.

Hardware: Se refiere a los componentes físicos de un sistema. Por lo general, estos componentes requieren de energía para funcionar. Por ejemplo, CPU, memoria, disco duro, entre otros.

JVM: La máquina virtual de Java es el entorno donde se ejecutan las instrucciones generadas por un compilador, con el fin de que puedan ser compatibles en diferentes plataformas sin tener que ser recompiladas.

Memoria RAM: La memoria RAM, es la memoria de acceso aleatorio, que es utilizada en los sistemas como una forma de almacenamiento volátil, necesario para la ejecución de programas en un sistema.

REST: Es una interfaz utilizada para conectar diferentes sistemas por medio de comunicaciones a través de la red utilizando el protocolo HTTP. Se basa en operaciones que facilitan la comunicación entre distintos servicios o componentes. Algunas de las operaciones son crear (POST), consultar (GET), editar (PUT) y eliminar (DELETE).

Software: Conjunto de instrucciones que se ejecutan en un sistema en forma de programas y aplicaciones.

Sostenibilidad: Es la capacidad de mantener un equilibrio ambiental al momento de satisfacer las necesidades poblacionales actuales sin comprometer las necesidades de las futuras generaciones.

Bibliografía

Al-Ghussain, L. (2019). Global warming: Review on driving forces and mitigation. *Environmental Progress & Sustainable Energy*, 13-21.

Amazon Web Services, Inc. (s.f.). *¿Qué es el SDLC?* Obtenido de https://aws.amazon.com/es/what-is/sdlc/

Amazon Web Services, Inc. (s.f.). *What is an Event-Driven Architecture?* Obtenido de https://aws.amazon.com/event-driven-architecture/

Amazon Web Services, Inc. (s.f.). *What Is Batch Processing?* Obtenido de https://aws.amazon.com/what-is/batch-processing/

Belkhir, L., & Elmeligi, A. (2018). Assessing ICT global emissions footprint: Trends to 2040 & recommendations. *Journal of cleaner production*, 448-463.

Calero, C., Mancebo, J., García, F., Moraga, M., Berná, J., Fernández-Alemán, J., & Toval, A. (2019). 5Ws of green and sustainable software. *Tsinghua Science and Technology*, 401-414.

Cockburn, A. (s.f.). *Hexagonal architecture* . Obtenido de Alistair Cockburn: https://alistair.cockburn.us/hexagonal-architecture/

Frick, T. (2016). *Designing for sustainability: a guide to building greener digital products and services.* O'Reilly Media, Inc.

García- Martin, E., Rodrigues, C. F., Riley, G., & Grahn, H. (2019). Estimation of energy consuption in machine learning. *Journal of Parallel and Distributed Computing*, 75-88.

Georgiou, S., Rizou, S., & Spinellis, D. (2019). Software development lifecycle for energy efficiency: techniques and tools. *ACM Computing Surveys*, 1-33.

Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Holmes, D., & Lea, D. (2006). *Java Concurrency in practice.* Pearson Education.

Google Cloud. (s.f.). *What is a relational database?* . Obtenido de https://cloud.google.com/learn/what-is-a-relational-database

Green Software Foundation. (s.f.). *Manifesto*. Obtenido de Green Software Foundation: https://greensoftware.foundation/manifesto

Hort, M., Kechagia, M., Sarro, F., & Harman, M. (2021). A survey of performance optimization for mobile applications. *IEEE Transactions on Software Engineering*, 2879-2904.

Hussain, A. (s.f.). *Microservices*. Obtenido de Princples.Green: https://principles.green/principles/applied/microservices/

Katal, A., Dahiya, S., & Choudhury, T. (2022). Energy efficiency in cloud computing data centers: a survey on software technologies. *Cluster Computing*, 1-31.

Kweku, D., Bismark, O., Maxwell, A., Desmond, K., Danso, K., Oti-Mensah, E., & Adormaa, B. (2018). Greenhouse effect: greenhouse gases and their impact on global warming. *Journal of Scientific research and reports*, 1-9.

Lago, P., Gu, Q., & Bozzelli, P. (2014). *A systematic literature review of green software metrics.* VU Technical Report.

Martin, R. C. (2018). *Clean Architecture: A craftsman's guide to software structure and design.* Boston: Pearson Education, Inc.

Mehra, R., Sharma, V. S., Kaulgud, V., Podder, S., & Burden, A. P. (2022). Towards a green quotient for software projects. *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, (págs. 295-296).

Mourão, B., Karita, L., & do Carmo Machado, I. (2018). Green and sustainable software engineering - a systematic mapping study. *In proceedings of the 17th Brazilian Symposium on Software Quality.*, 121-130.

Ometov, A., Bardinova, Y., Afanasyeva, A., Masek, P., Zhidanov, K., Vanurin, S., & Bezzateev, S. (2020). An overview on blockchain for smartphones: State-of-the-art, consensus, implementation, challenges and future trends. *IEEE Access*.

Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2021). Ranking programming languages by energy efficiency. *Science of Computer Programming*.

Richards, M. (2015). *Software architecture patterns (Vol. 4, p. 1005).* 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Incorporated.

Sicari, S., Rizzardi, A., & Coen-Porisini, A. (2022). Security&privacy issues and challenges in NoSQL databases. *Computer Networks*.

Smith, D. (08 de September de 2021). *The art of long-term support and what LTS means for the Java ecosystem* . Obtenido de Java Magazine: https://blogs.oracle.com/javamagazine/post/java-long-term-support-lts

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv*.

Sukhija, N., & Bautista, E. (2019). Towards a framework for monitoring and analyzing hight performance computing environments using kubernetes and prometheus. *IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City innovation*, 257-262.

The Apache Software Foundation. (s.f.). *Apache JMeter*. Obtenido de https://jmeter.apache.org/index.html

United Nations Climate Change. (s.f.). *What is the United Nations Framework Convention on Climate Change?* . Obtenido de United Nations Climate Change: https://unfccc.int/process-and-meetings/what-is-the-united-nations-framework-convention-on-climate-change

University of Cambridge Judge Business School. (2023). *Cambridge Bitcoin Electricity Consumption index*. Obtenido de The Cambridge Centre for Alternative Finance : https://ccaf.io/cbeci/index

Vural, H., & Koyuncu, M. (2021). Does domain-driven design lead to finding the optimal modularity of a microservice? *IEEE Access*, 32721-32733.

Wedyan, F., & Abufakher, S. (2020). Impact of design patterns on software quality: a systematic literature review. *IET Software*, 1-17.

Zhelev, S., & Rozeva, A. (2019). Using microservices and event driven architecture for big data stream processing. *In AIP Conference Proceedings.* AIP Publishing LLC.

Zhong, Y., Li, W., & Wang, J. (2019). Using event sourcing and CQRS to build a high performance point trading system. *In Proceedings of the 2019 5th International Conference on E-Business and Applications.*, 16-19.

Anexos

Anexo 1: Encuesta de sistemas de venta de entradas en línea

1.1 Detalles de la encuesta:

1. Encuestados 85 personas
2. Siete días permaneció la encuesta activa.

1.2 Preguntas de la encuesta:

1. ¿Qué tan frecuente realiza compras de tiquetes/entradas en línea?
    a. Muy frecuente (Al menos una vez por semana)
    b. Frecuente (Una vez al mes)
    c. Poco Frecuente (Una vez cada 6 meses)
    d. Casi nunca (Una vez al año)
    e. Otro
2. ¿Cómo ha sido su experiencia realizando compras de tiquetes en línea?
    a. Excelente (Nunca he tenido problemas)
    b. Buena (Solo una vez he tenido problema)
    c. Regular (He tenido problemas de 2 a 5 veces)
    d. Malo (He tenido problemas más de 5 veces)
    e. Otro
3. ¿Ha tenido un familiar o conocido cercano que ha tenido problemas comprando tiquetes/entradas en línea?
    a. Si
    b. No

c. Tal vez

4. Describa brevemente el problema que ha tenido o escuchado.
5. Seleccione los aspectos más importantes para la compra de tiquetes en línea
    a. Tiempos de respuesta óptimos
    b. Seguridad de datos de tarjeta
    c. Que sea confiable y siempre esté disponible
    d. Buen soporte técnico
    e. Servicio post-venta (cambios-devoluciones)

La intención de las preguntas anteriores era poder identificar las siguientes características:

1. La frecuencia con la que las personas compran entradas en línea, esto con el fin de identificar cuántas personas actualmente conocen el mecanismo de compra en línea.
2. La percepción de las personas en cuanto al servicio que ofrecen las plataformas actuales de ventas de entradas en línea. Para ello se le permitió al encuestado expresar los problemas que ha tenido y del mismo modo contabilizar la frecuencia de esos problemas.
3. Los elementos más importantes desde la perspectiva del cliente a la hora de realizar compras en línea, esto con fin de tomar esa retroalimentación en cuenta a la hora de planificar los requerimientos de software.

1.3 Respuestas de la encuesta

Respuesta 1:

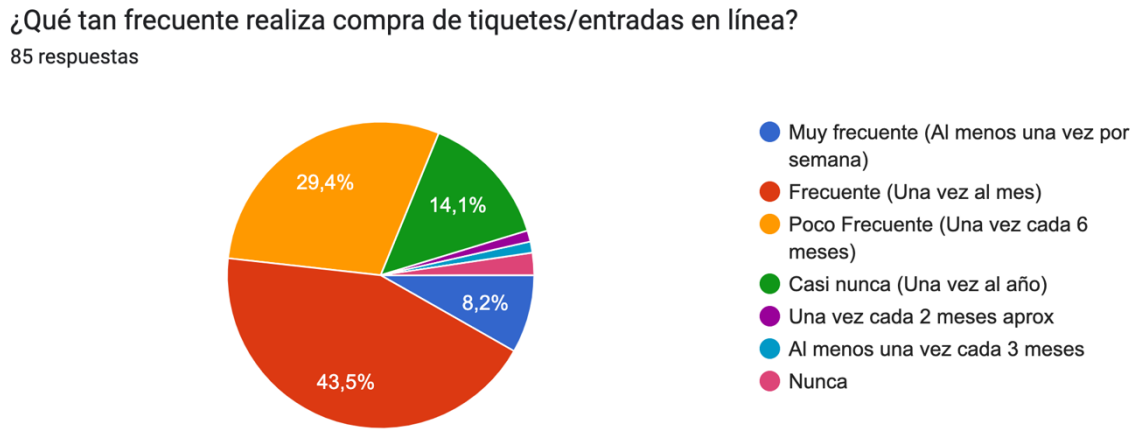

Ilustración 84 Respuesta 1 de encuesta de satisfacción de sistemas de compras en línea. Fuente: Elaboración propia

Respuesta 2:

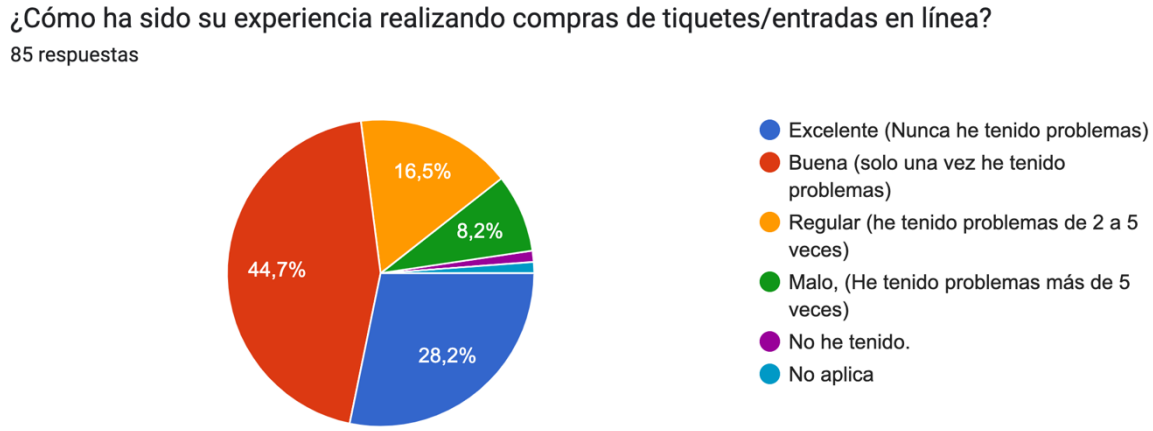

Ilustración 85 Respuesta 2 de encuesta de satisfacción de sistemas de compras en línea. Fuente: Elaboración propia

Respuesta 3:

Ilustración 86 Respuesta 3 de encuesta de satisfacción de sistemas de compras en línea. Fuente: Elaboración propia

Respuesta 4:

Después de analizar los resultados de la encuesta se logran identificar los siguientes problemas más frecuentes con los que se encuentran los clientes:

1. Tiempos de respuesta altos
2. Errores en el sistema
3. Fallos en el procesamiento de la compra
4. Cobros indebidos
5. No hay confirmación de compra (emisión de tiquetes)
6. Duplicados de tiquetes y sobreventa

7. Falta de comunicación
8. Complejidad

Respuesta 5:

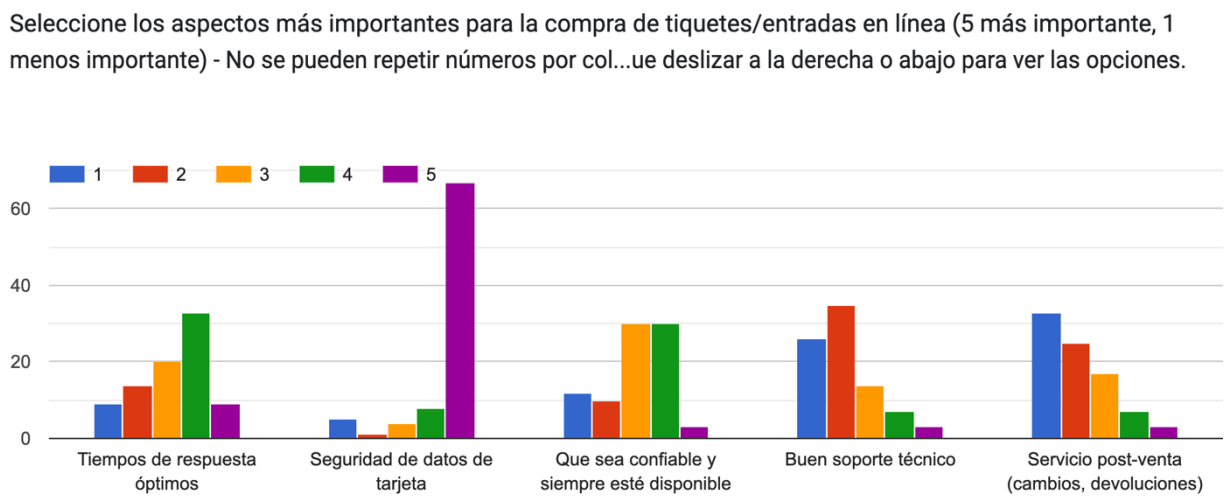

Ilustración 87 Respuesta 5 de encuesta de satisfacción de sistemas de compras en línea. Fuente: Elaboración propia

1.4 Conclusiones de las respuestas de la encuesta

Después de analizar las respuestas de la encuesta se puede concluir los siguiente:

- Más del 50% de los encuestados tienen hábitos de compra de tiquetes en línea muy frecuente o frecuente, lo cual quiere decir que compran al menos una vez cada mes.
- Casi el 70% de los encuestados han manifestado haber tenido por lo menos un problema a la hora de comprar entradas en línea.

- Casi el 70% de los encuestados ha manifestado conocer a alguien cercano que también ha tenido algún problema realizando compras en línea.
- Los problemas más comunes son problemas referentes a tiempos de espera y fallos en el sistema en distintas operaciones.
- Las características más importantes que debe tener el sistema según las personas encuestadas es la seguridad de los datos de pagos, tiempos de respuesta óptimos y que el sistema sea confiable.

Anexo 2: Requerimientos simplificados de la aplicación hipotética

A pesar de que la aplicación hipotética no será desarrollada como parte de la presente investigación, se han recabado los siguientes requerimientos, con el propósito de ejemplificar las características que se pensaron durante el desarrollo del prototipo:

Requerimientos de usuario

- UR-01: Un usuario administrador de un evento debe ser capaz de vender entradas.
- UR-02: Un usuario administrador de un evento debe ser capaz de administrar los eventos que tiene planeado realizar en un futuro.
- UR-03: Un usuario administrador de un evento debe ser capaz de recibir reportes del estado de las ventas de un evento que tenga planeado realizar en el futuro.
- UR-04: Un usuario cliente debe ser capaz de consultar eventos de interés.
- UR-05: Un usuario cliente debe ser capaz de obtener beneficios a la hora de conseguir entradas.
- UR-06: Un usuario cliente debe ser capaz de comprar entradas sin problema alguno.
- UR-07: Un usuario cliente debe ser capaz de administrar su entrada una vez adquirida.

Requerimientos funcionales

- FR-01: Como usuario de cualquier tipo se debe proveer la capacidad de registrarse en línea para posteriormente autenticarse.
- FR-02: Como usuario de cualquier tipo se debe ser capaz de manejar un perfil en el cual pueda agregar información personal y también información financiera.
- FR-03: Como usuario registrado se debe tener la capacidad de pagar el plan de empresa para ingresar información de eventos.
- FR-04: Como usuario administrador se debe tener la capacidad de administrar eventos.
- FR-05: Como usuario administrador se debe tener la capacidad de administrar detalles de los eventos, como cantidad de personas, precios, restricciones de compra, etc.
- FR-06: Como usuario cliente o cliente potencial se debe tener la capacidad de consultar los distintos tipos de eventos que hay disponibles.
- FR-07: Como usuario cliente se debe tener la posibilidad de una preventa después de haber realizado un pago que le permita tener ese beneficio.
- FR-08: Como usuario administrador se debe ser capaz de dar seguimiento a la cantidad de entradas vendidas y poder visualizar un reporte de los eventos.
- FR-09: Como usuario cliente se debe poder realizar una compra de entradas y recibir el boleto emitido por correo electrónico.
- FR-10: Como usuario cliente se debe tener la capacidad de revender la entrada en el sistema una vez adquirida.

Requerimientos no funcionales

- NFR-01: El sistema debe tener la característica de implementar mecanismos de seguridad tanto para el inicio de sesión como para los pagos.
- NFR-02: La aplicación debe tener estar disponible siempre, por lo que se deben implementar mecanismos de fallback.
- NFR-03: Se deben implementar mecanismos de seguridad que permitan auditoría de las transacciones que se realizan.
- NFR-04: El sistema debe hacer una utilización eficiente de recursos del hardware.

Requerimientos de interfaz de usuario

- RIU-01: La interfaz de usuario debe ser diseñada con características que permita la accesibilidad, específicamente con personas que tengan algún tipo degradación visual.
- RIU-02: Las interfaces de usuario deben cumplir con criterios de usabilidad definidos.
- RIU-03: Las interfaces de usuario deben ser diseñadas con la finalidad de generar una buena experiencia de usuario en las personas.

Anexo 3: Justificación de mediciones del consumo energético

La atención en el consumo energético en la presente investigación, se derivó a que cualquier interacción que se tiene con una aplicación de software, repercute en el hardware que lo soporta y lo ejecuta. Por lo tanto, es importante entender cuáles son esas repercusiones.

De hecho, la manera de alcanzar sostenibilidad del software es principalmente mejorando su consumo energético. (Calero, et al., 2019). Por lo que el enfoque será identificar un modo en el que el software puede hacer uso responsable y óptimo del hardware que lo ejecuta, ya que, existe evidencia sólida de que el diseño de software puede alterar significativamente el consumo energético de los productos de tecnología de información. (Georgiou, Rizou, & Spinellis, 2019)

Las aplicaciones desarrolladas para dispositivos móviles son un ejemplo de porqué se debe prestar atención al consumo energético, ya que son totalmente dependientes de un dispositivo de hardware que tiene carga energética muy limitada. Por ello, necesitan ser muy eficientes en el uso de los recursos del dispositivo para garantizar un funcionamiento óptimo.

Según se investigó, para asegurar el éxito de una aplicación móvil los desarrolladores intentan maximizar la calidad de experiencia y satisfacción de usuario por medio del cumplimiento de requerimientos funcionales y no funcionales, siendo consumo de energía uno de los más importantes, puesto que los usuarios se sienten muy insatisfechos cuando alguna aplicación drena la batería del dispositivo en minutos. (Hort, Kechagia, Sarro, & Harman, 2021)

Incluso, la adopción de la energía como una métrica en el área de *Machine Learning* es necesario para un futuro sostenible y escalable. (García- Martin, Rodrigues, Riley, & Grahn, 2019)

Con la información anterior se pudo determinar la importancia de obtener una medición precisa del uso de los recursos del hardware sobre el cual se ejecuta cierto software y se logra justificar por qué es de suma importancia monitorear el estado del hardware ante la aplicación de distintas prácticas de desarrollo.

Anexo 4: Descripción y justificación de selección de componentes de hardware

Existen diversos recursos de hardware que se podrían monitorear durante la ejecución de una aplicación, sin embargo, en la presente investigación se decidió hacer un enfoque mayor en dos: CPU y memoria RAM.

La razón por la que se decidió utilizar esos dos está respaldada por una investigación en la cual se determinó que

CPU (Central Processing Unit)

El CPU es el recurso responsable de la interpretación y ejecución de las instrucciones del programa, por lo que tiene una función trascendental a la hora de desarrollar el software, ya que ejecuta las instrucciones que se escribieron como código en la aplicación.

Debido a lo anterior, se decidió monitorear el CPU para contrastar las distintas formas o prácticas de escribir instrucciones en el código.

Memoria

La memoria, específicamente la memoria RAM es usada para almacenar y consultar información que no es permanente, pero que es necesaria para mantener el estado de la aplicación.

Los recursos anteriores fueron seleccionados basados en una investigación en la que se determinó que a lo largo de los años esos eran los componentes más medidos por su injerencia e importancia en el funcionamiento del código y de hecho son los

componentes que más atención se le han dado en los años recientes a dicha investigación (Lago, Gu, & Bozzelli, 2014).

Anexo 5: Descripción y justificación de herramientas de medición seleccionadas y su justificación

Las herramientas de medición seleccionadas en esta investigación ayudaron a identificar diferencias en el porcentaje de uso del CPU y de la memoria RAM, ante la aplicación de las distintas prácticas de desarrollo de software, en el prototipo diseñado.

Después de evaluar las distintas opciones de instrumentación que existen, se tomó como base un trabajo en el cual se propone un framework para monitorear y manejar de manera proactiva operaciones de data centers (Sukhija & Bautista, 2019).

De tal forma que para esta investigación se decidió utilizar Prometheus, Grafana, JMeter y Spring Boot actuator para la realización de las pruebas pertinentes.

JMeter

De acuerdo a la página principal de Apache JMeter, esta es una aplicación open source diseñada en Java para realizar pruebas de carga enfocada en comportamiento funcional y medición de rendimiento. Jmeter permite realizar pruebas de rendimiento en recursos dinámicos y estáticos y puede ser utilizado para simular una carga pesada en algún recurso.

(The Apache Software Foundation, s.f.)

El propósito del uso de esta herramienta en el contexto de este proyecto, fue tener la capacidad de evaluar diferentes escenarios de prueba, que permitieran simular diferentes tipos de carga sobre un recurso ante distintos casos de uso.

Grafana

El uso de esta herramienta en la presente investigación permitió consultar, visualizar y recibir alertas sobre la información que generó prometheus. Dicha información se logró mostrar haciendo uso de gráficos que facilitaron la interpretación de los datos recibidos.

Prometheus

Prometheus es una herramienta, que recolecta y almacena métricas de una aplicación en específico durante períodos de tiempo para su posterior análisis.

Cuando se habla de métricas se refiere a mediciones numéricas que, en el caso del prototipo de la aplicación implementada, se relacionaron con el comportamiento de los distintos componentes del hardware mencionados con anterioridad ante la aplicación de prácticas de desarrollo de software.

Implementación

Una vez implementado el prototipo, se agregó al microservicio, una librería de Spring Boot llamada Spring Boot Actuator. Esta librería permitió que se encontraran ciertas funcionalidades adicionales para monitorear la aplicación.

Posteriormente, se importó en el microservicio una librería de Prometheus para expandir los endpoints de monitoreo que se habilitaron con Spring Boot Actuator.

Finalmente se conectó grafana con Prometheus y se diseñó un dashboard para mostrar los componentes de hardware que se decidieron monitorear

Anexo 6: Descripción del prototipo realizado

Como ya se ha dicho, el presente trabajo se ha realizado con un caso hipotético en mente, el cual es un sistema de compra de tiquetes en línea. La decisión de esa aplicación es derivada al estudio de percepción que se realizó de los sistemas de compra de tiquetes en línea que existen en la actualidad (Apéndice 1) y en las posibles mejoras que se podrían realizar en una aplicación similar.

Debido a lo anterior, el prototipo que se utilizó en las pruebas, es un microservicio que se llama administrador de actividades (activity management), el cual es el encargado de todo lo relacionado con el manejo de las eventos de cualquier tipo que se podrían efectuar en el país.

Características principales

Las principales características implementadas en el microservicio descrito se encuentran:

- Administración de eventos o actividades, lo cual incluye, creación y modificación de eventos.
- Administración de facilidades o localidades en donde se desarrollará el evento. De igual forma esto incluye creación y modificación.
- Administración de sectores que se habilitarán en las facilidades creadas con anterioridad, cada sector tiene una capacidad específica, lo cual se utiliza a la hora de la creación de los tiquetes.

- Habilitación de actividades, lo cual incluye la creación de los tiquetes.
- Compra de tiquetes creados para una actividad en específico.
- Reportes de actividades, facilidades, sectores y tiquetes.

Entidades

Las entidades que se diseñaron son las siguientes:

- **Activity:** Tiene que ver con todo lo relacionado con los datos de la actividad, por ejemplo: nombre, lugar, fecha y hora.
- **Facility:** Se refiere al lugar en donde se realizará el evento, por ejemplo: Estadio Nacional.
- **Sector:** Cada facilidad, como por ejemplo un estadio, tiene diferentes sectores o localidades tales como: platea norte, palco preferencial, etc, por lo que esta entidad hace alusión a esos sectores.
- **ActivitySeating:** Son los tiquetes que se crean y ponen a la venta una vez que se habilitan las actividades.

Base de datos

Se ha decidido utilizar la base de datos MySQL para el almacenamiento de datos. El diseño de las tablas se puede encontrar en la siguiente ilustración:

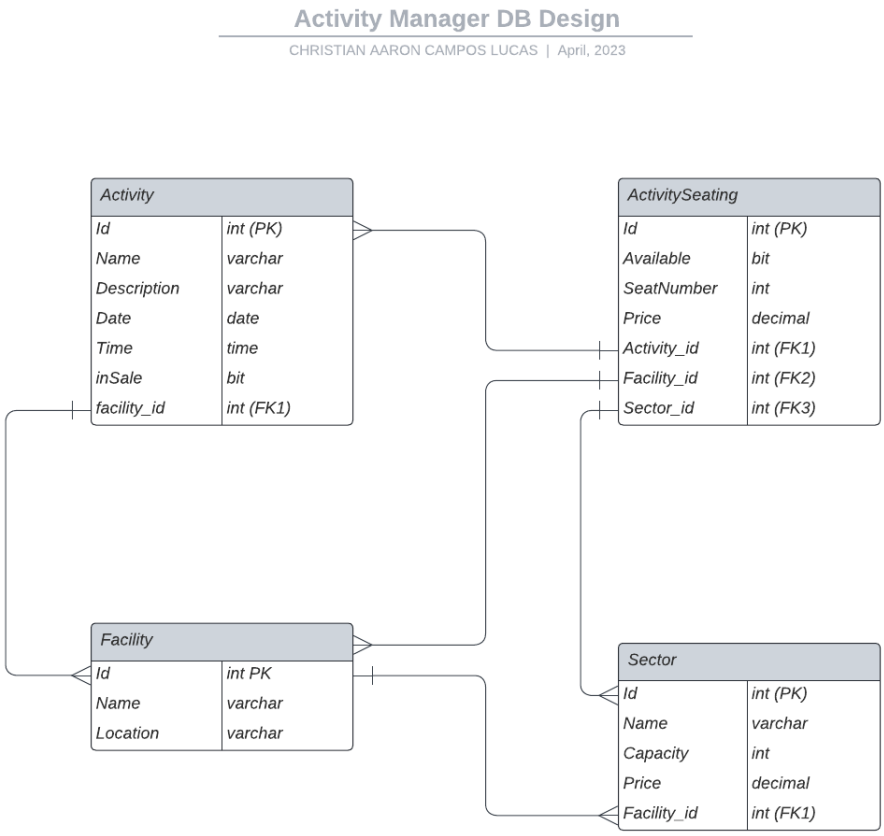

Ilustración 88 Diseño de base de datos para el microservicio prototipo llamado Activity Manager. Fuente: Elaboración propia

Anexo 7: Descripción y justificación de prácticas de desarrollo de software seleccionadas

Lenguajes de programación:

La escogencia del lenguaje de programación es de vital importancia a la hora de poder desarrollar aplicaciones verdes. Por ejemplo, seleccionar Java sobre Python para desarrollar aplicaciones puede reducir en un 97.4% el consumo energético. (Mehra, Sharma, Kaulgud, Podder, & Burden, 2022)

Basado en el trabajo de (Pereira, et al., 2021) podemos identificar de la siguiente tabla los lenguajes de programación que podrían tomarse en consideración a la hora de desarrollar el prototipo para la aplicación propuesta en el presente trabajo:

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Ilustración 89 Resumen de resultados de lenguajes de programación más eficientes energéticamente. Fuente: Pereira, et al., 2021

En el trabajo mencionado se realizaron comparaciones de lenguajes de programación tanto para medir la energía, el tiempo y la memoria, y basado en eso se puede identificar el lenguaje de programación más verde.

Dada la información obtenida de los resultados de la investigación de (Pereira, et al., 2021) para efectos de esta investigación se ha seleccionado el lenguaje de programación Java. La razón es básicamente el hecho de que, en el consumo energético y en el tiempo, el resultado fue muy favorable e incluyó el lenguaje Java entre los cinco mejores lenguajes en esas categorías.

Adicionalmente a eso, la escogencia del lenguaje de programación Java va respaldada por el uso mundial que se le ha dado a Java desde hace más de 25 años y la forma en la que se ha diseminado a lo largo de los años gracias a su robustez y adaptabilidad, haciéndolo uno de los lenguajes más populares.

Java tiene distintas versiones desde que se creó hace más de 25 años. Sin embargo, recientemente, en el año 2018, se anunció un nuevo modelo de despliegue a producción de nuevas versiones de Java. Antes del 2018, Java sacaba a producción nuevas versiones, por lo general, entre cada tres o cuatro años, sin embargo, a partir de la fecha mencionada se diseñó un modelo en el que cada seis meses se mandaba a producción una versión nueva, y posteriormente cada tres años la nueva versión sería designada como versión de soporte a largo plazo, o en inglés Long Term Support (LTS) (Smith, 2021).

Las últimas dos versiones de soporte de largo plazo corresponden a las versiones de Java 11 y Java 17, por lo tanto, para efectos de esta investigación se realizaron

comparaciones entre esas dos versiones y, además, se incorporó la versión 8, ya que es una versión ampliamente utilizada y que cuenta con soporte para el 2030.

Diseño de arquitectura

Con el fin de satisfacer los requerimientos de la aplicación, se pensaron en dos posibles diseños que se podrían utilizar: diseño de arquitectura en capas y diseño de arquitectura hexagonal.

Diseño hexagonal

Con respecto al diseño de arquitectura hexagonal, se propuso para la presente investigación con el fin de explotar las ventajas que ofrece, tales como, separación de responsabilidades, ya que la lógica de negocios se encuentra aislada de cualquier forma de comunicación de entrada o salida, permitiendo que sea más sencillo de probar.

Adicionalmente, la flexibilidad que ofrece este tipo de diseño hace posible que se pueda utilizar diversos tipos de tecnologías, como por ejemplo, REST o Kafka para la comunicación con el mundo exterior del microservicio, sin necesidad de afectar la lógica de negocios.

Finalmente, es sumamente portable y fácil de mantener, por tal motivo se propuso para esta investigación como una alternativa que permitiera satisfacer muchos de los requerimientos no funcionales de la aplicación hipotetica. (Cockburn, n.d.)

Diseño en capas

Este tipo de diseño es ampliamente aceptado y de hecho es el más común, convirtiéndose en el diseño por defecto en aplicaciones modernas.

Debido a lo anterior, se propuso utilizar este diseño en la presente investigación porque corresponde a un diseño muy conocido por la mayoría de arquitectos, diseñadores y desarrolladores. (Richards, 2015)

Este tipo de diseño permite que se divida la aplicación en capas, lo cual ayuda a que cada una pueda ser encapsulada en su marco de responsabilidades. Por ejemplo, la capa de acceso a datos, solo se preocupa por la interacción de la aplicación con la base de datos y no se preocupa por la presentación de la información.

Contenedores

Los contenedores son una tecnología que permite que microservicios puedan ser empaquetados y ejecutados de una forma sencilla e independiente.

Las pruebas realizadas con contenedores no tenían la intención de determinar si es una buena idea o no utilizar contenedores, sino más bien, entender la importancia de realizar un diseño adecuado de la cantidad de microservicios que se deben implementar a la hora de desarrollar una aplicación.

De hecho, según la Green Software Foundation, hay ciertas estrategias que se pueden utilizar en la arquitectura de microservicios con el fin de tener aplicaciones amigables con el ambiente, una de esas es reducir el número de microservicios.

La idea de reducir el número de microservicios está enfocado en evitar congestión de red y latencia a la hora de comunicarse entre microservicios, especialmente, si uno o más microservicios están muy acoplados entre sí. (Hussain, n.d.)

Para ello, el enfoque de *Domain Driven Design* es apto para que el número de microservicios pueda ser el adecuado, y de esta forma disminuir el consumo energético innecesario.

Estilos de comunicación

Existen diversos enfoques de diseño con los cuales diferentes microservicios de una aplicación pueden comunicarse entre sí. Por ejemplo, se puede utilizar REST (Representational State Transfer), con el cual un cliente, puede hacer peticiones a un API de otro cliente por medio del protocolo de comunicación HTTP.

Por otro lado, un estilo de comunicación es el de mensajería entre microservicios. En el caso de la presente investigación se decidió utilizar RabbitMQ, el cual permite realizar comunicaciones asíncronas entre un servicio que publica un mensaje y otro que lo recibe por medio de canales a los cuales distintos clientes se pueden suscribir.

Rendimiento y tiempos de respuesta

Basado en el estudio de percepción realizado, sobre las aplicaciones actuales similares a la que se propone en esta investigación, y tomando en cuenta los resultados de la encuesta disponible en el apéndice 1, se puede identificar que una de

las mayores críticas y quejas de los usuarios es con respecto al rendimiento de la aplicación y sus tiempos de respuesta al momento de utilizarla.

Por tal razón, se dio un enfoque fuerte en esta investigación a prácticas que propicien un mejor resultado en tiempos de respuesta como por ejemplo:

Batches

Los batches se refiere a un grupo de operaciones que se agrupan con el fin de que sean procesadas juntas, en lugar de que el procesamiento sea individual. Esta técnica es particularmente útil cuando se necesitan realizar muchas transacciones de escritura a la base de datos, con el fin de evitar la sobrecarga de la misma.

Hilos

Los hilos son de gran importancia en las aplicaciones modernas, ya que permiten la descomposición de tareas complejas en otras más pequeñas que pueden ser ejecutadas en paralelo, permitiendo que los tiempos de respuesta sean menores, y permitiendo un mayor aprovechamiento del recurso del CPU.

Cache

Es una de las prácticas de desarrollo que permite un acceso a la información de una forma más rápida, ya que, almacena datos en una capa de almacenamiento para su futuro acceso, permitiendo reducir los tiempos de respuesta al evitar realizar consultas adicionales a la base de datos.

Además, se consideró en la presente investigación, por su compatibilidad con el diseño CQRS, el cual divide la responsabilidad de acceso a datos tal y como se ha explicado en el documento.

Anexo 8: Evidencia de pruebas para definir la versión de Java

8.1 Evidencia de pruebas

Binary Tree Windows

Ejecución 1

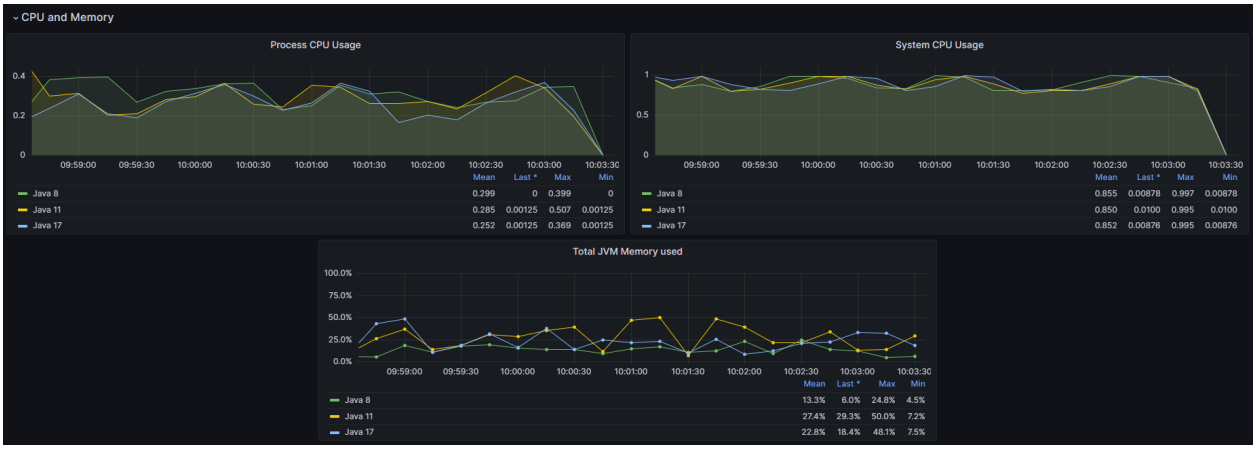

Ilustración 90 Ejecución en Windows algoritmo Binary tree 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 91 Ejecución en Windows algoritmo Binary tree 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 92 Ejecución en Windows algoritmo Binary tree 3. Fuente: Elaboración propia

Binary Tree MacOS

Ejecución 1

Ilustración 93 Ejecución en macOS algoritmo Binary tree 1. Fuente: Elaboración propia

Ejecución 2

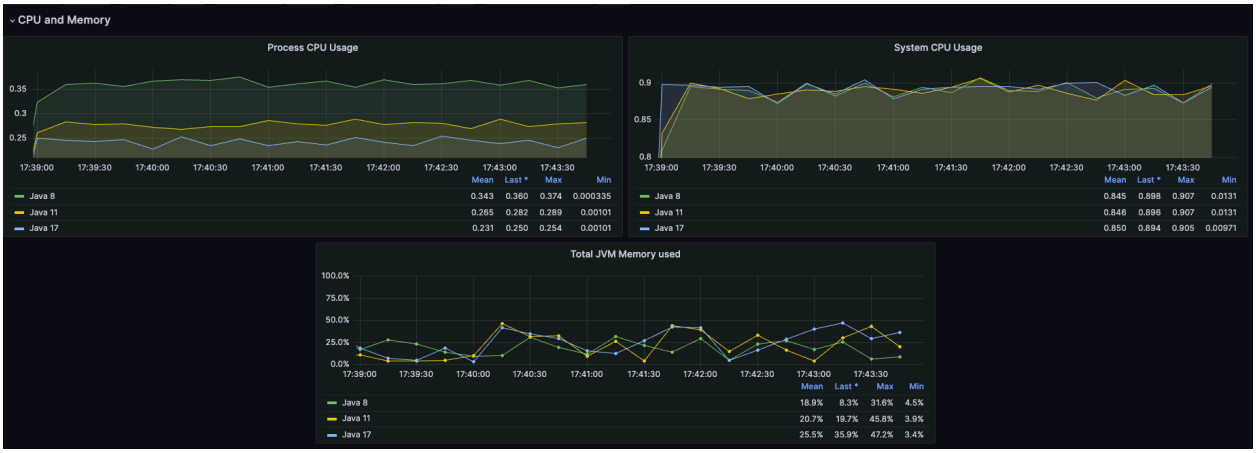

Ilustración 94 Ejecución en macOS algoritmo Binary tree 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 95 Ejecución en macOS algoritmo Binary tree 3. Fuente: Elaboración propia

Redux Windows

Ejecución 1

Ilustración 96 Ejecución en Windows algoritmo redux 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 97 Ejecución en Windows algoritmo redux 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 98 Ejecución en Windows algoritmo redux 3. Fuente: Elaboración propia

Redux MacOS

Ejecución 1

Ilustración 99 Ejecución en macOS algoritmo redux 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 100 Ejecución en macOS algoritmo redux 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 101 Ejecución en macOS algoritmo redux 3. Fuente: Elaboración propia

Quick Sort Windows

Ejecución 1

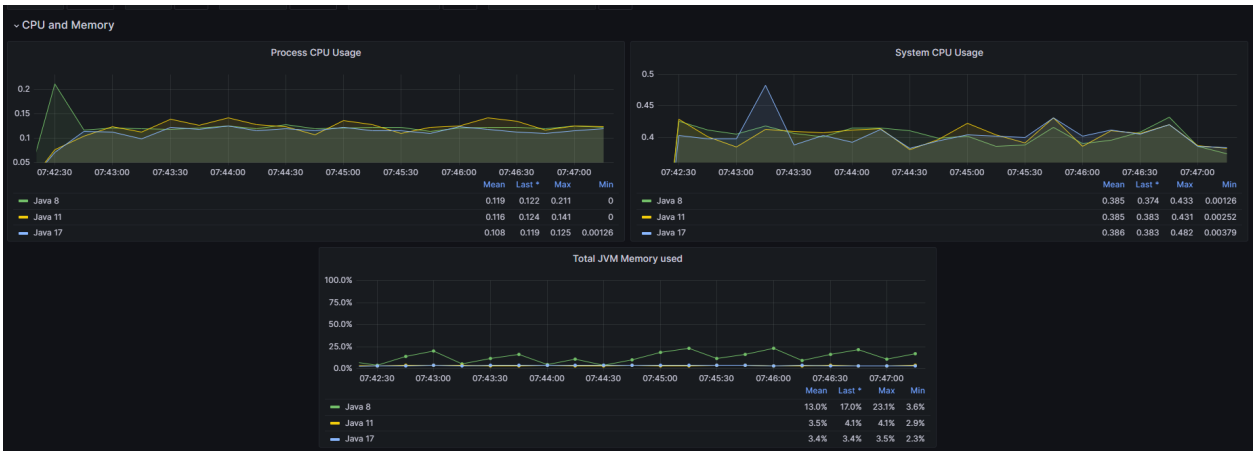

Ilustración 102 Ejecución en Windows algoritmo quick sort 1. Fuente: Elaboración propia

Ejecución 2

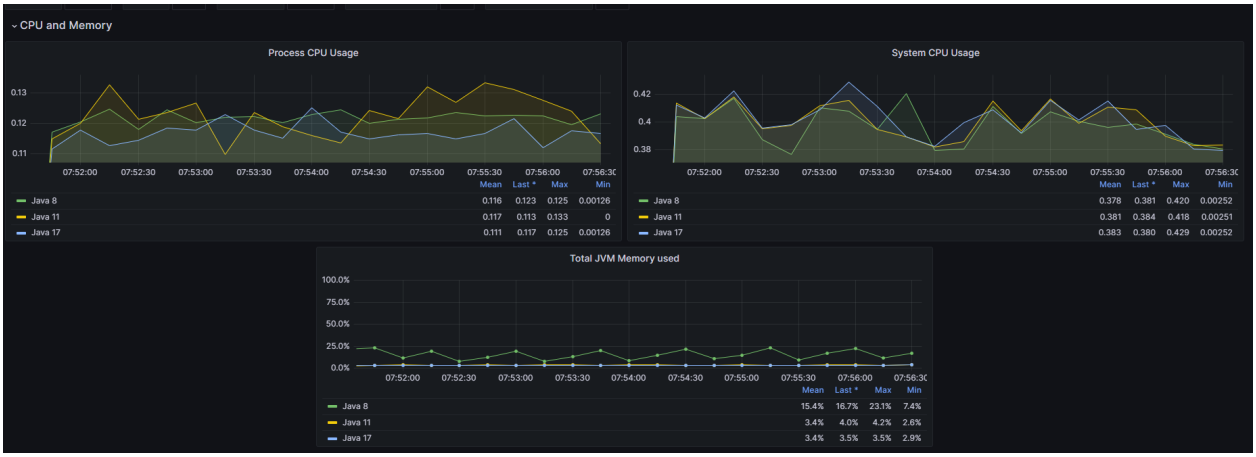

Ilustración 103 Ejecución en Windows algoritmo quick sort 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 104 Ejecución en Windows algoritmo quick sort 3. Fuente: Elaboración propia

Quick Sort MacOs

Ejecución 1

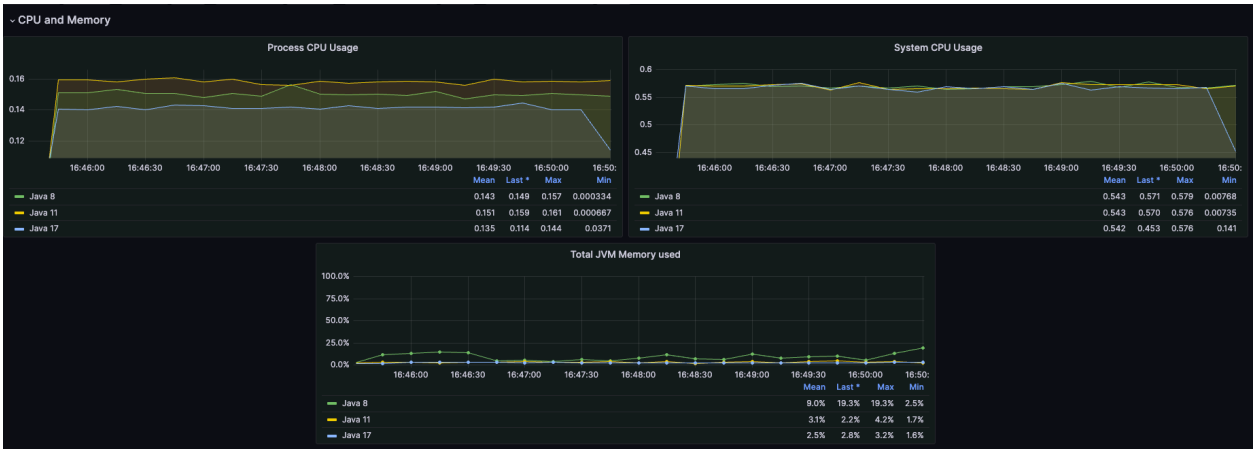

Ilustración 105 Ejecución en macOS algoritmo quick sort 1. Fuente: Elaboración propia

Ejecución 2

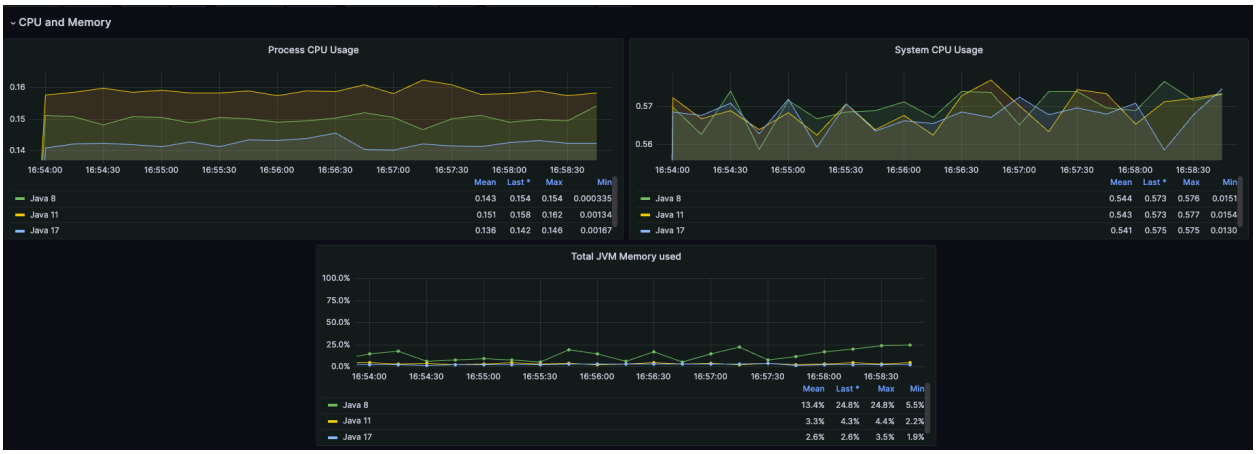

Ilustración 106 Ejecución en macOS algoritmo quick sort 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 107 Ejecución en macOS algoritmo quick sort 3. Fuente: Elaboración propia

Anexo 9: Evidencia de pruebas para definir diseño de arquitectura

Diseño de arquitectura hexagonal vs diseño de arquitectura en capas

Windows

Ejecución 1

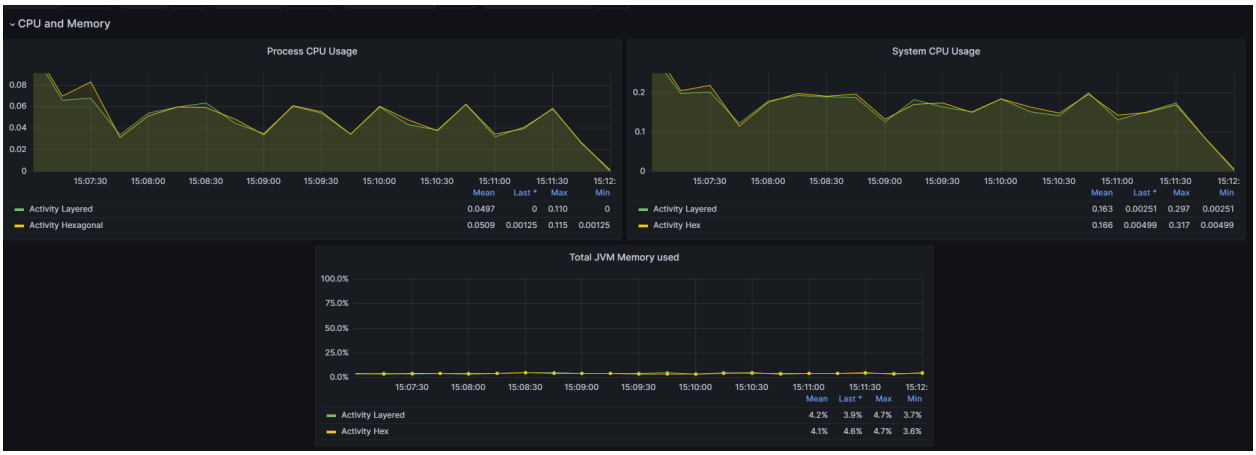

Ilustración 108 Ejecución de diseño de arquitectura en Windows 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 109 Ejecución de diseño de arquitectura en Windows 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 110 Ejecución de diseño de arquitectura en Windows 3. Fuente: Elaboración propia

Ejecución 4

Ilustración 111 Ejecución de diseño de arquitectura en Windows 4. Fuente: Elaboración propia

Ejecución 5

Ilustración 112 Ejecución de diseño de arquitectura en Windows 5. Fuente: Elaboración propia

MacOS

Ejecución 1

Ilustración 113 Ejecución de diseño de arquitectura en macOS 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 114 Ejecución de diseño de arquitectura en macOS 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 115 Ejecución de diseño de arquitectura en macOS 3. Fuente: Elaboración propia

Ejecución 4

Ilustración 116 Ejecución de diseño de arquitectura en macOS 4. Fuente: Elaboración propia

Ejecución 5

Ilustración 117 Ejecución de diseño de arquitectura en macOS 5. Fuente: Elaboración propia

Anexo 10: Evidencia de pruebas de rendimiento y tiempos de respuesta

Pruebas de rendimiento y tiempos de respuesta en ejecución e insersión de datos a la base de datos

Windows

Ejecución 1

Ilustración 118 Ejecución de pruebas de rendimiento de escritura de datos en Windows 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 119 Ejecución de pruebas de rendimiento de escritura de datos en Windows 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 120 Ejecución de pruebas de rendimiento de escritura de datos en Windows 3. Fuente: Elaboración propia

MacOS

Ejecución 1

Ilustración 121 Ejecución de pruebas de rendimiento de escritura de datos en macOS 1. Fuente: Elaboración propia

Ejecución 2

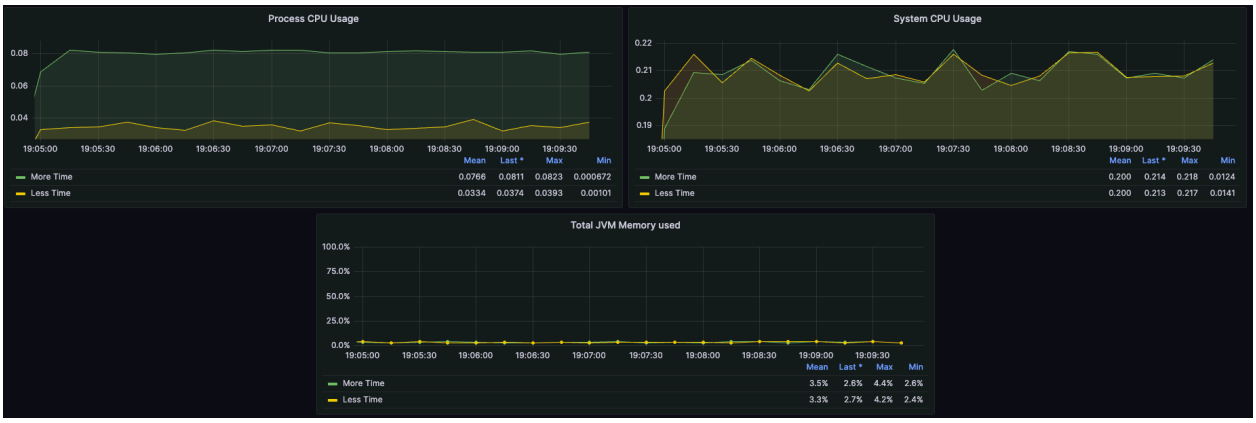

Ilustración 122 Ejecución de pruebas de rendimiento de escritura de datos en macOS 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 123 Ejecución de pruebas de rendimiento de escritura de datos en macOS 3. Fuente: Elaboración propia

Windows

Ejecución 1

Ilustración 124 Ejecución de pruebas de rendimiento de lectura de datos en Windows 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 125 Ejecución de pruebas de rendimiento de lectura de datos en Windows 2. Fuente: Elaboración propia

Ejecución 3

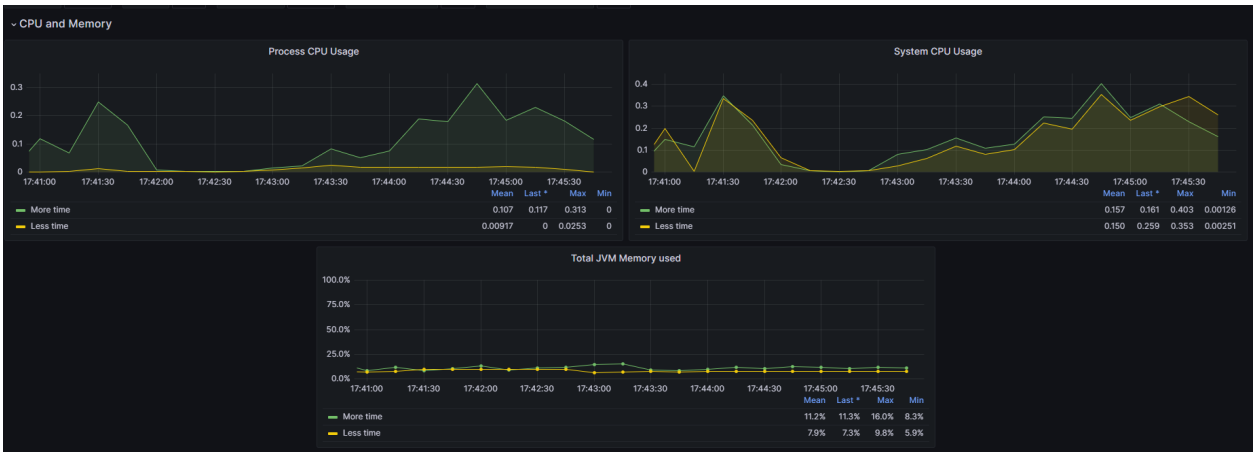

Ilustración 126 Ejecución de pruebas de rendimiento de lectura de datos en Windows 3. Fuente: Elaboración propia

MacOS

Ejecución 1

Ilustración 127 Ejecución de pruebas de rendimiento de lectura de datos en macOS 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 128 Ejecución de pruebas de rendimiento de lectura de datos en macOS 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 129 Ejecución de pruebas de rendimiento de lectura de datos en macOS 3. Fuente: Elaboración propia

Anexo 11: Evidencia de pruebas de aplicaciones creadas con contenedores

Docker

Windows

Ejecución 1

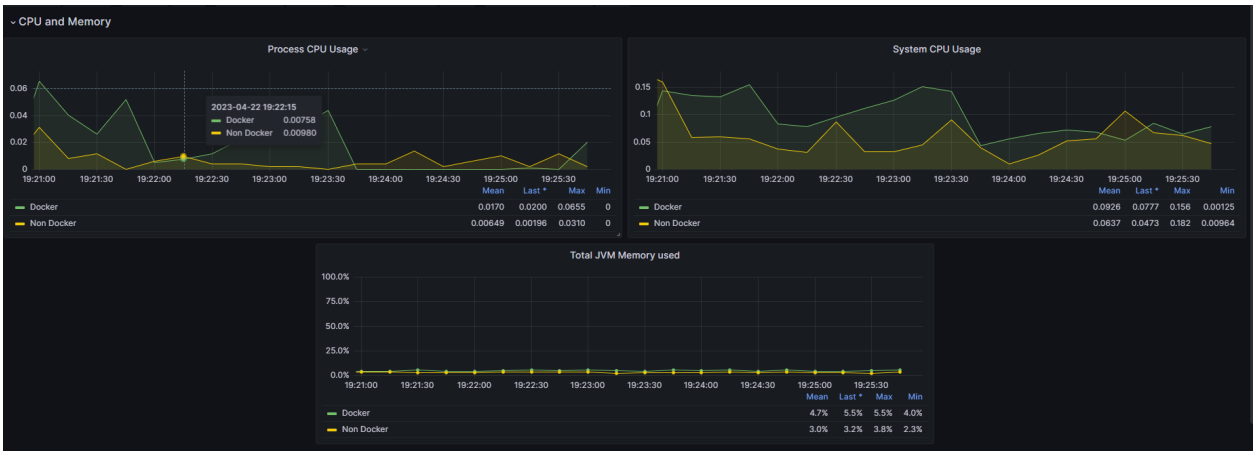

Ilustración 130 Ejecución de pruebas de docker en Windows - 1. Fuente: Elaboración propia

Ejecución 2

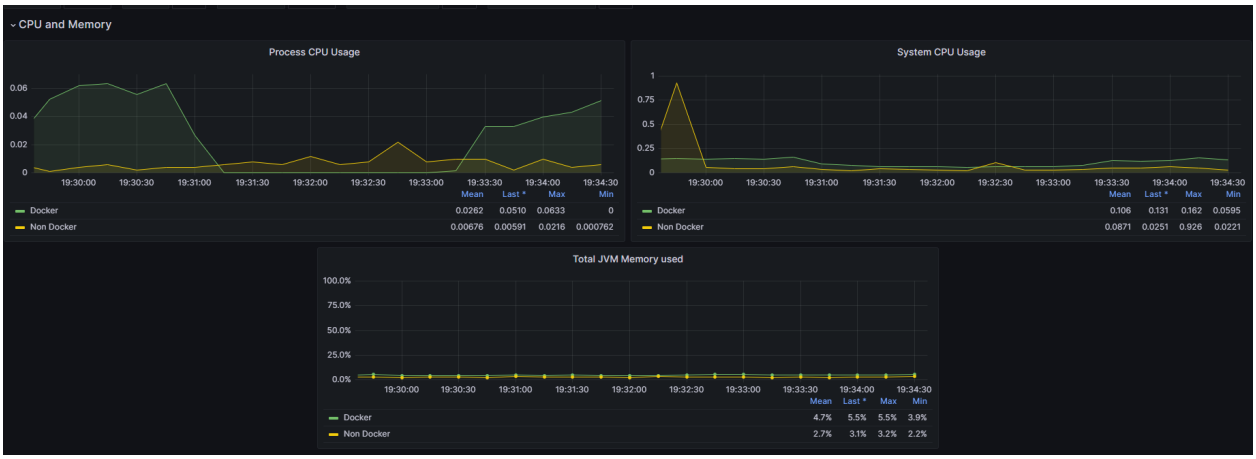

Ilustración 131 Ejecución de pruebas de docker en Windows - 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 132 Ejecución de pruebas de docker en Windows - 3. Fuente: Elaboración propia

MacOS

Ejecución 1

Ilustración 133 Ejecución de pruebas de docker en macOS - 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 134 Ejecución de pruebas de docker en macOS - 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 135 Ejecución de pruebas de docker en macOS - 3. Fuente: Elaboración propia

Anexo 12: Evidencia de pruebas de aplicaciones usando batches

Uso de batches

Windows

Ejecución 1

Ilustración 136 Ejecución de pruebas con batches en Windows - 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 137 Ejecución de pruebas con batches en Windows - 2. Fuente: Elaboración propia

MacOS

Ejecución 1

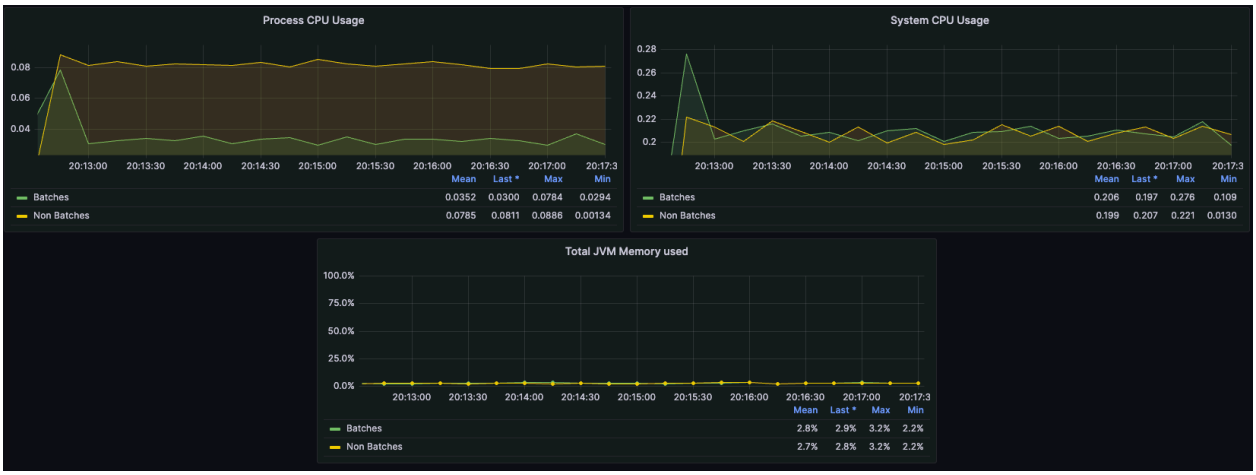

Ilustración 138 Ejecución de pruebas con batches en macOS - 1. Fuente: Elaboración propia

Ejecución 2

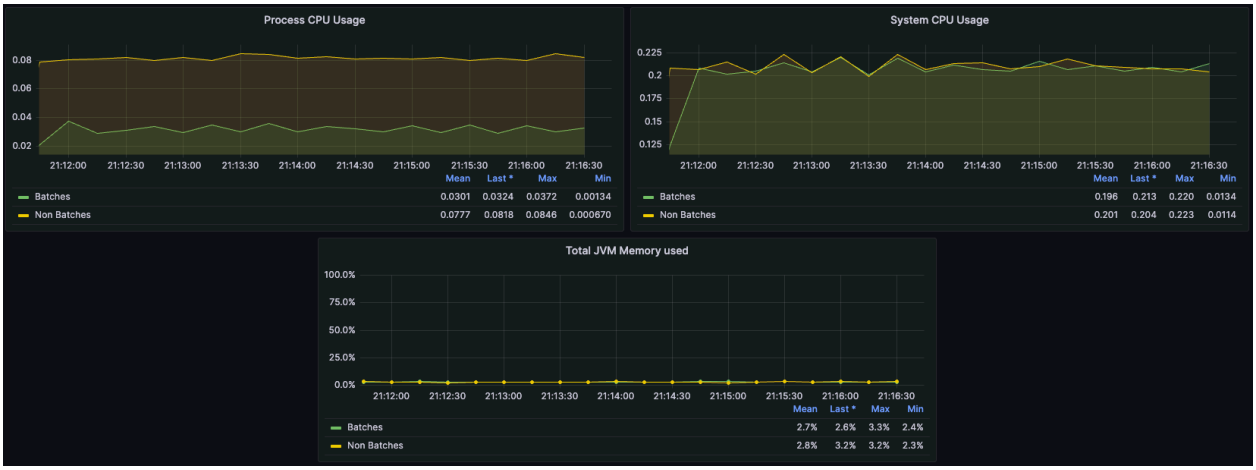

Ilustración 139 Ejecución de pruebas con batches en macOS - 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 140 Ejecución de pruebas con batches en macOS - 3. Fuente: Elaboración propia

Anexo 13: Evidencia de pruebas de aplicaciones usando hilos

Uso de hilos

Windows

Ejecución 1

Ilustración 141 Ejecución de pruebas con hilos en Windows - 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 142 Ejecución de pruebas con hilos en Windows - 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 143 Ejecución de pruebas con hilos en Windows - 3. Fuente: Elaboración propia

MacOS

Ejecución 1

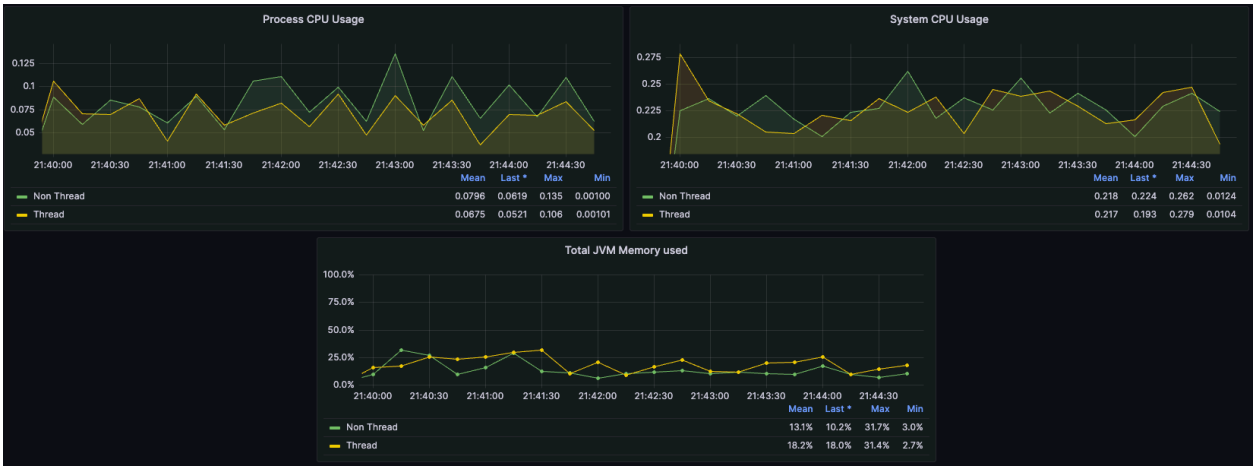

Ilustración 144 Ejecución de pruebas con hilos en macOS - 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 145 Ejecución de pruebas con hilos en macOS - 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 146 Ejecución de pruebas con hilos en macOS - 3. Fuente: Elaboración propia

Anexo 14: Evidencia de pruebas de aplicaciones usando cache

Redis con queries simples

Windows

Ejecución 1

Ilustración 147 Ejecución de pruebas con redis con consultas simples en Windows - 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 148 Ejecución de pruebas con redis con consultas simples en Windows - 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 149 Ejecución de pruebas con redis con consultas simples en Windows - 3. Fuente: Elaboración propia

Redis con queries que toman más tiempo

Windows

Ejecución 1

Ilustración 150 Ejecución de pruebas con redis con consultas con latencia en Windows - 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 151 Ejecución de pruebas con redis con consultas con latencia en Windows - 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 152 Ejecución de pruebas con redis con consultas con latencia en Windows - 3. Fuente: Elaboración propia

MacOS

Ejecución 1

Ilustración 153 Ejecución de pruebas con redis con consultas con latencia en macOS - 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 154 Ejecución de pruebas con redis con consultas con latencia en macOS - 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 155 Ejecución de pruebas con redis con consultas con latencia en macOS - 3. Fuente: Elaboración propia

Anexo 15: Evidencia de pruebas de aplicaciones usando comunicación por mensajes o por HTTP

REST vs RabbitMQ

Windows

Ejecución 1

Ilustración 156 Ejecución de pruebas con de enfoque de comunicación entre servicios en Windows - 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 157 Ejecución de pruebas con de enfoque de comunicación entre servicios en Windows - 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 158 Ejecución de pruebas con de enfoque de comunicación entre servicios en Windows - 3. Fuente: Elaboración propia

MacOS

Ejecución 1

Ilustración 159 Ejecución de pruebas con de enfoque de comunicación entre servicios en macOS - 1. Fuente: Elaboración propia

Ejecución 2

Ilustración 160 Ejecución de pruebas con de enfoque de comunicación entre servicios en macOS - 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 161 Ejecución de pruebas con de enfoque de comunicación entre servicios en macOS - 3. Fuente: Elaboración propia

Anexo 16: Evidencia de prueba final

Prueba final

Windows

Ejecución 1

Ilustración 162 Ejecución de pruebas con recomendaciones finales en Windows - 1. Fuente: Elaboración propia

Ejecución 2

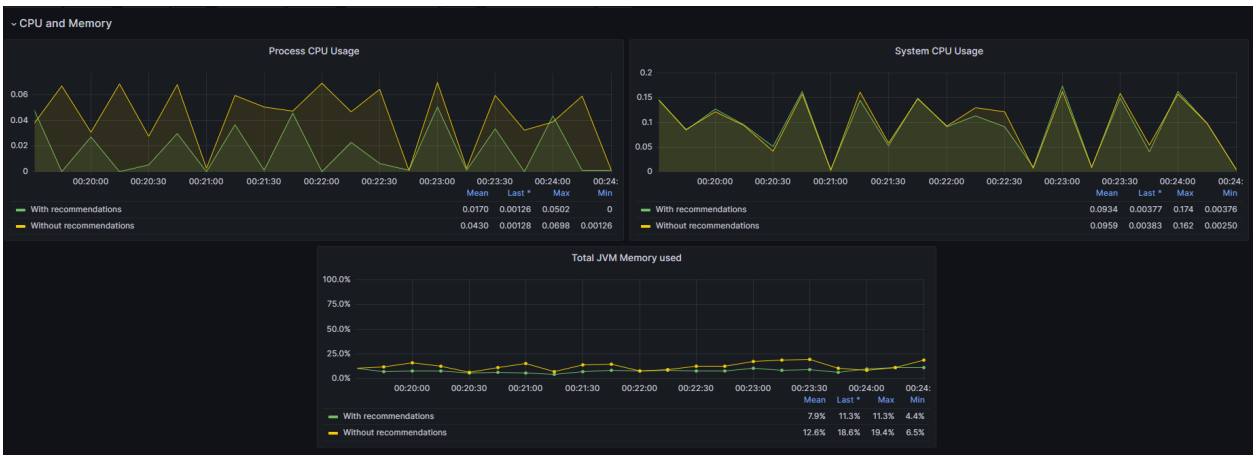

Ilustración 163 Ejecución de pruebas con recomendaciones finales en Windows - 1. Fuente: Elaboración propia

Ejecución 3

Ilustración 164 Ejecución de pruebas con recomendaciones finales en Windows - 3. Fuente: Elaboración propia

MacOS

Ejecución 1

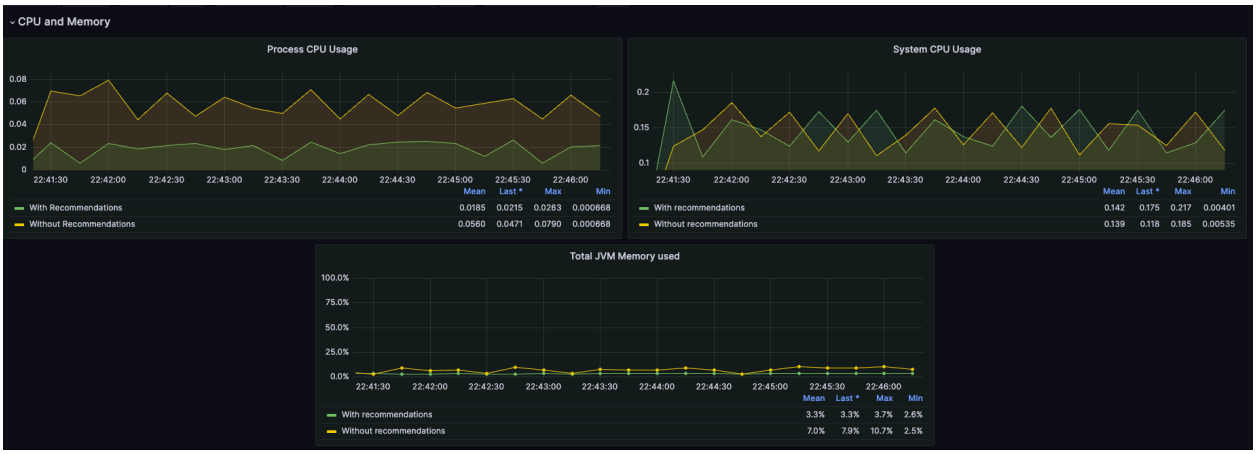

Ilustración 165 Ejecución de pruebas con recomendaciones finales en macOS - 1. Fuente: Elaboración propia

Ejecución 2

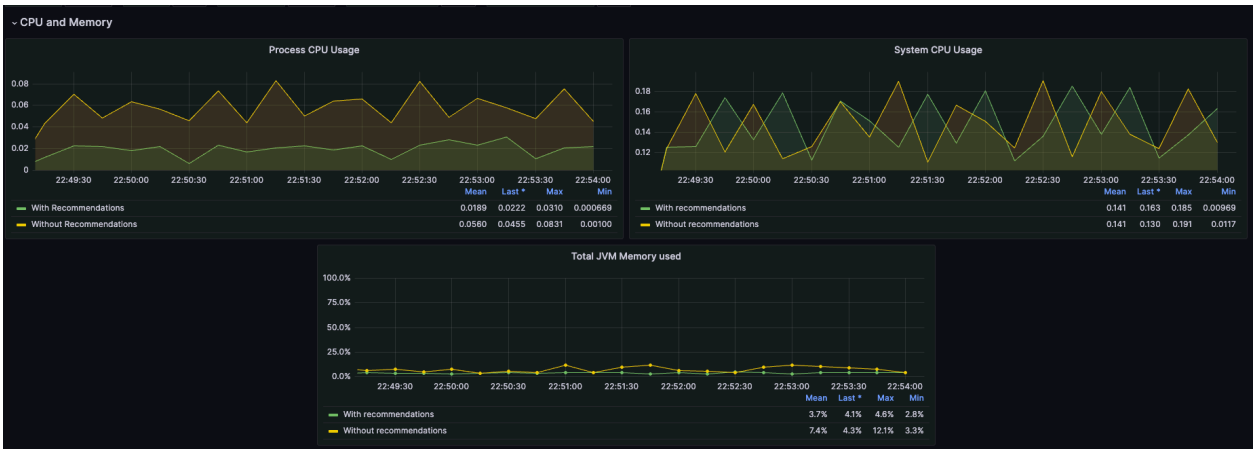

Ilustración 166 Ejecución de pruebas con recomendaciones finales en macOS - 2. Fuente: Elaboración propia

Ejecución 3

Ilustración 167 Ejecución de pruebas con recomendaciones finales en macOS - 3. Fuente: Elaboración propia

Anexo 17: Resultado de entrevista

16.1 Detalles de la entrevista:

1. Entrevistados 20 personas
2. La muestra se tomó de un grupo ingenieros de software, o carreras a fin, con amplia experiencia en microservicios y desarrollo de software.

16.2 Resultados de la entrevista:

Ilustración 168 Resultado de entrevista sobre desarrollo de software verde - Pregunta 1

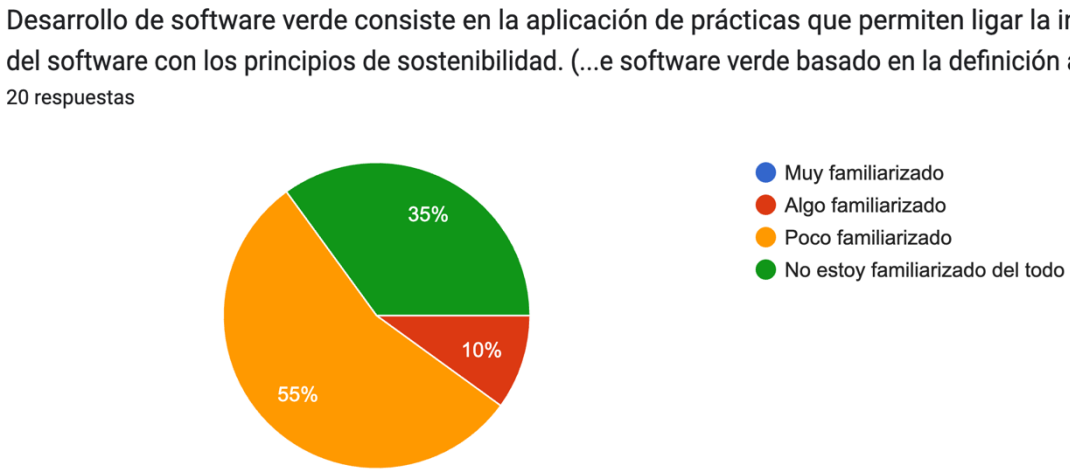

Ilustración 169 Resultado de entrevista sobre desarrollo de software verde - Pregunta 2

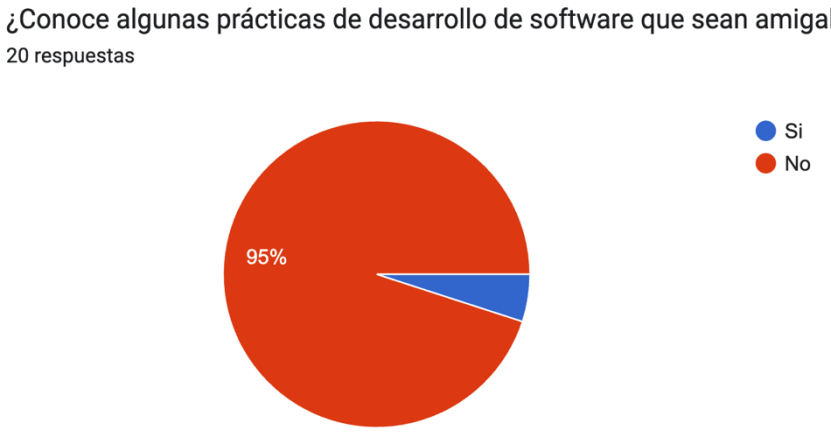

Ilustración 170 Resultado de entrevista sobre desarrollo de software verde - Pregunta 3

Las respuestas de la pregunta anterior se muestran a continuación:

- Reutilización de código y el empleo de técnicas para minimizar el tiempo de desarrollo
- La programación de algoritmos eficientes y el uso responsable de los recursos computacionales
- Diseño de sistemas computacionales eficientes, desde la escogencia de los lenguajes de programación e intérpretes de código, sistemas de B.B.D.D. a utilizar, hasta la arquitectura y sistemas de hardware (datacenters / cloud providers) utilizados

¿Considera importante aprender e implementar prácticas de desarrollo de software sostenible en sus proyectos? Justifique su respuesta.

- Si
- No realmente
- Si es importante todo lo que se pueda aportar para mitigar el cambio climático. Pero realmente no sé cómo dimensionar ese aporte.
- No veo que impacto puede producir el desarrollo de software en el medio ambiente que sea mayor que otras fuentes de contaminación o al menos significativa
- Si, para ayudar al medio ambiente
- Sí, para ayudar a cuidar el medio ambiente
- La verdad no había contemplado como el desarrollo de software puede contribuir de una manera razonable a la sostenibilidad del medio ambiente, por ende si me

parece bueno el ir aprendiendo y optimizar más el consumo energético y la adquisición de artefactos eléctricos para el desarrollo de software

- Si, porque hoy en día debemos cuidar más el planeta y evitar seguir destruyéndolo
- Si, cualquier perjuicio que generemos al medio ambiente, debe de ser tomado en consideración y remediado en la medida de lo posible
- Sí. El uso responsable de los recursos limitados debe ser una prioridad para todos. No solo permite sostenibilidad y que nuevas generaciones puedan disfrutar también de los recursos; sino que ahorra tiempo y recursos computacionales, lo que al final también se traduce en beneficios económicos.
- Sí lo considero importante porque el software, al igual que cualquier otra industria, debe contribuir en la resolución de los desafíos que enfrentamos como país y como civilización; y la sostenibilidad es clave en ello
- No, el hardware de la maquina tiene un consumo mínimo de operación, aunque el procesador no esté realizando ningún procesamiento. Cuando el procesador y la memoria consumen más energía ya de previo hay que resolver problemas en el desempeño de la aplicación mucho más notorios e importantes
- Si debido a que se debe de tener noción del impacto de cualquier desarrollo en el medio ambiente
- Sí, el mundo tiene que buscar prácticas sostenibles y bajar el consumo de energía
- Si, me parece que todos los proyectos sean de software o no deberían de tener conciencia sostenible con el ambiente y así evitar más contaminación
- Si, siempre y cuando no aumenten los tiempos de desarrollo.

- Si, fuera bueno aprender acerca del tema y de sus beneficios para brindar ayuda al medio ambiente también en esta área
- La sostenibilidad es un tema que ha venido en auge en los años para salvaguardar los recursos q utilizamos, y si considero importante aprender e implementar estas prácticas.
- No.

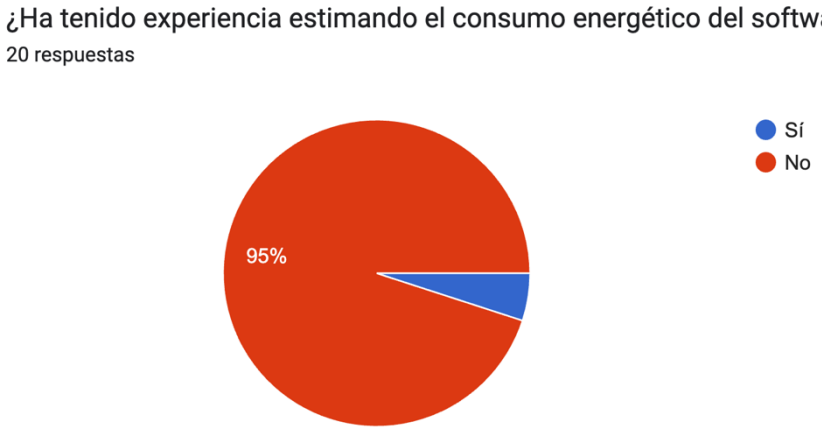

Ilustración 171 Resultado de entrevista sobre desarrollo de software verde - Pregunta 5

Ilustración 172 Resultado de entrevista sobre desarrollo de software verde - Pregunta 6