Revista de investigación de sistemas e informática
15(2): 107-122 (2022)
DOI: https://doi.org/10.15381/risi.v15i2.24315

ISSN: 1815-0268 (Impreso) / ISSN: 1816-3823 (Electrónico)
Facultad de Ingeniería de Sistemas e Informática - UNMSM

Análisis comparativo de algoritmos concurrentes basado en tecnología web para optimizar el proceso de suscripción en los servicios de streaming

Comparative analysis of web-based concurrent algorithms to optimize the subscription process for streaming services

Luz Elena Torres Talaverano
https://orcid.org/0000-0001-6465-0430
luzelena.torres@unmsm.edu.pe

Andrea Mariana Huarhuachi Ortega
https://orcid.org/0000-0003-0384-0235
andrea.huarhuachi@unmsm.edu.pe

Giajahira Cristhel Quito Cucho
https://orcid.org/0000-0002-4069-2182
giajahira.quito@unmsm.edu.pe

Felix Alfonso Fernandez Caillahua
https://orcid.org/0000-0003-2758-5478
felix.fernandez2@unmsm.edu.pe

Gustavo Adolfo Alania Inga
https://orcid.org/0000-0002-6445-7515
gustavoadolfo.alania@unmsm.edu.pe

Ivan Carlo Petrlik Azabache
https://orcid.org/0000-0002-1201-2143
ipetrlika@unmsm.edu.pe

Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática. Lima. Perú

Recibido: 18/11/2022 - Aceptado: 24/12/2022 - Publicado: 30/12/2022

RESUMEN

El presente artículo tiene como finalidad realizar un análisis comparativo para optimizar el proceso de suscripción en los servicios de streaming a través de algoritmos concurrentes. La problemática de esta investigación es la dificultad de la suscripción en los clientes de las diversas compañías de streaming, la cual se debe a la falta de mantenimiento y gestión de suscripciones masivas, generando colapsos en las plataformas que ofrecen este servicio. Los algoritmos concurrentes han demostrado ser una solución eficiente al problema en la gestión de recursos en sistemas altamente visitados. El diseño de la investigación fue cualitativo y el tipo fue descriptivo, el resultado demuestra que por medio de la comparación de los algoritmos concurrentes en la plataforma web, tenemos un algoritmo desarrollado en Go (Sincronización, Message Passing y Barreras binarias), que al ejecutar el testeo, indica que con 1000 solicitudes para el proceso de suscripción se obtiene un tiempo de respuesta medio de 1402 ms con un rendimiento de 249.2 procesos por segundo en cambio el otro algoritmo desarrollado en PHP , que al ejecutar el testeo , indica que con 1000 solicitudes para el proceso de suscripción se obtiene un tiempo de respuesta medio de 1689 ms con un rendimiento de 524 procesos por segundo. Finalmente, llegamos a la conclusión del uso del algoritmo concurrente desarrollado en Go bajo la plataforma web, optimiza el proceso en base a sus indicadores de tiempo de respuesta medio y rendimiento para la suscripción en los servicios de streaming a través de la Sincronización, Message Passing y Barreras binarias.

Palabras clave: Tecnología web, algoritmos concurrentes, concurrencia, rendimiento, tiempo de respuesta, Go.

ABSTRACT

The purpose of this article is to perform a comparative analysis to optimize the subscription process in streaming services through concurrent algorithms. The problem of this research is the difficulty of subscription in the customers of the various streaming companies, which is due to the lack of maintenance and management of massive subscriptions, generating collapses in the platforms that offer this service. Concurrent algorithms have proven to be an efficient solution to the problem of resource control in highly visited systems. The research design was qualitative, and the type was descriptive, the result shows that through the comparison of concurrent algorithms on the web platform, we have an algorithm developed in Go (Synchronization, Message Passing, and Binary Barriers), which when running the test, indicates that with 1000 requests for the subscription process an average response time of 1402 ms is obtained with a throughput of 249. 2 processes per second, on the other hand, the other algorithm developed in PHP, which indicates that with 1000 requests for the subscription process, an average response time of 1689 ms is obtained with a throughput of 524 processes per second. Finally, we conclude that the use of the concurrent algorithm developed in Go under the web platform, optimises the process based on its average response time and performance indicators for subscription streaming services through Synchronisation, Message Passing and Binary Barriers.

Keywords: Web technology, concurrent algorithms, concurrency, performance, response time, Go.

I. INTRODUCCIÓN

Durante la pandemia del COVID-19 se determinó el aislamiento social obligatorio, es decir, la imposibilidad de salir de los hogares. Este suceso marcó un antes y después en los hábitos sociales de entretenimiento y derivó en la aplicación de acciones para dar respuesta a la pandemia, a través de los medios digitales que jugaron un papel central Bao, H., Cao, B., Xiong, Y., & Tang, W. (2020). Estos medios digitales proporcionaron contenidos y con los avances tecnológicos y sumado la pandemia han cambiado la forma de producir, vender, distribuir y consumir este tipo de contenidos audiovisuales según Iglesias, E. L. (2022), el fenómeno de consumo que ha abarcado no solamente en la forma de decidir y consumir como también en la industria de producción son las plataformas digitales, de las cuales es lo que ve el usuario en la pantalla. Existen empresas como Netflix, Disney, Amazon Prime Video que presentan un modelo de suscripción orientados a servicios de televisión en streaming la cual consiste, que, a través de un contrato con el usuario, brindan el acceso de forma ilimitada a diversos contenidos, es decir que las personas suscritas puedan reproducir los videos en el momento que desee (Llanes M. et al.,2021). Asimismo, Díez de Paz (2021), afirma que las plataformas presentaron un gran incremento durante el primer trimestre respecto al año 2020, esto es debido a la emergencia sanitaria en donde hubo un gran incremento de usuarios que desearon adquirir una suscripción de manera simultánea, ralentizando el proceso.

En Latinoamérica, se está utilizando los servicios de streaming en la que el 63% de usuarios consideran que el principal motivo para suscribirse en alguna plataforma streaming es para acceder a diversos contenidos audiovisuales, como las películas que recién se estrenan y series que sean de su agrado (Sherlock C. et al., 2021). En Perú, el 52% de peruanos utilizan el servicio de streaming por video, en donde la región de Lima tiene la mayor incidencia con un 71% (Datum I. et al., 2020). En base a los datos encontrados, se infiere que las plataformas de streaming han tenido mayor acogida durante la pandemia debido a las medidas de confinamiento, eso condujo a las personas a consumir entretenimiento a través de estas plataformas, por lo cual, hasta estos días son muy demandadas por los usuarios. Asimismo, Oliveira et al., (2020) analizó los consumidores de Netflix en los países de Brasil y Portugal, aquí se evidenció que aproximadamente el 50% de usuarios de Brasil tienen problemas en el uso de la plataforma, en contraste con el de los usuarios de Portugal en la que se obtuvo un 36.9%, los problemas que abarcan son en relación al acceso de la cuenta, visualización del contenido y dificultad para realizar cambios en la cuenta del usuario. Por lo tanto, los consumidores han suscitado quejas de este tipo de plataformas de suscripción y el motivo principal se debe a la falta de mantenimiento y gestión de suscripciones masivas, la cual genera colapso y no permite al cliente acceder al plan deseado.

Si estas dificultades continúan podría haber una falla masiva en el sistema, generando que muchos usuarios quiten su suscripción en estas plataformas y las empresas de streaming pierdan ingresos económicos. Por consiguiente, se planteó la realización de un análisis comparativo para optimizar el proceso de suscripción a algoritmos concurrentes en la plataforma web de servicios de streaming. A continuación, se va a presentar los antecedentes de la respectiva investigación:

Según Wang, Q. (2019), la gran afluencia de usuarios en los sistemas e-commerce no permite lograr un alto rendimiento y eficacia en los recursos utilizados. Por ende, esta investigación tiene como objetivo demostrar la importancia de los recursos blandos (threads y conexiones del servidor) en la disminución de la carga de trabajo al procesador, de igual manera sus efectos en la disminución significativa del tiempo de respuesta resultante del cambio implícito de la concurrencia del procesamiento en las solicitudes en el sistema. Los resultados demuestran que el uso del modelo concurrente disminuye los tiempos de respuesta en un 30% y la carga al procesador a comparación de otro modelo (Amazon EC2-AutoScale) que no usa concurrencia. Asimismo, Duda, J. y Dlubacz, W. (2019), tienen como objetivo el análisis de cómo esas técnicas pueden contribuir a la mejora del rendimiento de una computación distribuida basada en la web. El sistema tiene un servidor web en lenguaje de programación Go y buscadores el cual ejecuta tareas de computación. Las tecnologías actuales en el entorno web son HTML5 Web Workers y JavaScript WebAssembly estas se usan para optimizar la aplicación de los recursos adecuados por los clientes. Los resultados obtenidos indican que en muchos casos la tecnología WebAssembly puede acelerar significativamente la ejecución de las funciones. Por otro lado, Whitney J., Gifford C. & Pantoja M. (2018) plantean como objetivo principal de su investigación, facilitar a los programadores de Go para ejecutar código en un sistema distribuido también en uno multinúcleo, implementando una librería (Gluster), que abstrae los detalles y dificultades de distribuir código. Aquí se desarrolló un modelo de programación que permite a los usuarios distribuir fácilmente el trabajo entre máquinas mediante el uso de subprocesos del sistema operativo con bibliotecas similares a Pthreads y OpenMP. Teniendo como resultados que Gluster puede con éxito acelerar la ejecución de funciones mediante la utilización de un grupo de máquinas. El desempeño escala bastante bien y agregar más máquinas proporciona una aceleración casi lineal dentro de un grupo. Continuando con otros estudios, los autores Liu J., Zhang S., Wang Q. & Wei J. (2022), proponen en esta investigación un modelo SCT (Scatter-Concurrency-Throughput) que puede determinar rápidamente la asignación de recursos blandos casi óptima para cada servidor en el sistema utilizando mediciones de desempeño y concurrencia en tiempo real de cada servidor. Además, se implementó un sistema de escalado automático (ConScale) que integra el modelo SCT para redistribuir rápidamente los recursos blandos a los servidores críticos del sistema para maximizar la capacidad de recursos del nuevo hardware después del escalado del sistema. Seguidamente, los autores William G., Anthony R. & Purnama J. (2020) nos proponen distintas soluciones en la parte del backend de un sistema de ventas de ropa por internet, estas soluciones se compararon de acuerdo a la cantidad de usuarios conectados y el tiempo de respuesta de cada solución, entre las propuestas se encuentra NodeJS, PHP y Python-Web. El objetivo de esta investigación es encontrar la mejor solución con respecto al tiempo de respuesta a pesar de tener una cantidad muy alta de usuarios. Los resultados evidencian que el tiempo de respuesta con NodeJs fue de 0.5 ms por cada petición realizada, PHP y Python-Web de 5 ms y 6 ms; respectivamente.

II. MATERIALES Y MÉTODOS

En la sección de materiales, se han desarrollado dos aplicaciones web que permiten la suscripción en los servicios streaming, en los lenguajes de programación GO y PHP. Según Andrawos, M., & Helmich, M. (2017), GO es un lenguaje multiplataforma moderno que es muy potente a la vez que sencillo. Es una opción excelente para microservicios y aplicaciones en la nube. En cambio, PHP es un lenguaje maduro y potente que se ha convertido en la tecnología más utilizada para crear sitios web dinámicos (Powers, D, 2022).

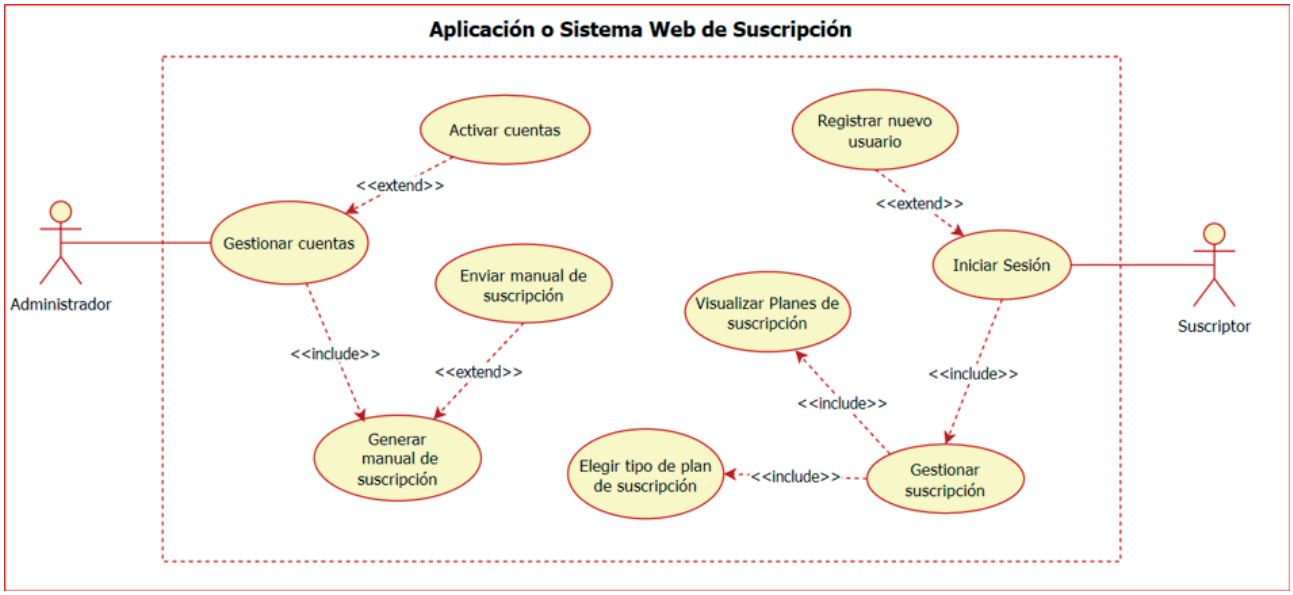
Las aplicaciones que fueron desarrolladas contemplan los requisitos funcionales expresadas en el lenguaje UML, que según Garcia, J. (2018), UML es la combinación de las tres mejores metodologías del modelado de objetos (método Booch, OMT y OOSE) en donde se agrega nuevos conceptos y visiones de lenguaje. En base a la premisa anterior, se eligió este lenguaje de modelado, mediante el cual se va a presentar los respectivos diagramas:

A. Diagrama de casos de uso de la Aplicación

Según la fig. 1, observamos al actor administrador, en la cual se encarga de la gestión de diversas cuentas que almacena la aplicación. Este caso de uso posee un “extend” que permite activar las diversas cuentas de los suscriptores en caso sea necesario. Por otra parte, tomando como caso de uso base gestionar cuentas, este se relaciona con un “include” respecto a la generación de un manual de suscripción; y este caso de uso se relaciona con un “extend” la cual deriva al envío del manual de suscripción. Con respecto al actor suscriptor interactúa directamente con el caso de iniciar sesión, lo cual se relaciona (extend) con el caso registrar nuevo usuario. Asimismo el caso de uso de iniciar

Figura 1

Diagrama general de casos de uso de la Aplicación o sistema web de Suscripción



sesión relaciona (include) de manera obligatoria con el caso de uso gestión de suscripción, y este mismo se relaciona (include) con el caso de uso visualizar planes de suscripción y con otra relación (include) para el caso de uso elegir tipo de plan de suscripción.

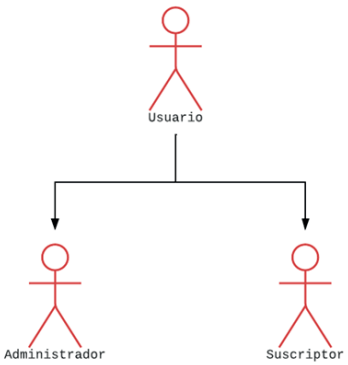
B. Actores de la aplicación

Seguidamente mostramos los actores del sistema o aplicación web de suscripción en los servicios streaming:

Según la fig. 2, observamos la escenificación de los actores del sistema o aplicación web, conformado por el actor usuario que es la generalización de los actores administrador y suscriptor.

Figura 2

Actores del sistema o aplicación web



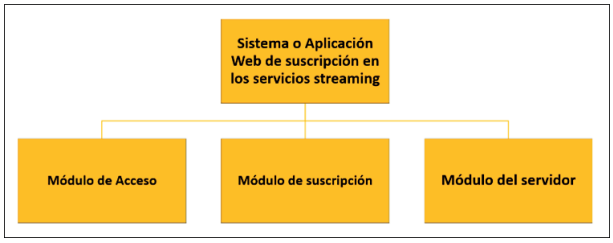
C. Módulos del sistema o aplicación web

A continuación, presentamos los módulos del sistema o aplicación web de la siguiente manera:

Según la fig. 3, observamos los módulos del sistema o aplicación web, comprendido por el módulo de acceso que permite el ingreso de los datos personales para acceder a la plataforma y realizar el proceso de suscripción, asimismo el usuario ya registrado ingresa el correo y contraseña para acceder al sistema; módulo de suscripción que permite mostrar una interfaz donde figura los planes de suscripción disponibles en la plataforma, además tiene un botón en cada plan para seleccionar la suscripción que desea; y el módulo del servidor , aquí interviene el usuario administrador. También, se muestra todos los correos logeados de los usuarios la cual servirá para una buena gestión. Asimismo, el administrador tendrá la posibilidad de activar las cuentas y enviar el manual de pago al usuario suscriptor.

Figura 3

Módulos del sistema o aplicación web



D. Pantallas del sistema o aplicación web

Las pantallas de la aplicación o sistema web, están centrados específicamente en el proceso de suscripción y lo vamos a relacionar en base a los módulos que ya se han presentado anteriormente:

1. Módulo de acceso

Según la fig. 4 se muestra el login con los campos de correo electrónico y contraseña por parte del usuario, con estas credenciales podrá acceder a través de la validación del sistema de suscripción.

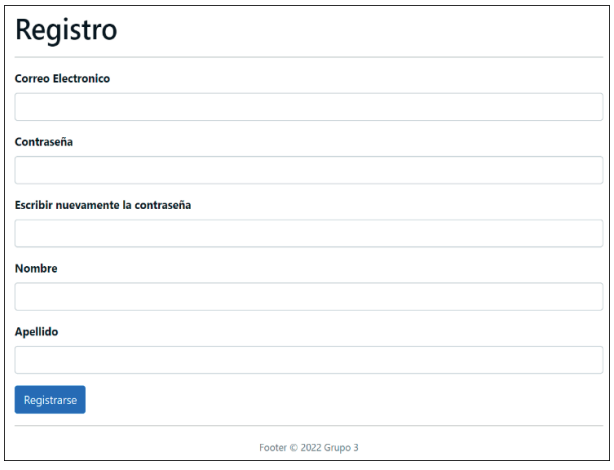
Figura 4

Login de la plataforma web

Según la fig. 5 se observa los diferentes campos que el suscriptor debe llenar para poder registrarse en el sistema y así acceder a un plan de suscripción.

Figura 5

Registro de nuevo usuario en la plataforma web

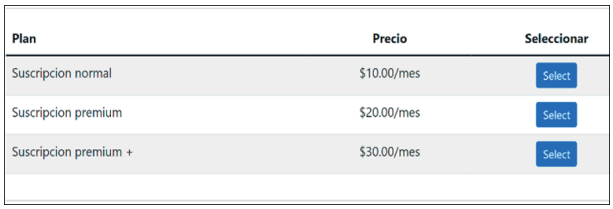


2. Módulo suscripción

Según la fig. 6, se muestran los diferentes tipos de planes con el que cuenta el sistema de suscripción, en esta sección el usuario podrá elegir la opción o plan que desea adquirir.

Figura 6

Planes de suscripción en la plataforma web

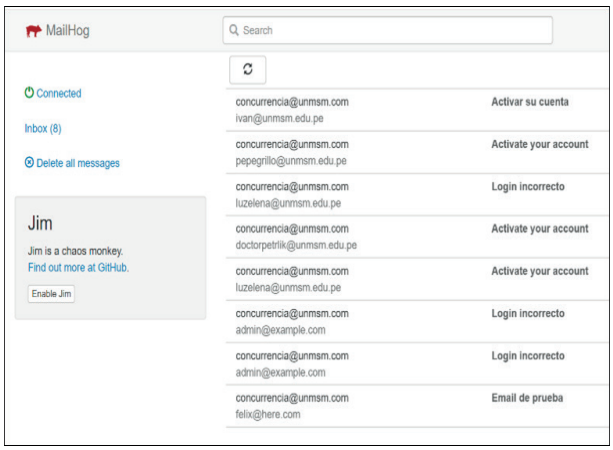


3. Módulo del servidor

Según la fig.7, se muestra el STP de la página donde los administradores podrán activar las cuentas de los usuarios, asimismo, proteger sus cuentas a través de un mensaje de aviso y enviar el plan de suscripción (tipo manual) al correo del usuario.

Figura 7

STP de la plataforma web



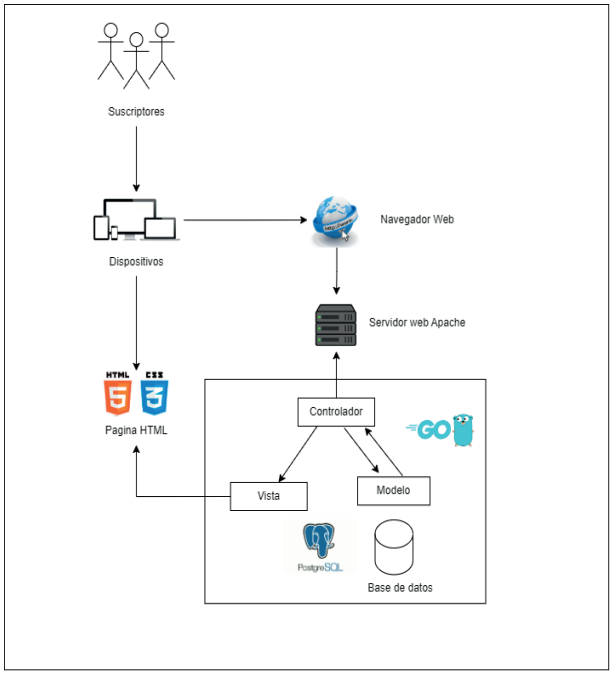
E. Arquitectura de la Aplicación de suscripción

A nivel de diseño se va a presentar la arquitectura del software de la siguiente manera:

Según la fig. 8, se muestra todo el proceso desde que el usuario realiza una acción hasta que obtenga una respuesta. Los usuarios que deseen suscribirse acceden a la plataforma web a través del navegador en múltiples dispositivos, mediante las tecnologías de desarrollo web, según Jiménez, J. (2022), HTML y CSS, se utilizan para elaborar diseños y estructuración del sitio web, mostrando el desarrollo de módulos y el contenido del sistema.

Figura 8

Arquitectura del Software



F. Desarrollo de la Aplicación de suscripción

Para iniciar el desarrollo de la respectiva aplicación o sistema web de suscripción, se realizaron dos versiones en los lenguajes de programación Go y PHP.

1. Desarrollo con el Lenguaje GO

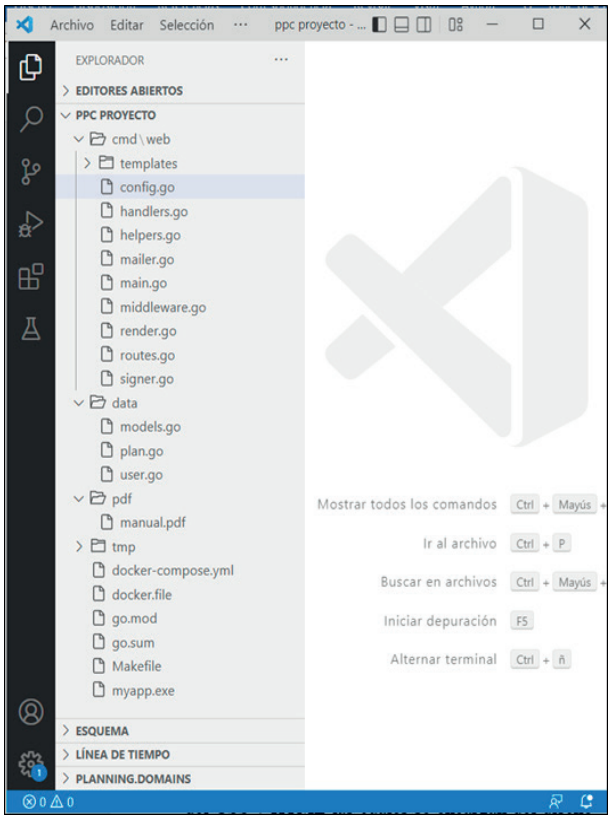
Según Bodner, J. (2021), Visual Studio Code es un buen entorno de desarrollo gratuito e incluye el soporte adecuado con Go, para ello hay que instalar una extensión que permita incluir las herramientas necesarias para la codificación del proyecto en Go. Por lo tanto, en base a esta elección del lenguaje de programación, se ha utilizado el IDE Visual Code, en la cual se evidencia la estructura del proyecto de la siguiente manera:

Según la fig. 9, se observa la estructura del proyecto comprendido a través de un patrón de diseño "Singleton". Según Contreras, M. (2017) el patrón Singleton es fácil de recordar. Como su nombre indica, le proporcionará una única instancia de un objeto y garantizará que no haya duplicados. única instancia de un objeto, y garantiza que no hay duplicados. Por lo tanto, en el presente proyecto se ha aplicado bajo ese patrón ya que se va a crear un modelo de inicialización llamado "módulo go" para realizar instancias de las clases de manera abstracta y sean llamadas cuando se necesite.

El proyecto consta de interfaces que se encuentran en la carpeta templates, estas se muestran según el contenido que quiera acceder el usuario suscriptor. En la carpeta data se observa la implementación y la estructura de la base de datos a través del código Golang (Go). También se muestra una carpeta llamada pdf que contiene la plantilla del manual. Y finalmente, una carpeta con el nombre tmp en las que se encuentran los archivos que genera la base de datos implementadas por Docker.

Figura 9

Estructura del proyecto en GO a través de la herramienta Visual Code



Seguidamente, la aplicación desarrollada en GO, está soportado por una serie de algoritmos concurrentes para mejorar su nivel de respuesta y rendimiento que a continuación se va a mostrar en la tabla 1.

A continuación, en función a la estructura del proyecto desarrollado en Go de la figura 9, vamos a especificar cada uno de los algoritmos concurrentes que mejoran el rendimiento del proceso, tiempo de respuesta y el número de solicitudes, plasmado en códigos en el lenguaje Go, especificados en la tabla 1.

Tabla 1

Algoritmos concurrentes que soportan la aplicación en GO

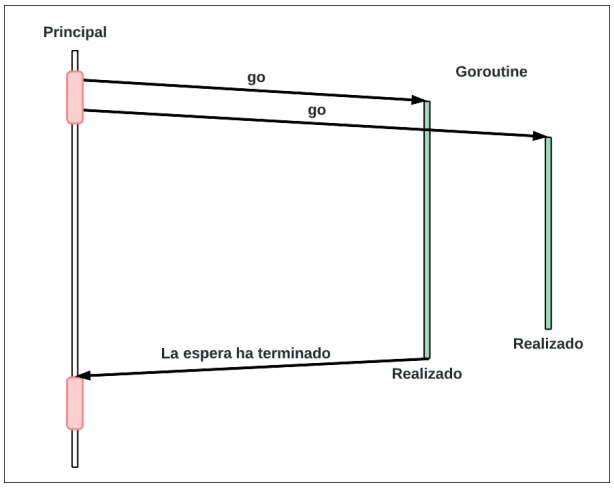
Algoritmos	Descripción
Sincronización	Según Moreno, G. (2022), la sincronización maneja interacciones entre las unidades de ejecución de cada proceso. Sin esto, el programa no podría actuar de manera controlada originando problemas de interbloqueo, bloqueo activo o inanición. En base a lo mencionado, en el propio lenguaje Golang (GO), se puede realizar a través de los threads y la sincronización de clases, logrando evitar las condiciones de carrera.
Message passing	Según Santiago, M. (2019), el paso de mensajes permite que los procesos intercambien mensajes entre sí mismos mediante el uso de operaciones de envío (send) y recibo (receive), el cual pasa a través de canales (channels) de comunicación; esto quiere decir, que en el lenguaje Golang (GO), estos mensajes pasan a través de los canales permitiendo la realización del algoritmo.
Barreras binarias	Según Raiturkar, J. (2018) los canales en Golang son simplemente colas con una interfaz simple, estas operaciones se pueden bloquear de manera eficiente mediante un puntero, que es representado mediante una barrera binaria; de esto se deduce que en el lenguaje Golang (GO), se plantea que cada operación sólo utilice un proceso sincronizando las rutinas a través del bloqueo de algunos canales.

Algoritmo de Sincronización en GO

Una forma de solucionar el problema de la sincronización y condiciones de carrera son los waitgroups. Estos fueron creados con el propósito de esperar que el goroutine principal termine de ejecutar todos los goroutines antes de seguir con su funcionamiento normal. En la fig. 10 se visualiza la representación gráfica del funcionamiento de un waitgroup en Go.

Figura 10

Funcionamiento del waitgroup en GO



Asimismo, se puede observar el respectivo funcionamiento expresado en código GO de la siguiente manera:

Este fragmento de código se encuentra ubicado en el archivo “main.go”. Según la fig. 11, se muestra la implementación del waitgroup en la estructura principal de la aplicación. Ahí se utilizó con el objetivo de que el hilo principal espere al goroutine de envió de mensaje al STP.

Figura 11

Código GO expresado a través del algoritmo de sincronización

```
//waitgroup para procesos concurrentes
wg := sync.WaitGroup{}

app := Config{
    Session:  session,
    DB:       db,
    InfoLog:  infoLog,
    ErrorLog: errorLog,
    Wait:     &wg,
    Models:   data.New(db),
    ErrorChan: make(chan error),
    ErrorChanDone: make(chan bool),
}
```

A continuación, se muestra el waitgroup en la función de envió de mensaje.

Esta función creada se encuentra dentro del archivo “helpers.go”. Según la fig. 12, se muestra que la función de envió de mensaje se ejecuta sin alterar el goroutine principal implementado a través del waitgroup.

Figura 12

Función para enviar los mensajes y el uso del waitgroup para sincronizar

```
func (app *Config) sendEmail(msg Message) {
    app.Wait.Add(1)
    app.Mailer.MailerChan <-msg
}
```

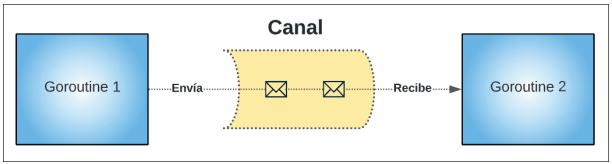
Algoritmo de Message Passing en GO

Una forma de implementar el algoritmo de message passing es usando canales (channels en su término inglés). Estos fueron creados con el propósito de servir como conductores en las que se conectan los goroutines concurrentes, a través de estos, se pueden enviar valores a los canales desde un goroutine y recibir esos valores en otro goroutine. En la fig. 13 se visualiza la representación gráfica del canal en el lenguaje Go.

En la fig 13, se puede observar su implementación en el proyecto de suscripción, primero a través de la creación de goroutines y después mediante los canales.

Figura 13

Funcionamiento del canal en Go.



Este fragmento de código se encuentra en el archivo "main.go". Según la fig. 14, los goroutines fueron implementados en la estructura del presente proyecto de investigación, estos servirán para enviar los mensajes a través de la función *crearcorreo()* en la cual utiliza los canales para la comunicación de los goroutines. Además, la función *crearcorreo()* se procede a crear los respectivos canales. A continuación, se muestra el algoritmo de message passing a través de estos.

Figura 14

Estructura de los goroutines en código

```
// Enviar el mail por channels
app.Mailer = app.crearcorreo()
go app.listenForMail()

go app.listenForShutdown()

go app.listenForErrors()

app.serve()
```

Este fragmento de código se encuentra en el archivo "main.go". Según la fig. 15, se crearon tres canales para la realización de envío de mensajes, el primero *(errorChan)* envía un mensaje en caso ocurra un error, el segundo *(mailerChan)* envía el mensaje y el tercero *(mailerDoneChan)* indica que le mensaje se envió correctamente.

Figura 15

Estructura de los canales en código

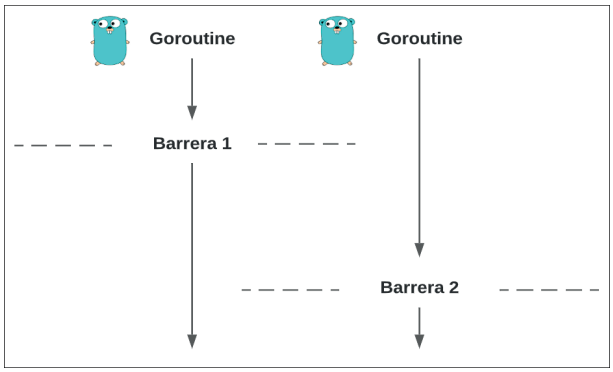
```
func (app *Config) crearcorreo() Mail {
    // crear canales
    errorChan := make(chan error)
    mailerChan := make(chan Message, 100)
    mailerDoneChan := make(chan bool)

    m := Mail{
        Domain: "localhost",
        Host: "localhost",
        Port: 1025,
        Encryption: "none",
        FromName: "Info",
        FromAddress: "concurrencia@unmsm.com",
        Wait: app.Wait,
        ErrorChan: errorChan,
        MailerChan: mailerChan,
        DoneChan: mailerDoneChan,
    }

    return m
}
```

Algoritmo de Barreras Binarias en GO

Se implementó el algoritmo de barreras binarias a través de los canales. En este lenguaje cada proceso envía un mensaje a su canal en cuanto llegue al punto de sincronización y a continuación espera un mensaje en el canal del otro proceso. En la fig. 16 se visualiza la representación gráfica de las barreras binarias en GO.

Figura 16

Funcionamiento de las barreras binarias en GO



A continuación, se puede observar su implementación en el proyecto de suscripción, a través del uso de los canales.

Este fragmento de código se encuentra en el archivo "main.go". Según la fig. 17, se cerraron los canales a través de un asignamiento "*true*" tanto para *app.Mailer.DoneChan* y en *app.ErrorChanDone*, para evitar problemas de sincronización al momento de iniciar nuevamente el aplicativo.

Figura 17

Uso de barreras binarias en la estructura de la aplicación

```
func (app *Config) shutdown() {
    // realizar acciones de limpieza
    app.InfoLog.Println("Limpiar las tareas")

    // bloqueo hasta que ese vacio
    app.Wait.Wait()

    app.Mailer.DoneChan <- true
    app.ErrorChanDone <- true

    app.InfoLog.Println("cerrar los canales y apagar la aplicacion..")

    close(app.Mailer.MailerChan)
    close(app.Mailer.ErrorChan)
    close(app.Mailer.DoneChan)
    close(app.ErrorChan)
    close(app.ErrorChanDone)
}
```

2. Desarrollo con el lenguaje PHP

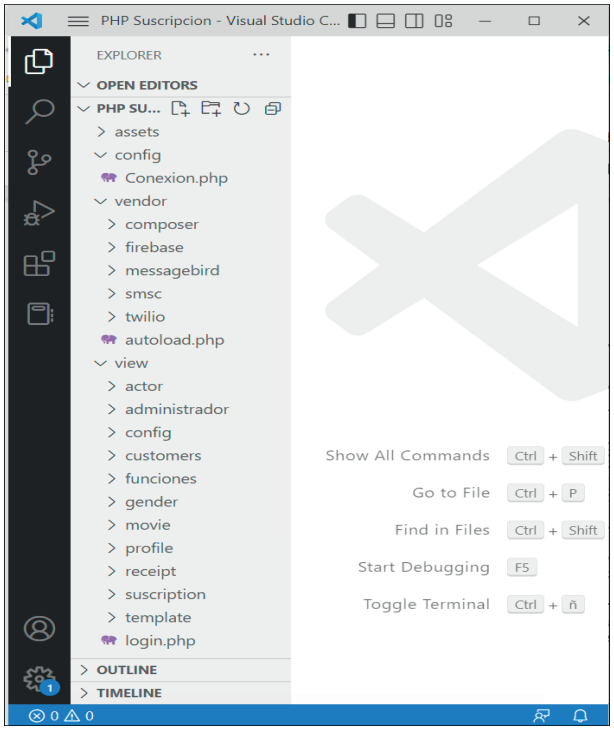
La siguiente plataforma web, con la que se va a comparar el presente proyecto, se ha desarrollado

en el lenguaje PHP utilizando el mismo el mismo IDE en el desarrollo de la aplicación en GO (Visual Code), según **Aguirre, S. (2022)** menciona que Visual Studio Code es un editor de código recomendable para programar en PHP, ya que ayuda a desarrollar proyectos de diferentes lenguajes, además de instalar plugins para la programación y corregir la sintaxis al codificar.

En la fig. 18, se observa la estructura del proyecto comprendido realizado con un patrón de diseño MVC, según Sunardi, A. (2019) menciona que la arquitectura MVC se divide en tres componentes, el Modelo que revela el área lógica, la vista que presenta la interfaz de usuario, y el control que gestiona los cambios en la vista. El proyecto consta de una configuración, en la que se pondrá manualmente la conexión con la base de datos, a su vez cuenta de vistas que contempla las interfaces del presente proyecto. También, en la carpeta vendor se encuentra implementado las librerías y frameworks que se utilizaron en la aplicación y en assets se tiene la hoja de estilos, JavaScript e imágenes del proyecto.

Figura 18

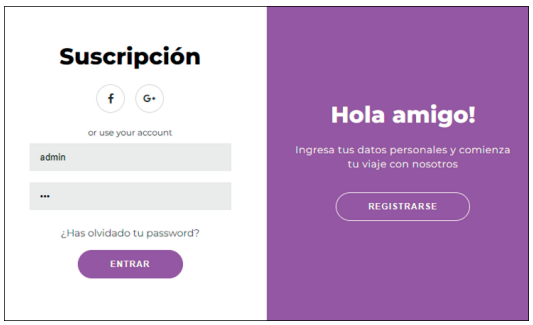
Estructura del proyecto en PHP a través de la herramienta Visual Code



A continuación, se puede observar en la fig. 19 el módulo de acceso del proyecto en PHP. Aquí el usuario puede registrarse o iniciar sesión para proceder con la suscripción.

Figura 19

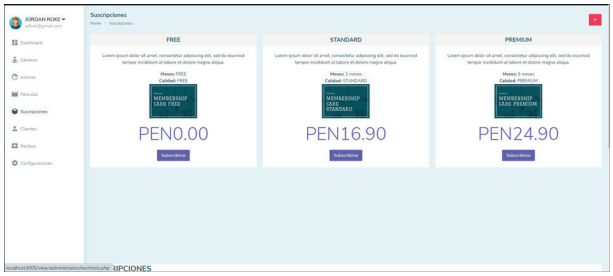
Módulo de acceso del proyecto PHP



Seguidamente, pasamos al módulo suscripción. En la fig. 20 se muestra la elección del plan en la cual cuenta con tres tipos de suscripción.

Figura 20

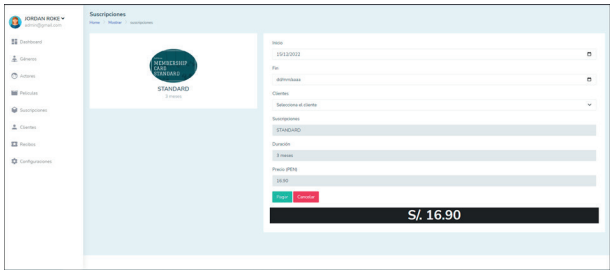
Planes de suscripción del proyecto PHP



Cuando el usuario suscriptor elige el plan de suscripción, se procede a realizar el pago. En la fig. 21 muestra la información del pago de la suscripción previamente seleccionada.

Figura 21

Pago de suscripción del proyecto PHP



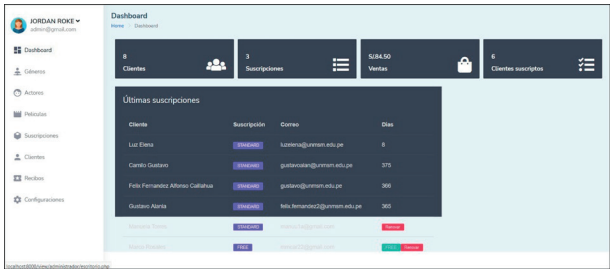
Toda la información que los suscriptores realizan se ven almacenada en el módulo servidor. En la fig. 22 se muestra la gestión de los suscriptores de la plataforma.

En contraste con el proyecto de Go, este proyecto en PHP se ha realizado sin considerar el uso de algoritmos concurrentes, por lo que podría presentar problemas y fallos en la aplicación al momento de

ser utilizado por una gran cantidad de usuarios de forma simultánea o concurrente.

Figura 22

Módulo del servidor del proyecto PHP



G. Procedimiento de testeo con la herramienta JMETER

Para obtener los resultados, se utilizó la herramienta para pruebas de rendimiento JMeter, y el navegador Mozilla Firefox. Según Li, Y. (2019) considera que JMeter es una buena herramienta de testeo, que automatiza pruebas de rendimiento para un conjunto de pruebas complejas, permitiendo su rápida ejecución y reducción del esfuerzo humano. Por ende, con esta herramienta se hizo el testeo de las dos aplicaciones mostradas anteriormente (Aplicaciones desarrolladas en GO y PHP) con el fin de comparar y seleccionar cuál de estas tecnologías analizadas es mejor con relación al rendimiento en el proceso de suscripción.

En la fig. 23 y fig. 24 se muestran la ejecución de testeo de la herramienta JMeter con respecto a la aplicación GO:

Las métricas obtenidas a través del testeo con la herramienta JMeter, permite identificar los respectivos indicadores que a continuación de va a detallar:

Según la tabla 2, se muestran los dos indicadores principales que se consideraron para calcular el rendimiento del proceso de suscripción, de las cuales se han definido el tiempo de respuesta medio y el número de solicitudes. Asimismo, para fines de este estudio, nos enfocamos en los puntajes finales

Figura 23

Testeo con la herramienta JMeter para el proyecto Go

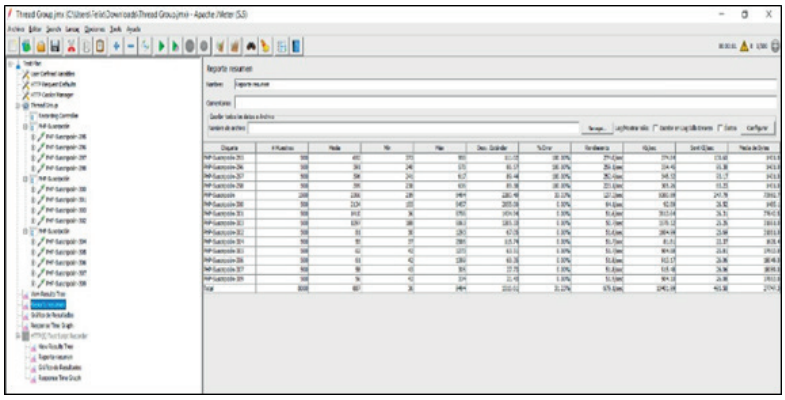
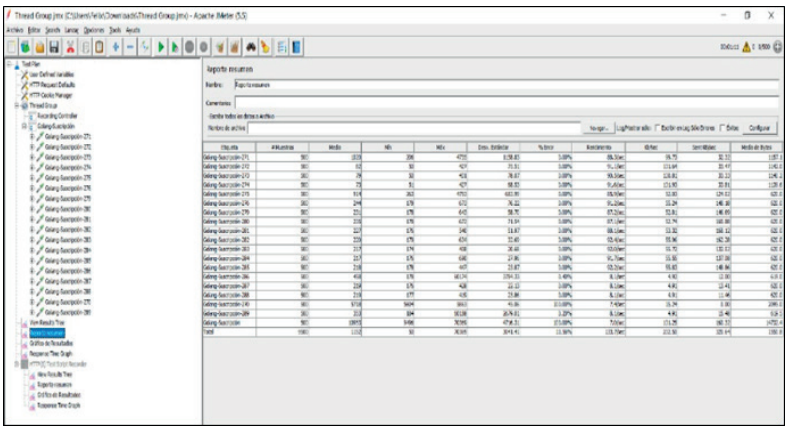


Figura 24

Testeo con la herramienta JMETER para el proyecto PHP



del tiempo mínimo (tiempo de respuesta medio) y el rendimiento de los proyectos de Go y PHP.

En la fig. 25 y fig. 26 se visualiza los puntajes finales con el ingresó 100 solicitudes; luego, en la fig. 27 y fig. 28 se observa los puntajes finales cuando se ingresó 500 solicitudes; finalmente, en la fig. 29 y fig. 30 se observa los puntajes finales cuando se ingresó 1000 solicitudes; la primera para el proyecto en Go, y la segunda, en PHP.

A continuación, en el respectivo artículo científico, su investigación es cualitativa, de las cuales el diseño de investigación es de tipo no experimental que según Hernández-Sampieri y Mendoza (2018), se implementan sin manipular variables, los fenómenos o variables ya ocurrieron (p.149). Además,

Tabla 2

Indicadores implementados en la tecnología web

Indicador	Descripción de la implementación
Media del tiempo de respuesta	Mide el tiempo de respuesta medio en función de elección de la suscripción, ya que son los más propensos a un alto tráfico según el registro de actividad del usuario.
Número de solicitudes	Es el número de usuarios activos que se encontraran accediendo a la plataforma de suscripción.

Figura 25

Testeo con 100 solicitudes de GO

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
Golang-Suscripción-271	100	185	164	237	15.60	0.00%	85.0/sec	96.15	31.14	1157.9
Golang-Suscripción-272	100	56	50	89	5.41	0.00%	98.3/sec	109.67	36.10	1142.1
Golang-Suscripción-273	100	56	49	148	10.72	0.00%	99.0/sec	110.28	36.45	1140.6
Golang-Suscripción-274	100	56	51	95	6.18	0.00%	99.1/sec	110.40	36.58	1140.6
Golang-Suscripción-275	100	313	269	391	24.52	0.00%	80.6/sec	48.79	116.31	620.0
Golang-Suscripción-276	100	216	177	291	19.22	0.00%	85.4/sec	51.71	138.69	620.0
Golang-Suscripción-279	100	217	181	256	17.06	0.00%	82.9/sec	50.16	139.32	620.0
Golang-Suscripción-280	100	218	178	298	19.28	0.00%	82.2/sec	49.75	156.47	620.0
Golang-Suscripción-281	100	217	181	277	17.03	0.00%	82.4/sec	49.87	157.26	620.0
Golang-Suscripción-282	100	220	180	389	25.36	0.00%	78.1/sec	47.30	137.18	620.0
Golang-Suscripción-283	100	219	176	685	49.89	0.00%	70.8/sec	42.85	101.53	620.0
Golang-Suscripción-284	100	218	176	304	19.01	0.00%	71.2/sec	43.12	106.42	620.0
Golang-Suscripción-285	100	217	177	376	25.59	0.00%	71.1/sec	43.03	114.73	620.0
Golang-Suscripción-286	100	217	180	263	18.22	0.00%	73.4/sec	44.42	108.19	620.0
Golang-Suscripción-287	100	217	177	262	18.56	0.00%	74.8/sec	45.29	123.59	620.0
Golang-Suscripción-288	100	216	177	279	18.97	0.00%	77.4/sec	46.86	109.37	620.0
Golang-Suscripción-270	100	5702	5618	5779	42.65	100.00%	14.6/sec	29.82	0.00	2095.0
Golang-Suscripción-289	100	231	186	351	23.85	0.00%	73.7/sec	44.62	140.62	620.0
Golang-Suscripción	100	8999	8785	9386	92.22	100.00%	9.9/sec	142.53	225.61	14736.2
Total	1900	947	49	9386	2261.74	10.53%	188.2/sec	285.05	451.23	1551.2

Figura 26

Testeo con 100 solicitudes de PHP

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
PHP-Suscripción-295	100	505	364	685	94.63	100.00%	65.3/sec	89.31	23.97	1401.0
PHP-Suscripción-296	100	395	242	534	83.76	100.00%	65.7/sec	89.83	24.17	1401.0
PHP-Suscripción-297	100	378	238	531	85.01	100.00%	60.9/sec	83.32	22.48	1401.0
PHP-Suscripción-298	100	393	244	535	88.13	100.00%	54.7/sec	74.89	20.42	1401.0
PHP-Suscripción	300	813	215	2102	627.33	33.33%	81.3/sec	5878.25	158.29	73997.6
PHP-Suscripción-300	100	176	148	259	18.45	0.00%	65.5/sec	95.05	27.12	1486.3
PHP-Suscripción-301	100	59	31	116	18.08	0.00%	71.6/sec	5427.57	36.49	77642.9
PHP-Suscripción-303	100	230	195	312	23.41	0.00%	63.9/sec	1988.13	31.82	31861.0
PHP-Suscripción-302	100	60	52	95	5.77	0.00%	70.6/sec	2195.78	35.15	31861.0
PHP-Suscripción-304	100	43	38	62	4.34	0.00%	71.4/sec	113.96	30.90	1633.8
PHP-Suscripción-305	100	49	44	82	5.25	0.00%	71.2/sec	1245.07	35.52	17913.0
PHP-Suscripción-306	100	49	44	81	5.01	0.00%	71.3/sec	1256.31	35.85	18049.0
PHP-Suscripción-307	100	48	44	60	3.76	0.00%	71.4/sec	1261.79	35.92	18089.0
PHP-Suscripción-309	100	48	44	58	3.47	0.00%	71.7/sec	1250.69	36.08	17853.0
Total	1600	305	31	2102	397.09	31.25%	433.8/sec	11756.50	316.59	27749.1

Figura 27

Testeo con 500 solicitudes de GO

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
Golang-Suscripción-271	500	1020	206	4735	1158.83	0.00%	88.3/sec	99.73	32.32	1157.1
Golang-Suscripción-272	500	82	50	427	75.51	0.00%	91.1/sec	101.64	33.47	1142.0
Golang-Suscripción-273	500	79	50	431	78.07	0.00%	90.5/sec	100.81	33.33	1140.2
Golang-Suscripción-274	500	73	51	427	68.53	0.00%	91.6/sec	101.93	33.81	1139.6
Golang-Suscripción-275	500	914	263	4793	683.99	0.00%	85.9/sec	52.03	124.02	620.0
Golang-Suscripción-276	500	244	179	673	76.22	0.00%	91.2/sec	55.24	148.18	620.0
Golang-Suscripción-279	500	231	178	643	58.70	0.00%	87.2/sec	52.81	146.69	620.0
Golang-Suscripción-280	500	235	178	672	71.54	0.00%	87.1/sec	52.74	165.88	620.0
Golang-Suscripción-281	500	227	176	540	51.97	0.00%	88.1/sec	53.32	168.12	620.0
Golang-Suscripción-282	500	220	179	634	32.69	0.00%	92.4/sec	55.96	162.28	620.0
Golang-Suscripción-283	500	217	174	408	20.68	0.00%	92.0/sec	55.72	132.02	620.0
Golang-Suscripción-284	500	217	176	690	27.96	0.00%	91.7/sec	55.55	137.08	620.0
Golang-Suscripción-285	500	218	178	447	23.87	0.00%	92.2/sec	55.83	148.86	620.0
Golang-Suscripción-286	500	458	178	60174	3784.33	0.40%	8.1/sec	4.92	12.00	619.0
Golang-Suscripción-287	500	219	176	428	22.13	0.00%	8.1/sec	4.91	13.41	620.0
Golang-Suscripción-288	500	219	177	419	23.86	0.00%	8.1/sec	4.91	11.46	620.0
Golang-Suscripción-270	500	5718	5604	5863	45.06	100.00%	7.4/sec	15.24	0.00	2095.0
Golang-Suscripción-289	500	353	184	60188	2679.01	0.20%	8.1/sec	4.91	15.48	619.5
Golang-Suscripción	500	10953	9496	70369	4716.31	100.00%	7.0/sec	101.25	160.32	14732.4
Total	9500	1152	50	70369	3041.41	10.56%	133.7/sec	202.50	320.64	1550.8

Figura 28

Testeo con 500 solicitudes de PHP

Etiqueta	# Muestras	Media	Min	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
PHP-Suscripción-295	500	692	373	958	111.02	100.00%	274.0/sec	374.84	100.60	1401.0
PHP-Suscripción-296	500	391	240	578	86.57	100.00%	259.1/sec	354.45	95.38	1401.0
PHP-Suscripción-297	500	396	241	617	89.44	100.00%	252.4/sec	345.32	93.17	1401.0
PHP-Suscripción-298	500	399	238	636	89.38	100.00%	223.1/sec	305.26	83.23	1401.0
PHP-Suscripción	1500	2366	219	9494	2280.40	33.33%	127.3/sec	9200.99	247.79	73992.7
PHP-Suscripción-300	500	2134	155	6457	2055.00	0.00%	64.1/sec	92.89	26.52	1485.1
PHP-Suscripción-301	500	1410	36	8756	1434.94	0.00%	51.6/sec	3913.64	26.31	77642.5
PHP-Suscripción-303	500	1297	188	6063	1205.33	0.00%	50.7/sec	1578.12	25.26	31861.0
PHP-Suscripción-302	500	81	50	1293	67.05	0.00%	51.6/sec	1604.99	25.69	31861.0
PHP-Suscripción-304	500	55	37	2586	115.74	0.00%	51.7/sec	81.81	22.37	1620.4
PHP-Suscripción-305	500	62	42	1273	63.51	0.00%	51.7/sec	904.88	25.81	17913.0
PHP-Suscripción-306	500	61	42	1399	68.35	0.00%	51.8/sec	913.17	26.06	18049.0
PHP-Suscripción-307	500	58	43	306	27.73	0.00%	51.8/sec	915.48	26.06	18089.0
PHP-Suscripción-309	500	56	42	334	21.43	0.00%	51.9/sec	904.10	26.08	17853.0
Total	8000	887	36	9494	1516.61	31.25%	679.1/sec	18401.99	495.58	27747.3

Figura 29

Testeo con 1000 solicitudes de GO

Etiqueta	# Muestras	Media	Min	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
Golang-Suscripción-271	1000	2516	422	6155	1795.66	100.00%	154.2/sec	210.94	56.46	1401.0
Golang-Suscripción-272	1000	394	236	616	87.58	100.00%	152.9/sec	209.14	56.13	1401.0
Golang-Suscripción-273	1000	392	237	693	92.10	100.00%	146.2/sec	199.97	53.81	1401.0
Golang-Suscripción-274	1000	396	237	679	92.30	100.00%	146.5/sec	200.38	54.06	1401.0
Golang-Suscripción-275	1000	678	270	7263	426.37	0.00%	78.6/sec	47.57	113.41	620.0
Golang-Suscripción-276	1000	229	177	790	52.56	0.00%	85.7/sec	51.86	139.11	620.0
Golang-Suscripción-279	1000	228	179	654	43.92	0.00%	85.3/sec	51.64	143.44	620.0
Golang-Suscripción-280	1000	226	177	682	46.51	0.00%	85.0/sec	51.45	161.82	620.0
Golang-Suscripción-281	1000	287	175	60175	1895.19	0.10%	16.5/sec	9.97	31.45	619.8
Golang-Suscripción-282	1000	226	176	666	38.41	0.00%	16.4/sec	9.96	28.88	620.0
Golang-Suscripción-283	1000	221	178	647	28.09	0.00%	16.4/sec	9.96	23.59	620.0
Golang-Suscripción-284	1000	222	176	693	33.86	0.00%	16.4/sec	9.96	24.57	620.0
Golang-Suscripción-285	1000	220	177	351	20.26	0.00%	16.4/sec	9.96	26.55	620.0
Golang-Suscripción-286	1000	223	177	373	24.30	0.00%	16.4/sec	9.96	24.26	620.0
Golang-Suscripción-287	1000	222	179	344	22.63	0.00%	16.4/sec	9.96	27.17	620.0
Golang-Suscripción-288	1000	223	178	370	23.62	0.00%	16.4/sec	9.95	23.22	620.0
Golang-Suscripción-270	1000	6029	5611	11052	655.67	100.00%	15.1/sec	30.83	0.00	2095.0
Golang-Suscripción-289	1000	388	184	60183	1955.93	0.10%	15.3/sec	9.26	29.20	619.8
Golang-Suscripción	1000	13328	10855	75582	3288.96	100.00%	13.1/sec	201.88	298.83	15758.5
Total	19000	1402	175	75582	3298.32	31.59%	249.2/sec	403.75	597.66	1658.8

Figura 30

Testeo con 1000 solicitudes de PHP

Etiqueta	# Muestras	Media	Min	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
PHP-Suscripción-295	1000	4753	478	26537	5592.79	100.00%	36.6/sec	56.28	11.89	1573.3
PHP-Suscripción-296	1000	543	236	11313	833.54	100.00%	36.6/sec	50.13	13.43	1404.0
PHP-Suscripción-297	1000	397	238	669	89.03	100.00%	36.6/sec	50.13	13.53	1401.0
PHP-Suscripción-298	1000	396	239	729	90.63	100.00%	36.7/sec	50.17	13.68	1401.0
PHP-Suscripción	1000	4504	214	27720	5364.98	36.27%	98.3/sec	7091.32	188.53	73907.7
PHP-Suscripción-300	1000	2904	147	24840	4452.93	7.00%	35.3/sec	54.52	13.58	1583.0
PHP-Suscripción-301	1000	1817	29	20268	2018.63	0.00%	37.6/sec	2848.87	19.15	77642.5
PHP-Suscripción-303	1000	2165	184	23565	2346.27	1.80%	37.5/sec	1148.36	18.35	31339.5
PHP-Suscripción-302	1000	154	51	6111	491.71	0.00%	38.6/sec	1200.72	19.22	31861.0
PHP-Suscripción-304	1000	118	37	5880	344.54	0.00%	38.4/sec	60.54	16.62	1613.9
PHP-Suscripción-305	1000	81	43	6940	274.96	0.00%	38.4/sec	671.91	19.17	17913.0
PHP-Suscripción-306	1000	63	42	2639	118.96	0.00%	38.4/sec	677.04	19.32	18049.0
PHP-Suscripción-307	1000	58	42	1605	59.56	0.00%	38.4/sec	678.49	19.32	18089.0
PHP-Suscripción-309	1000	58	43	2474	83.96	0.00%	38.4/sec	669.68	19.32	17853.0
Total	16000	1689	29	27720	3562.96	32.35%	524.0/sec	14182.64	377.06	27715.4

es transeccionales o transversales, porque la medición es en un tiempo único (Hernández-Sampieri & Mendoza, 2018, p.149) y finalmente es descriptivo porque indagan las incidencias de las modalidades, categorías o niveles de una o más variables en una población; son estudios puramente descriptivos (Hernández-Sampieri & Mendoza, 2018, p.178).

III. RESULTADOS

En la fig. 31 y fig. 32 se muestran los resultados realizados con la herramienta JMeter teniendo como variable independiente el número de solicitudes entrantes y como variables dependientes al rendimiento y tiempo de respuesta media, en ambos se puede observar que mientras más solicitudes son ingresadas, mayor es la carga en la tecnología web, alcanzando el proyecto PHP el punto máximo en el número de procesos por segundo (rendimiento).

En la fig. 33 y fig. 34 se puede observar la comparación entre las dos tecnologías tomando a las variables tiempo de respuesta medio, rendimiento y número de solicitudes, de estos se deduce que el número de solicitudes es directamente proporcional al tiempo de respuesta, mas no al rendimiento. Además, se observa que el proyecto PHP es más susceptible al número de solicitudes que el proyecto GO.

Según la tabla 3 se detalla los resultados aplicando JMeter en donde indica que usando el lenguaje Golang, el tiempo de respuesta medio es menor, a comparación de PHP donde se incrementa significativamente el tiempo de respuesta medio debido al número de procesos del sistema. Esto sucede a que PHP solo hace uso de un solo hilo, en cambio en la implementación de Go se agregó diferentes goroutines (equivalente a hilos) añadiendo concurrencia en la plataforma de suscripción.

Figura 31

Resultado del testeo general de la herramienta JMeter con Golang

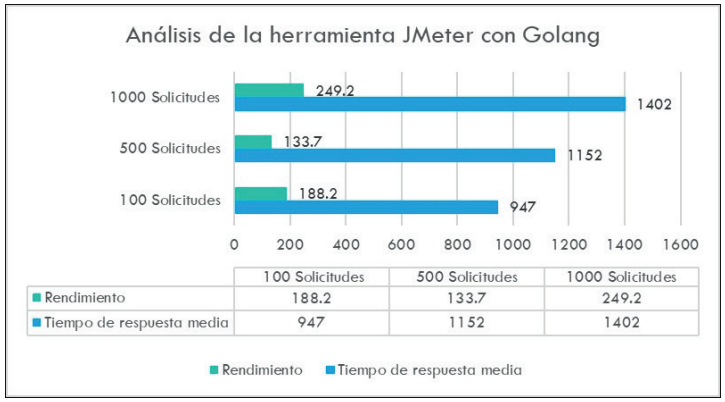


Figura 32

Resultado del testeo general de la herramienta JMeter con PHP

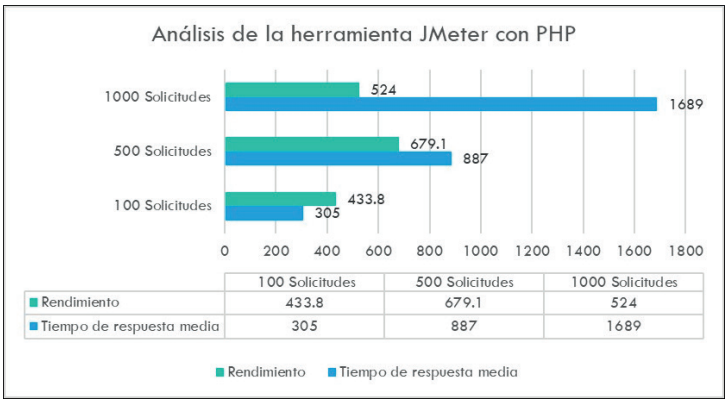


Figura 33

Resultado del testeo del tiempo de respuesta medio de la herramienta JMeter para los proyectos PHP y Go

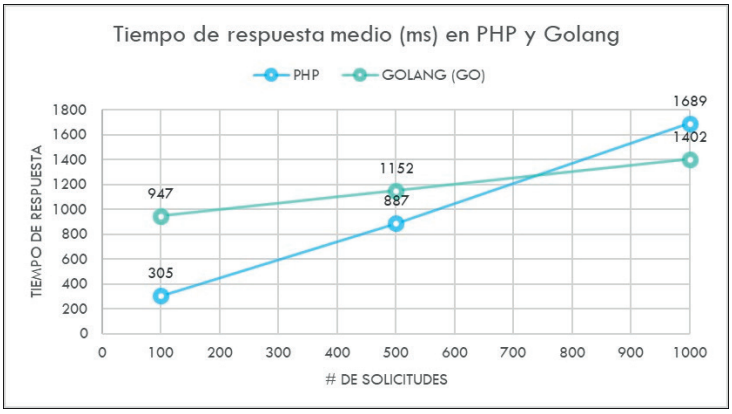


Figura 34

Resultado del testeo de rendimiento de la herramienta JMeter para los proyectos PHP y Go

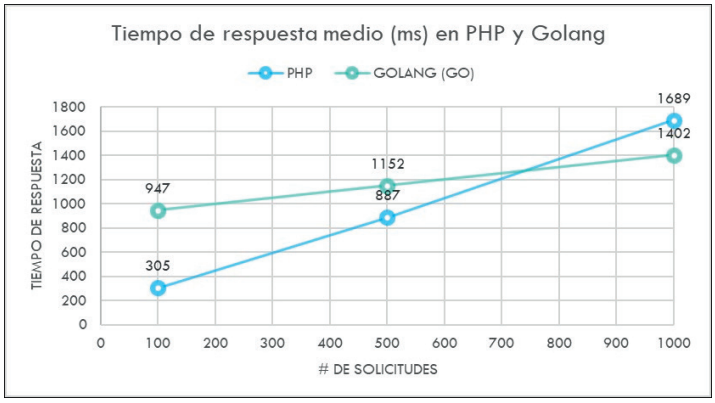


Tabla 3

Resultados del tiempo de respuesta medio en base al número de solicitudes usando PHP y Golang

Número de Solicitudes	Tiempo de respuesta medio (ms)	
	PHP	GOLANG (GO)
100	305 ms	**947 ms**
500	887 ms	**1152 ms**
1000	1689 ms	**1402 ms**

De igual forma, según la tabla 4 se observa que el rendimiento de Golang (GO) es mejor que en PHP, debido a que este último se encuentra procesando una mayor cantidad de procesos, en contraste de Golang donde el número de procesos es menor porque aplica la concurrencia generando el uso menor de recursos para el sistema, y que sea significativamente más rápido y eficiente en la plataforma de suscripción.

Tabla 4

Resultados del rendimiento en base al número de solicitudes usando PHP y Golang

Número de Solicitudes	Rendimiento (Procesos/segundo)	
	PHP	GOLANG (GO)
100	433.8	**188.2**
500	679.1	**133.7**
1000	524.0	**249.2**

El resultado demuestra que por medio de la comparación de los algoritmos concurrentes en la plataforma web, tenemos un algoritmo desarrollado en Go (Sincronización, Message Passing y Barreras binarias), que al ejecutar el testeo, indica que con 1000 solicitudes para el proceso de suscripción se obtiene un tiempo de respuesta medio de 1402 ms con un rendimiento de 249.2 procesos por segundo en cambio el otro algoritmo desarrollado en PHP, que al ejecutar el testeo, indica que con 1000 solicitudes para el proceso de suscripción se obtiene un tiempo de respuesta medio de 1689 ms con un rendimiento de 524 procesos por segundo.

IV. CONCLUSIONES

Se evidencia en los resultados que Golang (GO) es un lenguaje eficiente para incluir en el desarrollo de plataformas que tienen como necesidad el acceso de una gran cantidad de usuarios al mismo tiempo. Además, Golang (GO) es un lenguaje que permite codificar fácilmente algoritmos concurrentes y esto se demuestra en este estudio, donde incluye tres algoritmos fundamentales tales como la sincronización, message passing y barreras binarias, estos ayudan en la optimización del rendimiento de las plataformas de servicios de streaming.

También, se pudo evidenciar el nivel del rendimiento respecto a dos lenguajes de programación PHP y Golang, en donde mediante el uso de algoritmos concurrentes; Golang es mejor; y esto es evidenciado con el testeo de la herramienta JMeter, ya que mientras más aumenta el número de procesos mayor es el tiempo de respuesta medio. El resultado implementado con Go indica que con 1000 solicitudes para el proceso de suscripción se obtiene el tiempo de respuesta medio de 1402 ms con un rendimiento de 249.2 procesos por segundo.

Por último, se evidencia que el uso del algoritmo concurrente desarrollado en Go bajo la plataforma web, optimiza el proceso a través de sus indicadores

de tiempo de respuesta medio y rendimiento para la suscripción en los servicios de streaming a través de la Sincronización, Message Passing y Barreras binarias.

V. REFERENCIAS

[1] Aguirre, S. (2022). *PHP Avanzado Base de Datos - Vol.1: PDO. Encripactión. Sistema de Login.* (1° ed.., Vol 1). Editorial Claudio Peña.

[2] Andrawos, M., & Helmich, M. (2017). *Cloud Native Programming with Golang: Develop microservice-based high performance web apps for the cloud with Go.* Packt Publishing. https://books.google.at/books?id=NvNFDwAAQBAJ.

[3] Bao, H., Cao, B., Xiong, Y., & Tang, W. (2020). Digital media's role in the COVID-19 pandemic. *JMIR MHealth and UHealth*, *8*(9), e20156. https://doi.org/10.2196/20156.

[4] Bodner, J. (2021). *Learning Go: An Idiomatic Approach to Real-world Go Programming.* Editorial O'Reilly.

[5] Burgues, J. E. G. (2018). *Aprende a Modelar Aplicaciones Con UML - Tercera Edición (It Campus Academy, Ed.)*. Createspace Independent Publishing Platform.

[6] Castro, M. (2017). *Go Design Patterns.* Packt Publishing. https://books.google.at/books?id=BVQoDwAAQBAJ

[7] Datum Internacional S.A. (2020). *Informe de Encuesta de Opinión Pública a nivel nacional Septiembre - Octubre 2020.* http://admin.datum.com.pe/datum/descarga/20201011221123.pdf

[8] Díez de Paz, L. (2021). *El streaming como forma de entretenimiento = Streaming as an entertainment source.*

[9] Duda, J., & Dlubacz, W. (2019). Improving efficiency of aweb-baseddistributed evolutionary computing system. *Proceedings of the International Conferences Big Data Analytics, Data Mining and Computational Intelligence 2019; and Theory and Practice in Modern Computing 2019* , , 143-150. https://doi.org/10.33965/tpmc2019_201907L018

[10] Galli, R. (2015). *Principios y algoritmos de concurrencia.* Ricardo Galli. https://books.google.at/books?id=cLXfCQAAQBAJ

[11] Hernández-Sampieri, R & Mendoza, C. (2018). *Metodología de la Investigación: las rutas cuantitativas, cualitativas y mixta.* México D.F.: McGRAW-HILL.

[12] Iglesias, E. L. (2022). Consumption prediction on Netflix: Audience tracking analysis based on the recommendation algorithm in times of pandemic. En *Predictive Technology in Social Media* (1st Edition, pp. 52–74). CRC Press.

[13] Jimenez, J. (2020). *Diseño y desarrollo de una aplicación web sobre lenguaje etiquetas en HTML para un portal de acompañamiento a la crianza.*[Trabajo de pregrado, Universidad Cooperativa de Colombia]. Repositorio Institucional UCC. https://repository.ucc.edu.co/handle/20.500.12494/20076

[14] Liu, J., Zhang, S., Wang, Q., & Wei, J. (2022). *Coordinating fast concurrency adapting with autoscaling for SLO-oriented web applications. IEEE transactions on parallel and distributed systems: a publication of the IEEE Computer Society,* 33(12), 3349–3362. https://doi.org/10.1109/tpds.2022.3151512

[15] Li Y. (2019). *Design and implementation of an automated testing system based on pytest and JMeter.* Collection, 1.

[16] Llanes, M. S. (2021). *El impacto de los servicios de streaming. Puantástica,* (1), 89-99.

[17] López Delgado, D. (2018). *Estudio de las plataformas de streaming*. https://idus.us.es/handle/11441/87550

[18] Moreno, G. (2022). *Programación paralela y distribuída. Arquitecturas paralelas y distribuídas. Programación de aplicaciones multiproceso.* [Trabajo fin de Máster, Universidad de Jaén]. https://tauja.ujaen.es/handle/10953.1/17697.

[19] Netflix (2021). *A cooperative approach to content delivery.* https://openconnect.netflix.com/Open-Connect-Briefing-Paper.pdf

[20] Oliveira, A., Azevedo, A., & Maria da Silva, S. (2020). *Streaming services consumer behaviour: A Netflix user case study in Brazil and Portugal.* Proceedings of the 17th

[21] Oliveira, A. D. F. (2019). *Comportamento de consumidores de serviços de streaming: um estudo de caso de usuários da Netflix no Brasil e em Portugal* (Doctoral dissertation).

[22] Powers, D. (2022). *What Is PHP 8? En PHP 8 Solutions (pp. 1–7).* Apress. DOI: 10.1007/978-1-4842-7141-4_1.

[23] Raiturkar, J. (2018). Hands-On Software Architecture with Golang: Design and architect highly scalable and robust applications using

Go. Packt Publishing. https://books.google.at/books?id=fmd-DwAAQBAJ

[24] Santiago, M. (2019). *Innovaciones tecnológicas en las ciencias computacionales*. Editorial Montiel & Soriano Editores S.A. de C.V. ISBN: 978-607-7512-97-4

[25] Sherlock Communications. (2021). *Informe de Consumo de Streaming 2021.* https://mcusercontent.com/4b2c98cfb207cdaae9e24e227/files/7e8cb872-2229-7452-9e2c-2b5fd455adb6/EBOOK_STREAMING_2021_SPA.pdf

[26] Solayman, H. S., AL thanoon, A., & Aldabagh, G. (2021). *Message passing applications: A review. IRAQI JOURNAL OF STATISTICAL SCIENCES,* 18(34), 22–39. https://doi.org/10.33899/iqjoss.2021.169963.

[27] Sunardi, A., & Suharjito. (2019). *MVC architecture: A comparative study between laravel framework and slim framework in freelancer project monitoring system web based.* Procedia Computer Science, 157, 134–141. https://doi.org/10.1016/j.procs.2019.08.150.

[28] Wang Q., Chen H., Zhang Z., Hung L., & Palanizamy B. (2019, 1 abril). *Integrating Concurrency Control in n-Tier Application Scaling Management in the Cloud.* IEEE Journals & Magazine | IEEE Xplore. DOI: 10.1109/TPDS.2018.2871086

Fuentes de financiamiento:

Propia.

Conflictos de interés:

Los autores declaran no tener conflictos de interés.